Modeling, Simulation and Control

Finn Aakre Haugen
TechTeach

http://techteach.no/books/modsimcon

ISBN 978-82-91748-20-7

Edition 10 February 2024

Contents

Preface

The main topic of this book is automatic control – how to use controllers, which in practice are special computers – to automatically manipulate mechanical, thermal, chemical, or electrical procesess so that they behave as you want, i.e. they follow specified references (setpoints) despite disturbances from the process environment. Some examples:

- Cruise control of a car: The controller adjusts automatically the power acting on the wheels to make the measured car speed equal to the speed reference, e.g. 80 km/h, despite varying road and air friction and varying road slope (uphill, downhill).
- Temperature control of a room: The controller adjusts automatically the heating or cooling of the air in the room to make the measured temperature in the room equal to the temperature reference, e.g. 22 °C, despite disturbances as varying outdoor temperature, varying sun irradiance, and varying number of persons in the room.
- Level control of the a wood chips storage tank in a paper factory: The controller adjust automatically the inflow rate of wood chips to the tank to make the measured level of the wood chips in the tank equal to the level reference, e.g. 10 meter in a 15 meter tall tank, despite disturbances as varying wood chips density and varying consumption of chips out of the tank.
- Position control (so-called dynamic positioning) of ships: The controller adjusts automatically the propeller and thuster forces and the rudder angle to make the measured ship position equal to the position reference, e.g. over a subsea installation or in a specified distance from a platform, despite disturbances as forces from wind and water current.

These and lots of other examples indicate that automatic control is of crucial importance in industrial and other kinds of technical systems.

In my own practical control projects, I have had great use of mathematical models, mainly as the basis for building (programming) simulators. With a simulator you can design, analyse, and test your systems without exhaustive and perhaps dangerous experimentation on the physical system; the physical system may even not exist! Furthermore, with models you can implement methods for monitoring (state and parameter estimation), you can design advanced, model-based controllers, and you can carry out model-based tuning of standard PID controllers[1]. Models and simulators are simply great engineering tools for

[1] Proportional-Integral-Derivative controllers – the standard automatic control function in industry

control. So, I decided to name this book “Modeling, simulation and control”.

You will learn algorithms for simulation, control, filtering, and estimation which you can program yourself in an appropriate programming environment.

The book may serve as a textbook in bachelor courses and master courses in automatic control, and as a reference book for professionals.

You can find problems with detailed solutions at the end of each chapter.

The book contains simulated plots from Python programs, from OpenModelica models, and from SimView simulators. There are links to Python programs and OpenModelica models and SimView simulators in the text. Python and OpenModelica are open software. SimView is a collection of executable files (exe files) which I have developed in LabVIEW[2]. The Python programming language, OpenModelica, and SimView are introduced in respective appendices in the book.

Information about the style of mathematical symbols used in the book is in Appendix 37.

The book exists only in pdf file format.

The book is freely available from its home page on http://techteach.no/books/modsimcon, but a modest payment is welcome (see the home page for more information).

If you see errors or have suggestions or other comments about the book, you are welcome to send them to me in email.

A few words about my background: I have a MSc degree from former Norwegian Institute of Technology (Norwegian: Norges tekniske høgskole) and a PhD degree from former Telemark University College (Norwegian: Høgskolen i Telemark). I have experience as university teacher and researcher, textbook author, and participant in industrial and research projects about modeling, simulation and control. I work in my one-person firm TechTeach, and have part-time positions as professor[3] at the University of South-Eastern Norway and at OsloMet (Oslo Metropolitan University). I am also teaching in vocational education, and I give courses for the industry.

I enjoy the field of modeling, simulation and control and the development and programming of algorithms needed to implement theoretical methods on simulators and ultimately on practical systems. Without that enjoyment, there would not be a book.

Finn Aakre Haugen
http://techteach.no/fh
finn@techteach.no

[2] but you do not need LabVIEW to run the simulators

[3] Norwegian title: dosent

Part I

INTRODUCTION

Chapter 1

A flash course in automatic control

With this flash course in automatic control I want to introduce right away the basic principles and practical methods and physical components of automatic control systems. It should give you a good overview over the field, and hopefully make you motivated to read on to learn more – much more – about both methods of automatic control.

1.1 What can be obtained with control?

Figure 1.1 compares typical results of "passive" control with "active" control. In the figure, the term *control error* is defined as follows:

Control error (e) = Difference between reference value (r) and actual value of the process variable (y)

Or, briefly:

$$e = r - y \tag{1.1}$$

Figure 1.1 illustrates the following:

- ***Passive control***, i.e. the process is excited with a fixed or constant control signal. With passive control, *the control error may be too large.*
- ***Active control***, i.e. the process is manipulated actively with either a manual control signal generated by a human or an automatic control signal generated by an automation device – typically a computer. *With active control, the control error may be kept within specified limits – ideally zero.*

Automatic control is important in a large number of practical industrial and technical systems. With automatic control we may obtain:

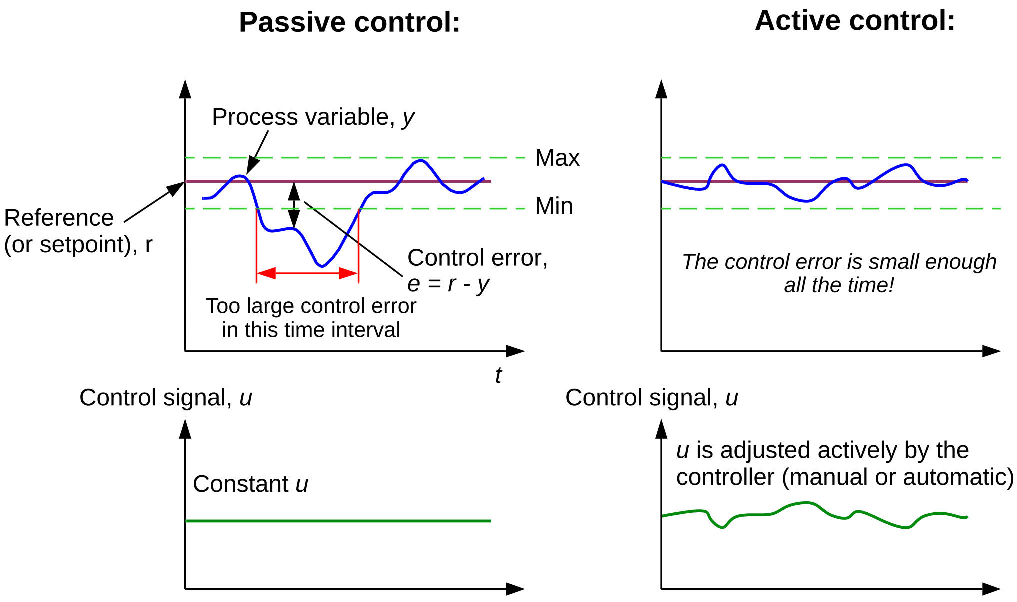


Figure 1.1: Active control can ensure that the control error is small enough.

- **Good product quality**: A product will have acceptable quality only if certain process variables are sufficiently close to their references. One example: In artifical (chemical) fertilizers the pH value and the composition of Nitrogen, Phosphate and Potassium are factors which express the quality of the fertilizer (for example, too low pH value is not good for the soil). Therefore the pH value and the compositions must be controlled.

- **Good production economy**: The production economy will become worse if part of the products has unacceptable quality so that it can not be sold. Good control may maintain the good product quality, and hence, contribute to good production economy. Further, by good control it may be possible to tighten the limits of the quality so that a higher price may be taken for the product!

- **Safety**: To guarantee the security both for humans and equipment, it may be required to keep variables like pressure, temperature, level, and others within certain limits– that is, these variables must be controlled. Some examples:

  - An aircraft with an autopilot (an autopilot is a positional control system).
  - A chemical reactor where pressure and temperature must be controlled.

- **Environmental care**: The amount of poisons to be emitted from a factory is regulated through laws and directions. The application of control engineering may help to keep the limits. Some examples:

  - In a wood chip tank in a paper pulp factory, hydrogene sulfate gas from the pulp process is used to preheat the wood chip. If the chip level in the tank is too low, too much (stinking) gas is emitted to the atmosphere, causing pollution. With level control the level is kept close to a desired value (set-point) at which only a small amount of gas is expired.

  - In the so-called washing tower nitric acid is added to the intermediate product to neutralize exhaust gases from the production. This is accomplished by controlling the pH value of the product by means of a pH control system. Hence, the pH control system ensures that the amount of emitted ammonia is between specified limits.
  - Automatically controlled spray painting robots avoid humans working in dangerous areas.

- **Comfort**:
  - The automatic positional control which is performed by the autopilot of an aircraft to keep a steady course contributes to the comfort of the journey.
  - Automatic control of indoor temperature may give better comfort.
- **Feasibility**: Numerous technical systems could not work or would even not be possible without the use of control engineering. Some examples:
  - An exothermal reactor operating in an unstable (but optimal) operating point
  - Launching a space vessel (the course is stabilized)
  - A dynamic positioning system holds a ship at a given position without an anchor despite the influence of waves, wind and current on the ship. The heart of a dynamic positioning system is the positional control system which controls the thrusters which are capable of moving the ship in all directions.
- **Automation**: Computers and other kinds of hardware and software implementing control solutions can accomplish tedious and dangerous operations for the benefit of human operators. Also, automation may reduce costs in a factory, thereby indirectly reducing product prices, to customer's benefit.

1.2 Process variables that are typically controlled

Below is a list of process variables which typically are controlled to follow their references (or setpoints):

- **Level** (in a storage tank)
- **Temperature** (in a room; in the fluid passing a heat exchanger; in a reactor; in a greenhouse)
- **Flow** (of feeds into a reactor)
- **Pressure** (of gas in an oil-water-gas separator)
- **Chemical composition** (of nitric acid; fertilizers)

- **Position** (of a ship; a painting robot arm; the tool of a cutting machine; a rocket)
- **Speed** (of a motor; a car; a fan)

1.3 Feedback control

1.3.1 Manual feedback control

Let's start with a shower!

Imagine that you are to take a shower, and you want to have the temperature of the shower water as you desire, see Figure 1.2. The figure also shows responses, as explained below.

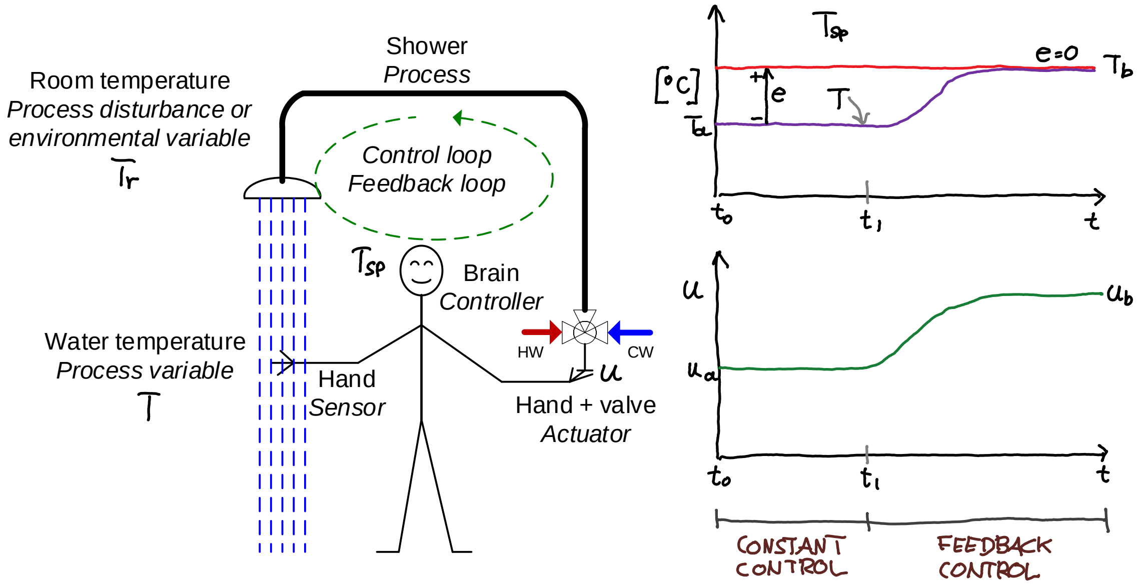


Figure 1.2: Controlling the water temperature of a shower. (HW = hot water. CW = cold water.)

In control terminology, the shower is here the *process*. The water temperature, T, is the *process variable* which we want to follow its desired temperature – the temperature *reference* or *setpoint*, r_T. The room temperature, T_r, makes an impact on T, and we can therefore say that T_r is a *process disturbance*. To control T, you can use your hand to manipulate a mixing valve which sets the ratio between the hot and cold water flows. The hand and the valve constitute the *actuator*. Let u be the setting of the mixing value.

In the shower example, the control error is

$$\text{control error} = \text{temperature reference} - \text{actual temperature} \tag{1.2}$$

or, using symbols,

$$e = r - T \tag{1.3}$$

Typically, the aim of control is $e = 0$, or $e \approx 0$ in practice as e will inevitably vary somewhat.[1]

Let us consider the following two alternative temperature control strategies:

- ***Passive control:*** Consider the time interval between t_0 and t_1 in Figure 1.2. Assume you took a shower yesterday. The shower temperature was as you desired, i.e. $T = r_T$. Say that a valve setting of $u = u_a$ gave the desired temperature. At time t_0 (today) you enter the shower. Naturally, you try the successful setting $u = u_a$ also today. If T_r is the same as yesterday, using $u = u_a$ gives $T = r_T$ also today. But assume that T_r is actually lower today than yesterday, maybe because you take the shower with an open window today while the window was closed yesterday. Consequently, $u = u_a$ gives $T = T_a$ which is less than r_T, or in other terms: $e > 0$. Assume that keeping $u = u_a =$ constant makes you freeze, and that you conclude that constant control is not a good control strategy. This makes you step aside of the cold water for a moment to think about a better strategy.

- ***Active, error-driven control, or feedback control:*** At time t_1, while freezing, you decide to improve the control to make T reach r_T, or in other terms, to obtain $e = 0$. How would you improve the control? I guess you decide to measure the temperature with say your right hand, which is a *sensor*. That measurement, also denoted T here as we assume it represents the actual temperature, is detected in your brain. Your brain – the *controller* – then adjust the nerve signal to the left hand to change the setting, u, of the mixing valve until $e \approx 0$, and then you are ready to take the shower – with the desired temperature. Since it the control error that "drives" the control, we can denote this control strategy as error-driven control. However, a more common term is feedback control, which is explained below. Also, the term closed loop control is used, and for constant control, open loop control is an alternative term.

This excellent temperature control system is a *manual* control system since *you* (a human being) are the controller. In an *automatic* control system you are replaced by a computer which can generate e.g. an electric control signal to the valve, and the sensor with an industrial temperature sensor, for example a Pt100 sensor which sends an electric signal to the controller. More details about an automatic control system for the shower are given in Section 1.3.2.

In Figure 1.2, I have indicated the *control loop*. It consist of the following three main components needed to control the process. These are the main components also in purely technical control loops.

- Sensor
- Controller
- Actuator

[1] However, in so-called averaging level control of buffer tanks, we actually want a relatively large e for the level, as this makes the tank compliant to inflow variations. Averaging level control is described in Ch. 16.4.

Sometimes also a measurement filter, which attentuates measurement noise, is included as component (following the sensor) in the control loop. Alternatively, the filter can also be regarded as a part of (included in) the sensor.

As I mentioned above, control loops are also denoted feedback (control) loops. The reason for this name is that there is a connection from the process output, which is the water temperature, *back* to the process input, which is the control signal to the valve, via the sensor and the controller.

An alternative term to feedback control, is error-driven control as it is the control error that "drives" the control action.[2] It is tempting to use the term "measurement-driven control". However, that term is ambiguous because also feedforward control, cf. Section 1.4, and not only feedback control is driven by measurements.

1.3.2 Automatic feedback control

Maybe you dream about an *automatic* temperature control system of your shower. In that system, you only need to specify the reference (setpoint) of shower outlet temperature, and a controller manipulates the valve *automatically* to make the actual (real) temperature become equal to the reference. Figure 1.3 shows a possible implementation of such an automatic temperature control system.

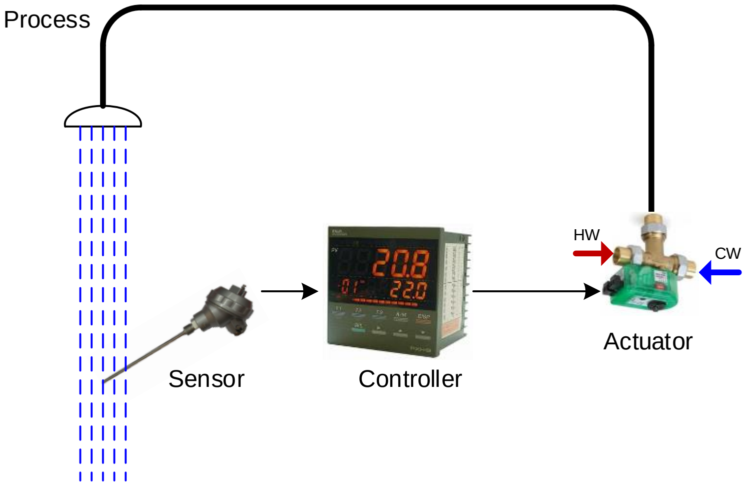


Figure 1.3: The shower water temperature control system implemented with only technical components. (Sensor: Autek. Controller: Fuji. Valve: Taco.)

The components shown in Figure 1.3 are presented briefly below.

[2] Personally, I like the term error-driven control better than the term feedback control since it expresses better the principle of control systems.

Actuator

The actuator is an electronically controlled three-way valve with two inlets, namely hot water and cold water, and one outlet, namely mixed water. Generally, it is typical that the actuator is manipulated by the controller with an electrical current in the range 0-20 mA generated by the controller.

Sensor

In Figure 1.3, the temperature sensor, which measures the water temperature out of the shower, is a Pt100 sensor. The measuring principle of a Pt100 sensor is that the resistance value of the electric resistor, which is made of Platinum, varies in a known way with the temperature. Included in the sensor is a temperature transmitter. The transmitter detects the resistance value, and generates typically either an electrical current or a voltage representing the temperature.

Controller

The controller in 1.3 is an industrial process controller containing a computer which executes a program implementing the control function. The measurement signal from the sensor is connected to the controller. You can set the temperature reference, r, using buttons on the front panel of the controller. The controller adjusts the control signal, u, to the valve automatically to make the measured temperature, y, become equal to r – without any human interaction. Thus, the control system shown in Figure 1.3 is an automatic temperature control system.

Piping & Instrumentation Diagram – P&ID

In the industry, it is common to document the structure of control systems with Piping & Instrumentation Diagrams (P&I Ds). Figure 1.4 shows a Piping & Instrumentation Diagram (P&I D) of the shower water temperature control system. P&I Ds are described in more detail in Appendix 2.

In Figure 1.4:

- TT = Temperature Transmitter, which is the standard letters of temperature sensors. A transmitter is actually not a sensor, but a device which sends the measurement signal, which in general may be electric or pneumatic or hydraulic or digital, to the controller. Still, TT here represents the temperature sensor.
- TC = Temperature Controller.
- The actuator – a mixing valve – is shown with a representative symbol. There is no general symbol for actuator. If the actuator were a pump, a pump symbols should be shown.

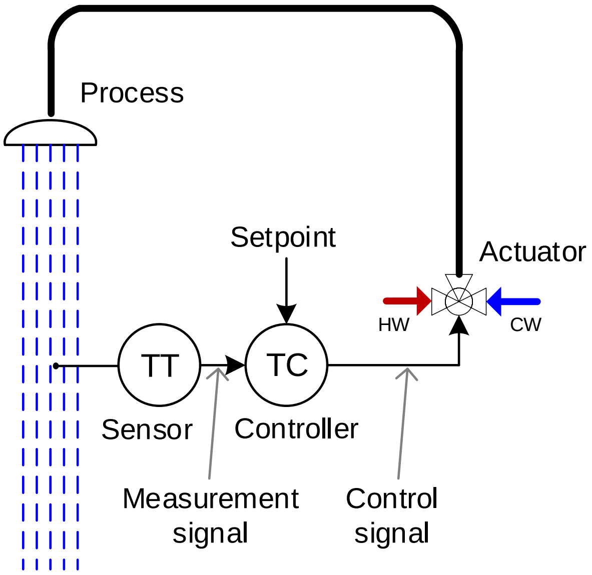


Figure 1.4: Piping & Instrumentation Diagram (P&I D) of the temperature control system of the shower.

- Process flows in e.g. pipelines are drawn with relatively thick lines.
- Signals, as measurement signals and control signals, are drawn with relatively thin lines. If necessary, you can use special dashes to indicate the signal type, i.e. electric, digital, etc. A line without dashes does not indicate any special signal type – it is just a signal.
- Strictly, signal lines are not drawn with arrow heads, just pure lines. I draw with arrow heads in this book because it makes the diagram easier to read, although it breaks with the standards of P&I Ds.

Block diagram of feedback control loop

In many contexts, e.g. in teaching, it is useful to draw a block diagram of a control system. Block diagrams are useful for showing the various variables (signals) and the components of a control system. The level of detail of block diagrams may vary, depending on what to show in the diagram. There are no specific standards for block diagrams.

Figure 1.5 shows a block diagram of the temperature control system of the shower. General control-related terms are used, with terms specific to the temperature control system in parenthesis. Note the three components needed to implement any feedback control system: Controller, actuator and sensor.

Some comments to Figure 1.5:

- The controller as a physical component is represented with a frame with a dashed line, while the controller function is represented with the blue block inside the dasjed

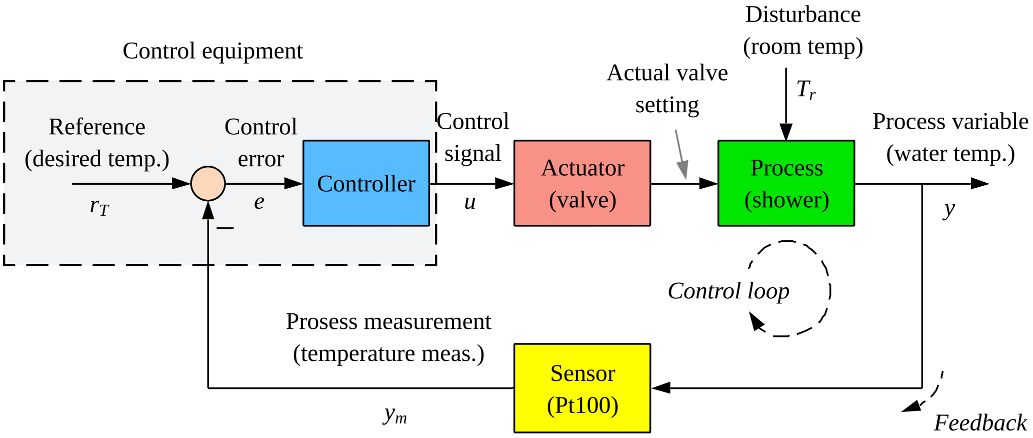


Figure 1.5: Block diagram of the temperature control system of the shower.

frame. Often the dashed frame representing the (physical) controller is not drawn in block diagrams.

- The circle to the left is an adder. The negative sign indicates the measurement enters the adder with a negative sign. Therefore, the output of the adder is reference minus measurement, i.e. the control error, $e = r - y_m$.
- The symbol y_m represents the temperature measurement signal. However, if it is assumed that the measurement gives a precise representation of the process variable, y, the same symbol, y, may be used also for the measurement.

Let's take a look at an industrial example of a control system.

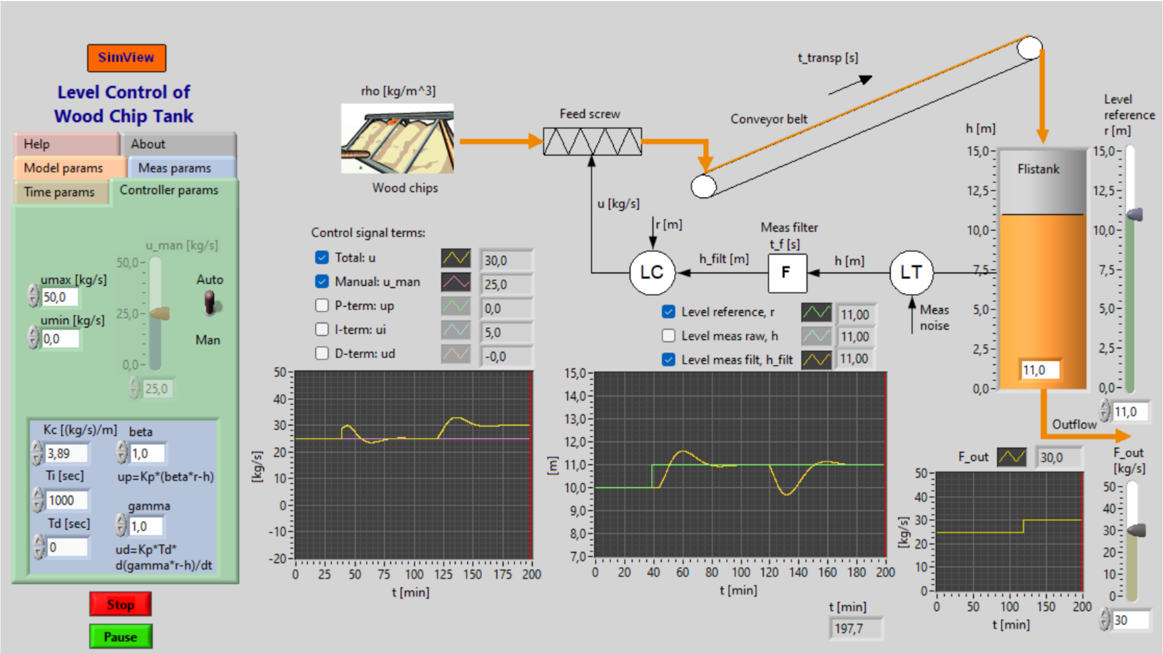
Example 1.1 *Level control of a wood chips tank*

Figure 1.6 shows the front panel of a simulator of a real level control system of a wood chips tank with feed screw and conveyor belt. The belt runs with constant speed. The tank is[3] in the production line of a paper pulp producing factory. There is a continuous outflow of wood chips which constitutes a disturbance on the chips level in the tank. A level controller (LC) manipulates the feed screw. The conveyor belt makes up a transport delay of 250 sec of the chips from screw to tank. A model with parameter values is described in Appendix 39.2.[4]

The simulator is in the SimView simulator library, and is available on:

[3] Actually: *was*, since the factory in Norway – Södra Cell at Tofte – has been shut down.

[4] The model used in the simulator differs from the model in the appendix as some of the model parameters are in different units.



umax [kg/s] 50,0

umin [kg/s] 0,0

u_man [kg/s] 25,0

Auto

Man

Kc [(kg/s)/m] 3,89

beta 1,0

Ti [sec] 1000

up=Kp*(beta*r-h)

gamma 1,0

Td [sec] 0

ud=Kp*Td*d(gamma*r-h)/dt

Control signal terms:

☑ Total: u 30,0

☑ Manual: u_man 25,0

□ P-term: up 0,0

□ I-term: ui 5,0

□ D-term: ud -0,0

☑ Level reference, r 11,00

□ Level meas raw, h 11,00

☑ Level meas filt, h_filt 11,00

t [min] 197,7

h [m] 11,0

Level reference r [m] 11,0

F_out 30,0

F_out [kg/s] 30

Figure 1.6: Level control of wood chips tank: Simulated responses due to a step change of the level reference and a step change of the outflow (disturbance).

http://techteach.no/simview/level_control_woodchips_tank

Figure 1.7 shows a block diagram of the level control system.

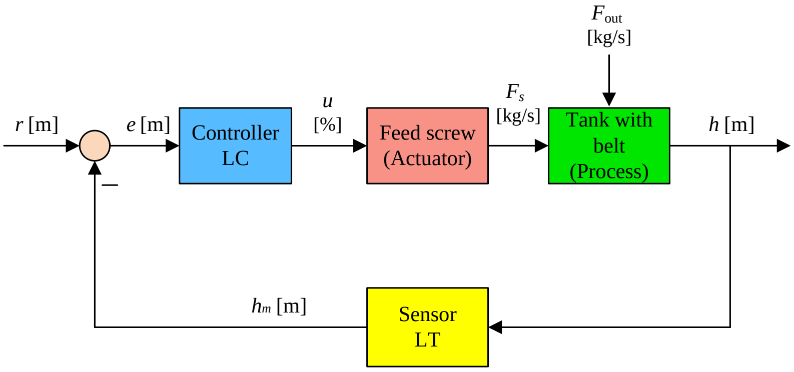


Figure 1.7: Block diagram of the level control system of the wood chips tank.

Figure 1.6 shows simulated responses in the level controlled tank. The level reference (or setpoint) is changed as a step from 10 to 11 m at $t = 40$ min, and the outflow (level disturbance) is changed as a step from 25 to 30 kg/s at $t = 120$ min. We observe that the level control system is able to bring the level control error to zero in steady state after the reference change and after the disturbance change.

[End of Example 1.1]

1.3.3 Components of feedback control loops

1.3.3.1 Block diagram of a general control loop

Figure 1.8 shows a detailed block diagram of a general feedback control system where the controller is realized with a computer.

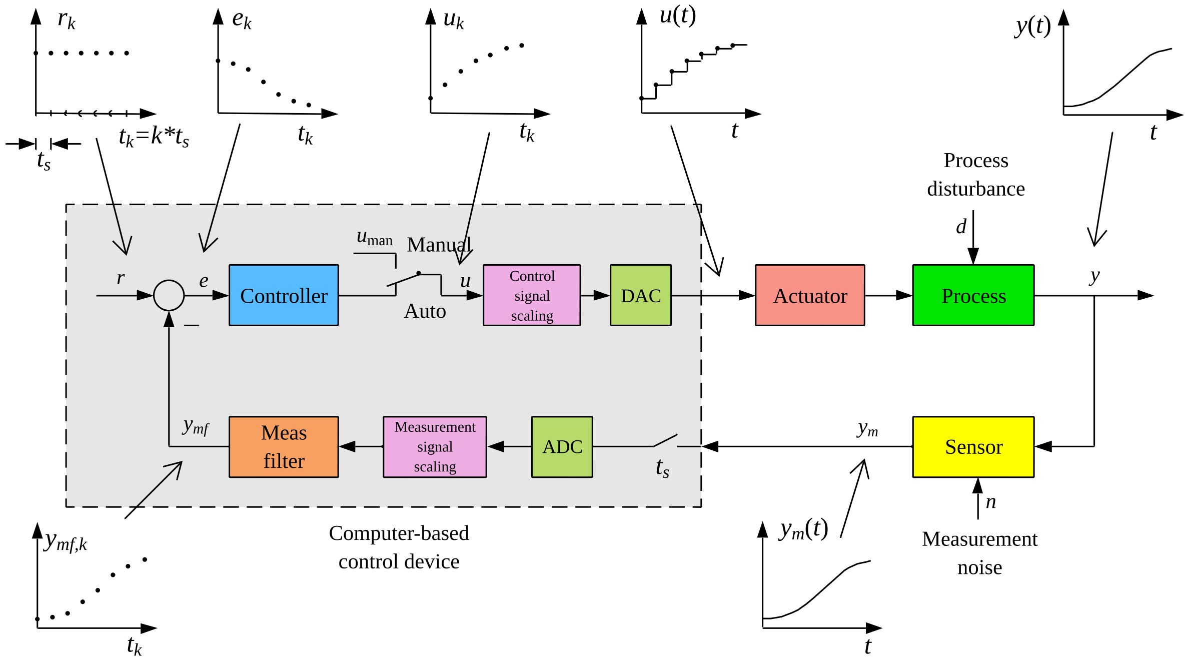


Figure 1.8: A computer-based control system. (DAC = Digital-Analog Converter. ADC = Analog-Digital Converter.)

Below are brief descriptions of each of the components of the control system

1.3.3.2 Process

The process is the physical system to be controlled, e.g. a tank where the level is to be controlled, a ship where the position on the sea is to be controlled, a biogas reactor where the temperature is to be controlled, etc.

1.3.3.3 Automation hardware

Automation hardware is the physical equipment where the control function is implemented, and where signal processing of the measurement signal and the control signal takes place. Nowadays it is common that automation hardware is computer-based. (More info in Section 3.2.)

1.3.3.4 Sensor

The sensor detects the process value, and generates typically either an electrical current or a voltage. Typically the current signal is in the standard range of 4-20 mA (milliampere). The voltage signal may be in the range 1-5 V, or some other range.

1.3.3.5 AD converter

AD converter (analog-digital converter, ADC) converts the process measurement signal – typically a current signal (milliampere) or a voltage signal from the sensor into a digital value that can be used by the computer. (More info in Section 3.4.3.)

1.3.3.6 Measurement signal scaling

The measurement signal scaling block scales the digital signal from the AD converter into a digital value expressed with the relevant physical unit, e.g. meters, degrees C, or %. For the shower temperature control system, assume that the measurement signal, m, is a voltage in the range 1-5 V representing a temperature (y) in the range 0–100 °C, with a linear relation between the ranges, see Figure 1.9. Then, the value of y can be calculated from any registered value of m with the following linear scaling formula:

$$y = a \cdot m + b \tag{1.4}$$

Slope a is

$$a = \frac{100^{\circ}\text{C} - 0^{\circ}\text{C}}{5\ \text{V} - 1\ \text{V}} = 25\ \frac{\text{K}}{\text{V}} \tag{1.5}$$

Intercept or constant b can be calculated from the first point (alternatively from the second point) by solving (1.4) for b:

$$b = y - a \cdot m = 0 - 25^{\circ}\text{C} = -25^{\circ}\text{C} \tag{1.6}$$

For example, a measurement value of $m = 3.0$ V indicates that the temperature is $y = 25 \cdot 3.0 - 25 = 50$ °C.

With industrial controllers, you can assume that transformation formulas like (1.4) are already implemented, and in such cases you only have to specify on the controller the temperature range that the measurement range (in mA or V) represents. While if you build a computer-based measurement and control system yourself (based on e.g. Python or LabVIEW), you will implement the transformation formula (1.4) in computer code. More information about sensors is given in Ch. 3.3.

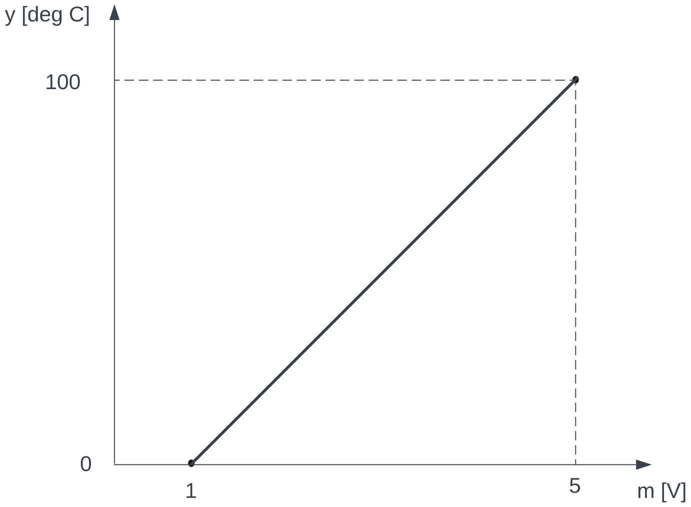


Figure 1.9: Scaling of temperature measurement signal.

1.3.3.7 Measurement signal filter

The measurement signal filter – or just measurement filter – shown in Figure 1.8 is an important part of the feedback control loop. The purpose of the filter is to filter out, or attenuate, the more or less random measurement noise from the measurement signal, so that the measurement signal used by the controller is more smooth, in turn causing a smoother control signal.

For illustration, Figure 1.10 shows the response of a lowpass filter output due to the following input signals:

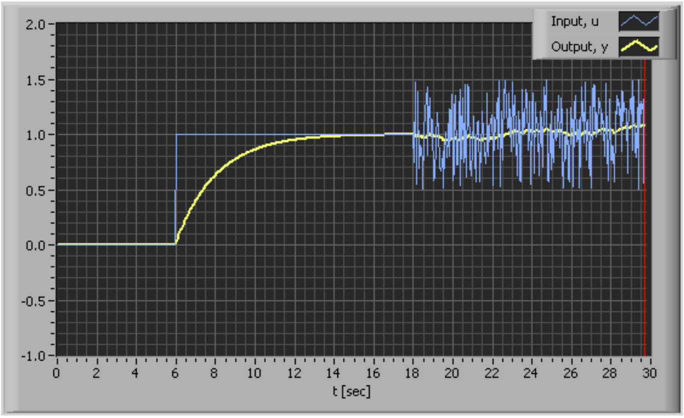


Figure 1.10: Response in a lowpass filter output due to a input step change at time 6 sec and random noise at the input from time 18 s.

- $0 < t < 6$ s: Zero input (no noise).
- $t = 6$ s: A step change of the input.
- $t \geq 18$ s: Random noise at the input

Comments to the responses shown in Figure 1.10:

- The filter attenuates the noise, as we want it to do.
- The filter has a sluggish response to the step change at the input. If this step represents the real changes of the process variable (although in practice such step changes would probably not appear), the filter actually have removed some information about the behaviour of the process variable, which may be unfortunate in an application.
- From the above two points, we can conclude that it is important to tune a filter for good noise attenuation while avoiding attenuation of variations in the process variables.

There are various types of measurement filters. The most relevant filters for use in control systems are:

- The moving average (MA) filter, which is presented in Section 3.4.6.1.
- The time constant filter, which is presented in Section 3.4.6.2.

It turns out that the discrete time filter algorithm ready for programming is identical for these two filters. Since the MA filter is simpler to describe, let's here take a brief look the MA filter.

The MA calculates the filtered value, y_{mf},as the average of a number, N_f, of the most recent measurement samples. N_f is the length of the filter. The time window of the filter moves as time goes. The size of the time window is here denoted t_f – the filter time constant. Assume as an example that $N_f = 5$. Figure 1.11 shows unfiltered measurement values (white circles) and the filtered value (black circle). The filtered measurement is calculated as the average of the present and the last four measurement samples – totally five samples:

$$\begin{aligned} y_{\text{mf},k} &= \frac{1}{5}\left(y_{\text{m},k} + y_{\text{m},k-1} + y_{\text{m},k-2} + y_{\text{m},k-3} + y_{\text{m},k-4}\right) \\ &= \frac{1}{5}\sum_{i=0}^{4} y_{\text{m},k-i} \qquad (1.7) \end{aligned}$$

Obviously, this averaging will smooth – i.e. filter – the measurement signal.

The general formula for $y_{mf,k}$ is

$$y_{mf,k} = \frac{1}{N_f}\sum_{i=0}^{N_f-1} y_{m,k-i} \qquad (1.8)$$

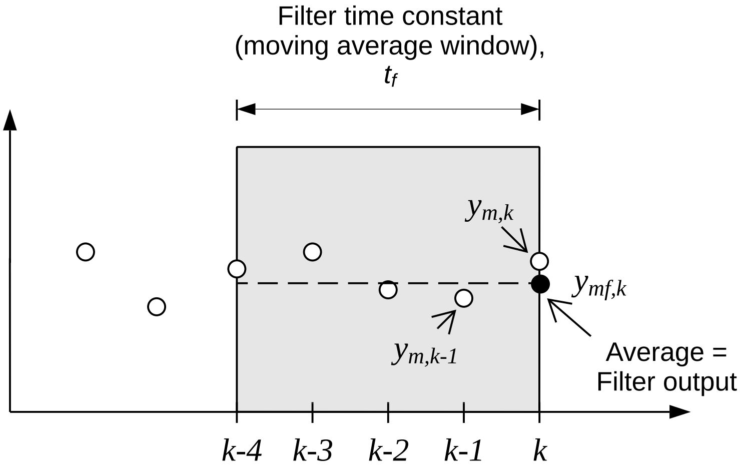


Figure 1.11: Moving average filter with length $N_f = 5$. t_f is the moving average time window.

For a given filter time constant (time window) t_f, the filter length is

$$N_f = \frac{t_f}{t_s} + 1 \tag{1.9}$$

For example, with $t_s = 0.1$ and $t_f = 4$ s, $N_f = 5$ (as in Figure 1.11).

In Section 3.4.6.1 it is shown that (1.10) shown below is a recursive – or online – realization of (1.8). (1.10) is a MA filter algorithm – ready for programming.

MA filter algorithm

$$y_{mf,k} = (1 - a)\, y_{mf,k-1} + a y_{m,k} \tag{1.10}$$

which is a filter algorithm ready for programming.

In (1.10), the filter parameter a is

$$a = \frac{1}{N_f} = \frac{t_s}{t_f + t_s} \tag{1.11}$$

For example, with $t_s = 0.1$ and $t_f = 0.4$ s (as in Figure 1.11), a is 0.2.

1.3.3.8 Controller

The controller calculates the control signal manipulating the actuator. Note: The terms "controller" here denote the control *functions*, and not the physical realization of the control function. The controller shown in Figure 1.3 is one realization of the control functions. Other realizations, i.e. physical controllers, are presented in Ch. 3.2.

The two most common control functions are the PID controller (proportional-integral-derivative) and the On-off controller. They are presented in detail in Sections 1.3.4 and 1.3.5, respectively.

Note: In Figure 1.8 the subtraction point for calculating the control error is shown ahead of the controller block. This is a common way to draw block diagrams of control systems, but in practice, the subtraction is calculated in the program code in the block.

1.3.3.9 Auto/man-switch

Industrial controllers can be switched between *automatic mode* (auto) and *manual mode* (man):

- Auto: The control signal is generated automatically by the controller function (algorithm) programmed in the built-in computer in the controller.
- Man: The built-in controller function is deactivated, and the control signal can be adjusted by a human via the user interface of the controller.

1.3.3.10 Control signal scaling

The control signal scaling block converts the calculated control signal, which may in units of e.g. % or kg/min, to a digital value of the signal to be applied to the actuator. The digital value may be in units of e.g. 0-20 mA for current signals, and e.g. 0.5 V for voltage signals. (More information is in Section 3.6.1.)

1.3.3.11 DA converter

The DA converter (digital-analog converter, DAC) converts the digital control signal into a physical control signal manipulating the actuator – typically a current signal in mA, but possibly a voltage signal. (More info is in Section 3.6.2.)

1.3.3.12 Actuator

The actuator manipulates the process based on the control signal generated by the controller. Examples of actuators are control valves, pumps, heating elements, motors, etc. (A number of actuators are described in Section 3.5.)

1.3.4 PID controller

1.3.4.1 Continuous-time PID controller

In Section 1.3.4.3 we shall develop a discrete-time PID controller – or a PID algorithm – ready for programming. This is the contents of the controller block in Figure 1.8. In the present section, I present the continuous-time, or analog, PID controller since it is the basis of the discrete-time PID controller.

The continuous-time PID controller is:

$$\begin{aligned} u &= u_{\text{man}} + \underbrace{K_c e}_{u_p} + \underbrace{\frac{K_c}{T_i}\int_0^t e\,d\tau}_{u_i} + \underbrace{K_c T_d\, e_f{}'}_{u_d} \qquad (1.12) \\ &= u_{\text{man}} + \text{P} + \text{I} + \text{D} \end{aligned}$$

where e is the control error:

$$e = r - y_{mf} \tag{1.13}$$

where y_{mf} is the filtered (smoothed) process measurement signal.

Some times it is convenient to use the simpler notation y instead of y_{mf}, and then (1.13) becomes

$$e = r - y \tag{1.14}$$

The parameters in (1.12) are defined in Section 1.3.4.2.

In (1.12),

- u_{man} is the nominal value of the control variable. It is the control signal available for adjustment by the operator while the controller is in manual mode. While the controller is in automatic mode, u_{man} can usually not be adjusted. u_{man} provides a reasonable initial value of the control signal at the moment of switching from manual to automatic mode. When the controller is switched from automatic to manual mode, u_{man} can be given the value of u just before the switching thereby providing the actuator with an assumeably appropriate control signal value.

- u_p is the P term. It is proportional to the control error.

- u_i is the I term, with time $t = 0$ being the latest time when the controller was set to automatic mode. The I term is the time integral of the control error. The integration starts when the control is set to automatic mode.

- u_d is the D term. The D term takes the time derivative, i.e. the rate of change, of the control error,

- e_f used in the D term is the lowpass filtered (smoothed) control error e. The filter is denoted the D filter. The filter is used to smooth the abrupt noise in the control error. This noise is due to the measurement noise present in any practical measurement signal. There is remaining noise even in after the measurement signal has been filtered

> with the measurement filter block shown in Figure 1.8. The D term takes the time derivative, i.e. the rate of change, of the control error. Since noise vary abruptly, the derivative can get large amplitudes. These amplitudes cause unfortunate variations in the D term, and consequently in the total control signal generated by the PID controller in which the D term is a part. To reduce the problem of the "noisy" D term, a lowpass filter is used.

The PID controller (1.12) is denoted the parallel PID controller since the terms appear in parallel in a mathematical block diagram of the controller, see Figure 1.12. It is sometimes denoted the academic controller.[5] There is also a serial form, which is discussed in Section 14.8.5.2.

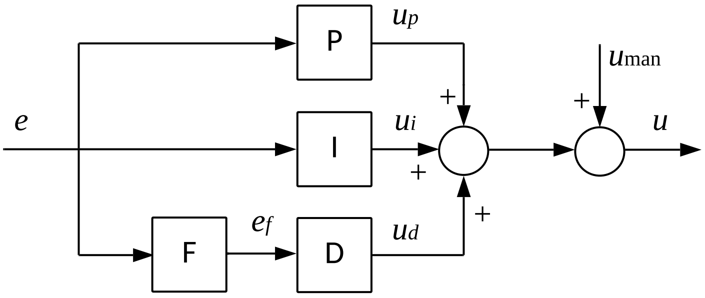


Figure 1.12: A block diagram illustrating why the PID controller (1.12) is denoted the parallel PID controller. ("F" is filter.)

1.3.4.2 PID parameters

The controller parameters in (1.12) are commented in detail below.

Proportional gain

K_c is the controller gain. An alternative symbol of this gain is K_p (p for proportional).

K_c is in unit of [unit of u/unit of y_{mf}]. Example: If u is in unit of W and y_{mf} is in unit of °C, K_c is in unit of W/°C, or strictly speaking, unit of W/K.

Several commercial controllers use the *proportional band* P_B, denoted just P in some industrial controllers, as a parameter instead of the proportional gain, K_c. The relation

[5] To a non-academic this name may indicate that the form is not useful in practice. However, it is indeed useful in practice!

between P_B and K_c is

$$P_B = \frac{100\%}{K_c} \tag{1.15}$$

where K_c is the controller gain, which here is assumed to be dimensionless. (It will be dimensionless if the control error e and the control variable u have the same unit, typically percent). It is typical that P_B has a value in the range of $10\% \leq P_B \leq 500\%$, which corresponds to K_c being in the range of $0.2 \leq K_c \leq 10$. It is important to note that P_B is *inversely proportional* to K_c. Thus, a small P_B corresponds to a large K_c, and vice versa.

Why the term *proportional band*? One explanation is that P_B is the change of the control error interval Δe (or the size of the measurement signal interval) which gives a change of the control signal interval Δu equal to 100% (i.e. a full range change of u): From the P-term $u = K_c e$ we see that $\Delta e = \Delta u / K_c = 100\%/K_c = P_B$.

Integral time

T_i [s] (or som other time unit, e.g. minutes) is the integral time.

In some controllers the value of $1/T_i$ is used instead of the value of T_i. The unit of $1/T_i$ is repeats per minute. For example, 5 repeats per minute means that $T_i = 1/5 = 0.2$ min. The background of the term repeats per minute is as follows: Assume that the control error e is constant, say E. The P-term has value $u_p = K_c E$. During a time interval of 1 minute, the I-term equals $\frac{K_c}{T_i} \int_0^1 E \, d\tau = K_c E \cdot 1[\text{min}]/T_i = u_p \cdot 1/T_i$. Thus, the I-term has *repeated* the P-term $1/T_i$ times.

Derivative time

T_d [s] is the derivative time.

P controller and PI controller

A P controller (proportional controller) is achieved by setting $T_i = \infty$ (or to a very large value) and $T_d = 0$. In some industrial controllers, you set T_i to 0 (zero) to deactivate the integral term, although this is an “unacceptable” numerical value of T_i because division by zero is not acceptable mathematically, cf. (1.12).

A PI controller (proportional-integral controller) is obtained by setting $T_d = 0$.

Alternative parameterization

Some controllers use the following alternative parameterization of the PID control function:

- Integral gain:

$$K_i = \frac{K_c}{T_i} \tag{1.16}$$

- Derivative gain:

$$K_d = K_c T_d \tag{1.17}$$

1.3.4.3 Discrete-time PID algorithm

Using the continuous-time PID controller, (1.12), as the basis, we will now develop a discrete-time PID controller – or a PID algorithm – ready for programming. The PID algorithm has the same basic form as (1.12), i.e.

$$u_k = u_{\text{man},k} + u_{p,k} + u_{i,k} + u_{d,k} \tag{1.18}$$

where each of the terms are discretized version of their corresponding continuous-time terms.

In the following, the discrete-time control error is calculated as

$$e_k = r_k - y_{\text{mf},k} \tag{1.19}$$

In the following, t_s is the sampling time (or time step, or cycle time) of the controller. In industrial PID controllers, a typical value of t_s is 0.1 s.

The manual term

u_{man} is a constant set by the operator (you?) when the controller is in manual mode. u_{man}, is "passive" when the controller is in automatic mode.

The P term

It is straightforward to discretize the P term:

$$u_{p,k} = K_c e_k \tag{1.20}$$

The I term

The continuous-time I term is

$$u_i(t) = \frac{K_c}{T_i} \int_0^t e(\tau)\, d\tau \tag{1.21}$$

It has become a tradition to discretize this integral with the Euler Backward method, which gives

$$\begin{aligned} u_{i,k} &= \frac{K_c}{T_i}\int_0^{t_k} e\,d\tau \\ &= \frac{K_c t_s}{T_i}\left(e_1 + \ldots + e_{k-1} + e_k\right) \qquad (1.22) \\ &= \underbrace{\frac{K_c t_s}{T_i}\left(e_1 + \ldots + e_{k-1}\right)}_{u_{i,k-1}} + \frac{K_c t_s}{T_i} e_k \\ &= u_{i,k-1} + \frac{K_c t_s}{T_i} e_k \qquad (1.23) \end{aligned}$$

which is a recursive version of the I term algorithm, which I repeated here for easier reference:

$$u_{i,k} = u_{i,k-1} + \frac{K_c t_s}{T_i} e_k \tag{1.24}$$

It is much more practical to implement (1.24), which is a recursive or online algorithm, than the batch algorithm (1.22), although they are equivalent. In (1.24), the only term that we have to store is $u_{i,k-1}$, while in (1.22) we have to store all older control errors, from $t = 0$ (i.e. the last time when the controller was set to automatic mode), whenever back in time that is (years maybe).

The D term

The continuous-time D term is, cf. (1.12),

$$u_d = K_c T_d\, {e_f}' \tag{1.25}$$

It has become a tradition in the control community to discretize ${e_f}'$ with the Euler Backward method, which gives the following discrete-time D term:

$$u_{d,k} = K_c T_d \frac{e_{f,k} - e_{f,k-1}}{t_s} \tag{1.26}$$

Above, e_f is lowpass filtered (smoothed) control error e, filtered with the D term filter. The filter is commonly either a moving average (MA) filter with filter parameter (cf. Section 3.4.6.1) or a time constant filter (cf. Section 3.4.6. As shown in Section 3.4.6, the discrete time algorithms of both these filters are identical. With any of these filters, the D term filter algorithm is

$$e_{f,k} \;=\; (1-a)e_{f,k-1} + a e_k \tag{1.27}$$

Filter parameter a is

$$a = \frac{t_s}{t_f + t_s} \tag{1.28}$$

It is common to set the filter time constant t_f as a fraction of the derivative time, T_d, of the controller:

$$t_f = \alpha T_d \tag{1.29}$$

with the following typically setting:

$$\alpha = 0.1 \tag{1.30}$$

Obviously, the D term filter is needed only if the controller has an active D term, i.e. if T_d is different from zero. If you are to implement a P controller or a PI controller, you can just forget about both the D term and the D term filter.

The total control signal

Just a repeat: Once the P, I, and D terms are calculated as described above, the total control signal must be calculated with (1.18), which I repeat here for convenience:

$$u_k = u_{\text{man},k} + u_{p,k} + u_{i,k} + u_{d,k} \tag{1.31}$$

The PI algorithm

Assume you are to program a PI controller yourself. Below is a PI control algorithm that you can use, expressed in pseudo code. u_{min} and $u_{\text{max}}$are minimum and maximum values of the control signal, respectively. For example: u_{min}may be 0 % or 0 V or 0 mA, and u_{max}may be 100 % or 5 V or 20 mA, depending on the application.

PI control algorithm

- Initialization of the I term: Typically, you can set $u_{i,0} = 0$.
- In a for loop or a while loop running with cycle time (time step) dt [s], which is the real time between each loop iteration:
  - Read the process measurement from the sensor: $y_{\text{m},k}$.
  - Filter (smooth) the measurement $y_{m,k}$ to obtain $y_{mf,k}$.
  - Calculate the control error: $e_k = r_k - y_{\text{mf},k}$.
  - Calculate the P term: $u_{p,k} = K_c e_k$.
  - Update the I term: $u_{i,k} = u_{i,k-1} + t_s \cdot (K_c/T_i)\, e_k$.
  - Optional (may be dropped in a quick & dirty implementation): Implement anti integral wind-up by limiting the I term, cf. Ch. 1.3.4.3 for details.
  - Calculate the total control signal: $u_k = u_{\text{man},k} + u_{p,k} + u_{i,k}$.
  - Limit the control signal between u_{min} and u_{max}.

- Write (apply) u_k to the actuator.
- Time index shift (to prepare for the next iteration of the loop): Set $u_{i,k-1}$ equal to $u_{i,k}$.

Practical modifications of the PID controller

The PID controller presented above can be used in "quick & dirty" implementations of the controller. To make a full-fledged implementation, you should consider the following practical modifications of the PID controller, which are described in Section 11.3:

- Reducing P kick and D kick (Ch. 11.3.2)
- Integral anti wind-up (Ch. 11.3.3)
- Bumpless transfer between manual and automatic modes (Ch. 11.3.4)

1.3.4.4 How does the PID controller work?

The manual term, u_{man}, is constant, and hence "passive" when the controller is in automatic mode. Its contribution in automatic mode is to provide kind of a reasonable initial value at the moment of switching from manual to automatic mode.

The P term, u_p, contributes with a term in the total control signal, u, which is proportional to the control error, e. It brings some speed to the control. However, assuming u_{man} is not "perfect" to give zero control error, i.e. $e = 0$, the P term by itself can not ensure $e = 0$ either This is because, with $e = 0$, $u_p = 0$, which mean no contribution from the P term. In other words, P controller can not ensure zero error in steady state.

The I term is the most important part of the PID controller because *it ensures zero steady state control error*, i.e. $e_s = 0$. How? Look at the I term, (1.24). As long as e is different from zero, u_i will change. In other words, e is an "improvement term". Or, e drives the control. This change keeps on until e has become zero, and then u_i,and u, are kept constant, until some disturbance or reference change causes e again to become nonzero, but then the improvement starts again.

The D term

- Assume that, for some reason, the control error, e, is increasing, i.e. the measurement is moving away from (lower than) the reference. The difference $e_{f,k} - e_{f,k-1}$ in 1.26) is then positive, making $u_{d,k}$ positive. So, the D term contributes positively to the total control signal, u_k, and we can expect *faster control* with the D term (PID control) comparing with no D term (only PI control).

- Now, assume that e is decreasing, i.e. the measurement is getting closer to the reference. The difference in 1.26) is now negative, making $u_{d,k}$ negative. So, the D term contributes negatively to the total control signal, u_k, and we can expect *a breaking or stabilizing control* with the D term (PID control) comparing with no D term (only PI control).

So, the D term may “press the gas pedal” when appropriate, and “press the break pedal” when appropriate.

One implication of the above is that the D term may stabilize a control system which otherwise can not be stabilized with a P or a PI controller. This is the case with dynamic positioning of ships. With a PI controller the control system is deemed to be unstable, while it is stable with a properly tuned PID controller.

There is one serious practical problem with the D-term: It amplifies the random measurement noise, causing large variations in the control signal. These variations will be reduced with a lowpass (smoothing) filter acting on the process measurement, cf. Section 1.3.3.7.

Example 1.2 *Temperature control of a liquid tank*

Figure 1.13 shows a simulation of a temperature control system of a liquid tank.

The tank can be e.g. a biogas reactor where the temperature in the tank is maintained at a temperature reference which is favourable to the microorganisms that form biogas (mainly methane gas) of the biological raw material. The temperature control, which is made by the controller TC (Temperature Controller) is based on feedback from the measured temperature in the tank provided by the sensor TT (Temperatuere Transmitter). The controller manipulates the tank temperature by adjusting the control signal to the heating element in the tank. A significant process disturbance, or environmental variable, is the varying inlet temperature T_{in}. The temperature reference, r_T, i.e. the desired temperature, is 40°C.

Figure 1.14 shows a block diagram of the temperature control system.

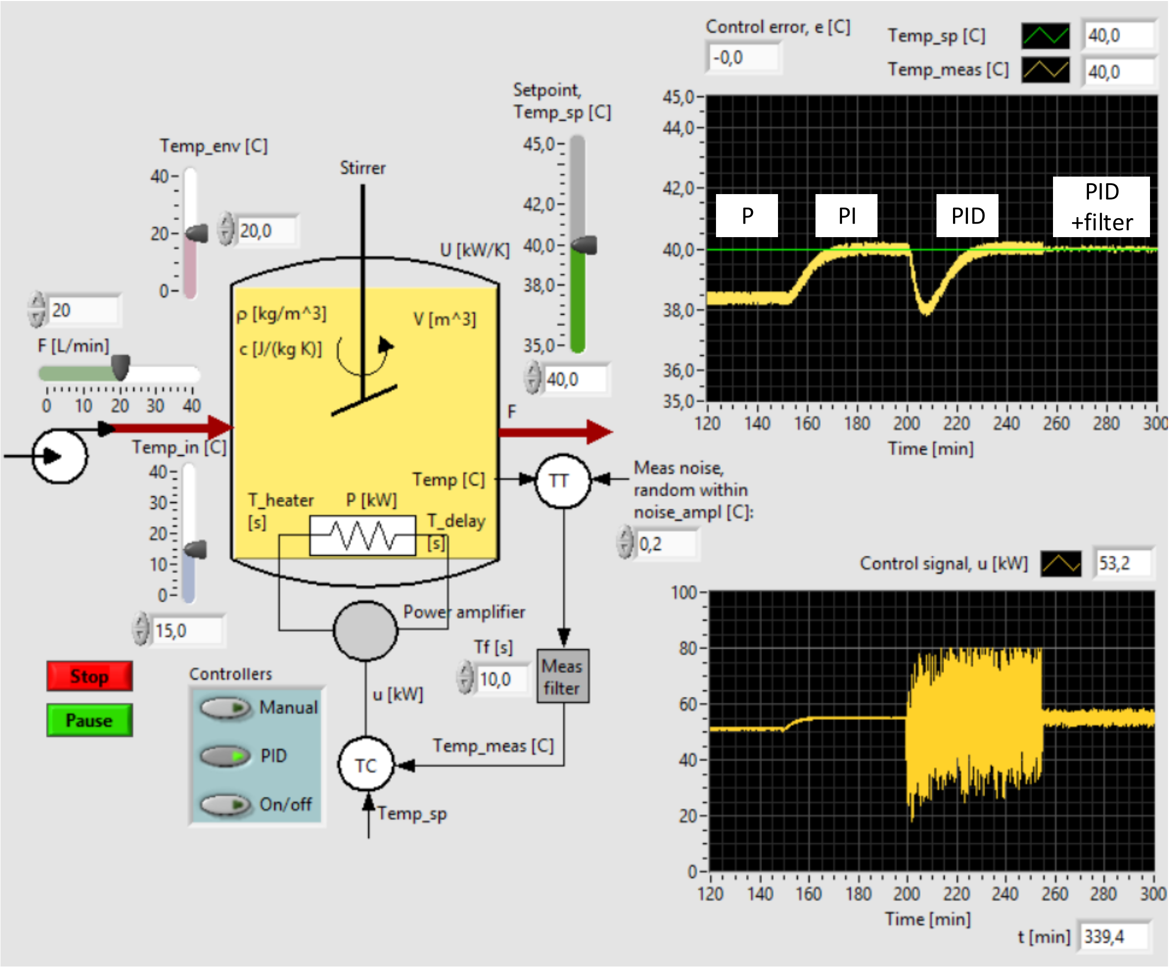
As seen in the simulation:

- The steady state control error is non-zero with the P controller. The non-zero error is due to the lack of the integral term of the PI controller.

- The steady state control error is zero with the PI controller. The zero error is due to the integral term of the PI controller.

- The control signal is very noisy when the PID controller is used. (The large transient response in the temperature around $t = 210$ s is due to the D term suddenly being activated, causing a jump in the control signal and hence in the temperature.)

- The measurement filter attenuates the noise, thereby reducing the noise in the control signal.



Temp_env [C] 20,0

F [L/min] 20

Temp_in [C] 15,0

Setpoint, Temp_sp [C] 40,0

U [kW/K]

Meas noise, random within noise_ampl [C]: 0,2

Tf [s] 10,0

Control error, e [C] -0,0

Temp_sp [C] 40,0

Temp_meas [C] 40,0

Control signal, u [kW] 53,2

t [min] 339,4

Figure 1.13: Temperature control with P, PI, PID controller without meas. filter, and PID with filter.

The simulations in this example are made with this SimView simulator:

http://techteach.no/simview/temp_control_pid_onoff

[End of Example 1.2]

Summing up the PID controller

From the above it can be concluded that the controller should definitely have an I-term, i.e. the controller should be either a PID controller or a PI controller, to ensure zero steady state control error. The D-term should be omitted if the control signal is too noisy even with a measurement lowpass filter. However there are processes where the D-term is

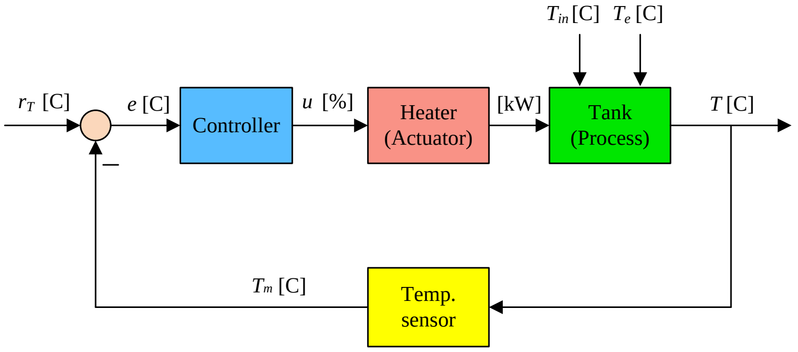


Figure 1.14: Block diagram of the temperature control system of the liquid tank

essential for obtaining a stable control system (as in position control of ships and other "free-body" mechanical systems where the controller manipulates force or torque acting on the body). In fact, the PI controller is by far the most used feedback controller in the industry. I have heard that more than 90% of PID controllers run as PI controllers.

1.3.5 On-off controller

1.3.5.1 The basic On-off controller

The On-off controller is probably the simplest controller there is. On-off controllers may be an alternative to PID controllers, especially in temperature control. For example, room temperature is commonly controlled with a thermostat, which is an On-off controller.

Figure 1.15 illustrates the On-off controller function.

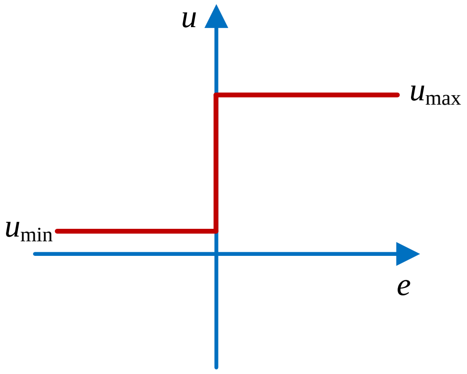


Figure 1.15: On-off controller.

In Figure 1.15, $u_{\min}$may be 0 % or 0 V or 0 mA, and $u_{\max}$may be 100 % or 5 V or 20 mA, depending on the application.

The On-off controller can be presented as an algorithm as follows.

On-off control algorithm

The algorithm is implemented in a for loop or a while loop running with cycle time (time step) t_s [s]:

- Read the filtered process measurement from the sensor: y_k.
- Calculate the control error: $e_k = r_k - y_k$.
- If $e_k \geq 0$, set $u_k = u_{\min}$.
- Else (i.e. if $e_k < 0$), set $u_k = u_{\max}$.
- Write (apply) u_k to the actuator.

The On-off controller works as follows: When the control error, e, is positive, i.e. when y is below r, u is $u_{\max}$, causing y to increase. Eventually y becomes larger than r, so that e becomes negative, which sets u to $u_{\min}$, causing y to decrease. Eventually y becomes less than r, so that e becomes positive, and then the scenario repeats. Hence, the On-off controller causes the control system to oscillate.

The period of the oscillations is given by the dynamic properties of the process. The amplitude of the oscillations in u and in y are also given by the dynamic properties of the process. We can reduce the amplitudes by reducing the difference between $u_{\max}$ and $u_{\min}$, but with the danger of limiting the control signal so that it will stay at either $u_{\max}$ or $u_{\min}$ in case of large process disturbances.

The benefits of the On-off controller are:

- It is a very “quick” controller. It compensates quickly for process disturbances – quicker than a PID controller.
- It is tuned very easily. In principle, the only tuning is selecting the values of $u_{\max}$ and $u_{\min}$.

The main drawbacks about the On-off controller are:

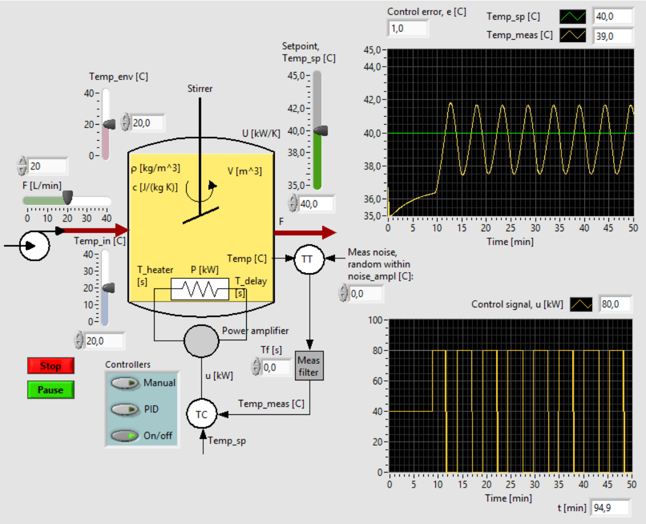
- The control loop has inavoidable oscillations, cf. Figure 1.16.
- The average value of the control error is non-zero, cf. Figure 1.16.

Example 1.3 *Temperature control with On-off controller*

In this example, the simulations are made with the following SimView simulator:

http://techteach.no/simview/temp_control_pid_onoff

Figure 1.16 shows a simulation of a temperature control system. $u_{\max} = 80$ kW, and $u_{\min} = 0$ kW. The oscillatory behaviour is clear. Note that the mean value of the control error is different from zero (the mean temperature is slightly below its reference), which is typical for On-off control.



Control error, e [C] 1,0

Temp_sp [C] 40,0

Temp_meas [C] 39,0

Setpoint, Temp_sp [C] 40,0

Temp_env [C] 20,0

F [L/min] 20

Temp_in [C] 20,0

Meas noise, random within noise_ampl [C]: 0,0

Tf [s] 0,0

Control signal, u [kW] 80,0

t [min] 94,9

Figure 1.16: Simulation of a temperature control system with an On-off controller.

[End of Example 1.3]

1.3.5.2 On-off controller for processes with negative process gain

Actually, the On-off controller shown in Section (1.3.5.1) applies only to processes which have a so-called positive process gain, i.e. the process measurement increases when the control signal increases (like in a "heating" thermal process where an increase of the control signal to the heater makes the process temperature increase).

If the process has a negative gain, i.e. the process measurement decreases when the control signal increases (like in a "cooling" thermal process where an increase of the control signal to the cooler makes the process temperature decrease), the On-off controller is "flipped" as shown in Figure 1.17.

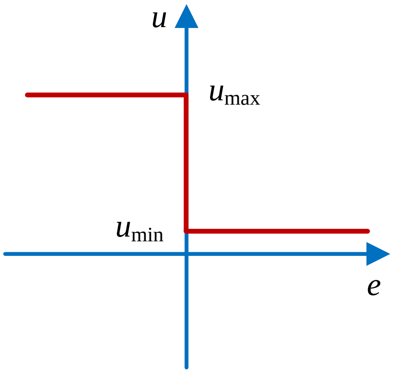


Figure 1.17: On-off controller for processes with negative process gain.

1.3.5.3 On-off controller with deadband

There is a practical problem with the On-off controller presented in Sections 1.3.5.1: If the process measurement, y, is noisy, also the controller error, e, is noisy. There will be some remaining noise even if the process measurement is filtered. A noisy e may make the control signal, u, switch abruptly between $u_{\min}$ and $u_{\max}$. This is unfortunate, especially if the actuator is mechanical.

Example 1.4 *Temperature control with On-off controller with noisy measurement*

Figure 1.18 shows a simulation of the same temperature control system as in Example 1.3, but where uniformly distributed random noise between ±1.0 °C has been added to the temperature measurement. We can see that the control signal varies abruptly when the error is around zero. (In this example, I have not used a measurement filter, to make the problem about the noise clearer.)

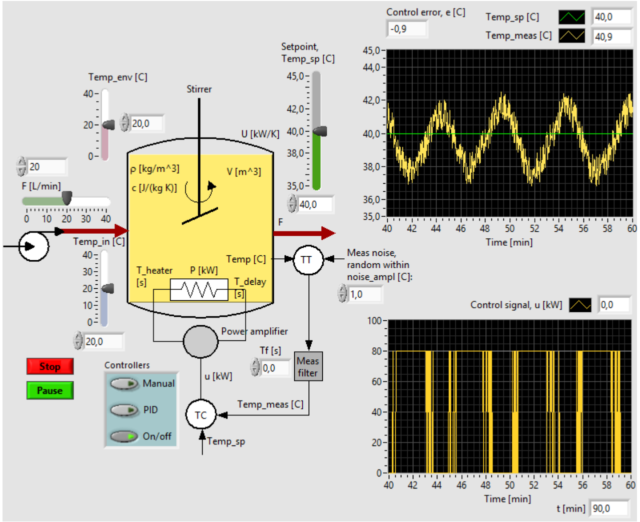
[End of Example 1.4]

In Example 1.4 we saw that measurement noise caused the On-off controller output to change abruptly between $u_{\min}$ and $u_{\max}$ when the control error varied around zero. Such abrupt changes are unfortunate, and should be avoided, particularly if the actuator is mechanical, like a mechanical relay, or a pump, or a valve, or a motor.

How can we avoid such abrupt changes? We can include a deadband in the On-off controller! The deadband should be larger than the maximum amplitude of the measurement noise. This solution is shown in Figure 1.19.

A drawback about including the deadband is that the amplitude of the oscillations in y will be somewhat larger than without deadband. Furthermore, the period of the oscillations will increase, but this is hardly a drawback.

Example 1.5 *On-off controller with deadband*



Control error, e [C] -0,9

Temp_sp [C] 40,0

Temp_meas [C] 40,9

Setpoint, Temp_sp [C] 40,0

Temp_env [C] 20,0

F [L/min] 20

Temp_in [C] 20,0

Meas noise, random within noise_ampl [C]: 1,0

Tf [s] 0,0

Control signal, u [kW] 0,0

t [min] 90,0

Figure 1.18: Simulation of a temperature control system where uniformly distributed random noise between ±1.0 °C of has been added to the temperature measurement.

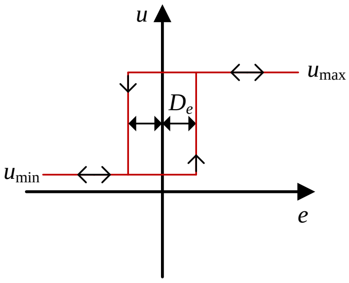


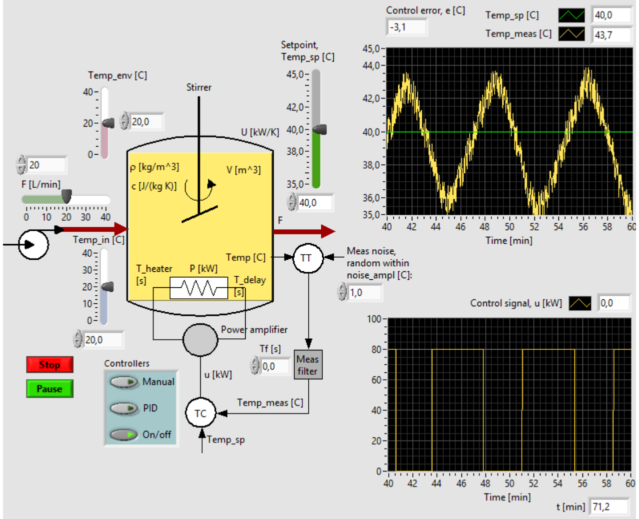
Figure 1.19: On-off controller with deadband.

Figure 1.18 in Example 1.4 shows a simulation of a temperature control system where uniformly distributed random noise between ±1.0 °C was added to the temperature measurement, causing u to change abruptly when e is around zero. Now we include a deadband with $D_e = 2$ °C in the On-off controller. Figure 1.20 shows the results of a simulation. As expected, the abrupt changes in u are now eliminated. The period of the oscillations is somewhat larger than without deadband, cf. Figure 1.18.

[End of Example 1.5]

1.4 Feedforward control

By now, we know that feedback control – or error-driven control – can bring the process output variable to or close to the reference in steady state (strictly: when time goes to infinity). Feedback control is in most cases a sufficiently good control method. But sometimes you want more. The "problem" with feedback is that there *has to exist a control error different from zero* for any change of the control signal to take place, since the control variable is adjusted as a function of the control error. Here, *feedforward control* come to



Control error, e [C] -3,1

Temp_sp [C] 40,0

Temp_meas [C] 43,7

Temp_env [C] 20,0

F [L/min] 20

Temp_in [C] 20,0

Setpoint, Temp_sp [C] 40,0

Meas noise, random within noise_ampl [C]: 1,0

Tf [s] 0,0

Control signal, u [kW] 0,0

t [min] 71,2

Figure 1.20: Temperature control with On-off controller with deadband.

help. Feedforward control is based on the following information about the process to be controlled:

- *Desired behaviour of the process in terms of the reference.* This is always known, and is needed also in feedback control.
- *Measurements of the process disturbances*
- *A mathematical model of the process.* A model is an abstract, mathematical representation of process, and expresses how the process behaves.

Using all this information to calculate the control signal, we may obtain excellent (accurate) control, i.e. very good reference tracking. However, the information is certainly more or less imprecise, causing the control to be more or less imperfect so that the control error becomes somewhat different from zero. But here comes feedback control to help: It reduces the control error which exist due to the inevitable imperfect feedforward control. Hence, when feedforward control is used, it is typically used together with feedback control.

Let's take a brief look at an interesting real application of feedforward control, see Example 1.6. Feedforwad control is presented more systematically and in more detail in Chapter 17.

Example 1.6 *Dynamic positioning of a ship*

Dynamic positioning (DP) of ships is automatic position control through the manipulation of the actuators which are the main propeller, maneuvering thrusters, and the rudder. DP systems are very important in various marine operations. DP systems make ships stay sufficiently close to e.g. platforms and other ships, and make ships follow a given position trajectory accurately. Obviously, DP systems increase the level of safety largely.

Let us see how feedforward control can be used in the dynamic positioning system of a ship. The main disturbances are:

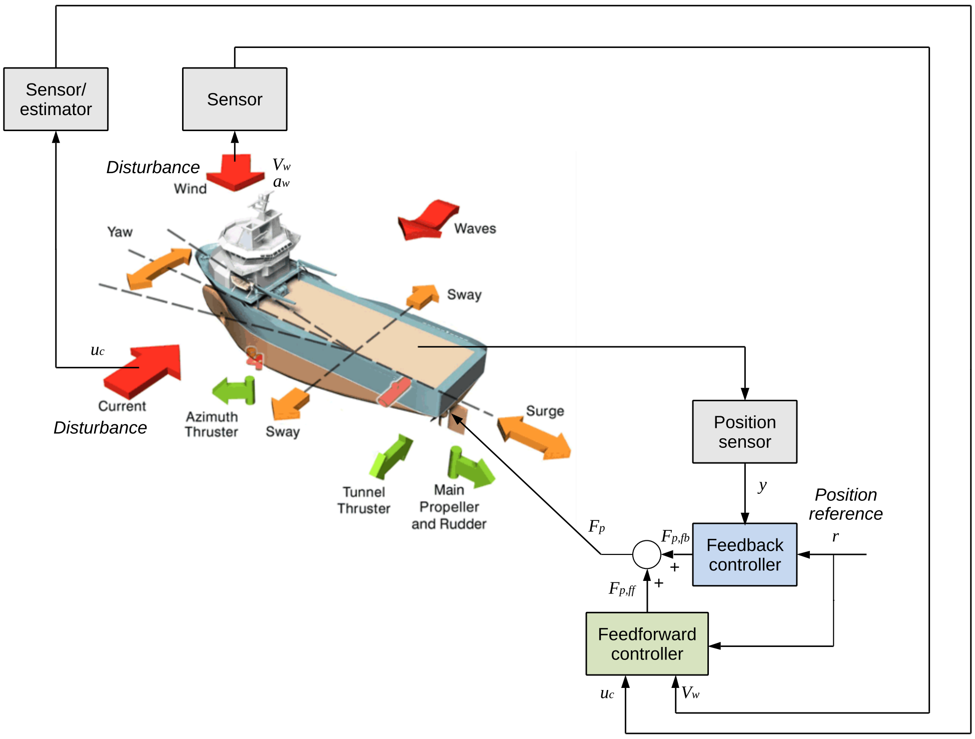


Figure 1.21: Dynamic positioning system with feedforward control added to the feedback control. (The drawing is based on a drawing originally made by Kongsberg Maritime AS.)

- Wind, represented with the wind speed, V_w [m/s], and the wind angle, a_w, causing a disturbing force, F_w [N], on the ship.
- Water current, represented with the water current speed, u_c [m/s], causing a disturbing hydraulic force, F_h, on the ship.

Assume that we can measure or estimate V_w and u_c and calculate F_w and F_h, respectively. This can actually be done with a wind force model and a water current model. Such models are presented in Ch. 39.9, but we skip details about how to design the feedforward controller using these models until Ch. 17. Assuming that we know F_w and F_h from measurements of V_w and u_c and the mentioned models. To compensate for these disturbing forces, we can increase the propeller force, F_p, with an amount which is equal to the negative sum of F_w and F_h, thereby cancelling out their impacts on the ship motion! In other words, we establish a technical coupling – a feedforward controller – that cancels out the natural coupling that the disturbances has on the ship (the process). The feedforward controller also uses the position reference, r, to calculate the feedforward control signal.

The resulting control signal (which is a propeller force demand) is the sum of the feedback control signal and the feedforward control signal:

$$F_p = F_{p,\mathrm{fb}}(r, x) + F_{p,\mathrm{ff}}(r, V_w, a_w, u_c) \tag{1.32}$$

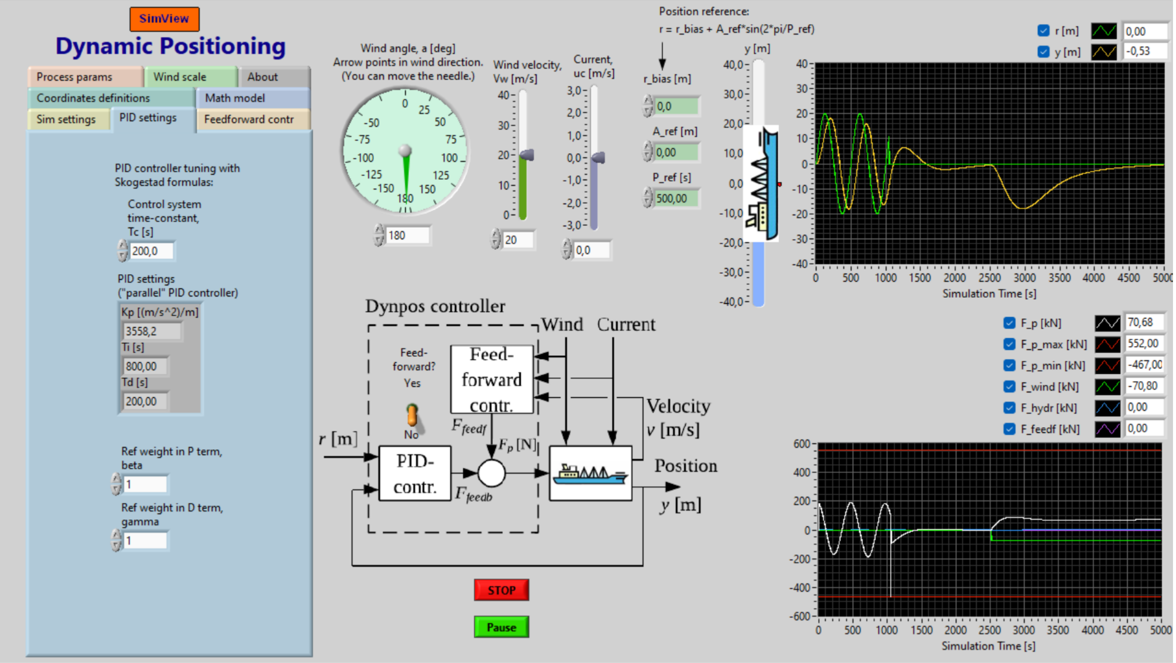
Figure 1.21 shows the dynamic positioning system with feedforward control, which is added to the feedback control based on position measurement from GPS sensors, but other position sensors may be used also depending on the operation conditions.

Feedback control

Let's first see how the DP system works without feedforward control – but with the compulsory feedback control. The feedback controller is a PID controller, which is tuned with the SIMC[6] method (Skogestad 2003), which is presented in Ch. 14.8.5.

I have programmed a simulator in Python which simulates the longitudinal or surge motion of a ship, using model parameter values provided by the company Kongsberg Maritime AS, Norway. The mathematical model is presented in Ch. 39.9.

Figure 1.22 shows simulated responses where the position reference r is changed softly as a sine function from from $t = 0$ s until $t \approx 1000$ s, then kept constant at 0 m, and at $t \approx 2500$ s the speed, V_w, of the wind acting on the ship is changed from as a step from 0 to 20 m/s.[7] The wind speed causes a wind force, F_w, which is a disturbance on the DP system. F_w is plotted in the lower subplot of Figure 1.22.



PID controller tuning with Skogestad formulas:

Control system time-constant, Tc [s]: 200,0

PID settings ("parallel" PID controller)

Kp [(m/s^2)/m]: 3558,2

Ti [s]: 800,00

Td [s]: 200,00

Ref weight in P term, beta: 1

Ref weight in D term, gamma: 1

Wind angle, a [deg]
Arrow points in wind direction.
(You can move the needle.): 180

Wind velocity, Vw [m/s]: 20

Current, uc [m/s]: 0,0

Position reference:
r = r_bias + A_ref*sin(2*pi/P_ref)

r_bias [m]: 0,0

A_ref [m]: 0,00

P_ref [s]: 500,00

☑ r [m] 0,00
☑ y [m] -0,53

☑ F_p [kN] 70,68
☑ F_p_max [kN] 552,00
☑ F_p_min [kN] -467,00
☑ F_wind [kN] -70,80
☑ F_hydr [kN] 0,00
☑ F_feedf [kN] 0,00

Figure 1.22: Simulation of position control of a ship (dynamic positioning) with PID feedback control.

The simulation is done with the following SimView simulator:

[6] SIMC = Simple Internal Model Control

[7] You may say that such a wind gust is unrealistically large. But keep in mind that this change may be due to the ship being in harsh weather with a strong side wind, and when the ship turns for some reason, the wind force suddenly changes direction and force relative to the ship.

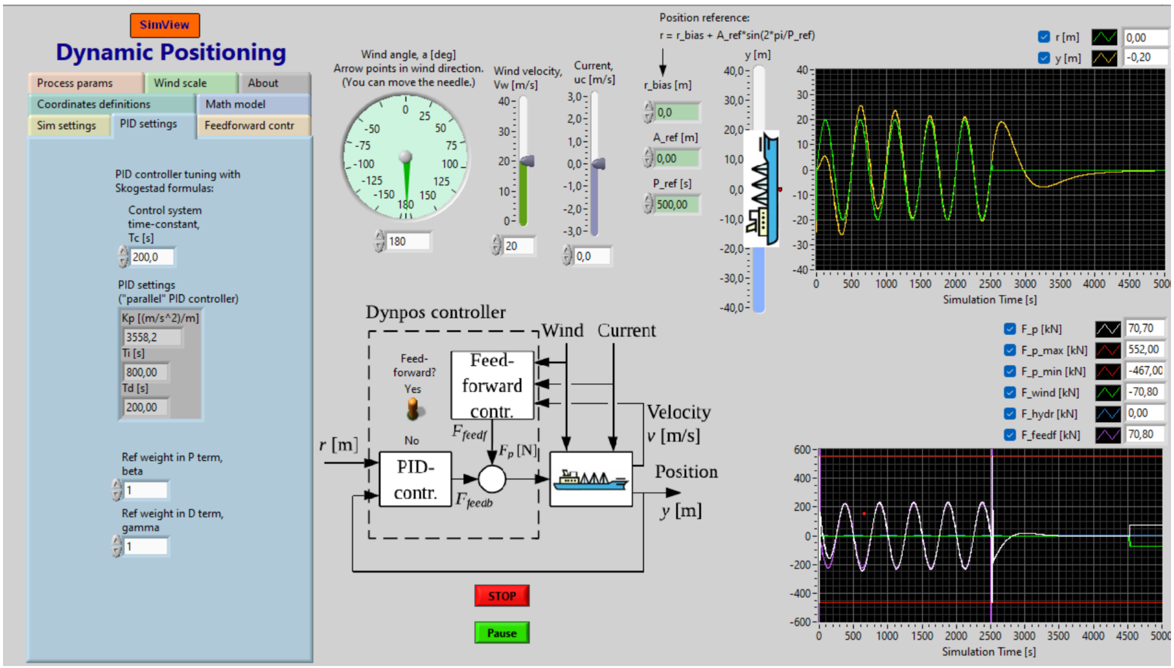
http://techteach.no/simview/dynpos

How does the control system work? Based on the control error, i.e. the difference between the position reference and the measured position, the controller generates a propeller force F_p in an attempt to keep the ship at the reference, see the lower plot in Figure 1.22. The ship tracks the varying reference with a maximum control error of a few meters, and the wind gust drives the ship off the reference with approximately 20 m. In steady state, the control error is zero.

Now, let's see if feedforward control can reduce the control error.

Feedback + feedforward control

Figure 1.23 shows a simulation of the ship with position control based on feedforward control combined with feedback control. The control system behaves excellently:



PID controller tuning with Skogestad formulas:
Control system time-constant, Tc [s]: 200,0

PID settings ("parallel" PID controller)
Kp [(m/s^2)/m]: 3558,2
Ti [s]: 800,00
Td [s]: 200,00

Ref weight in P term, beta: 1
Ref weight in D term, gamma: 1

Wind angle, a [deg]: 180
Wind velocity, Vw [m/s]: 20
Current, uc [m/s]: 0,0

Position reference:
r = r_bias + A_ref*sin(2*pi/P_ref)
r_bias [m]: 0,0
A_ref [m]: 0,00
P_ref [s]: 500,00

☑ r [m]: 0,00
☑ y [m]: -0,20

☑ F_p [kN]: 70,70
☑ F_p_max [kN]: 552,00
☑ F_p_min [kN]: -467,00
☑ F_wind [kN]: -70,80
☑ F_hydr [kN]: 0,00
☑ F_feedf [kN]: 70,80

Figure 1.23: Simulation of the ship with position control based on feedforward control combined with feedback control.

- After $t \approx 1500$ s, the position reference is tracked very well, and much better than with feedback control only.
- The wind speed is changed as a step at $t \approx 4500$ s is compensated for very effectively; the ship is not moving despite the wind gust. Notice that the propeller force, F_p,

counteracts exactly the wind force, causing the ship to stay still despite the heavy wind gust.

In practice, we can not expect such an excellent behaviour of the controlled ship because we can not (never) implement a perfect feedforward controller due to modeling and/or measurement errors. However, we can expect a *large improvement* of the control with a well-designed feedforward controller compated to not using feedforward.

In Problem 17.2 you are to tune the PID controller and design the feedforward controller, and implement a simulator of the DP system of the present example in Python.

[End of Example 1.6]

1.5 Performance measures of control systems

Some times it is useful to evaluate the performance of a control system with a numerical measure or index. There are several alternative measures:

- Control error based measures, which can be found experimentally.
- Stability margins, which can be found experimentally or from the frequency response of the control system.
- Bandwidth, which can be found from the frequency response of the control system.

Performance measures based on the control error are defined below. Stability margins are defined later in the book; in Sections 15.2 and 22.3, while bandwidth is defined in Section 22.3.

Figure 1.24 shows three common quantitative measures of the performance of control systems. These measures are based on the observed control error:

$$e = r - y \tag{1.33}$$

where r is the reference (or setpoint), and y is the (measured) process output.

Below are comments to each of the three performance measures.

Maximum control error

The smaller the $|e|_{\max}$, the better performance.

Example 1.7 *Maximum control error in a level control system*

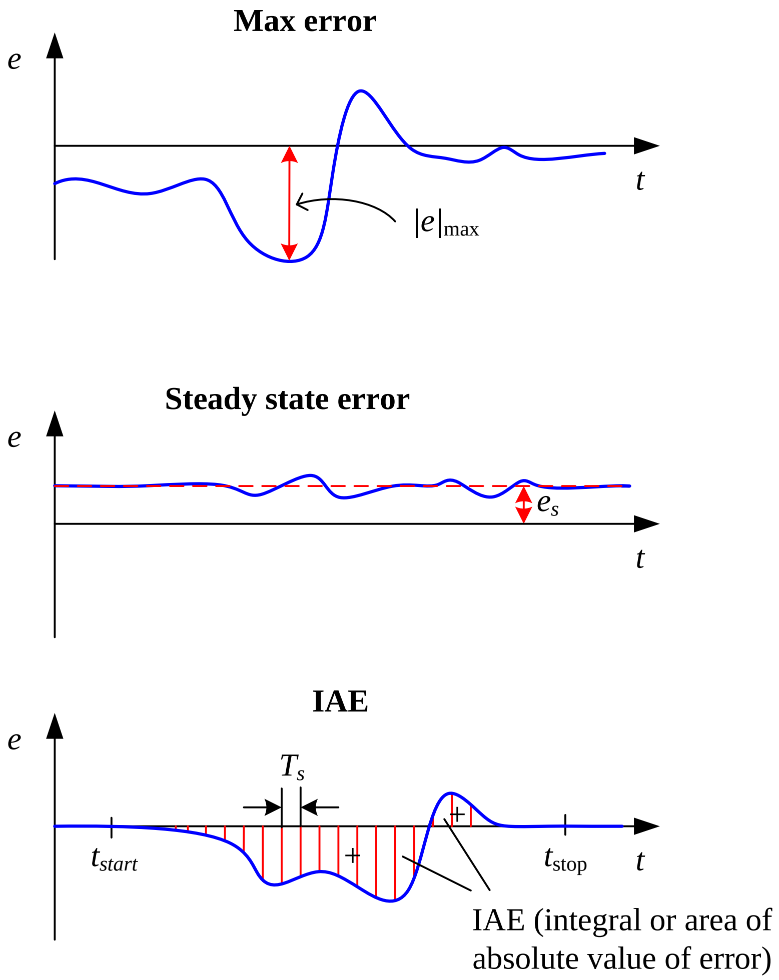


Figure 1.24: Three common quantitative measures of the performance of control systems.

See Figure 1.6. The level reference r is changed as a step from 10 to 11 m. From the plot of the wood chips level, we see that the maximum control error due to this reference step change is $|e|_{\max} = 1.3$ m.

[End of Example 1.7]

Steady state control error

The steady state control error, e_s, is the error when the error is approximately constant (when you disregard the variations due to measurement noise). The steady state control error can also be denoted the static control error.

The smaller the e_s, the better performance.

Example 1.8 *Steady state control error in a level control system*

See Figure 1.6. From the plot of the wood chips level, we see that the steady state control error when the level reference and the process disturbance (the outflow) are contants is $e_s = 0$ m.

[End of Example 1.8]

IAE

IAE is short for integral of absolute value of control error. The IAE performance index is widely used in literature (books and articles) where the performance of control systems is compared, but you will not find any IAE indicator in an industrial controller.

IAE is defined as

$$\text{IAE} = \int_{t_{\text{start}}}^{t_{\text{stop}}} |e(t)| \, dt \tag{1.34}$$

The IAE can be calculated numerically as

$$\text{IAE} = t_s \cdot [|e(t_0)| + |e(t_1)| + \cdots + |e(t_{k-1})| + |e(t_k)|] = t_s \sum_{t_0 = t_{\text{start}}}^{t_{N-1} = t_{\text{stop}}} |e(t_k)| \tag{1.35}$$

where t_s is the time step or samling time. t_{start} and t_{stop} are specified by the user. k is the time index (an integer). If you disregard the factor t_s (i.e. assume it 1), the IAE is the sum of the absolute errors.

The smaller the IAE index, the better performance.

The IAE value from an experiment does not tell that much, but it is useful when it is compared with the IAE values from other experiments.

The IAE index depends on the duration of the experiment, so you can compare IAE values from different experiments only when the experiments have the same duration. If you want an IAE index which is independent of the duration, you can use the following normalized index:

$$\text{IAE}_{\text{norm}} = \frac{1}{t_{\text{stop}} - t_{\text{start}}} \int_{t_{\text{start}}}^{t_{\text{stop}}} |e(t)| \, dt \tag{1.36}$$

which gives you the average absolute control error over the pertinent time interval.

1.6 *Problems for Chapter 1*

Problem 1.1 *Components of a car speed control system*

Suppose you are driving a car on a road with uphill and downhill slopes. The speed limit of the road is as shown in Figure 1.25. Assume that your drives currently at 50 km/h, but you want to get home quickly, and would therefore like to stay at the speed limit on this stretch of road. To that end, speed control is needed, and you are to carry out the control.

Figure 1.25: Speed limit (reference).

1. What is the reference?
2. What is the process variable?
3. What are the process disturbances?
4. What is the sensor?
5. What is the controller?
6. What is the (main) actuator?
7. How does the controller carry out the control?
8. Would the car have kept the speed limit without active control, i.e. with passive control (no manipulation of the accelerator).
9. Above, *manual* control was assumed. Do you know about an *automatic* car speed control system?

Problem 1.2 *Components of a speed control system*

Figure 1.26 shows the different components of a speed control system of an electric motor.

1. "Construct" a speed control system by connecting the components (draw a block diagram of the control system). Where is the control error in your block diagram?
2. How does the control system work? (Assume that the speed initially is equal to the speed reference (setpoint), and that the load torque is increased so that the motor speed is reduced.)

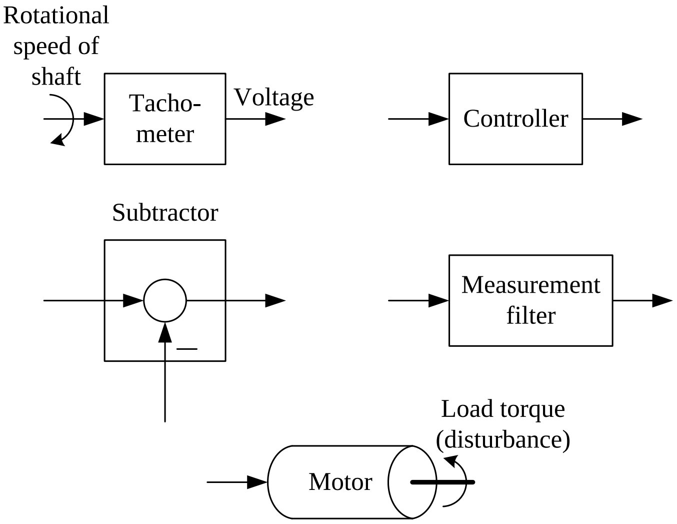


Figure 1.26: Components of a motor servo mechanism.

Problem 1.3 *Automatic bartender*

The weight control system shown in Figure 1.27 seems to be an automatic bartender.

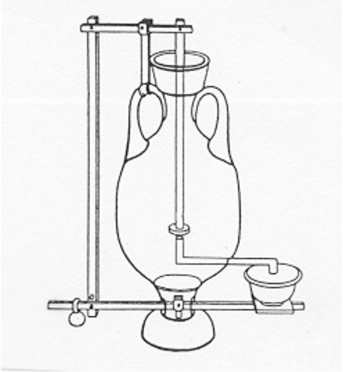


Figure 1.27: A weight control system from the Antics. An automatic bartender? (Mayr 1970)

Explain how the control system works. (Explain the feedback control action.)

Do you know any other process of your daily life which uses the same principle of level control?

Problem 1.4 *Evaporator*

Figure 1.28 shows an evaporator where the product is created by evaporating the feed. (As an example, an evaporator is used to remove water from the half-finished fertilizer in a fertilizer factory.)

Suggest a control structure by drawing a Process & Instrumentation Diagram of the evaporator according to the requirements listed below. (Process & Instrumentation

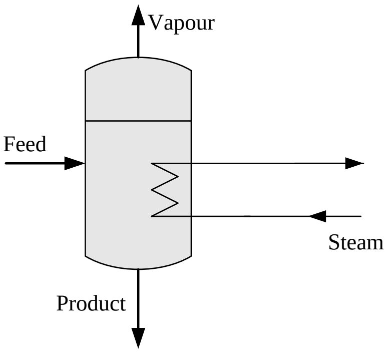


Figure 1.28: Evaporator.

Diagrams are covered in detail in Ch. 2, but I assume you have enough information from the examples in this chapter to draw such a diagram at this stage.)

- The feed flow is controlled to its reference. (Symbol of flow is F.)
- The liquid level is controlled to its reference. (Symbol of level is L.)
- The liquid temperature is controlled to its reference. (Symbol of temperature is T.)
- The vapour pressure is controlled to its reference. (Symbol of pressure is P.)
- You can use control valves as actuators. A symbol of a control valve is shown in Figure 1.29.

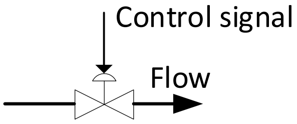


Figure 1.29: Symbol of a control valve.

Problem 1.5 *Examples of control systems*

Below are mentioned three processes which are supposed to be controlled. The process output variable is indicated in parenthesis. For each of the processes:

- What is the control (manipulating) variable?
- What are the disturbances or loads or environmental variables (these are alternative names)?

Make your own assumptions. The processes are as follows:

1. Robot arm or manipulator driven by an electric motor (arm position).

2. Steam heated heat exchanger with some process fluid to be heated (temperature of fluid outlet).

3. Ship positioned with thrusters (denoted a dynamic positioning system) (ship position).

Problem 1.6 *The sensor location is important!*

Figure 1.30 shows two alternative sensor locations for temperature control of an air heater, which is presented in Ch. 39.5. The temperature reference, r, is the reference of the temperature, T_1, at the outlet of the tube. The controller is a PI controller, which ensures that the average control error is zero under stationary conditions.

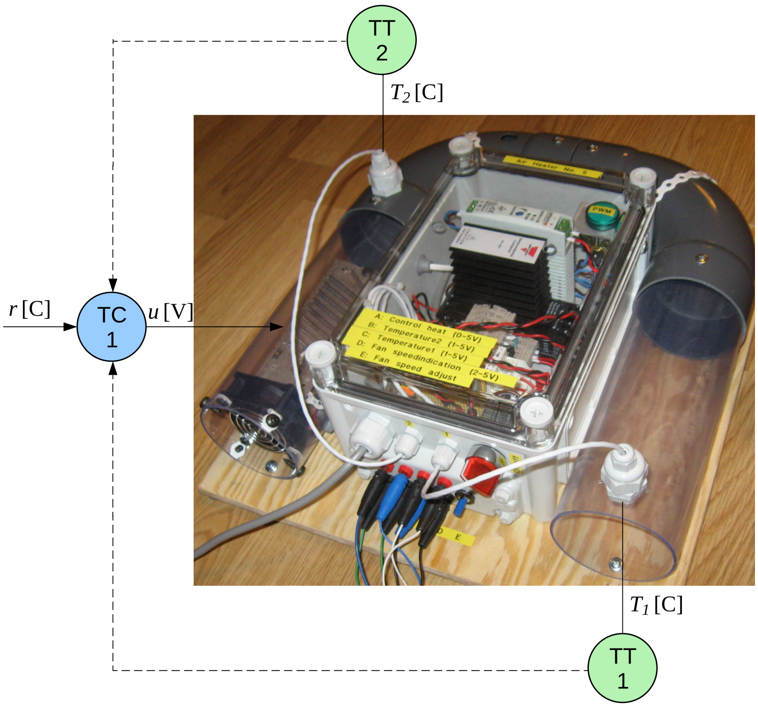


Figure 1.30: Two alternative sensor locations for temperature control of an air heater.

Figure 1.31 shows the temperature response in T_1 for the following two cases:

1. The temperature controller uses sensor TT1.

2. The temperature controller uses sensor TT2.

What is the value of the average control error (approximately) in each of the two cases? Explain the difference of the values!

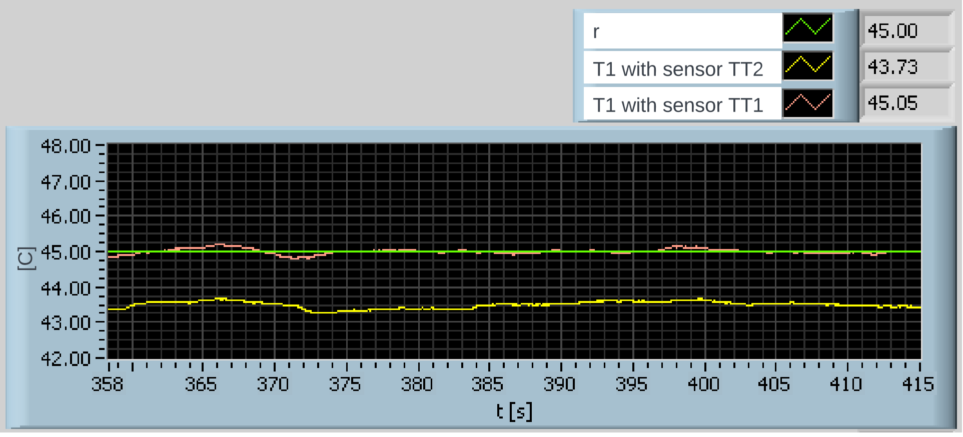


Figure 1.31: Temperature response in T_1 for two different locations of the temperature sensor used for control.

Problem 1.7 *Measurement scaling*

Given a level sensor which represents the level, h, in the range [0 – 15 m] with a measurement signal (current), m, in the range [4 – 20 mA], with a linear relation between these ranges.

1. Find the scaling function with m as input and h as output, and express it as a linear function on the standard form
$$h = a \cdot m + b \tag{1.37}$$
where a is the slope and b is the intercept (or constant term).
2. What is h if 8 mA?

Problem 1.8 *Manual calculation of filter output*

Given the following measurement signals, y_m, for the times t_0, t_1,t_2, respectively:

9.11

10.48

9.54

(The ideal (noise-free) measurement value is 10.)

Assume that the initial value of the filter output is 10. Manually calculate the filter output for times t_0, t_1, t_2 for a moving average filter. The filter parameter is $a = 0.1$.

Can you see from the results that the filter output is smoother (varies less) than the filter input?

Problem 1.9 *Filter length of an MA filter*

Given an MA filter with $t_f = 5.0$ s and $t_s = 0.05$ s. What is the filter length N_f of the filter? What is the filter parameter a?

Problem 1.10 *Temperature response with various controllers*

Figure 1.32 shows an air heater. A fan with fixed speed blows air through the pipe. The fan opening can be varied manually. The air is heated by a electric heater. The control signal u is the voltage signal which controls (adjusts) the power supplied to the heater. The temperature is measured with a thermistor which is a temperature-dependent resistance. In the experiments described below the controller is implemented in a PC with I/O-device (Input/Output-device). (In general, a control system should contain a measurement filter, but in this particular system a filter was not used.)

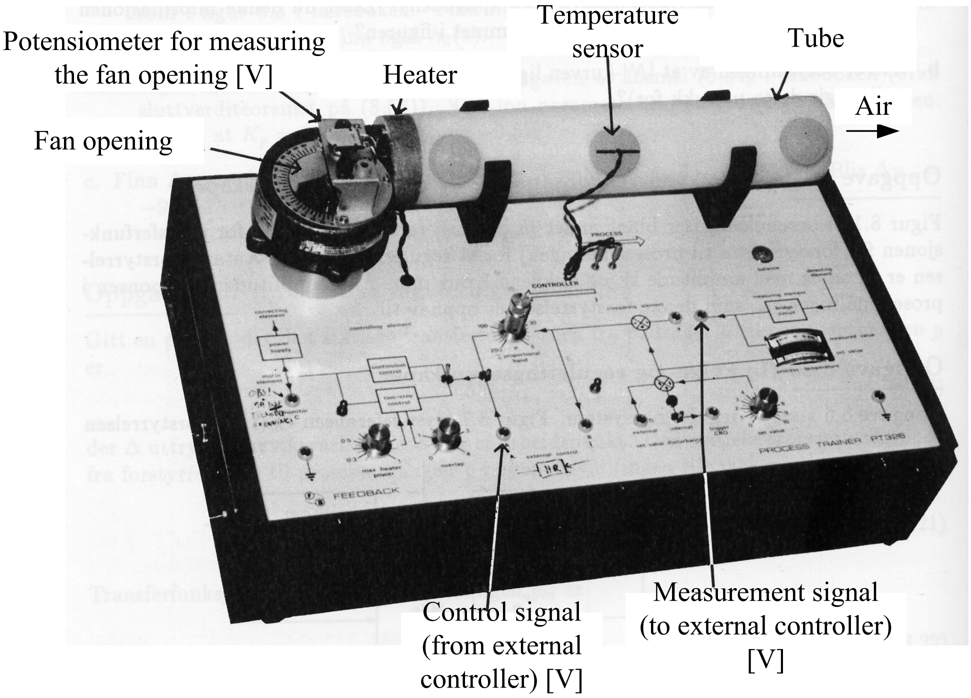

Potensiometer for measuring the fan opening [V]
Heater
Temperature sensor
Tube
Air
Fan opening
Control signal (from external controller) [V]
Measurement signal (to external controller) [V]
PROCESS TRAINER PT326
FEEDBACK

Figure 1.32: Process trainer

1. Draw a block diagram of the control system, including a measurement filter.
2. Figure 1.33 shows the response in the temperature y after a step in the temperature reference (setpoint) and after a step in the air inflow due to an increase of the fan

opening. The air inflow or – equivalently the fan opening – can be regarded as a disturbance. In this experiment the control signal is constant, hence there is no feedback (no measurement-based or error-driven) control. Explain why there is no response in the temperature due to the reference change. And explain why there *is* a response after the disturbance (fan opening) change.

3. Assume that the temperature is controlled with a PID-controller (with proper parameter settings). Draw the principal temperature response after a reference step and after a disturbance step. You can "add" your curves to 1.33.)

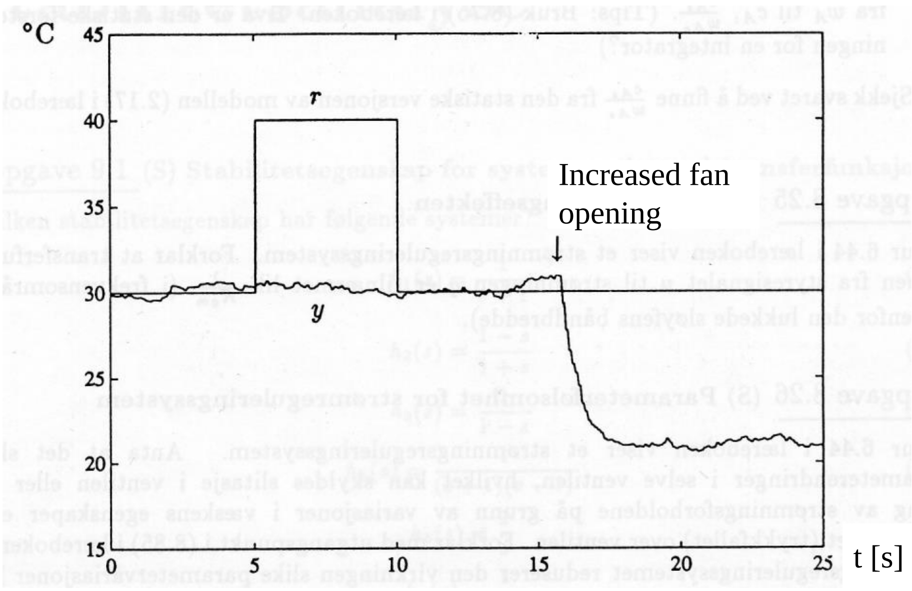


Figure 1.33: Temperature response with constant control signal

Problem 1.11 *Gain and PB*

What is the value of the controller gain K_c corresponding to proportional band PB $= 250$ %?

Problem 1.12 *Manual calculation of PI controller output*

In this problem, you will act as a discrete time PI controller.

Assume the following:

- Reference: $r_k = 60$.
- Filtered process measurement: $y_{mf,k} = 59.4$.
- Integral term from previous time-step: $u_{i,k-1} = 0.5$.
- Manual control signal: $u_{\text{man}} = 50$.

- Controller gain: $K_c = 2$.
- Integral time: $T_i = 10$ s.
- Time step of the controller: $t_s = 0.1$ s.

Calculate the control signal u_k according to the discrete time PI control agorithm.

Problem 1.13 *Step response of PI controller*

Suppose you want to verify that a PI controller works correctly according to the mathematical PI controller function. This can be done with a step response test, where a step is applied to the reference r input to the controller while the measurement input y_m is kept constant. Assume that

$$y_m(t) = A \tag{1.38}$$

and that the reference is increased from A to

$$r(t) = A + E \tag{1.39}$$

This implies that the control error e is increased as a step from zero to

$$e(t) = r(t) - y_m(t) = (A + E) - A = E \tag{1.40}$$

By comparing the observed (experimental) step response in the controller output u with the theoretical output, you can (hopefully) confirm that the mathematical operation of the controller is correct.

The PI controller function is

$$u = K_c e + \frac{K_c}{T_i} \int_0^t e \, dt \tag{1.41}$$

1. Calculate the step response in u (as a function of time) assuming that the control error is a step of amplitude E, and plot $u(t)$.
2. Figure 1.34 shows the step response in u for a given PI controller. The step amplitude of the control error was

$$E = 1 \tag{1.42}$$

   Calculate K_c and T_i from the step response.

Problem 1.14 *Filter time constant in D-term*

Given a PID controller with a lowpass filter acting on the derivative term. Assume that the derivative time T_d is 2.0 sec. Suggest a proper value of the filter time constant t_f.

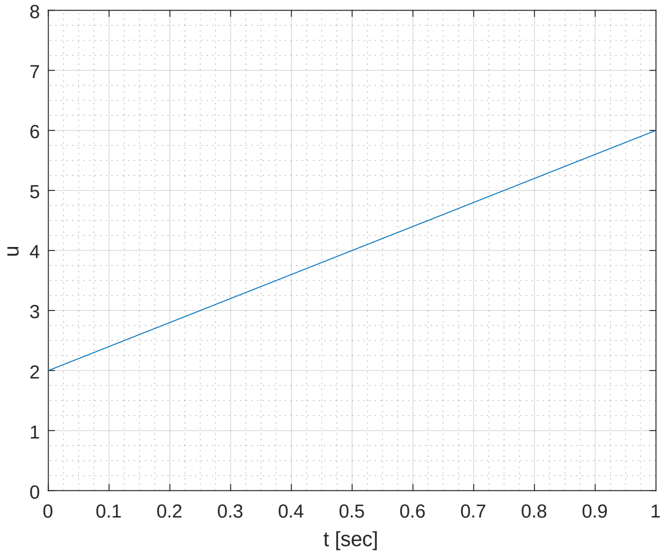


Figure 1.34: Step response in PI controller output.

Problem 1.15 *On-off control*

See Figure 1.16.

1. What is the period of the oscillations?
2. What is the average control error? Is it zero?

Problem 1.16 *IAE*

Assume that an experiment with control system number 1 has an IAE index of 148 calculated over the time interval 20-30 min after a step in the reference at time $t = 20$ min. The same experiment with another control system, number 2, has an IAE index of 97 over the same time interval, and the same experiment with control system no. 3 gives IAE index of 123 calculated over the time interval 20-40 min.

1. Can the IAE indexes of the control systems 1, 2 and 3 be compared to each other?
2. Which of the control systems has the best performance in terms of IAE?

1.7 *Solutions to problems for Chapter 1*

Solution to Problem 1.1

1. Reference: The speed limit indicated on the sign.
2. The process variable: The speed of the car.
3. Process disturbances:
   - Gravity force at uphill and downhill driving.
   - Air force
   - Friction force
4. Sensor: The speed sensor of the car.
5. Controller: You!
6. Actuator: Foot + accelerator.
7. The controller (you) keeps the car roughly at the speed reference - despite varying process disturbances (mentioned above) - by adjusting the accelerator on the basis of the control error, which is the difference between the speed reference and the actual car speed as measured by the speed sensor.
8. The car would not have stayed at the speed reference with passive control, since there would then not be any compensation for process disturbances. Active control, i.e. manipulation of the accelerator, is necessary.
9. Probably you have. A built-in cruise controller in the car!

Solution to Problem 1.2

1. Figure 1.35 shows the speed control system. The control error, e, is the output of the subtractor.

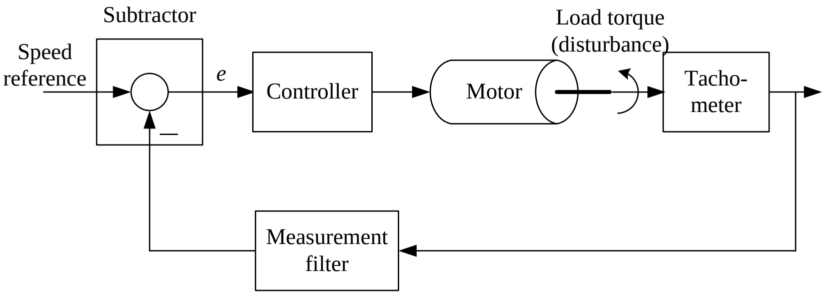


Figure 1.35: Speed control system.

2. When the speed is reduced due to the increased load torque, the control error becomes different from zero, and positive. This non-zero, positive control error causes the controller to increase the control signal acting on the motor, so that the speed is increased. If the controller is properly chosen (it must have integral action, actually), the controller is able to adjust the control signal to exactly the new value that is needed to compensate for the load torque, and consequently the control error become zero – in steady state.

Solution to Problem 1.3

The purpose of the system is to fill just the right amount of liquid into the cup, i.e. level control. The system works as follows: As long as the level is low, the inlet is open and the cup is being filled. The more liquid in the cup, the less opening in the inlet. Eventually, when the cup is full, the inlet is closed ands hence, the desired level (the level reference) is reached.

The system is a feedback control system, with feedback from the process output (liquid level) back to the process input (inflow).

Solution to Problem 1.4

The control structure is shown in Figure 1.36.

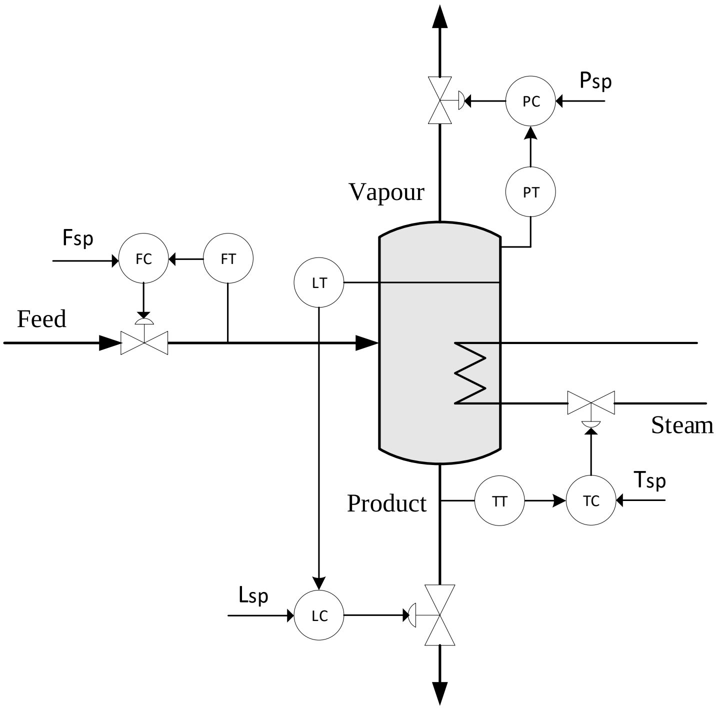


Figure 1.36: Control structure of an evaporator.

Solution to Problem 1.5

1. Robot arm: Control signal manipulates the motor. Disturbances: Torques due to the gravity and due to mechanical couplings to other links.
2. Heat exchanger: Control signal manipulates the valve. Disturbances: Temperature and pressure of inlet steam.
3. Ship: Control signal manipulates the propellers (thrusters). Disturbances: Wind, current, waves.

Solution to Problem 1.6

From Figure 1.31:

Case 1 (the temperature controller uses sensor TT1): The average control error is 0. It is 0 because the measurement that the controller uses is at the location for which the temperature reference is defined.

Case 2 (the temperature controller uses sensor TT2): The average control error is approximately 1.5 °C. It is non-zero because the measurement that the controller uses is *not* at the location for which the temperature reference is defined. The T_2 measurement is not at the outlet, but earlier in the tube. There will be a heat loss in the tube, through the walls, which causes a temperature drop from TT2 to TT1. The controller can not compensate for this temperature drop since the measurement that the controller uses does not capture the temperature drop. In other words, in this case, the sensor is misplaced.

This problem illustrates that it is important that the location of the sensor used for control the location at which the reference is defined.

Solution to Problem 1.7

1. The slope becomes

   $$a = \frac{15 \text{ m} - 0 \text{ m}}{20 \text{ mA} - 4 \text{ mA}} = \frac{15}{16} \frac{\text{m}}{\text{mA}} = 0.9375 \frac{\text{m}}{\text{mA}} \tag{1.43}$$

   The constant term becomes

   $$b = 0 \text{ m} - \frac{15}{16} \frac{\text{m}}{\text{mA}} \cdot 4 \text{ mA} = -3.75 \text{ m} \tag{1.44}$$

2. With $m = 8$ mA, (1.37) gives

   $$p = a \cdot m + b = \frac{15}{16} \frac{\text{m}}{\text{mA}} \cdot 8 \text{ mA} - 3.75 \text{ m} = 3.75 \text{ m} \tag{1.45}$$

Solution to Problem 1.8

The filter function is given by (3.49), which with $a = 0.1$ becomes

$$y_{mf}(t_k) = 0.9y_{mf}(t_{k-1}) + 0.1y_m(t_k)$$

At time t_0:

$$y_{mf}(t_0) = 0.9y_{mf}(t_{-1}) + 0.1y_m(t_0) = 0.9 \cdot 10 + 0.1 \cdot 9.11 = 9.911$$

At time t_1:

$$y_{mf}(t_1) = 0.9y_{mf}(t_0) + 0.1y_m(t_1) = 0.9 \cdot 9.911 + 0.1 \cdot 10.48 = 9.968$$

At time t_2:

$$y_{mf}(t_2) = 0.9y_{mf}(t_1) + 0.1y_m(t_2) = 0.9 \cdot 9.968 + 0.1 \cdot 9.54 = 9.925$$

Yes, it is quite clear that these values of the filter output vary less than the filter inputs.

Solution to Problem 1.9

From (1.9):

$$N_f = \frac{t_f}{t_s} + 1 = \frac{5.0}{0.05} + 1 = 101 \tag{1.46}$$

From (1.11):

$$a = \frac{1}{N_f} = \frac{1}{101} = 0.009901 \tag{1.47}$$

Solution to Problem 1.10

1. The block diagram is shown in Figure 1.37.
2. Since u_0 is not influenced by the temperature reference (setpoint), the control signal remains constant. Therefore, the reference does not influence the actual temperature. However, an increase of the fan opening will influence (reduce) the temperature because more cold air is blown into the pipe.
3. The PID-controller gives zero control error (in average), see Figure 1.38.

Solution to Problem 1.11

$$K_c = \frac{100}{\text{PB}} = \frac{100}{250} = 0.4 \tag{1.48}$$

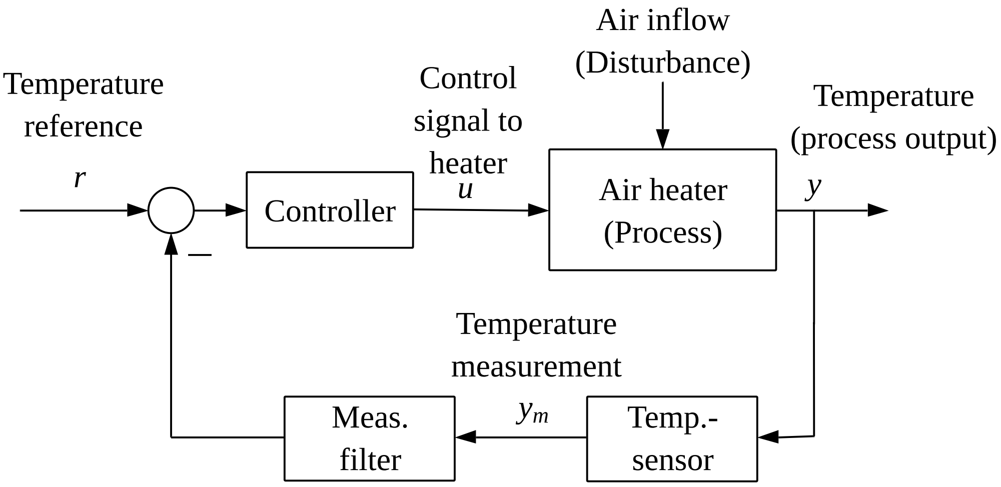


Figure 1.37: Block diagram of temperature control system.

Solution to Problem 1.12

Control error:

$$e_k = r_k - y_{\text{mf},k} = 60.0 - 59.4 = 0.6$$

P term:

$$u_{p,k} = K_c e_k = 2 \cdot 0.6 = 1.2$$

I term:

$$u_{i,k} = u_{i,k-1} + t_s \cdot (K_c/T_i)\, e_k = 0.5 + 0.1 \cdot (2/10) \cdot 0.6 = 0.512$$

Total control signal:

$$u_k = u_{\text{man},k} + u_{p,k} + u_{i,k} = 50 + 1.2 + 0.512 = 51.712$$

Solution to Problem 1.13

1. Setting $e = E$ in the controller function gives

$$u(t) = K_c E + \frac{K_c}{T_i} \int_0^t E\, dt = K_c E + \frac{K_c}{T_i} E t \tag{1.49}$$

   which is "constant plus ramp". Figure 1.39 shows this step response.

2. From Figure 1.34 we see that

$$u(0_+) = 2 = K_c E = K_c \cdot 1 = K_c \tag{1.50}$$

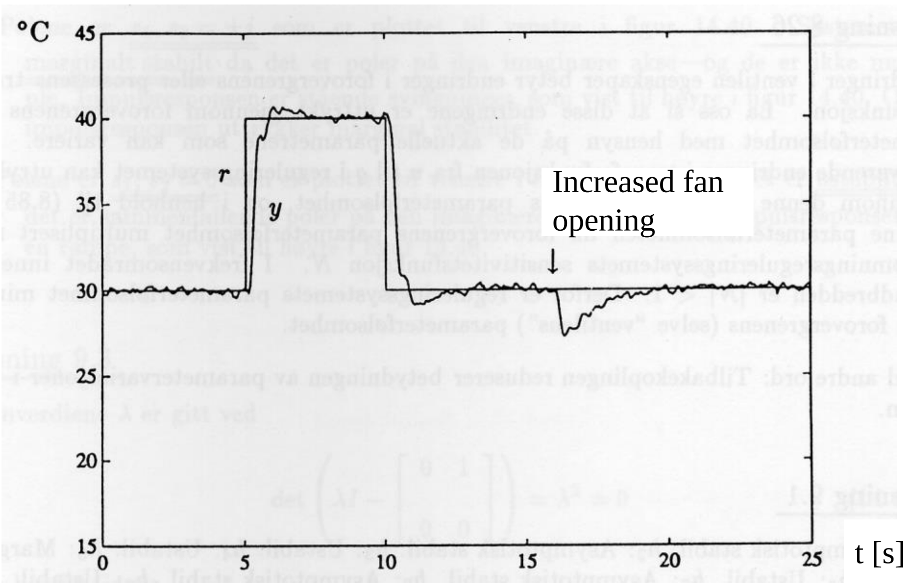


Figure 1.38: Temperature response with PID controller

And we see that the slope is

$$\text{Slope} = 4 = \frac{K_c}{T_i}E = \frac{2}{T_i} \cdot 1 \tag{1.51}$$

which gives

$$T_i = 0.5 \text{ sec} \tag{1.52}$$

Solution to Problem 1.14

$$T_f = 0.1T_d = 0.1 \cdot 2 = 0.2 \text{ s} \tag{1.53}$$

Solution to Problem 1.15

From Figure 1.20:

1. The period is approximately 7.5 min.
2. The max and min values of the the temperature is approximately 43.0 and 35.5 deg C. Then the average temperature is approximately (43.0 + 35.5)/2 = 39.25. And the average control error is 40.0 − 39.25 = 0.75 deg C, which is non-zero.

Solution to Problem 1.16

1. Only the IAE indexes of control systems 1 and 2 can be compared to each other since their IAE values are calculated over the same time interval.

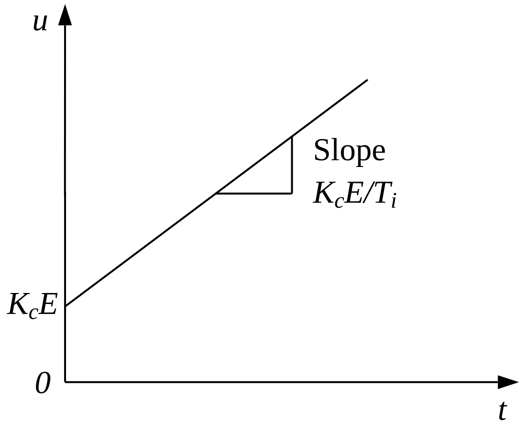


Figure 1.39: Step response of the PI controller.

2. Among control system 1 and 2, system 2 has best performance since it has smaller IAE value than system 1 has.

Part II

INSTRUMENTATION OF CONTROL SYSTEMS

Chapter 2

Piping & Instrumentation Diagrams

In the industry, Process & Instrumentation Diagrams – P&I Ds are widely used as documentation of control systems. A P&I D shows the structure of the control system. It contains easily recognizable drawings of the processes to be controller, e.g. tanks and heat exchangers, together with symbols for instrumentation equipment as sensors, controllers and actuators, e.g. valves and pumps. We have already seen examples of simple P&I Ds in the previous sections

This section gives a brief overview over codes and symbols used in the P&I D standard ISA-5.1 by International Society of Automation. There are other similar standars, both international, national and internal standards in factories.

2.1 Instrument codes

Instrument tags are used to give the instrument a unique name. The instrument code contains a letter code that expresses the function of the instrument (i.e. what it does), and a number code. For example, LC-102 is a Level Controller, while the LT-103 is a Level Transmitter (sensor). When we refer to the instrument code, we use a hyphen between the letter code and the numeric code, but when the instrument code is written in a P&I D, the hyphen is not included.

2.1.1 Letter codes

Table 2.1 shows the most commonly used letter codes used in Process & Instrumentation Diagrams.

Some examples:

- LC = Level Controller
- LIC = Level Indicator Controller = level controller with level indicator

Table 2.1: Common letter codes (identifiers) for instrument symbols in the ISA 5.1 standard

Letter code	1. letter	Subsequent modificator to 1. letter	Subsequent letter
A	Analysis		Alarm
B	Burner, Combustion		
C	User's choice		Control
D	User's choice	Differential	
E	Voltage		Sensor, Primary element
F	Flow rate	Ratio	
G	User's choice		Glass, Gauge
H	Hand		High
I	Current (electric)		Indicate
J	Power		
L	Level		Low
P	Pressure		
Q	Quantity	Integrate, Totalize	
R	Radiation		Record
S	Speed, Frequency		Switch
T	Temperature		Transmit
V	Vibration		Valve
W	Weight, Force		
Y			Computation
Z	Position	Safety Instrumented System (Interlock)	

- LE = Level Element = transducer; the primary device used to detect the process variable to be measured, e.g. an ultrasound level sensor. Note: In P&I Ds, we will use the letter code LT, not LE, to represent the level sensor.
- TT = Temperature Transmitter = temperature sensor
- PDT = Pressure Differential Transmitter
- TY = Temperature computation = some formula related to temperature measurement or regulation, e.g. a feedforward controller based on temperature measurement. The formula must be indicated on an appropriate place in the P&I D.

It is the function that determins which letter symbol to be used. Example: If a differential pressure (dp) cell is used to measure fluid flow in a pipeline, the letter symbol FT – not PT – must be used.

In P&I Ds where emphasis is placed on documenting control structures, and with little emphasis on details such as whether the controller contains a display for showing the process value, you can use the simplest letter code, e.g. LC instead of LIC.

2.1.2 Numeric codes

Numeric codes are used to number the instruments. The ISA 5.1 standard does not define any specific standard for numbering instruments (such as controllers and sensors), but still recommends choosing between so-called parallel and serial numbering:

- **Parallel numbering**: New numbering is started for each new first letter, with 1 being the lowest number.
  Examples: FIC-101. FT-102. LIC-101.
- **Serial numbering**: The numbering is continuous, regardless of letter.
  Examples: FIC-101. FT-102. LIC-103. LIC-104.

Serial numbering is used in the P&I Ds in this book.

The first digit of the number code may represent e.g.

- Area or field
- Unit
- Plant (factory)

The number of digits is not defined by the standard. It must be chosen so that all elements can have their unique instrument code.

2.1.3 Localization

You can use various symbols to indicate the localization of the instrument, so the user knows where she/he can find it (this can be important information particularly in emergency situations). Three different localizations are shown in Figure 2.1.

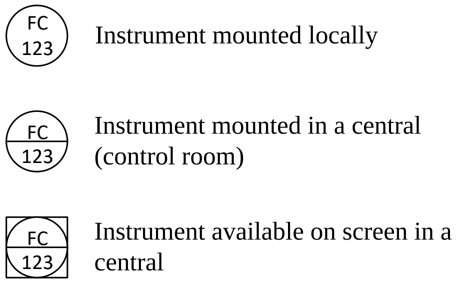

FC
123
Instrument mounted locally
FC
123
Instrument mounted in a central
(control room)
FC
123
Instrument available on screen in a
central

Figure 2.1: Main instrument symbols (FC123 is one example of instrument code)

If it is not important to show the location in a P&I D, you can use the simplest symbol, which is a circle with only the instrument code inside (the upper symbol in Figure 2.1).

2.2 Signals

Figure 2.2 shows various signal symbols.

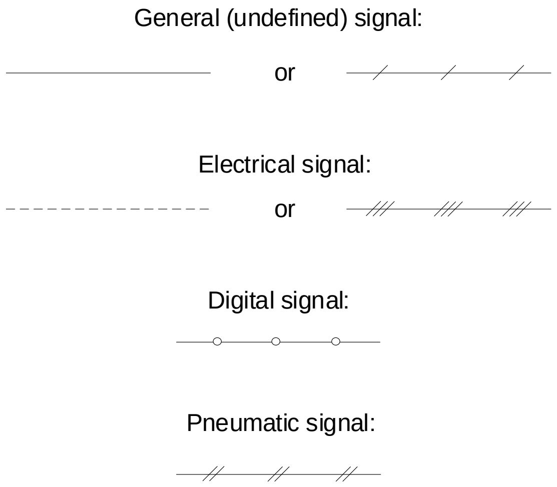
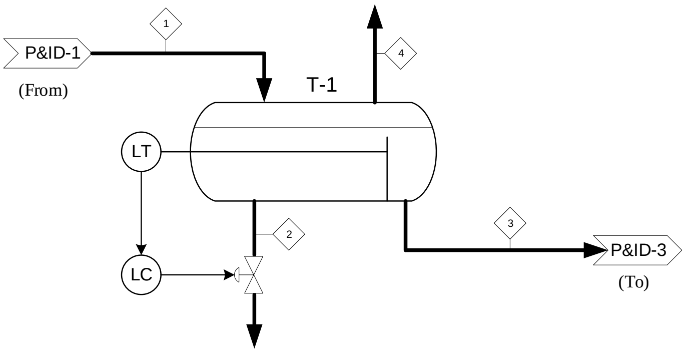


Figure 2.2: Signal symbols

2.3 Material flows

Figure 2.3 shows how material flows (process flows) can be drawn in a P&I D. Material flows should be drawn with lines that are clearly thicker than signal lines. If the plant is so extensive that more than one flowchart is needed for all of the units, references to the adjacent diagrams are indicated, as shown in Figure 2.3.



Material flows

1	2	3	4
Water/oil/gas	Water	Oil	Gas

Figure 2.3: Drawing of material flows in a P&I D. (The example is a water/oil/gas separator. The vertical line inside the separator is a so-called weir that is used to separate the oil and water. The oil flows over the weir.)

In Figure 2.3, the material flows are identified with labels on the pipelines, and the flows are defined at the bottom of the diagram. However, it is not required in the ISA 5.1 standard to label material flows in P&I Ds. Instead, material flows can be labelled in so-called Process Flow Diagrams (PFDs), which are diagrams similar to technical flow charts, but which show fewer details of the instrumentation. ISA's documentation guidelines encourages to avoid giving the same information in different diagrams. However, this rule is not absolute, and control structures may appear both in P&I Ds and PFDs, and you may define material flows in P&I Ds if you find it appropriate.

Note: In Figure 2.3, I have drawn arrow heads on the signal lines pertaining to the level control loop. In the ISA 5.1 standard, arrow heads are generally not used on signal lines, only ordinary lines. However, in this book I will still draw arrow heads to make the direction of the signal flow completely clear. In comprehensive P&I Ds it may cause extra stress to understand the signal direction from the symbols, although the direction is obvious, but implicit. For example, the signal direction between a LT and a LC is obvious – the signal goes from the LT to the LC.

2.4 Process equipment

2.4.1 Codes of process equipment

P&I Ds contain drawings of process equipment such as tanks, heat exchangers, pumps, valves, etc. These can also be represented by letter and number codes, e.g. H-1 for Heat Exchanger number 1. The ISA 5.1 standard does not define process equipment letters, but some common letter codes are shown in Table . The numbering is usually serial.

Table 2.2: Letter codes of process equipment

Code	**Equipment**
C	Column
D	Drum
F	Furnace
H	Heat exchanger
K	Compressor
M	Motor
P	Pump
R	Reactor
T	Tank
V	Valve. Vessel

2.4.2 Valves

Some common valve symbols are shown in Figure 2.4.

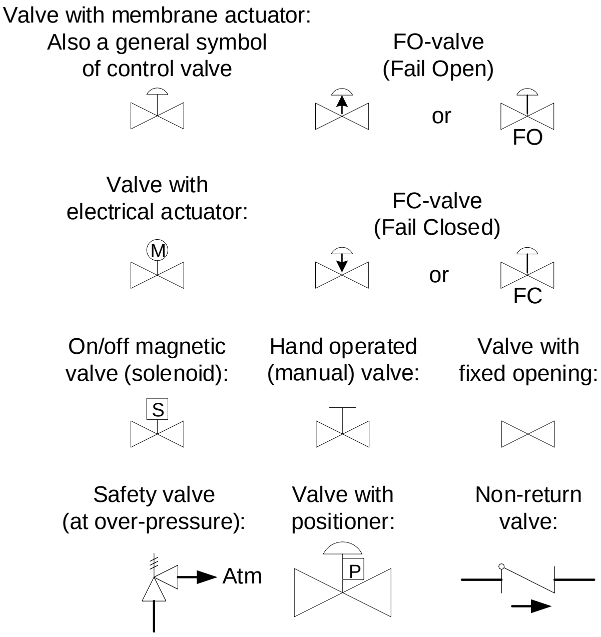


Figure 2.4: Valve symbols.

One comment:

- There are symbols for a Fail Open (FO) valve and a Fail Closed (FC) valve. The “fail” situation means that the power, e.g. air pressure, needed to operate the valve fails. So, a FC valve closes if the air pressure vanish.

2.4.3 Rotational flow components

Figure 2.5 shows various rotational flow components.

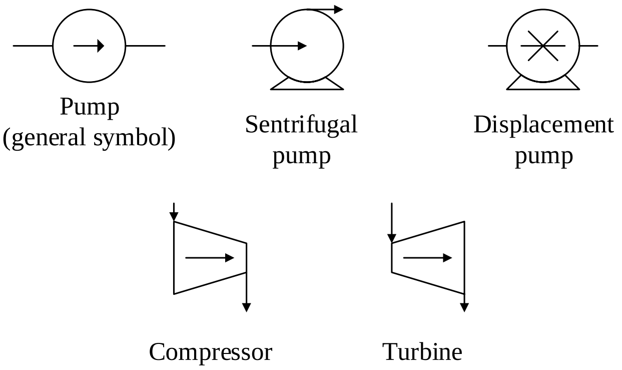


Figure 2.5: Symbols of pumps, compressors and turbines

2.4.4 Heat exchangers

Figure 2.6 shows symbols of heat exchangers.

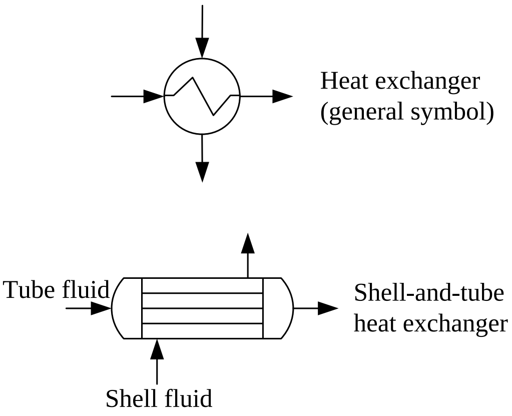


Figure 2.6: Symbols of heat exchangers

2.4.5 Vessels

Figure 2.7 shows symbols of various types of vessels.

2.4.6 Mathematical functions

Figure 2.8 shows two alternative ways of including mathematical functions in P&I Ds. In the example, the control signal u_{ff} is from a feedforward controller, to be added to the

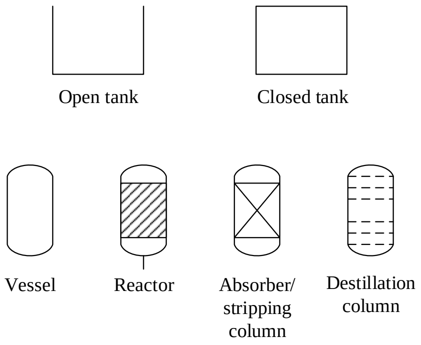


Figure 2.7: Symbols of various types of vessels

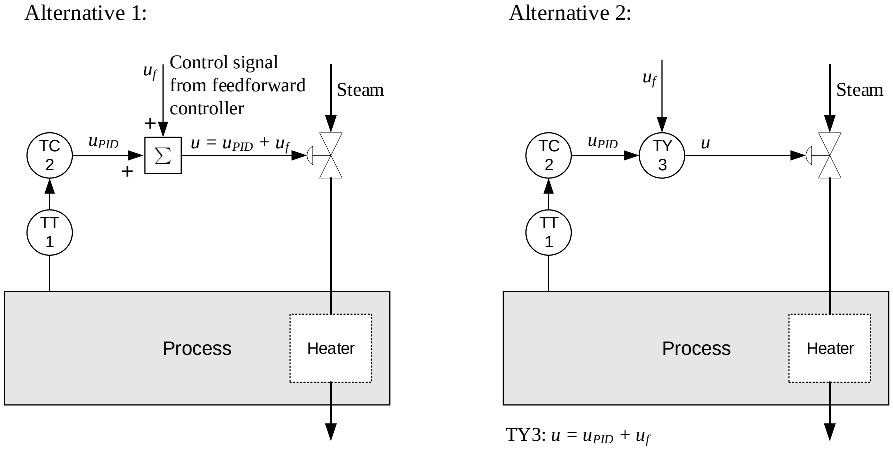


Figure 2.8: Two alternative ways of including mathematical functions in P&I Ds.

control signal u_{PID} generated by a feedback temperature controller (PID controller). The total control signal is $u = u_{\text{PID}} + u_{\text{ff}}$. Two alternatives are shown:

- **Alternative 1**: The mathematical function is specified directly with a suitable mathematical symbol.
- **Alternative 2**: A general letter symbol is used, here TY, where Y stands for "computation", cf. Table 2.1. In this alternative, the mathematical function must be specified at a suitable location in the P&I D.

2.4.7 Logical functions

In addition to arithmetic functions as discussed above, one can enter logical functions into the diagram e.g. to express that a valve should be opened if the temperature in a tank is above a certain limit. Such logical functions are used to implement safety actions, i.e. interlocks, which are activated in critical situations.

A special diagram type denoted System and Control Diagram (SCD) has been developed in Norway for the oil and gas industry. SCDs defines a uniform documentation of both logical control for safety, i.e. interlocks, and process control.

2.5 *Problems for Chapter 2*

Problem 2.1 *P&I D of a level control system*

Given a level control system for a water tank with inlet and outlet. The level control is based on manipulation of a pump in the inlet. The controller is accessible via a computer screen in a control room. Both the control signal and the measuring signal are electric. Draw a Piping & Instrumentation Diagram of the control system.

Problem 2.2 *P&I D of a temperature control system*

Draw a Piping & Instrumentation Diagram of a temperature control system of a process. You can select by yourself the process to be temperature controlled. It is assumed that the controller is accessible in the field. The instruments are numbered with parallell numbering with three digits. Both the control signal and the measurement signal are digital.

Problem 2.3 *P&I D of a separator*

Figure 2.9 shows an oil/water/gas separator. (The separation takes place by a sufficient retention time.) The vertical line inside the separator is a so-called weir that is used to separate the oil and water. The oil flows over the weir.

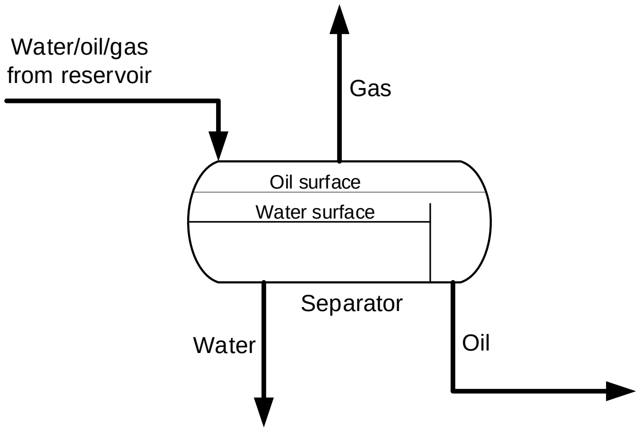


Figure 2.9: Oil/water/gas separator.

Draw a Process & Instrumentation Diagram for the separator according to the following requirements:

- The oil and water levels and the gas pressure in the separator are controlled to their references.

- The references can be shown explicitly with arrows.
- Signal lines may be drawn with arrows (although arrows are not normally used in P&I D standards).
- Control valves are used as actuators, and should be labeled.
- The separator should be labeled.
- Parallel numbering is used.

2.6 *Solutions to problems for Chapter 2*

Solution to Problem 2.1

See Figure 2.10.

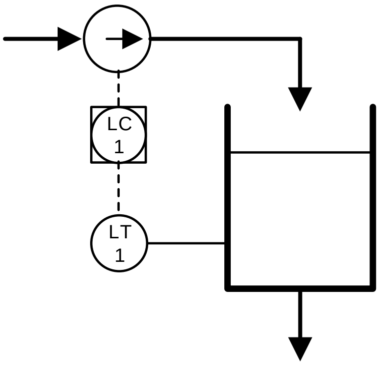


Figure 2.10: P&I Diagram of level control system.

Solution to Problem 2.2

See Figure 2.11.

Solution to Problem 2.3

The P&I D of the separator is shown in Figure 2.12.

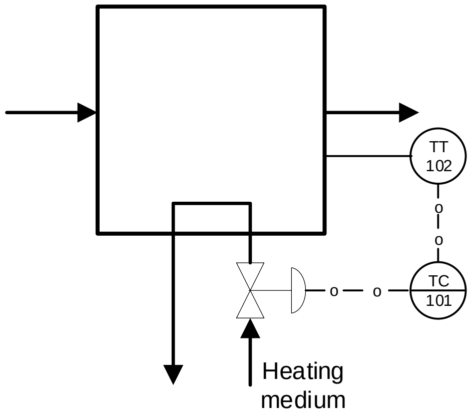


Figure 2.11: P&I Diagram of temperature control system.

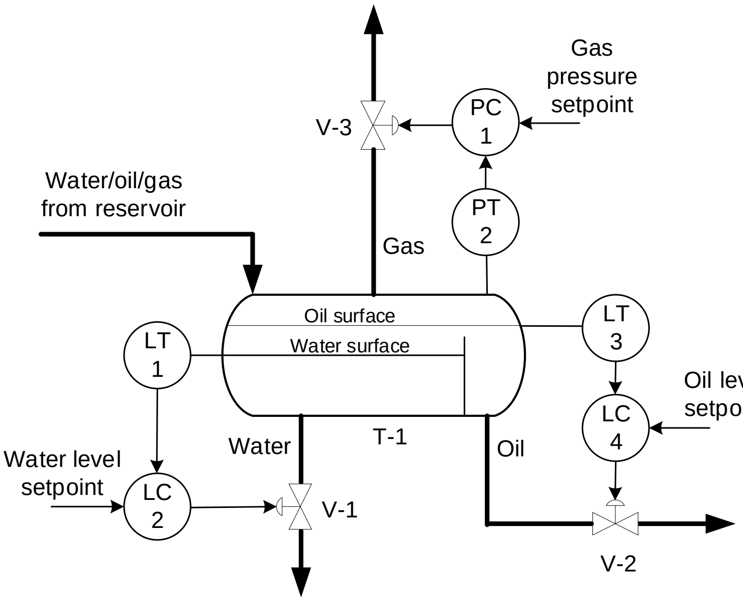


Figure 2.12: Control structure of an oil/water/gas separator.

Chapter 3

Components of control systems

3.1 Introduction

Section 1.3.3 provided a brief description of the components of such control systems. In the present chapter, we will take a more detailed look at these component. Figure 3.1 shows a block diagram of a general feedback control system. The diagram is the same as Figure 1.8, but repeated here for convenience.

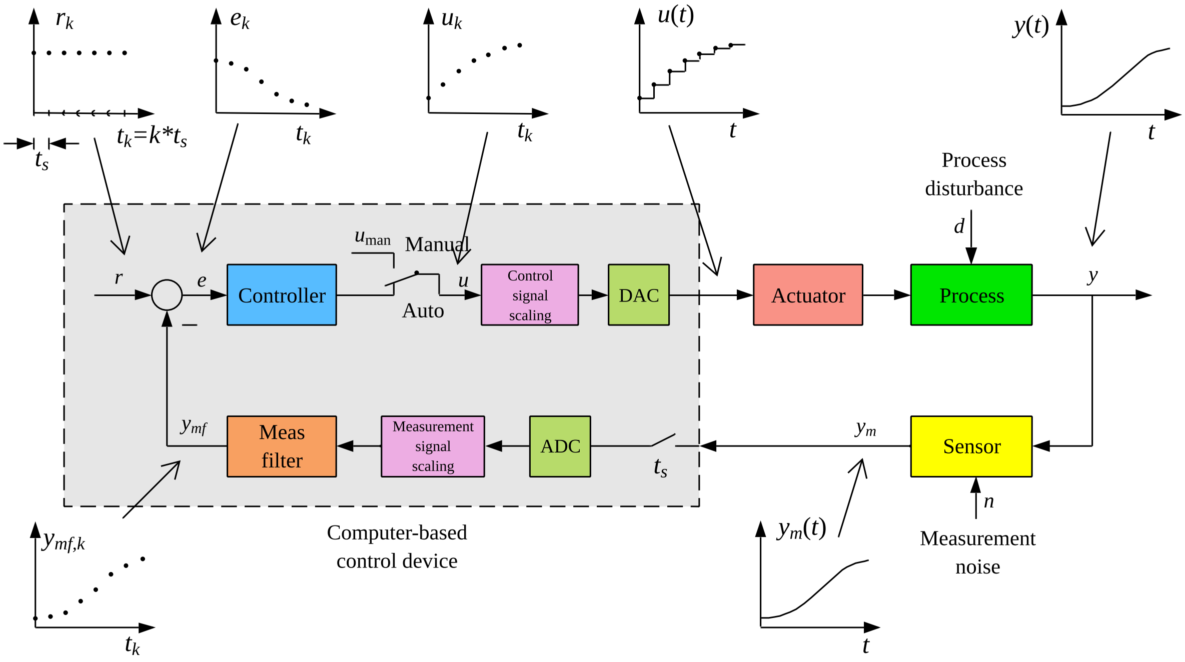


Figure 3.1: Block diagram of a control system.

3.2 Automation hardware

3.2.1 Introduction

The following sections presents various equipment implementing the component denoted automation hardware in Figure 3.1.

3.2.2 Process controllers

The term process controller is often used for standalone automation units that implement a small number of – typically one – PID control loops. Figure 3.2 shows an example of a process controller, namely the ABB CM50 [1]. Values of reference, process variable, and control signal are displayed on the front panel.

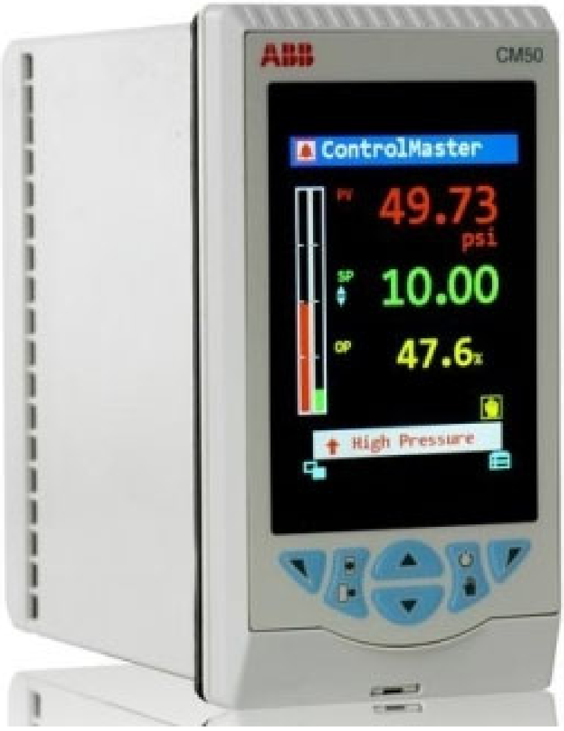


Figure 3.2: The process controller ABB CM50 (height 14.4 cm, width 7.6 cm).

Figure 3.3 shows the backplane of the controller. Sensors and actuators are connected with cables to terminals on the backplane.

[1] Control Master 50

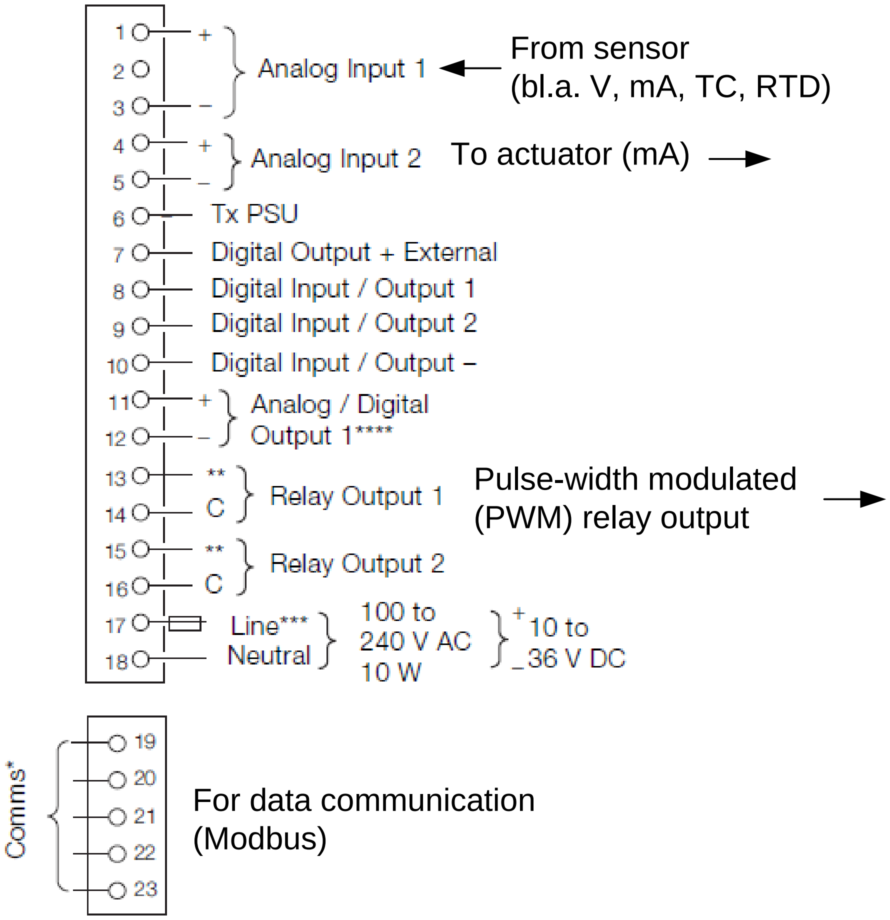


Figure 3.3: Backplane of the process controller ABB CM50.

Below is a list of some of the features of the ABB CM50 controller. The list is representative of process controllers, although this controller is quite advanced.

- Control functions:
  - PID controller with adjustable parameters:
    - Proportional band P in range 0-999%. (The relationship between the P and controller gain K_c is $P = 100/K_c$.)
    - Integral time T_i in range 0-10000 s.
    - Derivative time T_d in range 0-999.9 s.
  - Manual control with control signal adjustable in range 0-100%.
  - On/off controller, cf. Ch. 1.3.5.
- Control tuning:
  - Autotuner with the "relay method", cf. Ch. 14.6.
  - Gain scheduling, cf. Ch. 14.10.3.

  - Adaptive PID setting, i.e. continuous updating of the PID parameters on the basis of an estimated transfer function process model.

- Sampling time (time step): 0.125 s.

- Measurement filter: Moving average filter with averaging interval adjustable in the range 0-60 s, cf. Ch. 3.4.6.1.

- Reference (setpoint): Can be set locally on the controller, or received externally, e.g. from another controller or from a sensor. The reference can be given a ramp-shaped change between two different values to obtain a smooth shift from one reference value to another.

- Analog inputs: Measurement signals (from sensors), cf. Ch. 3.3:

  - Current in the range 0-50 mA (which covers the standard range 4-20 mA)
  - Voltage in the range 0-25 V
  - Millivolt signals in the range 0-150 mV
  - Resistance values in the range 0-55 Ω
  - Thermocouples (eng.: thermocouples = TC) of many types: J, K, L, etc.
  - Resistance thermometer: Pt100 elements within the range –200 - +600 °C.

- Analogue output signals (control signals to actuators):

  - mA in the range 4-20 mA or 0-20 mA
  - Pulse-width modulated control signal, cf. Ch. 3.6.3
  - Split-range control, cf. Ch. 16.3.

- Digital inputs from e.g. limit switches and push buttons.

- Digital outputs for controlling e.g. lamps, relays and motors.

- Logical control with logical operators as AND, OR, etc.

- Data communication: Connection to external devices with Modbus or Ethernet.

- Alarms: A warning can be given on the front panel if the process value exceeds alarm limits.

- Programming: From the front panel or from a computer (PC) using a wireless connection.

- Power supply to the regulator: Mains voltage or 24 VDC.

- Control structures: The CM50 controller contains ready-made templates for the following structures, see Example 3.1.

  – Single-loop control, cf. Ch. 1.3.2. (However, two independent regulation loops can be implemented.)

  – Cascade control, cf. Ch. 16.1.

  – Ratio control, cf. Ch. 16.2.

  – Feedforward control, cf. Ch. 17.

Example 3.1 *Control structure template in process controller CM50*

Figure 3.4 shows an example from the control structure templates, namely level control of a boiler. The process variable to be controlled to a setpoint is the water level in the boiler. The control signal from the controller is the control signal for the air-actuated valve. The control structure is cascade control. The primary loop is based on feedback from sensor LT. The secondary loop is based on feedback from flow sensor FT for the feedwater. Feedforward control is based on sensor FT of the steam flow.

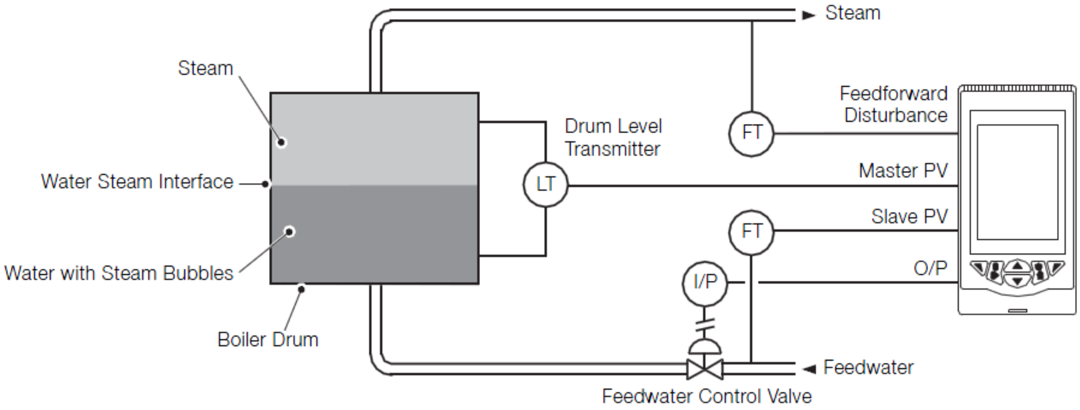


Figure 3.4: From the control structure templates of ABB CM50: Level control of a boiler.

[End of Example 3.1]

3.2.3 Programmable logic controllers (PLCs)

Programmable logical controllers (PLCs) is a type of automation equipment that is widely used – in process industry, discrete processes, building automation, water resource recover plants[2], motor control, traffic control, etc.

PLCs are available both as relatively small compact units, and as modular systems where special modules can be added as required. Figure 3.5 shows a PLC.

The name PLC suggests that PLCs are meant for logical control of valves, motors, lamps, etc., but typically modern PLCs can also run PID control.

Figure 3.6 shows the principle structure of a PLC. Program development usually takes place on a PC, possibly on a hand-held programming tool, and the finished program is then downloaded to the PLC where it runs on the microprosessor (see Figure 3.6), independently of the PC.

PLCs can be programmed in different types of programming languages. The open standard IEC 61131-3 standard (IEC = International Electrotechnical Commission) defines the following languages:

[2] Older term: Wasterwater treatment plants.

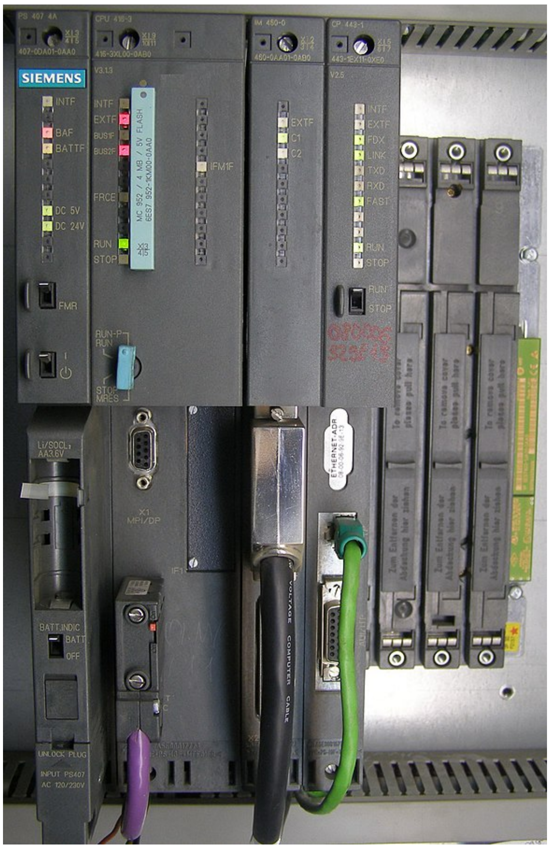


Figure 3.5: A Simatic PLC. (Public domain. https://en.wikipedia.org/wiki/Programmable_logic_controller.)

- Ladder diagram (LD), graphical programming
- Function block diagram (FBD), graphical programming
- Structured text (ST), textual programming
- Instruction list (IL), textual programming
- Sequential function chart (SFC), which is not really a separate language, but instead a standard for organizing the program in serial and parallel sequences.

Different vendors of PLCs may offer their own variation of the above languages.

3.2.4 Programmable automation controllers

Programmable automation controllers (PACs) can be considered as an alternative to a PLC. A PAC is a modular system for e.g. logical and sequential control and continuous PID control. In some ways, PACs are more manufacturer-specific than PLCs as they typically have their own "proprietary" programming languages. These languages may be flexible and powerful than PLC languages, and can therefore give PACs greater functionality than PLCs.

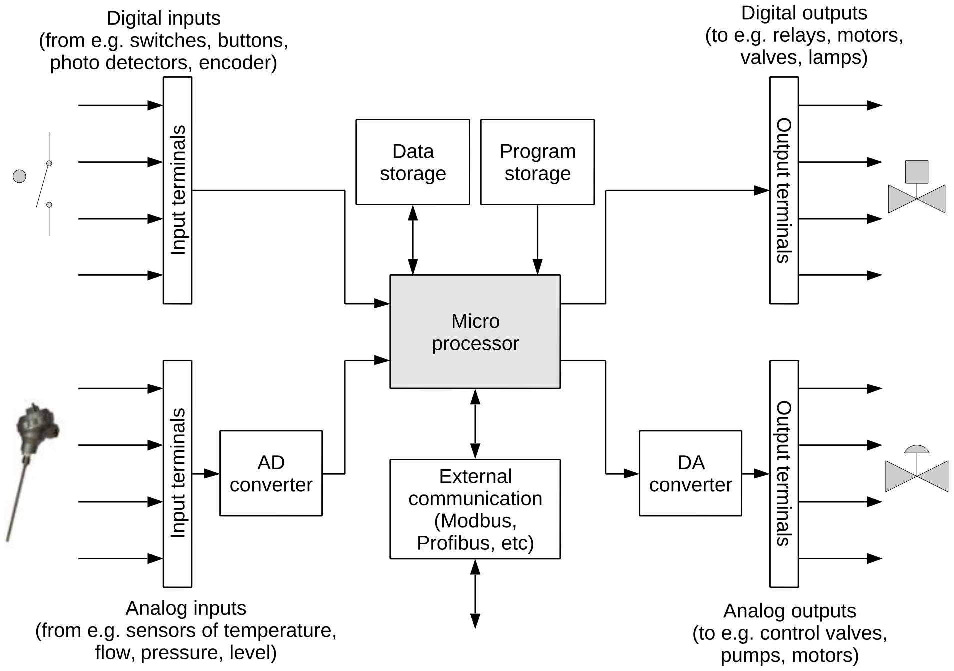


Figure 3.6: The structure of a PLC. (AD = analog-digital. DA = digital-analog.)

Figure 3.7 shows an example of a PAC, namely Compact FieldPoint (National Instruments). The control program is developed in LabVIEW on a PC and then downloaded to the PAC equipment where it runs independently of the PC.

National Instruments presents the CompactRIO as follows: “The CompactRIO platform features a range of embedded controllers with two processing targets: (1) a real-time processor for communication and signal processing and (2) a user-programmable FPGA to implement high-speed control and custom timing and triggering directly in hardware. Eliminate the need for separate subsystems by connecting directly to sensors, displays, cameras, and motors and take advantage of powerful development and run-time software.

3.2.5 Plantwide control systems

In modern factories, the control (or automation) systems are implemented in complex, structured computer-based systems.

In the past, when computer performance was generally far weaker than it is today, the control system was implemented using one central computer, possibly with one additional, redundant computer to increase security. These days, control systems instead have a distributed structure. This means that the implementation of the control system is distributed on a number of computers. These computers can be process controllers, PLCs, PACs and/or other types of automation devices.

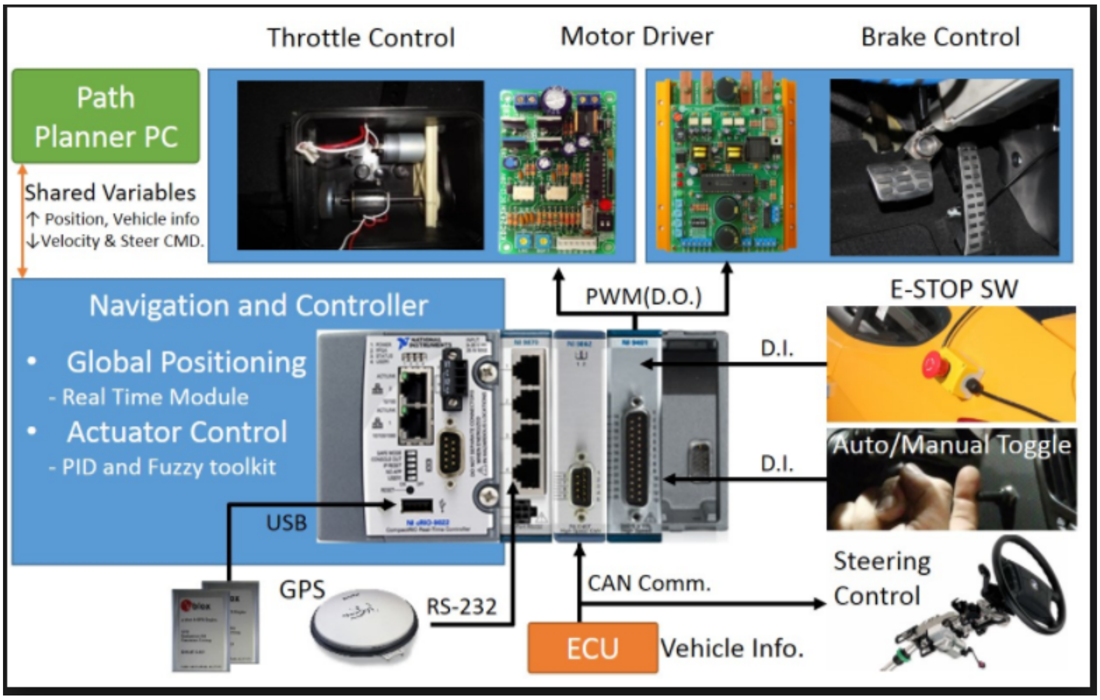


Figure 3.7: CompactRIO (National Instruments).

Plantwide control systems are modular and scalable. This implies that the existing system can be modified and expanded with new control devices modules to obtain the desired functionality of the system.

Figure 3.8 shows an example of a plantwide control system. The system consists of a number of levels:

- **Field level**. Here, the control system interacts with the physical process via sensors and actuators. In Figure 3.8, motors are the physical processes.
- **Control level**. This level includes the PLCs, PACs, process controllers and process stations. Different types of automation devices can communicate with each other using digital communication based on standardized communication protocols. It is common for control systems to consist of equipment from several different suppliers.
- **Operator level**. This level includes process operators and computers – operator stations -- which are used for process monitoring and possibly control in e.g. control room. The operator stations display process images with tanks, containers, valves, etc., as well as dashboards, alarm indicators (for example too high a level in a tank) and diagrams with plots of process data (for example the time course of a level in a tank).
- **Management level**. This level includes operations and company management and computers that are used there for overall monitoring and operational planning. This kind of planning is called ERP (enterprise resource planning). ERP software is a software tool for the overall management of business areas such as production, storage, sales, purchasing and finance.

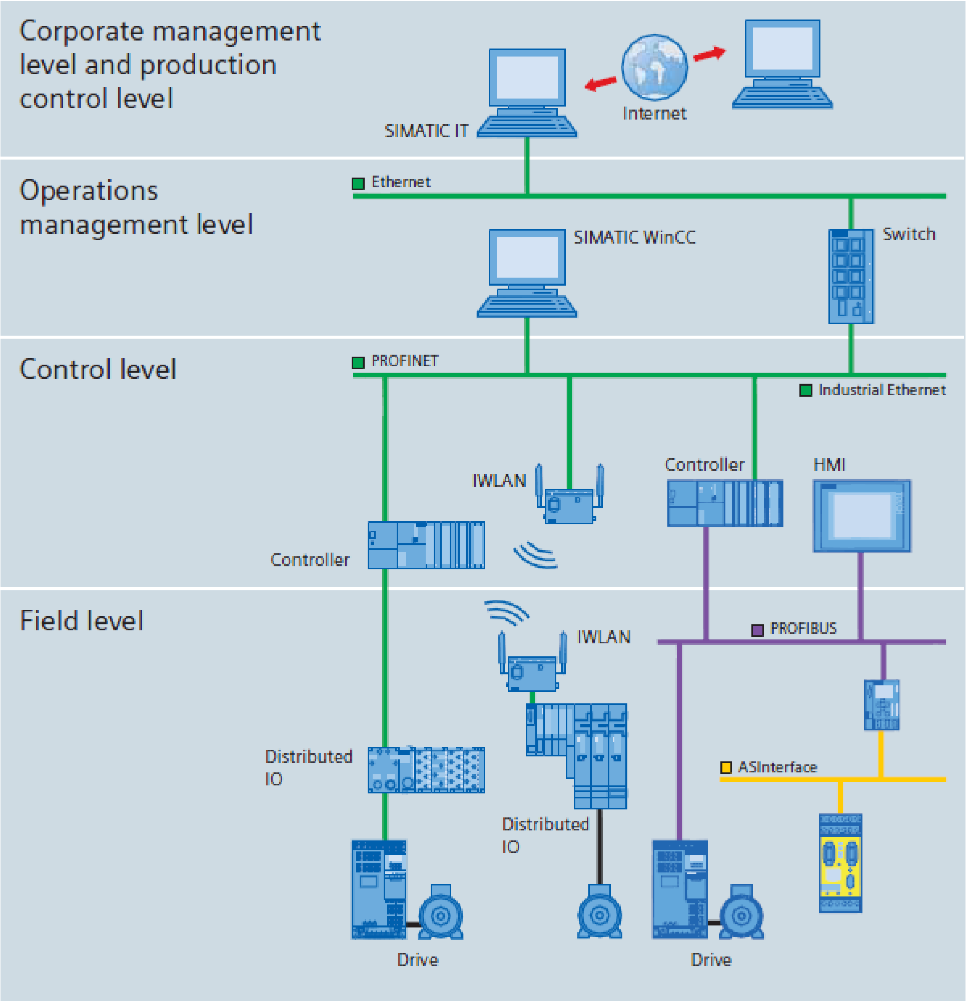


Figure 3.8: Example of a plantwide control system (PCS7, Siemens).

Between the various levels, data is exchanged digitally using standardized communication techniques. Analog signals (current or voltage) are only used at the field level, but also digital communication is common at field level.

Process data, which are logged values (time series) of process measurements, control signals, references or setpoints, alarm signals etc., are stored in a process database. Historical process data can later be retrieved for presentation and analysis on computers at the operator level and management level.

DCS and SCADA

A distinction is often made between DCS control systems and SCADA control systems, although there are not very clear lines between them. DCS is short for Distributed Control Systems; SCADA for Supervisory Control and Data Acquisition. Here are some special features of the two types of control systems:

- DCS systems consist of equipment and software which mainly originates from one total supplier where equipment, functionality and data communication are integrated and optimized.
- DCS systems can have special processing stations which are powerful computers in control cabinets. These computers can run advanced, computationally demanding algorithms for e.g. model-based control and simulation.
- A DCS system often has its own, proprietary function block-based language for developing control programs. Some of the function blocks may be advanced. For example, Emerson's DeltaV system has its own functions for MPC control (Model-based Predictive Control) and functions for process modeling with neural network models.
- SCADA systems use PLCs and/or RTUs (RTU = Remote Terminal Unit). An RTU is an I/O unit that can send out analogue and digital data to actuators and receive such data from sensors. The SCADA software running on operator-level PCs, may come from a different supplier than the PLC and RTU supplier(s).

Examples of DCS/SCADA suppliers/products are: Siemens/PCS7, Rockwell Automation/PlantPAX, Honeywell/Experion, ABB/System 800xA and Emerson/DeltaV, Schneider Electric/AVEVA.

3.2.6 Platforms for home-made automation systems

In the sections above, we took a look at various platforms industrial automation systems. The following sections gives a brief presentation of some platforms for home-made automation systems.

3.2.6.1 Standard PC with I/O device

In facilities where there are no strict industrial requirements for regularity and safety, a standard PC with an I/O device may be used to create a powerful and flexible computer-based control and monitoring system. The application program runs on the PC. Examples of such facilities are experimental setups for research and teaching which are used for a limited period of time, and which are not damaged if the PC hangs up. Such PCs are often used for things other than control and measurement, e.g. editing documents, calculations, internet searches, etc., and there is indeed a chance that the PC will hang or be disturbed in some other way.

The PC may be programmed with tools such as LabVIEW or Matlab/Simulink. Relatively inexpensive I/O may be purchased for reading measurement signals and generating control signals, typically in the form of voltage signals. The I/O device may communicate with the PC via USB. If the PC or the program stops, the I/O device may hold the last control signal, which prevents the system from a shut down.

Figure 3.9 shows an example of a PC-based control and monitoring system for an experimental facility of a biogas reactor. The PC runs a LabVIEW program. An I/O device with USB communication with the PC is mounted in a electronics cabinet. From the cabinet there are cables for transmitting voltage signals to actuators (here: pumps) and from sensors (here: sensors for temperatures, gas flow and gas concentration). The PC is connected to the internet. With suitable software, remote login can be carried out on the PC via any other PC on the Internet, to run and maintain the LabVIEW program and to download files with timeseries of data.

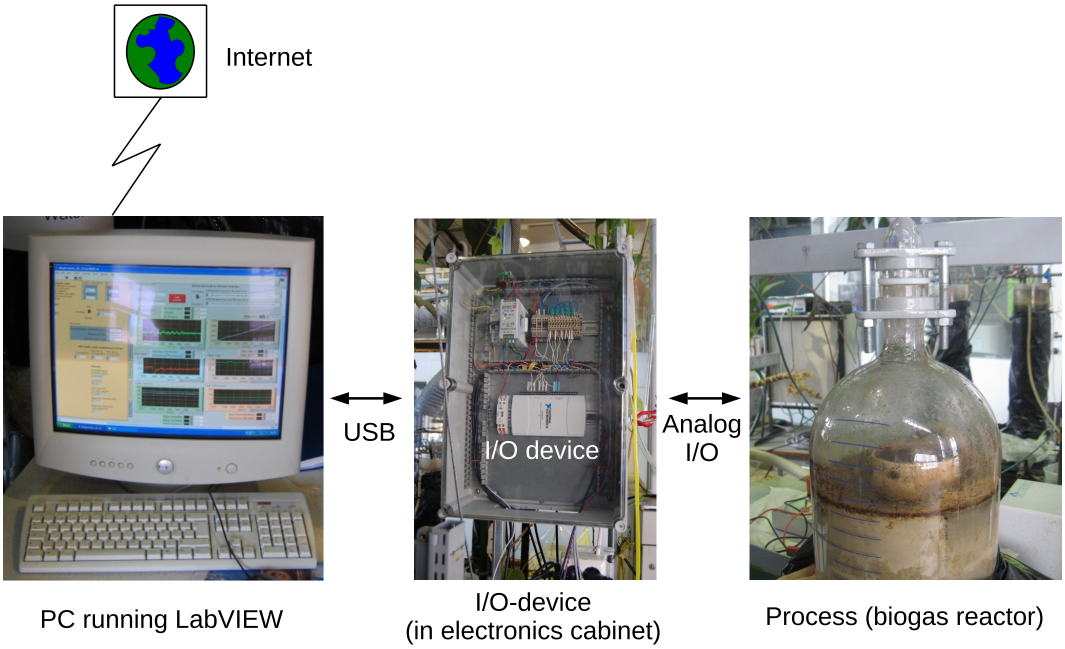


Figure 3.9: PC-based control and monitoring system for an experimental setup for a biogas reactor (University of South-Eastern Norway).

Alternatives to standard PCs are industrial PCs, which are PCs designed to be far more robust than normal PCs.

3.2.6.2 Raspberry Pi

Raspberry Pi is a an electronics board with a programmable microcontroller and various types of inputs and outputs. Figure 3.10 shows a Raspberry Pi 3. It can be connected to various kinds of extention equipment, e.g. electronic breadboards where various components like motors, switches, LEDs, etc. can be mounted and interconnected, displays (both standard monitors and small displays), and a keyboard.

Essential features of the Raspberry Pi:

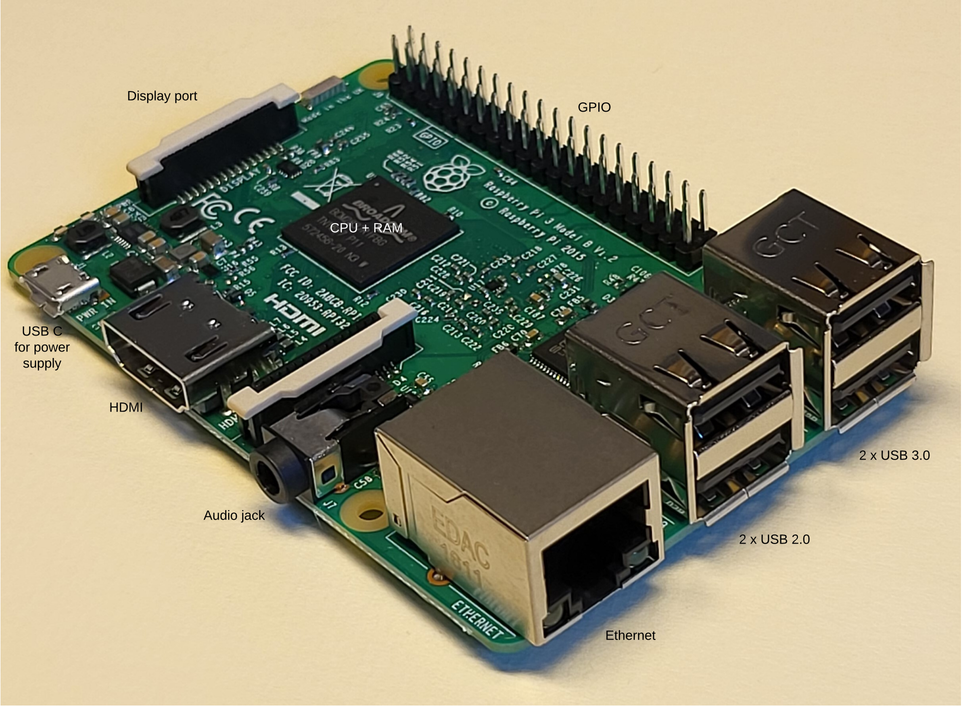


Figure 3.10: Raspberry PI 3.

- Physical dimensions: 8.6 x 5.7 cm.
- Digital inputs and outputs (I/O).
- Analog voltage inputs, to read measurement signals from sensors.
- No analog outputs! However, some of the digital outputs can be used for approximate analog output based on pulse-width modulation, cf. Section 3.6.3.
- The standard operating system is Raspberry Pi OS (formerly denoted Raspbian) which is based on Linux.
- The microcontroller can be programmed with Python code.
- Of course, any algorithm, e.g. a PID controller, a signal filter, etc. can be programmed on the Raspberry Pi.

The home page of Raspberry Pi is on

https://www.raspberrypi.org/

Raspberry Pi Pico

Raspberry Pi Pico is a very small electronics board with a programmable microcontroller and various types of inputs and outputs. Figure 3.11 shows a Raspberry Pi Pico with pre-solded headers (pins) mounted on a breakout board for connecting wires to external equipment. The Raspberry Pi Pico can also be mounted directly on electronics breadboards (a breadboard is shown in Figure 3.12).

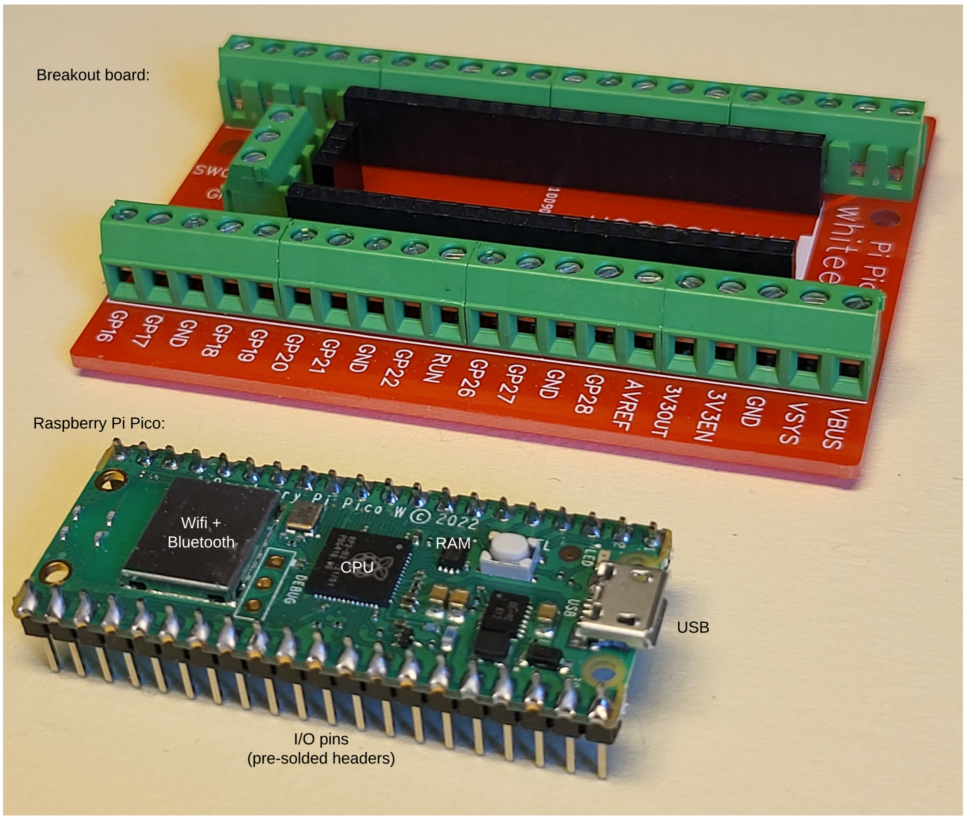


Figure 3.11: Raspberry Pi Pico with breakout board.

Essential features of the Raspberry Pi Pico:

- Physical dimensions: 5.1 x 2.1 cm.
- Digital inputs and outputs (I/O).
- Analog voltage inputs, to read measurement signals from sensors.
- No analog outputs! However, some of the digital outputs can be used for approximate analog output based on pulse-width modulation, cf. Section 3.6.3.
- The microcontroller can be programmed with MicroPython code – not full-fledged Python.

- Algorithm for PID control, signal filtering, etc. can be programmed.

3.2.6.3 Arduino

Arduino[3] is a an electronics board with a programmable microcontroller and various types of inputs and outputs. There are various versions of Arduino hardware. Figure 3.12 shows an Arduino UNO R3 with an electronics breadboard where various components like motors, switches, LEDs, etc. can be mounted and interconnected.

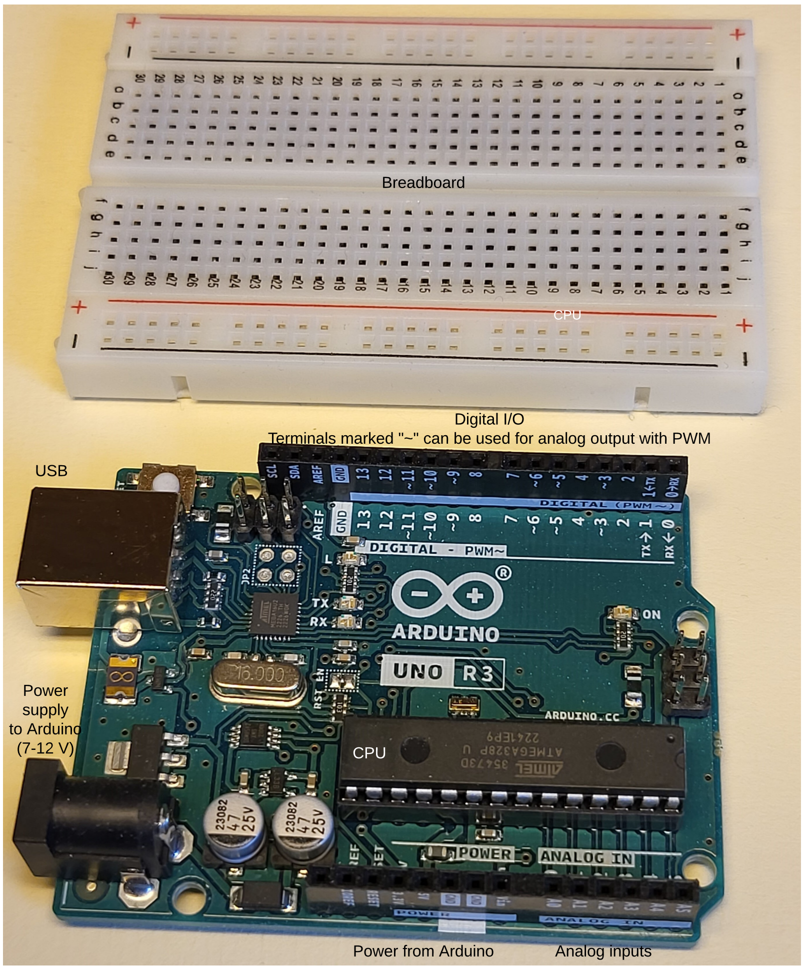

Breadboard
CPU
Digital I/O
Terminals marked "~" can be used for analog output with PWM
USB
Power supply to Arduino (7-12 V)
CPU
Power from Arduino
Analog inputs

Figure 3.12: Arduino Uno R3 with electronics breadboard.

Essential features of the Ardunio:

- Digital inputs and outputs (I/O).

[3] According to Wikipedia, the name Arduino stems from the bar Arduino in Ivrea, Italy where the founders of the project used to meet.

- Analog voltage inputs, to read measurement signals from sensors.
- No analog outputs! However, some of the digital outputs can be used for approximate analog output based on pulse-width modulation, cf. Section 3.6.3.
- The microcontroller can be programmed with the Arduino programming language, which is based on the C language. Both C and C++ code can be used in the programming.
- Algorithm for PID control, signal filtering, etc. can be programmed.

The home page of Arduino is on

https://www.arduino.cc/

3.3 Sensors

3.3.1 Introduction

This chapter describes examples of sensors for measuring the following common process variables:

- Temperature
- Pressure
- Level
- Liquid and gas flow
- Gas concentration
- Position
- Speed

In one of the problems at the end of this chapter, you are asked to find additional sensors.

3.3.2 Temperature sensors

The two most common temperature sensors – or thermometers – are thermocouple thermometers and resistance thermometers. Figure 3.13 shows some industrial temperature sensors including the transmitter, which is the "head" of the components shown in Figure 3.13. The transmitter typically outputs a current signal in the standard range of 4-20 mA and/or a digital measurement signal representing the temperature.

Figure 3.13: Industrial temperature sensors including transmitters (the “head” of the components). [Autek AS]

3.3.2.1 Resistance thermometer

Resistance thermometers – also denoted resistance temperature detectors – (RTD) are perhaps the most commonly used type of temperature sensor. Figure 3.14 shows the principal construction of an RTD.

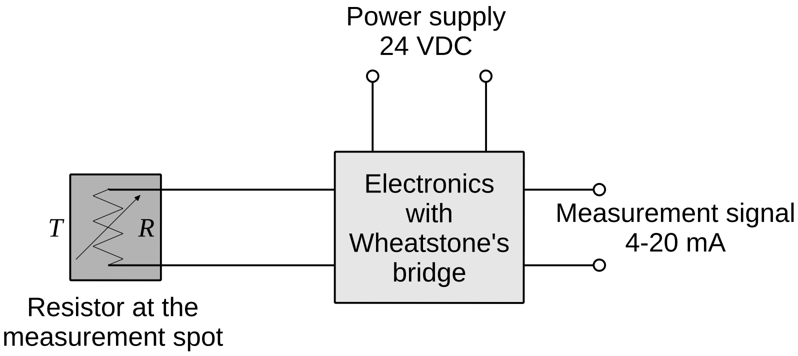

Power supply
24 VDC
T
R
Electronics with Wheatstone's bridge
Measurement signal
4-20 mA
Resistor at the measurement spot

Figure 3.14: Principal construction of a resistance temperature detector (RTD).

The temperature is detected using a resistor placed at the measurement spot. The resistance R [Ω] vary with the temperature T [°C] in a known way:

$$R(T) = R_0(1 + aT) \tag{3.1}$$

where R_0 is the resistance value at 0°C, and a is a constant that depends on which metal is used as resistance. R is found with a special electrical circuit called Wheatstone's bridge which is generally used for precise calculation of an unknown resistance value.

Pt100 sensors are the most commonly used type of RTDs. Pt is short for platinum. For

Pt100 sensors,

$$R_0 = 100\ \Omega \tag{3.2}$$

$$a = 3.9083 \cdot 10^{-3\mathrm{o}}\mathrm{C}^{-1} \tag{3.3}$$

Once, R is detected (with the Wheatstone's bridge), T can be calculated from (3.1):

$$T = \frac{(R/R_0) - 1}{a} \tag{3.4}$$

Since the measurement principle is based on an accurate measurement of the resistance R, variable resistance value in the signal lines due to e.g. temperature changes and aging can give an inaccurate temperature measurement. To increase the accuracy, one or two additional conductors can be connected between the bridge connection and the resistance at the measuring point. Two-wire sensors as in Figure 3.14 are the least accurate, while four-wire are the most accurate.

Resistance thermometers have an accuracy of approx. ±0.2 °C. They can be used for measuring temperatures in the range of approximately $[-100, 500]$ °C, but for a given sensor the range will be smaller, e.g. $[0, 100]$ °C.

As an alternative to a Pt100 sensor, you can use a Pt1000 sensor[4], cf. Example .

Example 3.2 *Temperature sensor and transmitter*

Figure 3.15 shows an electronics box including a Pt1000 sensor and a transmitter. (The box is used in a temperature control application of a kettle, but the kettle is not shown in the figure.) The Pt1000 sensor is connected to the transmitter, which generates a current measurement signal in the range 4-10 mA representing the temperature in the range 0-100 °C, with a linear relation between the ranges:

$$\text{4-10 mA} \Longleftrightarrow \text{0-100}^{\mathrm{o}}\mathrm{C} \tag{3.5}$$

A precision resistor of 250 Ω is mounted between the current signal terminals to convert the current signal into a voltage signal. The reason for this transformation is that the data acquisition device (not shown in Figure 3.15) can only read voltage signals, not current signals. According to Ohm's Law, the current signal and voltage signal ranges relate as follows:

$$\text{4-10 mA} \Longleftrightarrow \text{1-5V} \tag{3.6}$$

Then we have,

$$\text{1-5 V} \Longleftrightarrow \text{0-100}^{\mathrm{o}}\mathrm{C} \tag{3.7}$$

[End of example 3.2]

[4] I think you can guess what the number 1000 means here.

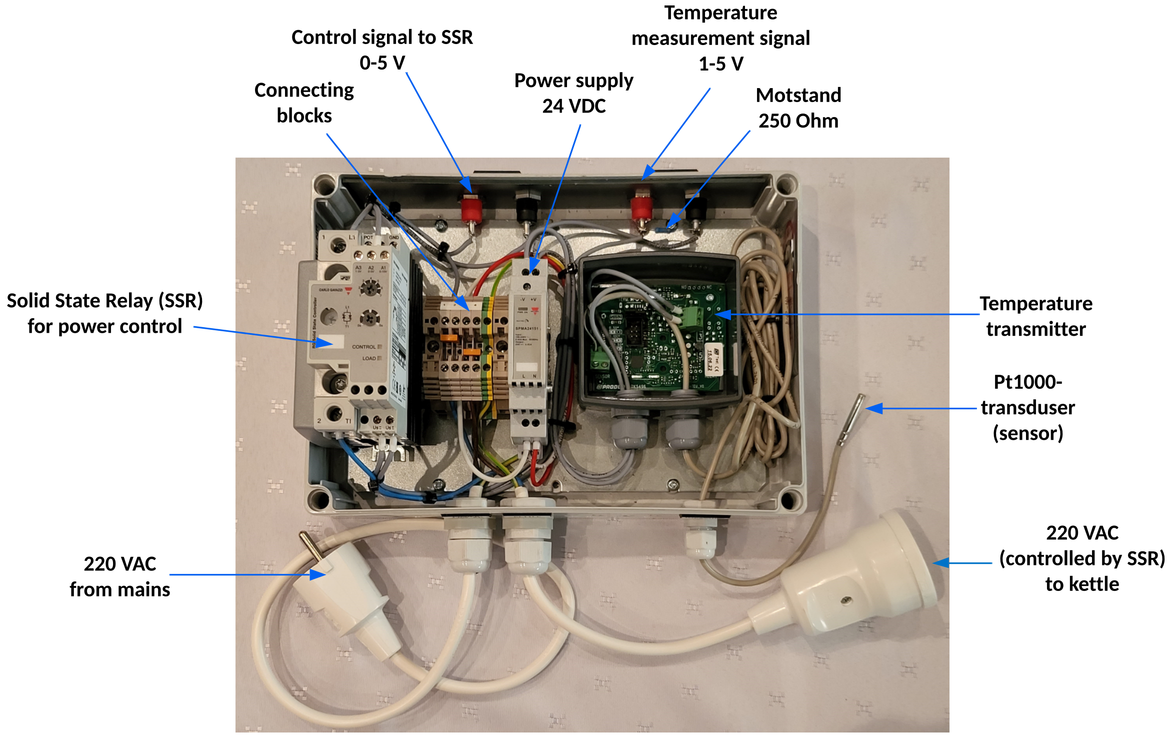

Control signal to SSR
0-5 V
Connecting blocks
Power supply
24 VDC
Temperature measurement signal
1-5 V
Motstand
250 Ohm
Solid State Relay (SSR) for power control
Temperature transmitter
Pt1000-transduser (sensor)
220 VAC from mains
220 VAC (controlled by SSR) to kettle

Figure 3.15: Electronics box including a Pt1000 sensor and a transmitter.

3.3.2.2 Thermocouple thermometers

The measuring principle for thermocouple thermometers is based on the Seebeck effect: An electromotive force (EMF) exists between two points of a metallic electric conductor if the points are at different temperatures. Some metals have greater EMF per temperature difference than others, i.e. different metals have different Seebeck constants, also denoted thermoelectric sensitivity, in unit μV/K. It is this phenomenon, i.e. that different metallic conductors have different Seebeck constants, that is exploited in thermocouples.

Figure 3.16 illustrates the principle of thermocouple thermometers. T_m is the temperature that the thermocouple thermometer is to be measured. Two conductors, which must be different metals, are connected together at the measuring point, also denoted the “hot junction”. An expansion cable of the same metal type as the measurement lead wires themselves can be used to connect the measurement element, which is in or near the process, to a voltage measurement meter that is typically at a distance off the measurement point, e.g. in a room. The place where the generated voltage v is detected, is called the reference point, or the “cold junction”. The temperature T_r at the reference point is assumed known. In modern thermocouples, T_r is measured with a temperature-sensitive semiconductor component, e.g. a thermistor, which is built into the thermocouple. T_m can now be found from the detected voltage v for a given thermocouple. However, we will not go into more details about how to get T_m from v.

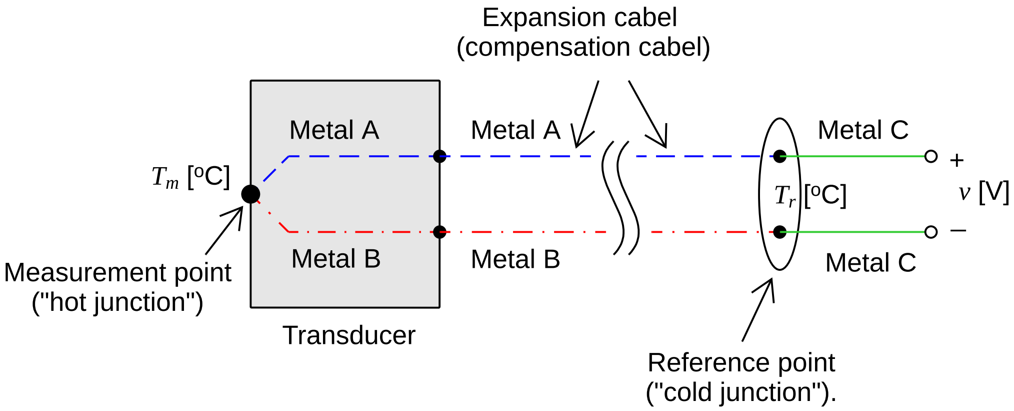


Figure 3.16: The principle of thermocouple thermometers.

The accuracy of thermocouple thermometers is approximately 2 Kelvin.

Thermocouple thermometers will of course not react instantaneously to temperature changes. A typical time constant (or response time, cf. Section 9.3) of thermocouples without encapsulation is 10-20 ms, while the time constant with solid encapsulation can be in the region of 10-20 s Johnson (2000).

Thermocouple thermometers come in different versions, each having a unique combination of metallic conductors. The different types are identified by a letter code: K, E, J, N, etc. The different types have different thermoelectric sensitivities, different measurement ranges, and different robustness against environmental effects. Here are some data of a type K thermocouple, which is a widely used type[5]:

- *Metal conductors*: Cromel + Alumel (both are alloys with mainly nickel).
- *Sensitivity*: It varies with temperature. As an example, for the range $[100, 110]$ °C, the thermoelectric sensitivities is 41.3 μV/K.
- *Temperature range of application*: $[-35, 1260]$ °C.

3.3.3 Pressure sensors

There are many types of pressure sensors. Here we will only look at the sensor type denoted dp (differential pressure) cells.

Dp cell pressure sensors are widely used as pressure sensors. They can be used also for measuring other process variables, such as

[5] Information by Autek AS, Norway

- Level, cf. Section 3.3.4
- Flow of liquid and gas, cf. Section 3.3.5

Figure 3.17 illustrates that one type of dp cell can be used for measuring feed water flow, liquid level and steam pressure.

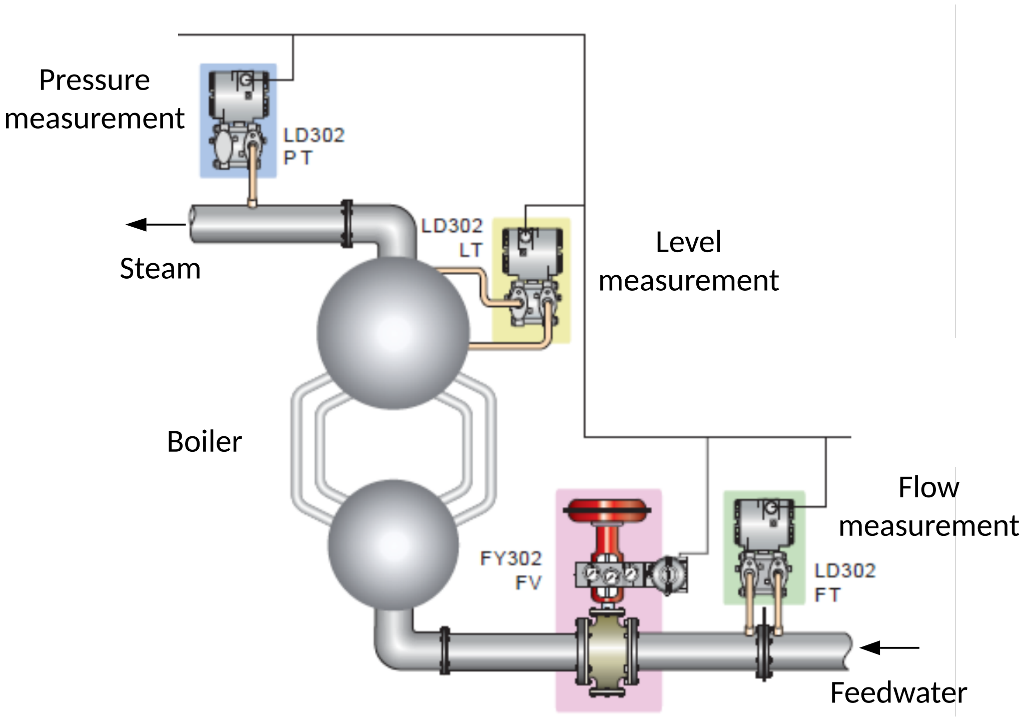


Figure 3.17: One type of dp cell is used for measuring feed water flow, liquid level and steam pressure in a boiler.

Figure 3.18 shows an example of the construction of a dP cell. The left part of the figure shows the structure of sensor or transducer itself. P_H is the pressure on the high pressure side, P_L on the low pressure side. When these two pressures are different, the pressure difference will cause the diaphragm, which is a plate that can hold an electric charge, to be displaced and to have a changed position in relation to two fixed metallized plates. The plates and diaphragm form an electrical capacitance. The displacement causes the capacitance value to change. This change is detected by an electronic circuit. From the measurement of the capacitance change, the differential pressure is calculated and presented on a display in the part that makes up the transmitter of the dp-cell. The dp value is available as a current signal in the standard range of 4-20 mA and as a digital measurement signal in various formats.

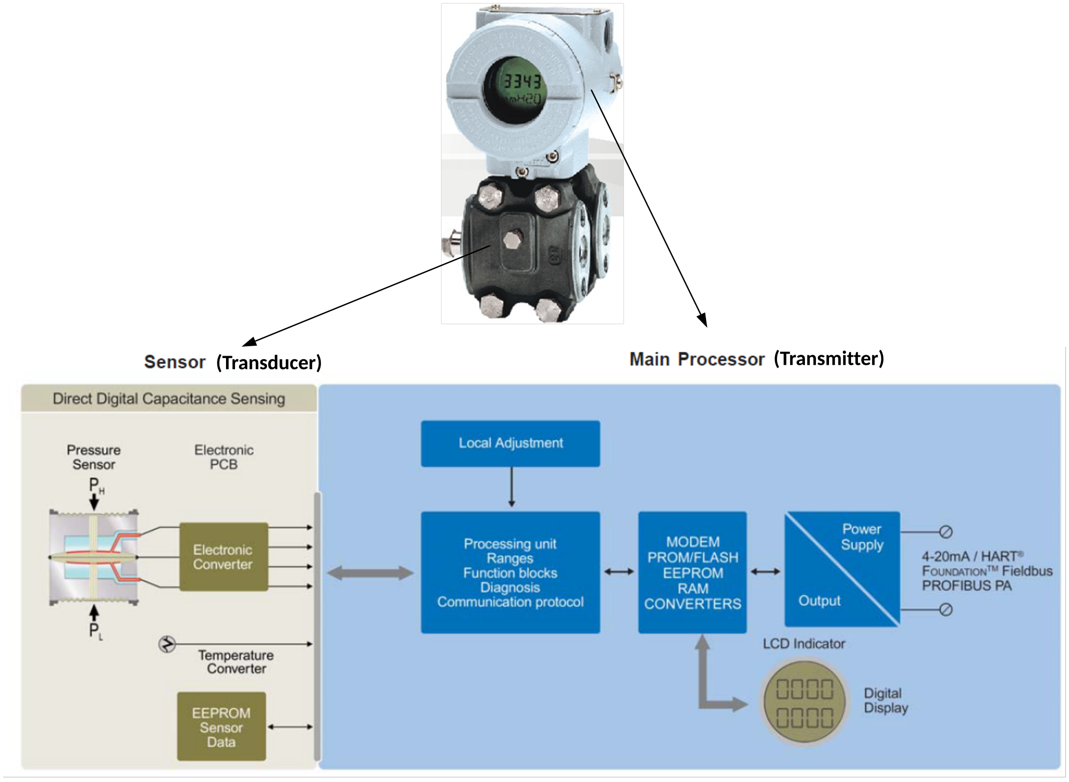


Figure 3.18: Example of construction of a dP cell. (Smar Pressure Transmitter. LD300 Series. Autek AS.)

3.3.4 Level sensors

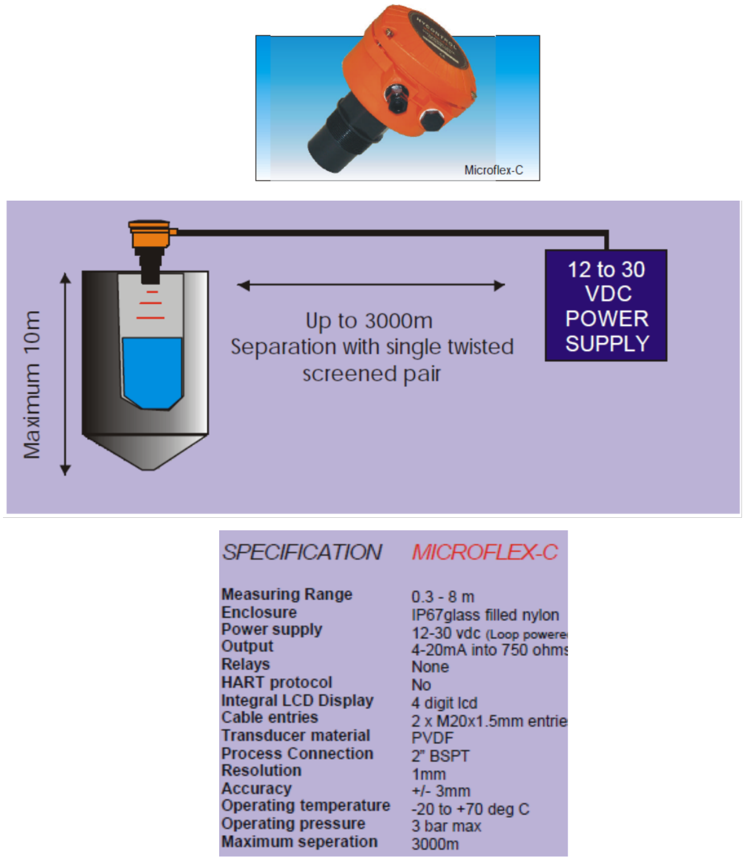
There are many different types of level sensors. Here we will look at two types:

- Ultrasonic level sensor, cf. Section 3.3.4.1.
- Dp cell-based level sensor, cf. Section 3.3.4.2.

3.3.4.1 Ultrasonic level sensor

Figure 3.19 shows an example of an ultrasonic level sensor and an application to measure the level of water in a tank. Also, the specifications of the sensor is shown in the figure.

The operating principle is that ultrasonic pulses are sent towards the surface of the liquid (or other material) of which the level is to be measured. The reflection time T_r [s], i.e. the time from emission to reception of the ultrasonic pulse, is detected. The speed of sound v [m/s] is known (330 m/s in air). Then, the distance L [m] from the sensor to the liquid



SPECIFICATION	MICROFLEX-C
Measuring Range	0.3 - 8 m
Enclosure	IP67glass filled nylon
Power supply	12-30 vdc (Loop powere
Output	4-20mA into 750 ohms
Relays	None
HART protocol	No
Integral LCD Display	4 digit lcd
Cable entries	2 x M20x1.5mm entrie
Transducer material	PVDF
Process Connection	2" BSPT
Resolution	1mm
Accuracy	+/- 3mm
Operating temperature	-20 to +70 deg C
Operating pressure	3 bar max
Maximum seperation	3000m

Figure 3.19: Ultrasonic level sensor. (Microflex, Autek AS.)

surface can be calculated:

$$L = \frac{vT_r}{2} \tag{3.8}$$

Ultrasonic level sensors are relatively easy to install in the process. They can, however, cause problems when mounted in too small containers as there may be unfavorable reflections of the pulses.

3.3.4.2 Dp cells as level sensor

Dp cells as pressure sensors were described in Section 3.3.3. Dp cells can be used for level measurement. The dp cell is then used to detect the hydrostatic liquid pressure at the measuring point. Since this pressure is a function of the liquid level above the measuring point, the level can be calculated from the pressure.

Figure 3.20 shows the principle of measuring the liquid level in a tank using a dp cell. The

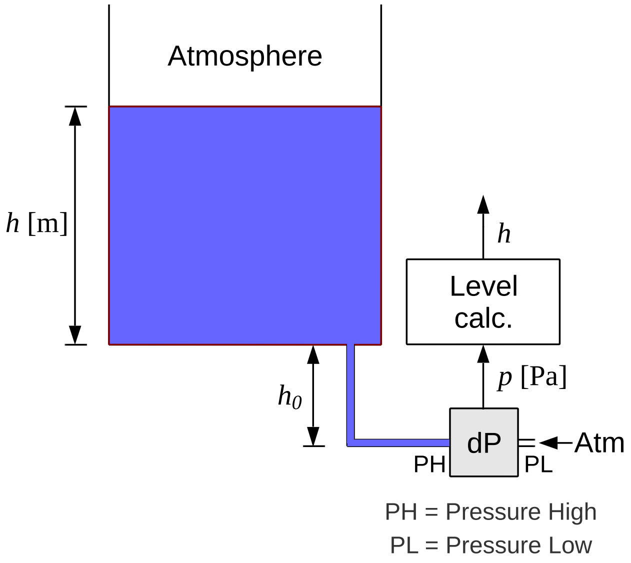


Figure 3.20: The principle of level measurement using a dp cell.

hydrostatic pressure p [Pa] at the measuring point, i.e. at the PH input of the dp-cell, is

$$p = \rho g(h + h_0) \tag{3.9}$$

where $h + h_0$ [m] is the total liquid height, g [m/s^2] is the gravity, ρ [kg/m^3] is the density of the liquid. From 3.9) we get the following formula for liquid level h:

$$h = \frac{p}{\rho g} - h_0 \tag{3.10}$$

3.3.5 Flow rate sensors

We will look at different principles and sensors for measuring material flow rate, also denoted flow velocity. Most of the principles presented can be applied to both liquid flows and gas flows.

3.3.5.1 Flow rate measurement with orifice and dp cell

Figure 3.21 shows how flow rate in a pipeline can be measured with an orifice (an annular constriction) mounted inside a pipline, and a dp cell. The flow rate is larger at the constriction than ahead of the constriction. In general, the pressure in the liquid decreases with increased flow rate. Therefore, the larger flow rate, the larger pressure drop at the constriction. For an orifice with a fixed orifice opening, the following applies:

$$F = k\sqrt{\Delta p} \tag{3.11}$$

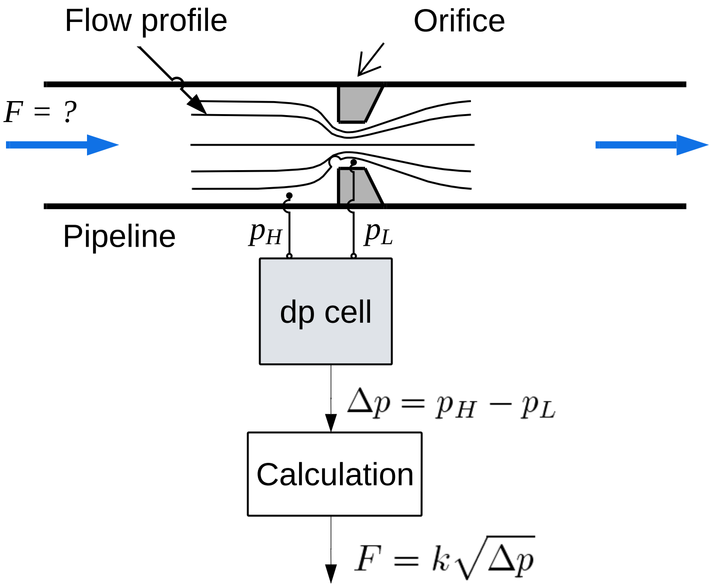


Figure 3.21: Flow rate measurement with an orifice and dp cell.

where F is the flow rate in an appropriate unit, e.g. m^3/s, and Δp is the pressure drop. k is a constant. The pressure drop can be measured with a dp cell, cf. Section 3.3.3.

3.3.5.2 Ultrasonic flow rate measurement

Ultrasonic flow rate measurement can be used for liquids and gases. An advantage, comparing with other methods, is that the sensor can be mounted "clamp-on", so an intervention on the pipeline is not necessary.

Figure 3.22 shows an industrial ultrasonic flow rate sensor and the basic structure of such a sensor. The sensor consists of two sender-receiver pairs, S_1-R_1 and S_2-R_2. The senders send ultrasonic pulses regularly – for example several hundred times a second – and at the same point of time. The receivers detect the transport time of the sound pulse it receives. In Figure 3.22, t_1 [s] is the transport time of a pulse sent downstream, i.e. from S_1 to R_2, while t_2 is the transport time of the pulse sent upstream, i.e. from S_2 to R_1. t_2 is less than t_1. The flow velocity v [m/s] is proportional to the difference between these two transport times, i.e.

$$v = k\,(t_2 - t_1) \tag{3.12}$$

From v, the volumetric flow rate Q [m^3/s] can be calculated with

$$Q = v \cdot A = k\,(t_2 - t_1) \cdot A \tag{3.13}$$

where A [m^2] is the internal cross-sectional area of the pipe. (Of course, units other than m^3/h and m^2 may be used here.)

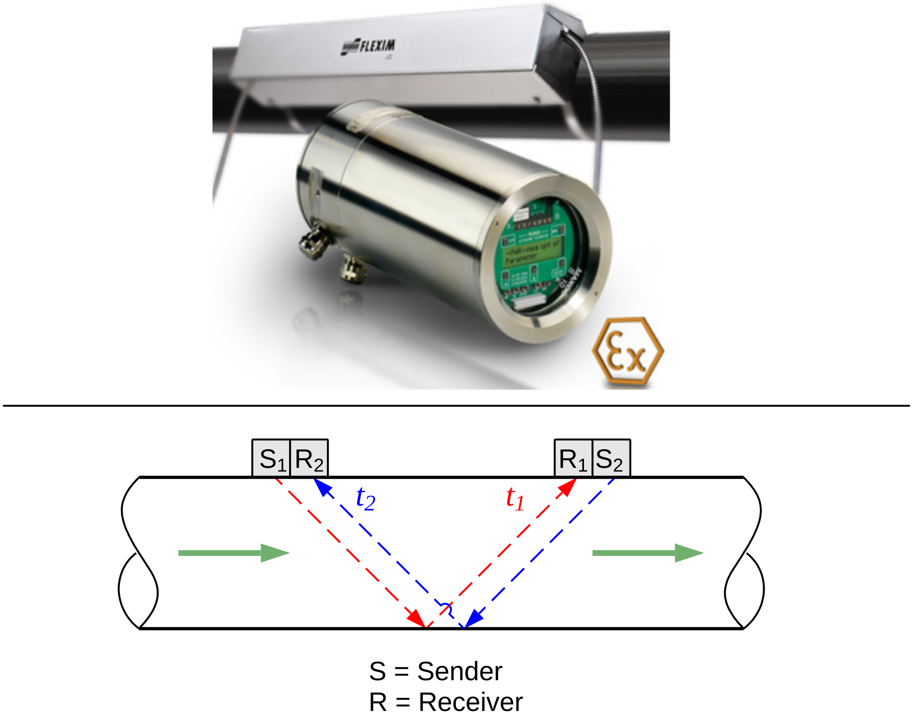


Figure 3.22: Top: An industrial ultrasonic flow rate sensor. (The symbol Ex means that this sensor is suitable for potentially explosive atmospheres.) (Fluxus, Flexim, Flow-Teknikk AS.) Bottom: The principle structure of ultrasonic sensors.

In Figure 3.22 there is one reflection. More than one reflection may be used; the ultrasonic pulse is then reflected in the pipe walls several times on its way from sender to receiver. This may increase accuracy, but at the same time the receiver will receive weaker signals. Ultrasonic sensors of the brand mentioned in Figure 3.22 are able to find the optimal number of reflections, i.e. the optimal sound path.

As an example of characteristic data of ultrasonic flow rate sensors, below are data of the ultrasonic flow rate sensor mentioned in Figure 3.22:

- *Update period of measurement signal*: 1 s.
- *Measurement range*: 0.01 ... 25 m/s.
- *Accuracy*: Approx. 1% of read value ±0.01 m/s.
- *Resolution*: 0.025 cm/s.

3.3.5.3 Coriolis flow rate sensor

Coriolis mass flow sensors are used to measure both mass flow rate and volume flow rate of liquids and gases. Figure 3.23 shows an industrial Coriolis mass flow sensor.

Figure 3.23: Industrial Coriolis flow rate sensor. (Heinrich's TMU. Flow-Teknikk AS.)

The principle of Coriolis flow rate sensors is as follows. See Figure 3.24. Inside the sensor, the medium – liquid or gas – passes through two pipe loops. With two loops, the forces arising due to the vibrations in the sensor, cancels each other, thereby prevening the sensor from vibrating. The motion actuators ensures that the loops are in periodic oscillations at the particular frequency being the natural or resonance frequency of the loops including the medium. The actuator ensures that the oscillations are in opposite phase, i.e. one loop is moved up when the other is moved down.

The two loops have a certain flexibility, i.e. they are twistable, as indicated in Figure 3.24. When the medium flows through the moving loops, a Coriolis force arises on the flowing medium. This force is due to the fluid being in a reference system that rotates around the axis shown in Figure 3.24.

For each of the two loops, the Coriolis force causes the loop to twist, and the direction of motion depends on whether the liquid moves towards the axis or away from the axis on its way through the loop. The twist on each side of the loop is detected by respective motion sensors, see Figure 3.24.

For Coriolis flow rate sensors the following applies:

- The mass flow rate F_m [kg/s] is a function of the phase shift ϕ [degrees] between the oscillations. This phase shift is equal to 360 degrees times the time shift dt [s] divided by the period P [s].
- The density of the liquid ρ [kg/m^3] is a function of the period P of the oscillations.

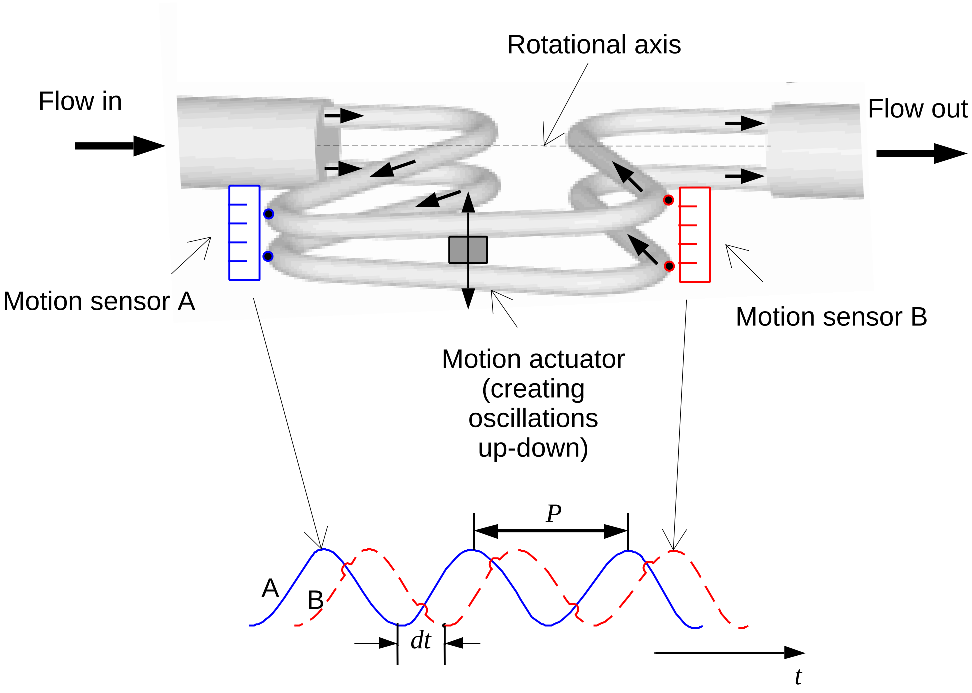


Figure 3.24: The principle of Coriolis mass flow sensors. (The figure showing the sensor itself can be found at http://en.wikipedia.org/wiki/Mass_flow_meter.

From the detected of mass flow rate and density, the volume flow rate F_v [m^3] can be calculated with

$$F_v = \frac{F_m}{\rho} \tag{3.14}$$

It is common for Coriolis flow rate sensors to output also a temperature measurement.

Coriolis flow rate sensors can cover large areas and are relatively accurate, but they are also relatively expensive. For example, the sensor shown in Figure 3.23 covers 0-2.2 tons/h with an accuracy of 0.1% of measurement range (span).

3.3.5.4 Magnetic flow rate sensor

Magnetic flow rate sensors are based on Faraday's law which expresses that an electromotive force (emf - electromotoric force), i.e. an electric potential, or voltage, is induced in a conductor that moves in a magnetic field. The voltage is proportional to the speed of the conductor in the field, i.e. the greater the speed, the greater the voltage. In general, liquids are electrical conductors. In magnetic flow rate meters, the fluid velocity , i.e. the flow rate, is measured with the voltage induced as the fluid moves in a magnetic field.

Figure 3.25 shows the basic structure of a magnetic current sensor and an image of an industrial sensor. A magnetic field is generated inside the sensor. Usually, magnets are used that are mounted outside the pipeline, and in that case the magnets must work through a piece of pipe that is not conductive; Otherwise the magnetic field would not pass through the liquid, but instead follow the pipe wall (like a Faraday cage). The sensor part of the pipeline usually consists of a plastic material. In the figure you can see a shows a fluid element that moves in the magnetic field. Remember that a fluid is an electric conductor. Due to the motion, a voltage is induced in the liquid element. This voltage is detected by electrodes. (An electrode is a piece of material that conducts electric current to or from a liquid, a gas, a body part, etc., or to another electrode which is in contact with the liquid.) Since the induced voltage is proportional to the velocity of the fluid element (conductor), the voltage will indicate the fluid velocity.

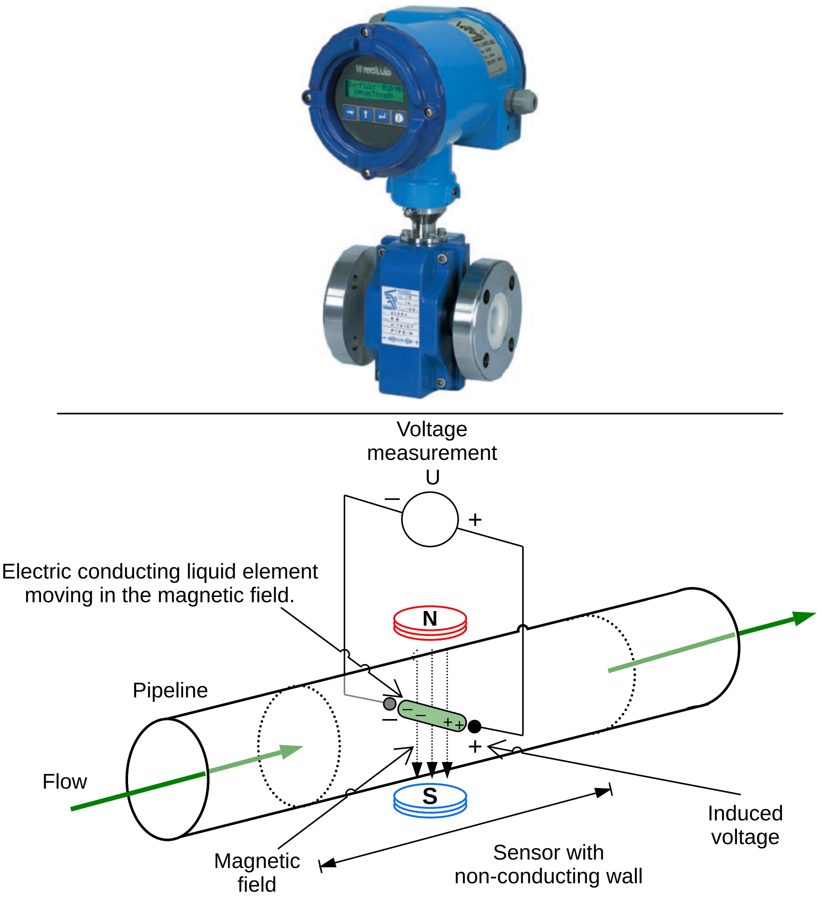

Voltage measurement
U
−
+
Electric conducting liquid element moving in the magnetic field.
N
Pipeline
Flow
−
++
+
S
Magnetic field
Sensor with non-conducting wall
Induced voltage

Figure 3.25: Magnetic flow rate sensor for liquid flow (Flomid FX, Techfluid. Flow-Teknikk AS.)

To prevent electrostatic conditions in the liquid itself from causing a systematic error in the measurement, the direction of the magnetic field is changed with a fixed frequency. The difference in induced voltage between periods with opposite field directions then forms the basis for measuring the fluid velocity.

Magnetic flow rate sensor can measure liquids with particles.

The measured the volume flow rate F [m^3/s] can be calculated with

$$F = Av \tag{3.15}$$

where v [m/s] is the fluid velocity and A [m^2] is the internal cross-sectional area of the pipeline. A must be precisely known so that the volume flow can be measured accurately.

Here are some data given in the data sheet for the sensor depicted in Figure 3.25:

- *Accuracy*: ±0.5 % of displayed flow rate value.
- *Recommended flow velocity*: Approx. 0.5-4 m/s. If the volume flow rate range of what is to be read is known, you can then use ([eq_qdv]) to calculate the appropriate pipe diameter in the sensor. (If the calculated sensor diameter is smaller than the diameter of the pipeline, narrowings are fitted on each side of the sensor. Guidelines are given for the narrowings' slope and distance from the sensor.)

3.3.5.5 Thermal flow rate sensor

Figure 3.26 shows an example of a thermal mass flow rate sensor. The shown sensor is for gases, e.g. biogas, but there are thermal flow rate sensors also for liquids. The device shown in Figure 3.26 also has a built-in control valve (see right part of the figure), as well as a controller, and therefore implements a control system for mass flow rate.

Figure 3.27 shows the principle of thermal mass flow rate measurement. A part of the flow of the medium (gas or liquid) is heated using heating coils in two subsequent heating sections. The temperature in each of the two sections, i.e. T_1 and T_2, is detected, see "sensor bridge" in the figure. The temperature difference between the two sections is proportional to the mass flow rate F [kg/s], see the diagram in the figure:

$$\Delta T = T_2 - T_1 = kF \tag{3.16}$$

So, assuming ΔT and k are known, F can be calculated with

$$F = \frac{\Delta T}{k} \tag{3.17}$$

3.3.5.6 Vortex flow rate sensor

When you see the flag flapping in the wind, you see the vortex principle in free dressage. The greater the wind speed, the higher the frequency in the flutter ring. The frequency basically gives an expression of the wind speed.

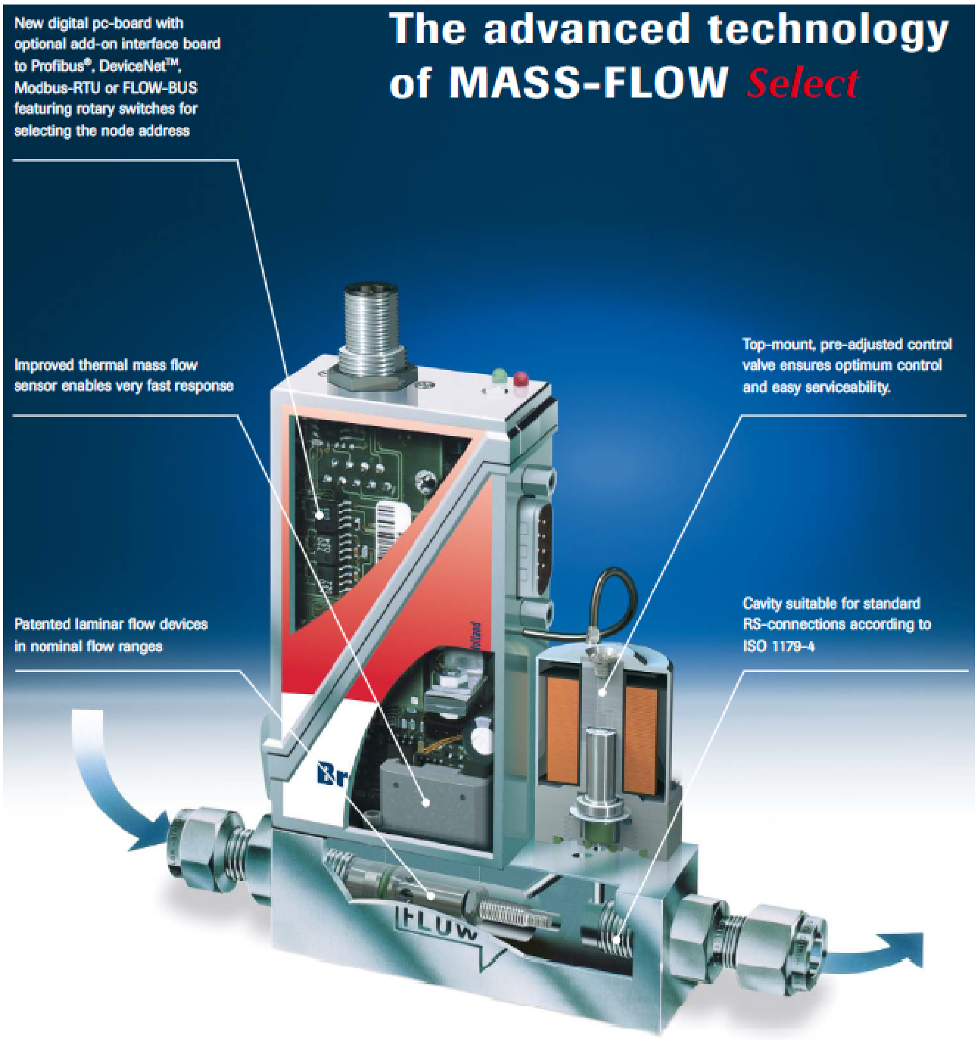

The advanced technology of MASS-FLOW Select
New digital pc-board with optional add-on interface board to Profibus®, DeviceNet™, Modbus-RTU or FLOW-BUS featuring rotary switches for selecting the node address
Improved thermal mass flow sensor enables very fast response
Patented laminar flow devices in nominal flow ranges
Top-mount, pre-adjusted control valve ensures optimum control and easy serviceability.
Cavity suitable for standard RS-connections according to ISO 1179-4
FLOW

Figure 3.26: Example of thermal mass flow sensor for gas. The unit also has a built-in control valve and a regulator and therefore implements a mass flow control system. (Bronkhorst. Flow-Teknikk AS.)

A vortex sensor is based on the fact that a restriction in a pipeline creates regular vortices, and the frequency of these vortices is proportional to the flow rate. By detecting the frequency, the flow rate can be measured. This principle applies to both liquid flow and gas flow.

Figure 3.28 illustrates the principle and shows an industrial vortex sensor. The frequency in the pressure vortices is usually detected with e.g. piezoelectric crystals, which are crystals where charges are produced when they are exposed to pressure.

The measurement accuracy for vortex sensors is around 1 % of the maximum measurement value.

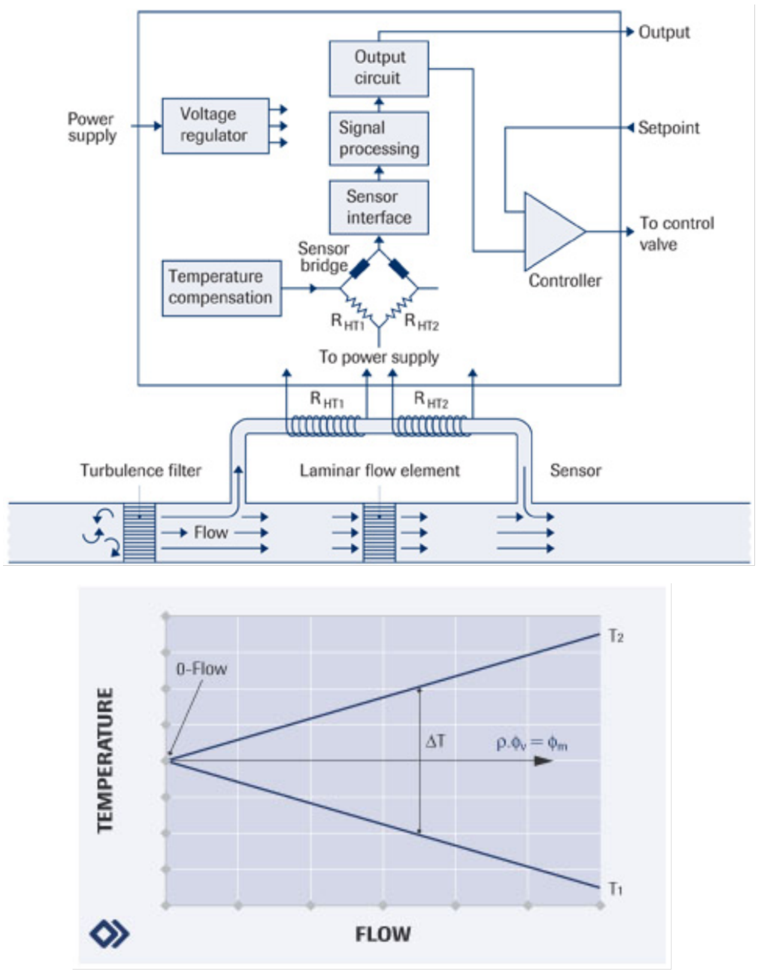


Figure 3.27: The principle of thermal mass flow rate measurement. (Bronkhorst. Flow-Teknikk AS.)

3.3.6 Sensors for gas concentration

Figure 3.29 shows an example where you need to measure gas concentration, namely a biogas reactor which converts various types of biological material, e.g. food waste, slaughter waste, manure, etc., into biogas, which mainly consists of methane and carbondioxide. The energy content of the biogas for combustion is the methane. Therefore, biogas reactors are usually equipped with methane gas flow rate sensors to indirectly measure the power production by the reactor. The methane gas flow rate can be calculated as the product of the total biogas flow rate using e.g. a thermal gas fom rate sensor, cf. Section 3.3.5.5, and the methane concentration.

The methane concentration can be detected with e.g. a spectroscopic concentration sensor where the concentration in percent is detected by measuring the intensity of light with a

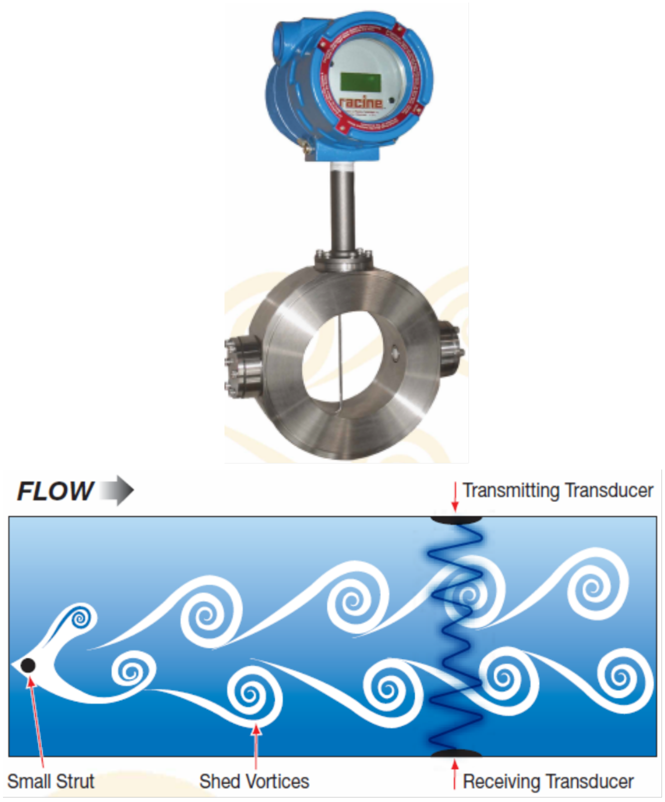


Figure 3.28: The principle of vortex sensor for measuring gas or liquid flow and an industrial vortex sensor. (Racine. Flow-Teknikk AS.)

specific wavelength – typically infrared (IR) light – is absorbed by the gas, see Figure 3.30.

The measuring accuracy of one example of a spectroscopic sensor is approximately ±2 % of the upper limit of the measurement range.

3.3.7 Position sensors

There are several types of position sensors. Here, I will present only the encoder.

3.3.7.1 Encoder

Encoders are popular for measuring of angular position. Figure 3.31 shows a principle sketch of an encoder. The encoder is attached to the object whose angular position is to be measured. The encoder has a number of evenly spaced slits that can let the light from the light source through. As the encoder disc rotates, light pulses are generated for each of channels A and B. These channels are 90 degrees (a quarter period) offset from each other

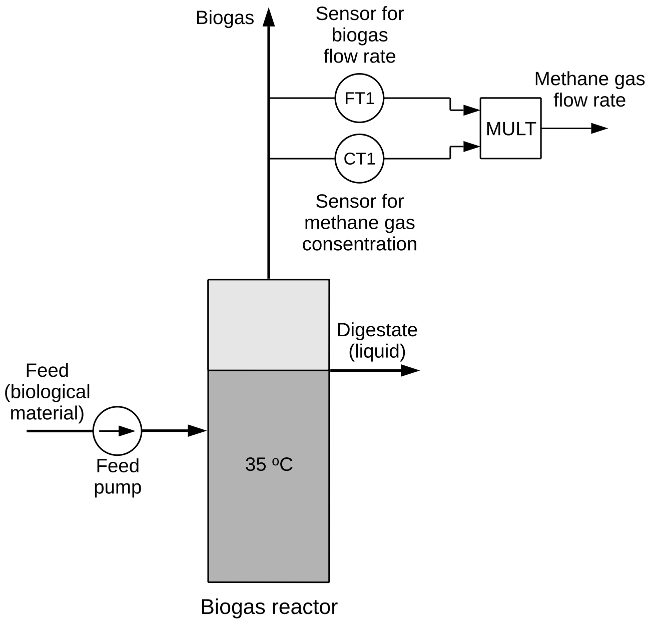

Biogas
Sensor for biogas flow rate
FT1
CT1
MULT
Methane gas flow rate
Sensor for methane gas consentration
Digestate (liquid)
Feed (biological material)
Feed pump
35 °C
Biogas reactor

Figure 3.29: Methane concentration measurement used to calculate the methane gas flow rate out of a biogas reactor.

to enable detecting an increasing as well as a decreasing position. The counter is a logic circuit that counts the pulses for channels A and B. If the direction of rotation is positive, the counter counts up. If it is negative, it counts down.

A typical number of pulses, n, also denoted bits, for each of the channels for one full rotation is 1024. This gives a resolution, angle change per pulse, of

$$r = \frac{360 \text{ degrees}}{n} \text{ [degrees/pulse]} \tag{3.18}$$

For example, if $n = 1024$,the resolution is $r = 360$ degrees/1024 pulses = 0.3516 degrees per pulse.

In addition to channels A and B, encoders often also have a Z channel, which gives one pulse for each time the code disc has rotated once. The Z channel can be used to define a reference position, so that for example position zero corresponds to a lifting arm standing in a vertical position.

Encoders can be used together with suitable data acquisition equipment from various suppliers. Figure 3.32 shows an example with equipment from National Instruments. The I/O module shown to the left can be connected to a encoder. The programming tool LabVIEW has functions for recording the position measurement signals from the encoder which, via the NI-9401 module, are read by LabVIEW running on a PC.

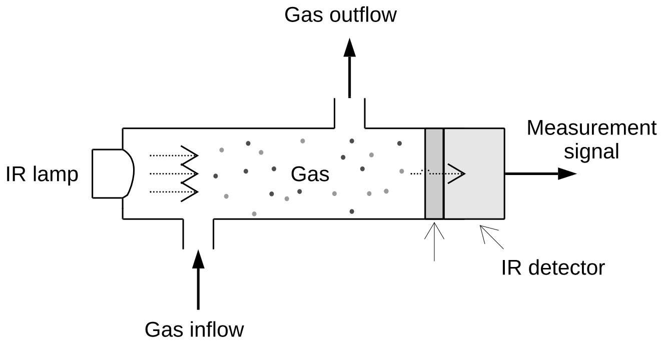


Figure 3.30: Spectroscopic measurement of gas concentration using IR lamp and detector. (IR = Infrared.)

3.3.8 Speed sensors

There are several types of speed sensors. Below I will present the encoder and the tachogenerator as sensors for rotational speed.

3.3.8.1 Encoder

Figure 3.31 shows an encoder. Assuming only one-directional rotation, the absolute value of the speed v [rpm] can be calculated from the detected frequency f [# pulses/s] of either pulse train A or B:

$$v = \frac{f}{n} \tag{3.19}$$

where n is the number of pulses or bits of the encoder for one full rotation, i.e. 1024.

If the rotation is bi-directional, the speed can be calculated as the time derivative of the position measurement:

$$v(t) = s'(t) \tag{3.20}$$

which in discrete-time form can be approximated with

$$v(t_k) \approx \frac{s\left(t_k\right) - s\left(t_{k-1}\right)}{dt} \tag{3.21}$$

where $s(t_k)$ is the position value at the current point of time and $s(t_{k-1})$ is the position value at the previous point of time. dt [s] is the sampling time of the position measurements.

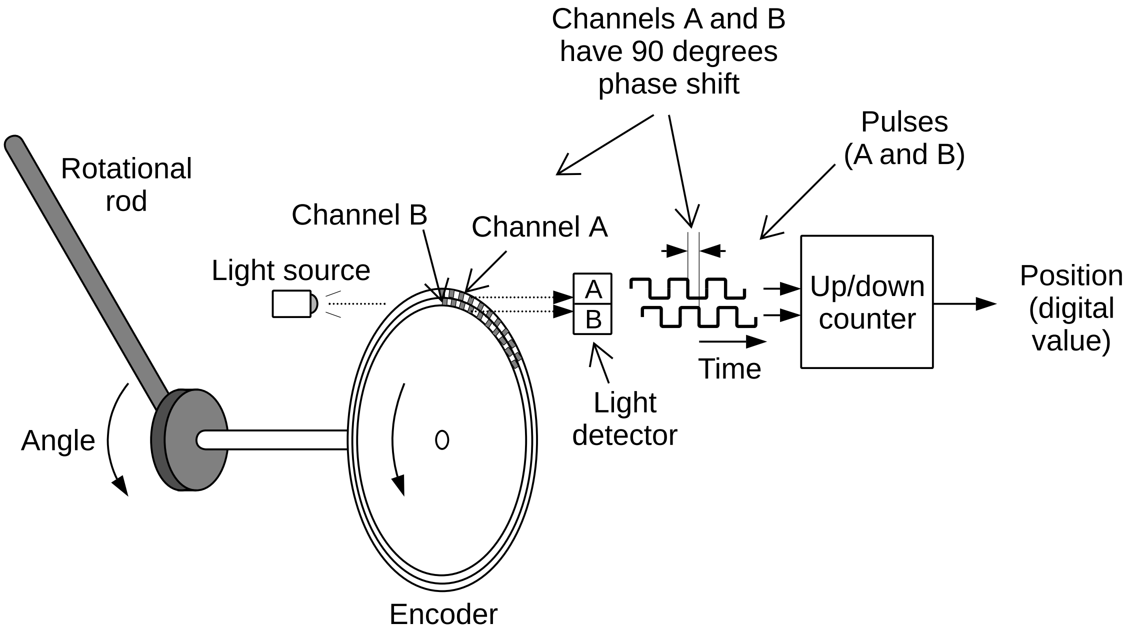


Figure 3.31: Encoder for position measurement.

3.3.8.2 Tachogenerator

A tachogenerator[6] is an electrical machine generating a DC voltage that is proportional to the rotational speed of the machine. When a tachogenerator is connected (fixed) to the shaft of some rotating device, e.g. a motor, the tachogenerator acts as a rotational speed sensor for that device. The generated voltage u_t [V] is approximately proportional to the rotational speed v [rpm = revolutions per minute]:

$$u_t = Kv \tag{3.22}$$

where K [V/rpm] is the tachogenerator constant. Figure 3.33 shows an application of a tachogenerator as speed sensor of a DC motor.

3.4 Signal conditioning of measurement signals

3.4.1 Analog measurement signals

Sensors output the measurement signal as an analog signal or a digital signal, and some sensors output both analog and digital signals. Below is a brief description of analog

[6] Tacho means speed in Greek.

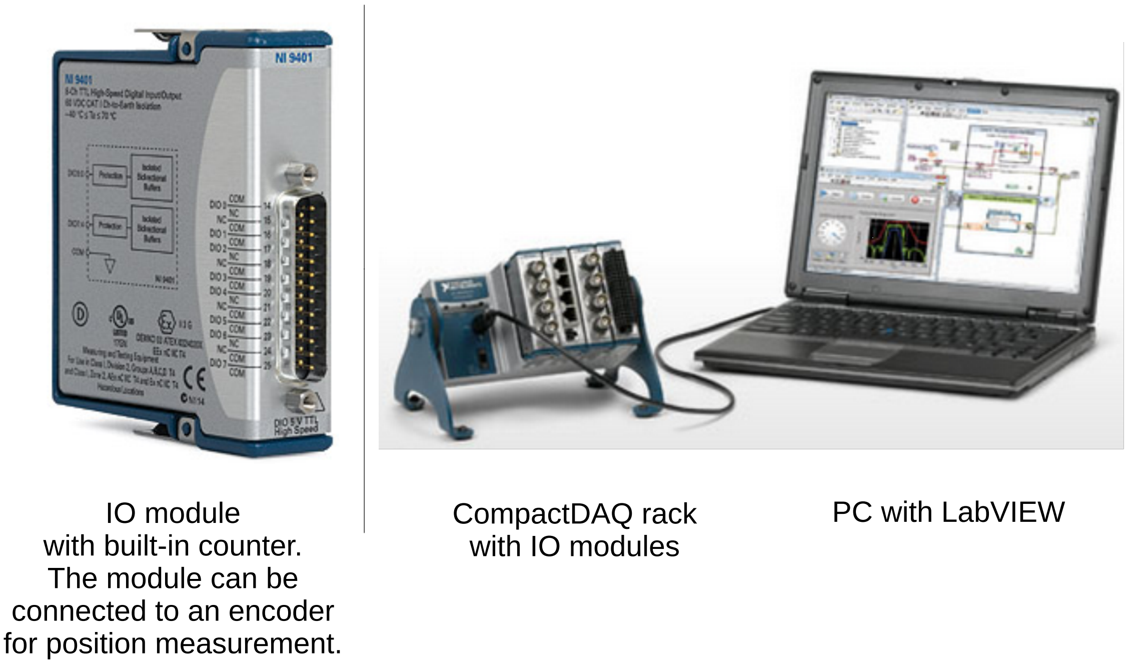


Figure 3.32: Example of equipment that can measure position measurement signals from an encoder. (National Instruments)

signals. There are many forms or standards for digital signals (RS-232/serial, RS-485/Modbus, HART, etc.), but these are not described in this book.

Current signals

A common process measurement signal form is electric current in the range of 4-20 mA (milliamperes), but 0-20 mA is also used.

Voltage signals

Although not as common as current signals, measurement signals can be in the form of electrical voltages, typically in the range 0-5 V or 1-5 V.

3.4.2 Current loop

In process instrumentation, current loops are common. Figure 3.34 shows a current loop.

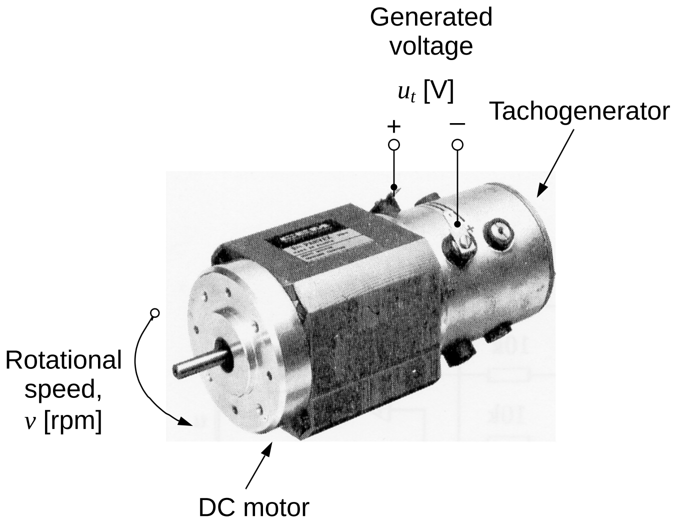


Figure 3.33: A tachogenerator connected to the shaft of a DC motor for speed measurement. (Electro-Craft S-19-3AT.)

Current loops comprise the following:

- A power supply delivering typically 24 VDC.
- A sensor supplied with 24 VDC from the power supply.
- The sensor generates a measurement signal in the form of an electrical current in the typical range of 4-20 mA, representing the value of the pertinent process variable, e.g. temperature, level, pressure, etc.
- A device registering a measurement signal in the form of an electrical voltage with a specified range, e.g. 1-5 V. Examples of devices are data acquisition devices, and controllers. Actually, there can be several devices in series in the current loop, although only one device is shown in Figure 3.34.
- A resistor for converting the current signal into a proportional voltage signal. The resistance needed can be found with Ohm's Law:

  $$R = \frac{U}{I} \ [\Omega] \tag{3.23}$$

  As an example, assume the signal ranges given above, the resistance is 5 V/20 mA = 250 Ω.

3.4.3 Analog-digital (AD) conversion

Figure 3.35 illustrates the operation of analog-digital converters (ADC). ADCs are included in automation hardward, but they are also available as separate electronic components.

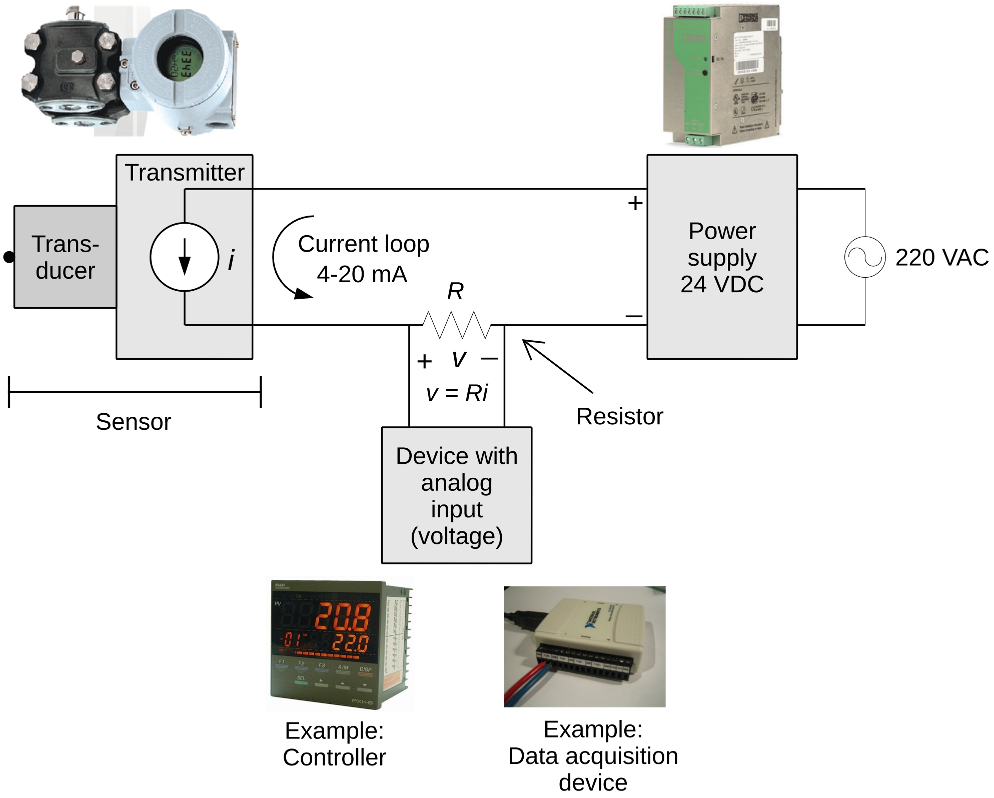


Figure 3.34: Current loop.

In process control applications, the sampling time t_s of an ADC is typically less than 1 sec, e.g. 0.1 s.

The AD converter will use a certain number of bits, n pieces, each with value either 0 or 1, to represent the analog signal. $n = 12$ is a typical value for AD converters. What is the meaning of these bits? They are coefficients for the digital value expressed in the total system. As an example, let's assume that $n = 12$. Assume that the AD converter will be used to convert analog values y_a which are in the range $[y_{a_{\min}}, y_{a_{\text{maks}}}]$, e.g. [4, 20] mA. It can be shown that the digital value can be expressed as a function of these bits as:

$$y_d = \frac{b_{11}2^{11} + b_{10}2^{10} + ... + b_1 2^1 + b_0 2^0}{2^{12} - 1} (y_{a_{\text{maks}}} - y_{a_{\min}}) + y_{a_{\min}} \tag{3.24}$$

where the numerator is actually the representation of the number in the total system. Since it is the values of the bits that vary as y_a varies, the digital signal can be represented in a more compact way as a set of bits:

$$y_d \sim b_{11}b_{10}...b_1b_0{}_2 \tag{3.25}$$

where subindex 2 means that this set of bits is actually a binary number in the binary number system.

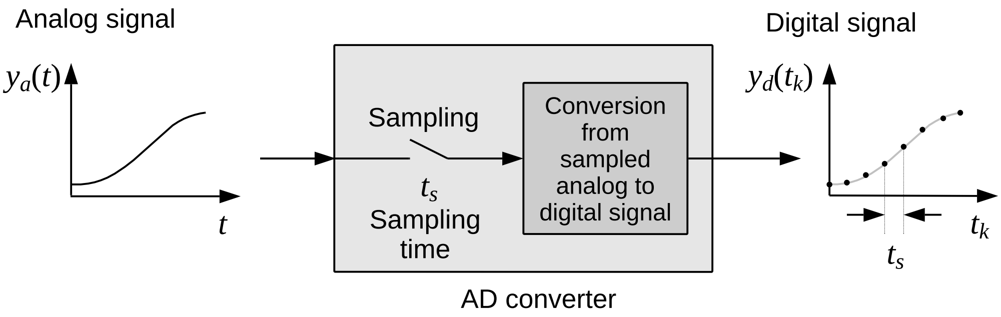


Figure 3.35: Analog-digital converter (ADC).

Bit $b_{n-1} = b_{11}$ is called MSB (most significant bit), while bit b_0 is called LSB (least significant bit).

Example 3.3 *Digital representation of analog value*

Assume that the bit set is

$$y_d \sim 010...00_2 \tag{3.26}$$

Inserting into (3.24) gir

$$y_d = \frac{0 \cdot 2^{11} + 1 \cdot 2^{10} + 0 \cdot 2^9 ... + 0 \cdot 2^1 + 0 \cdot 2^0}{2^{12} - 1} (y_{a_{\text{maks}}} - y_{a_{\text{min}}}) + y_{a_{\text{min}}} \tag{3.27}$$

$$= \frac{2^{10}}{2^{12} - 1} (20 - 4) + 4 = 8.001 \tag{3.28}$$

[End of Example 3.3]

ADC resolution

AD converters are of course very useful since they make analog signals available as digital numbers for computers. But not all information is retained in the digital numers. The finite number of bits limits how accurately the digital signal can represent the analog signal. This accuracy is in terms of the *resolution* of the ADC. From the specified resolution you select an appropriate ADC.

From (3.24) we can calculate the smallest change in y_a that y_d can detect. Assume that only the least significant bit in the digital signal, i.e. bit b_0 (LSB), changes value from 0 to 1. This change is $\Delta b_0 = 1$. From (3.24), where n is the number of bits, we see that the corresponding change in y_d becomes

$$\Delta y_d = \frac{\Delta b_0 = 1}{2^n - 1} (y_{a_{\max}} - y_{a_{\min}}) = \frac{y_{a_{\max}} - y_{a_{\min}}}{2^n - 1} = R \tag{3.29}$$

So, the resolution is

$$R = \frac{y_{a_{\max}} - y_{a_{\min}}}{2^n - 1} \tag{3.30}$$

Figure 3.36 illustrates the resolution R.

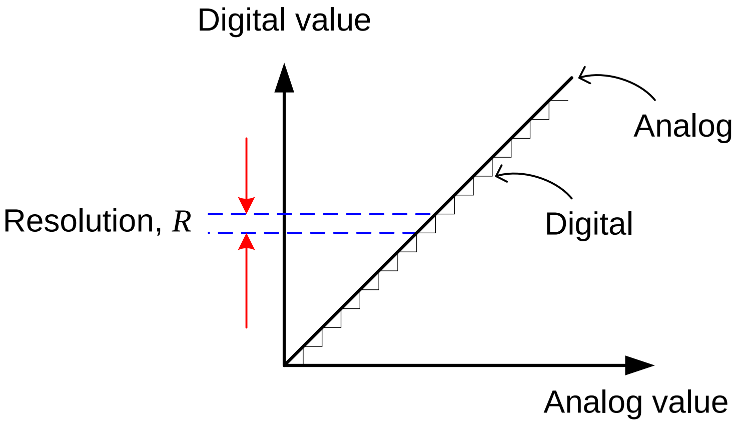


Figure 3.36: The resolution R.

The relative resolution calculated as a fraction of the analog signal range is

$$R_{\text{rel}} = \frac{1}{2^n - 1} \tag{3.31}$$

Example 3.4 *Resolution*

Given a 12-bit AD converter which covers the analog signal range from $y_{a_{\min}} = 4$ mA to $y_{a_{\max}} = 20$ mA. The resolution is then

$$R = \frac{y_{a_{\max}} - y_{a_{\min}}}{2^n - 1} = \frac{20 - 4}{2^{12} - 1} = 0.0039 \text{ mA} \tag{3.32}$$

The relative resolution is

$$R_0 = \frac{1}{2^n - 1} = \frac{1}{2^{12} - 1} = \frac{1}{4095} = 0.24 \cdot 10^{-3} = 0.024 \ \% \tag{3.33}$$

In the old times 8-bit AD converters were common. They had a relative resolution of

$$R_0 = \frac{1}{2^8 - 1} = \frac{1}{255} = 0.39 \ \% \tag{3.34}$$

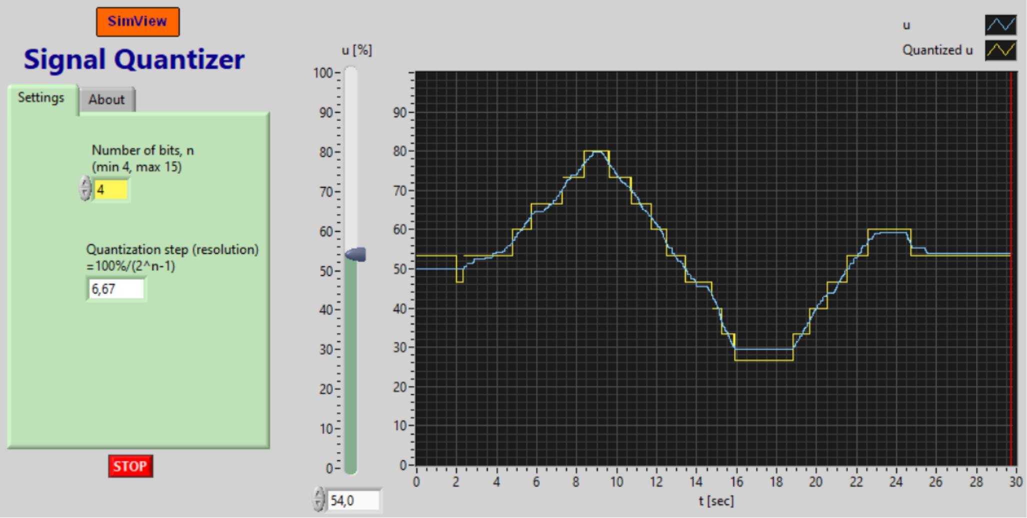
[End of Example 3.4]

Example 3.5 *Simulation of analog-digital conversion*

The simulations in this example are made with the following SimView simulator:

http://techteach.no/simview/signal_quantizer

Figure 3.37 shows a signal, u, in range 0-100 % and the corresponding quantized signal, with just $n = 4$ bits ADC conversion.



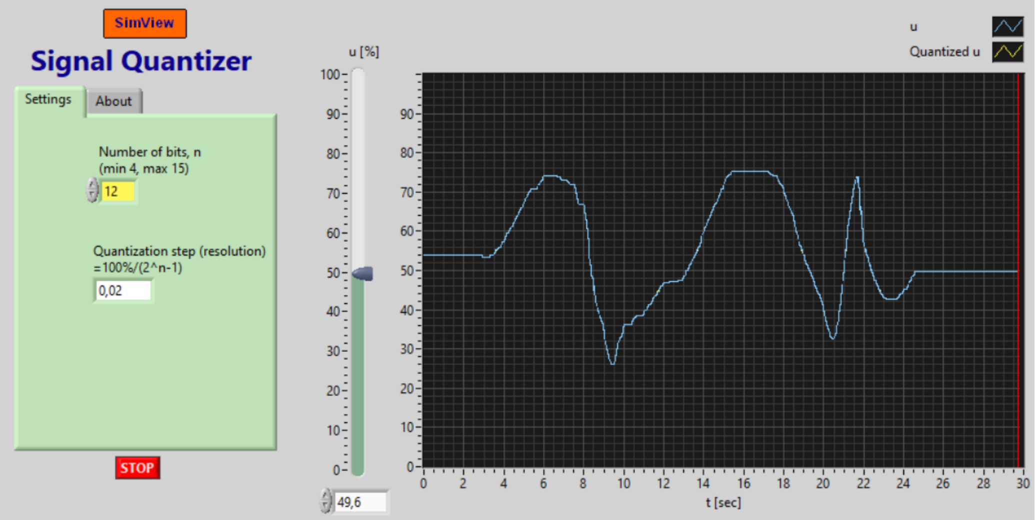
Number of bits, n (min 4, max 15): 4

Quantization step (resolution) =100%/(2^n-1): 6,67

54,0

Figure 3.37: $n = 4$ bits ADC conversion.

Figure 3.38 shows the signals with $n = 12$ bits ADC conversion. The resolution is clearly better with more bits.



Number of bits, n (min 4, max 15): 12

Quantization step (resolution) =100%/(2^n-1): 0,02

49,6

Figure 3.38: $n = 12$ bits ADC conversion.

[End of Example 3.5]

3.4.4 Scaling of measurement signals

We have already, in Section , seen how to scale a measurement signal with a linear scaling function. I will not repeat it here. Instead, I will present some new terms which are relevant to measurement scaling. Figure 3.39 shows an assumed linear measurement characteristic which expresses the process value p as a function of the measurement signal m:

$$p = a \cdot m + b \tag{3.35}$$

with slope

$$a = \frac{p_2 - p_1}{m_2 - m_1} \tag{3.36}$$

and intercept or constant (calculated from point 1; alternatively point 2 could have been used) is

$$b = p_1 - a \cdot m_1 \tag{3.37}$$

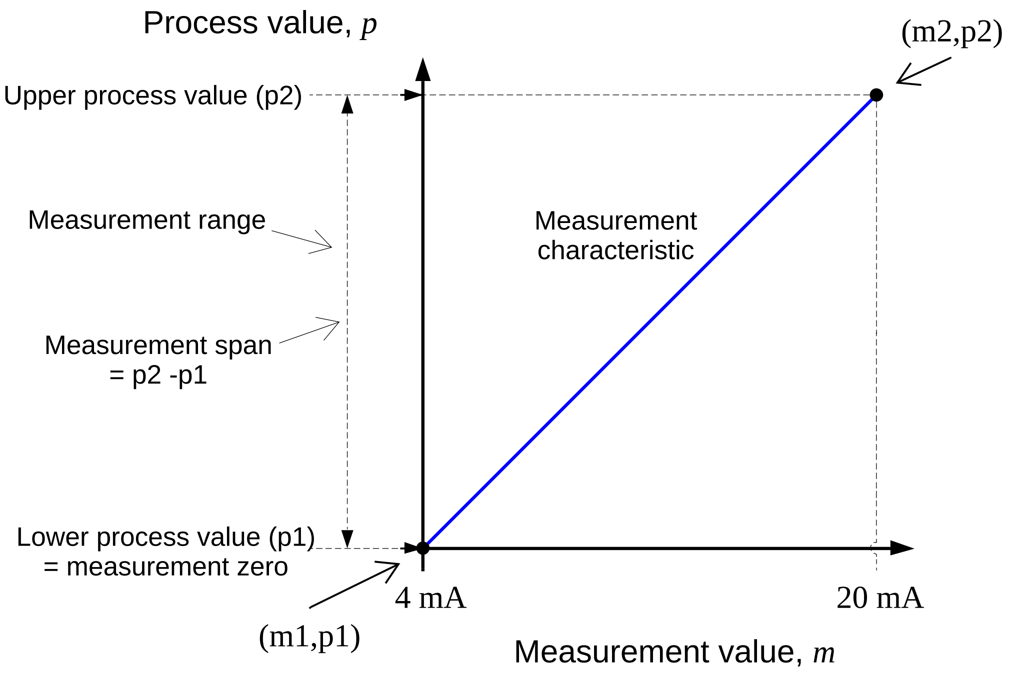


Figure 3.39: Measurement characteristic.

In Figure 3.39, several measurement parameters are defined: Span, zero, and range. These parameters may be adjustable on the transmitter of the sensor.

3.4.5 Accuracy of measurement signals

Measurement accuracy is the largest difference – as an absolute value – between the measurement and the real process value, see Figure 3.40.

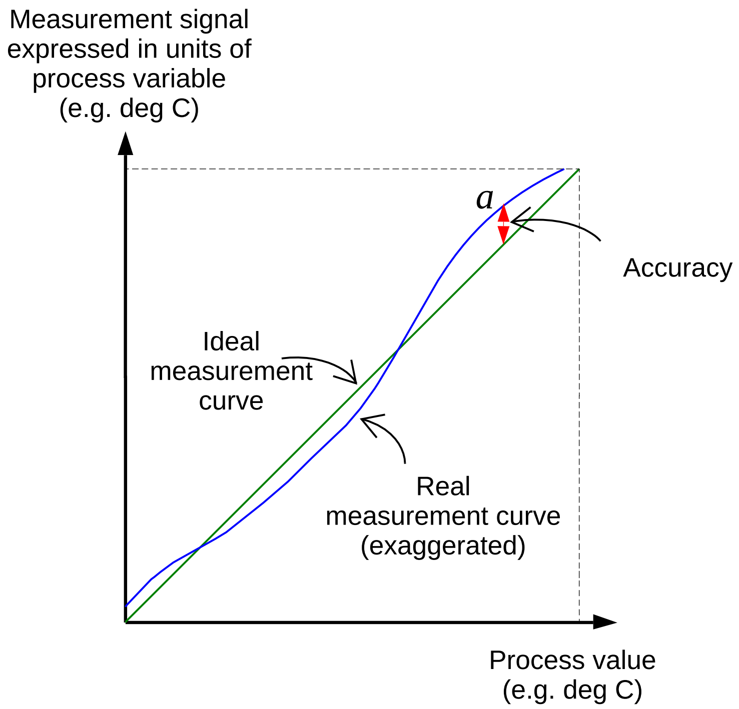

Measurement signal expressed in units of process variable (e.g. deg C)
a
Accuracy
Ideal measurement curve
Real measurement curve (exaggerated)
Process value (e.g. deg C)

Figure 3.40: Measurement accuracy.

The accuracy of a sensor can be found in the data sheet. The accuracy is usually expressed either as a fraction of the maximum measurement value (Upper Range Limit – URL), or of the full scale (FS) value to which the sensor has been adjusted to measure.

Example 3.6 *Measurement accuracy*

Given a gas flow rate sensor with URL = 600 L/d (liters per day). The sensor has an accuracy of 1.5 % of URL, i.e. $600 \cdot 1.5/100 = 9$ L/d. So you must assume that any measurement value has an error of 9 L/d.

[End of Example 3.6]

A couple of other types of accuracy terms are:

- *Temperature drift*. For example, the zero point (the lowest point) of the measurement range may change with temperature with an amount equal to the temperature drift.

- *Age drift.* For example, the zero point of the measurement range may change during 2 years with an amount equal to the age drift.

3.4.6 Measurement filters

A measurement lowpass filter is used to attenuate noise in the process measurement signal. In control systems, the following two lowpass filter types are common:

- *Moving average (MA) filter*, presented briefly already in Section 1.3.3, and presented in more detail in Section 3.4.6.1.
- *Time constant filter*, which is presented in Section 3.4.6.2.

These two filters can be realized with the same discrete time filter algorithm, so it does not matter much which one you use. Still, I like to present both filters since both are popular.

3.4.6.1 Moving averaging (MA) measurement filter (revisited)

The MA filter algorithm

The MA filter algorithm was already presented in Section 1.3.3.7, but is repeated here for convenience:

$$y_{\mathrm{mf},k} = \frac{1}{N_f}\sum_{i=0}^{N_f-1} y_{\mathrm{m},k-i} \tag{3.38}$$

where N_f is the filter length. The corresponding moving average time window of the filter is t_f– the filter time constant. Figure 3.41 illustrates the filter behaviour. (The figure is the same as Figure 1.11, and repeated here for convenience.) The relation between t_f and N_f is

$$N_f = \frac{t_f}{t_s} + 1 \tag{3.39}$$

Recursive implementation

To implement (3.38) we need to store $N_f - 1$ old samples. Although computers are powerful nowadays, it is regarded a good habit to minimize data storage. To this end, let's develop a recursive version of (3.38) which requires storage of only one variable.

(3.38) can be written as

$$\begin{aligned} y_{mf,k} &= \frac{N_f-1}{N_f}\left(\frac{1}{N_f-1}\sum_{i=1}^{N_f-1} y_{m,k-i}\right) + \frac{1}{N_f}y_{m,k} && (3.40) \\ &= \frac{N_f-1}{N_f}y_{mf,k-1} + \frac{1}{N_f}y_{m,k} && (3.41) \end{aligned}$$

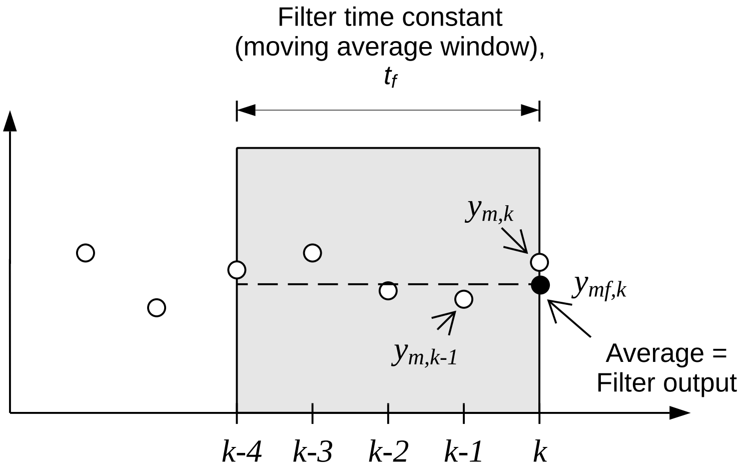


Figure 3.41: Moving average filter with length $N_f = 5$. t_f is the time window.

where $y_{mf,k-1}$ is the filter output from the previous time step. In (3.41) the previous filter output is updated with the present "raw" filter input.

(3.41) can be written as

$$y_{mf,k} = (1 - a)\, y_{mf,k-1} + a y_{m,k} \tag{3.42}$$

where the filter parameter a is

$$a = \frac{1}{N_f} = \frac{t_s}{t_f + t_s} \tag{3.43}$$

where t_s is the time step.

How to tune an MA filter

The tuning factor of the filter is the filter length, N_f. From (1.8) we can conclude:

- The larger N_f, the more samples are averaged, giving stronger filtering. Another way to see this, is that a large N_f gives a small a (closer to 0), which gives stronger filtering, cf. (3.42).
- The smaller N_f, the larger a (closer to 1), and the weaker filtering.

Thus, N_f should be as large as possible, to obtain strong noise filtering. But, it must not be set so large that also relevant process (e.g. temperature) information is filtered. In practice, various values of N_f can be tried until an ok value is found. In the process industries, a filter time constant t_f of value some seconds may be appropriate. As an example, assume that $t_f = 2.0$ s and that the sampling time of the filter is $t_s = 0.1$ s, then

$$N_f = \frac{t_f}{t_s} + 1 = \frac{2.0}{0.1} + 1 = 21 \tag{3.44}$$

Example 3.7 *MA filter implemented in Python*

Figure 3.42 shows a noisy sinusoidal input (blue curve) and the filter output (red curve) with an MA filter. The measurement noise is filtered effectively (the red curve is much smoother than the blue curve).

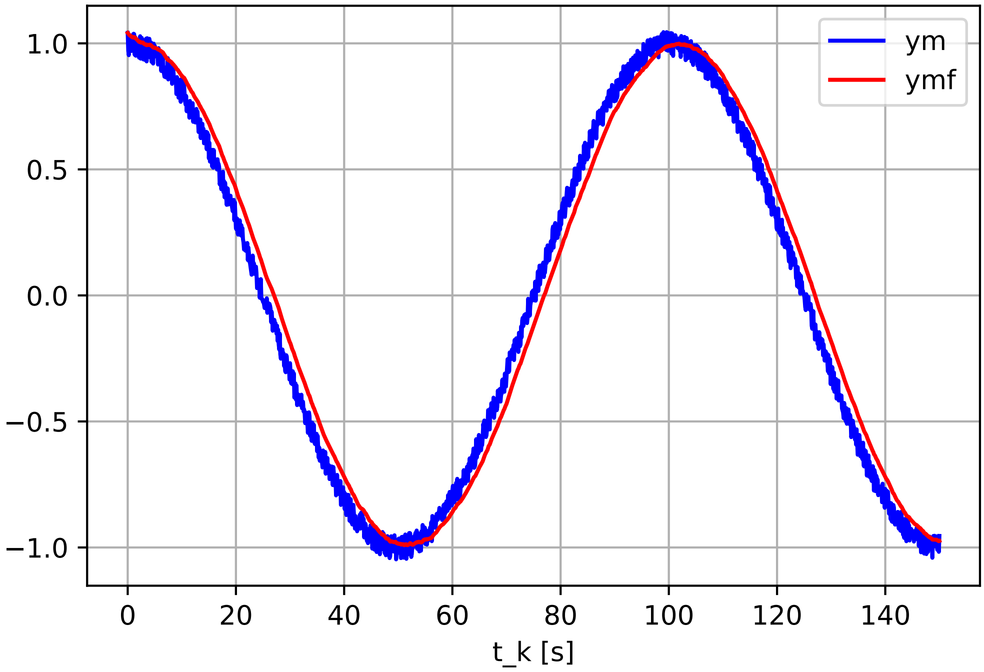


Figure 3.42: Input and output of an MA filter.

Comments to the simulations:

- The noise-free measurement, or “pure process signal, is

$$y_p(t) = A\cos(2\pi t/t_p) \tag{3.45}$$

  with amplitude $A = 1$ and period $t_p = 100$ s.

- Measurement noise is added to the noise-free measurement. This noise is random and uniformly distributed between $\pm A_n$ where $A_n = 0.05$.

- Filter time step is $t_s = 0.1$ s.

- Filter time window is $t_f = 2.0$ s.
- There is a small lag between the input signal and the output signal. A lag is inavoidable for online or real-time filters where the filter acts on new samples as they come. (For offline or batch filters the lag can be eliminated.)
- The initial value of the filter output, y_f, (red) is the same as the unfiltered input, or noisy measurement, y_m (blue) – namely approximately 1.0. Hence, the filter is "bumpless". In general, filters in control system should be bumpless, since the controller acts upon the filter output. Setting the initial filter output to e.g. zero would give an unfortunate control signal behaviour.

The MA filter is implemented in program 3.1.

https://techteach.no/control/python/ma_filter.py

Listing 3.1: ma_filter.py

```
"""
Moving average filter
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 08
"""

# %% Import

import matplotlib.pyplot as plt
import numpy as np

# %% Filter function

def fun_ma_filter(ym_k, yf_km1, Nf):

    a = 1/Nf
    yf_k = (1 - a)*yf_km1 + a*ym_k

    return yf_k

# %% Simulation time settings

ts = 0.1  # [s]
t_start = 0  # [s]
t_stop = 150  # [s]
N_sim = int((t_stop - t_start)/ts) + 1  # Num time-steps

# %% Preallocation of arrays for plotting

t_array = np.zeros(N_sim)
ym_array = np.zeros(N_sim)
ymf_array = np.zeros(N_sim)

# %% Params of signals
```

```
A = 1  # Amplitude of cosine
tp = 100  # [s] Period of cosine
An = 0.05  # Ampl of uniformly distributed random noise
num_samples = 1  # Number of samples from random generator

# %% Filter param

Nf = 21  # Filter length

# %% Simulation loop

for k in range(0, N_sim):

    t_k = k*ts  # Time

    # Signals:
    yp_k = A*np.cos(2*np.pi*t_k/tp)  # Noise free measurement
    n_k = np.random.uniform(-An, An, num_samples)[0]  # Meas noise
    ym_k = yp_k + n_k  # Noisy measurement

    # MA filter:
    if k == 0: ymf_km1 = ym_k  # Initially, filter out = filter in
    ymf_k = fun_ma_filter(ym_k, ymf_km1, Nf)

    # Arrays for plotting:
    t_array[k] = t_k
    ym_array[k] = ym_k
    ymf_array[k] = ymf_k

    # Time index shift:
    ymf_km1 = ymf_k

# %% Plotting

plt.close(’all’)
plt.figure(1)

plt.plot(t_array, ym_array, ’b’, label=’ym’)
plt.plot(t_array, ymf_array, ’r’, label=’ymf’)
plt.legend()
plt.xlabel(’t_k [s]’)
plt.grid()

plt.savefig(’response_ma_filt.pdf’)
plt.show()
```

[End of Example 3.7]

3.4.6.2 Time constant filter

Continuous-time filter function

The time constant filter can be represented by the following differential equation:

$$y'_{mf} = (y_m - y_{mf})/t_f \tag{3.46}$$

where:t_f [s] is the filter time constant, y_m is the unfiltered (raw) measurement signal, and y_{mf} is the filtered measurement signal.

In Section 3.4.6.2 you will see that t_f as defined in (3.46) has the same meaning in a discrete time version of (3.46) as the parameter t_f has in a moving average filter, cf. 3.4.6.1. This is the reason why I use the symbol t_f in both time constant filters and moving average filters.

Sometimes, the filter is represented by a transfer function corresponding to (3.46):

$$\frac{Y_{mf}(s)}{Y_m(s)} = H_f(s) = \frac{1}{t_f s + 1} \tag{3.47}$$

(In (3.47), s is the Laplace variable. Transfer functions are described in Ch. 8.)

The tuning parameter of the filter is the filter time constant, t_f. How to set t_f? That depends on how much noise smoothing you want and the dynamic properties of the process. If you do not have any other requirements, you can initially set it equal to one tenth of the time constant of the process to be controlled to avoid the filter adding too much sluggishness to the control loop. It turns out that a time constant of a few seconds is a typical value in industrial control loops (of e.g. temperature loops).[7]

A SimView simulator of a time constant filter is available on:

http://techteach.no/simview/lowpass_filter

Discrete-time time constant filter

In computer-based automation systems the filter is available as a function block containing program code that implements the filter algorithm. We will derive a filter algorithm with (3.46) as the basis. Let us assume that the time step of the filter algorithm is dt [s]. It is tradition to discretize a time constant filter using the Euler Backward method. Approximating the time derivative in (3.46) with Euler Backward approximation, gives

$$t_f \frac{y_{mf,k} - y_{mf,k-1}}{t_s} + y_{mf,k} = y_{m,k} \tag{3.48}$$

[7] In one of the Fuji PID temperature controllers the preset value of the filter time constant is 5 sec.

We need a formula for the filter output at time index k. Solving (3.48) with respect to $y_{mf,k}$ gives the filter algorithm:

$$y_{mf,k} = (1-a)y_{mf,k-1} + ay_{m,k} \tag{3.49}$$

with

$$a = \frac{t_s}{t_f + t_s} \tag{3.50}$$

The filter algorithm (3.49) is sometimes denoted the exponentially weighted moving average (EWMA) filter.

Comparing time constant filter with moving average filter

Filter algorithm (3.49) is identical to the MA filter algorithm (3.42). Therefore, if you implement (3.49) or (3.42), you can say that you implement an MA filter or (equally correct) a time constant filter.

How to tune a time constant filter

The tuning factor of the filter is the time constant t_f. From (3.49) and (3.50), we can conclude:

- The larger t_f,the smaller a (closer to 0), giving stronger filtering.
- The smaller t_f,the larger a (closer to 1), and the weaker filtering.

It is important that t_f is considerably larger than t_s, or in other words, that t_s is considerably smaller than t_f. Otherwise, the filter algorithm (3.49) may behave quite differently from the continuous-time filter (3.46) from which it is derived. We may set the lower bound on t_f as $10t_t$, i.e.

$$t_f \geq 10t_s \tag{3.51}$$

How can we choose an appropriate value of t_f? It should be as large as possible, to obtain noise filtering. But, it must not be set so large that also relevant process (e.g. temperature) information is filtered. In practice, various values of t_f should be tried until an ok value is found. In the process industries, typical t_f values are some seconds. At one particular industrial plant in Norway, the default value of t_f is 2 sec. And PID temperature controllers by Fuji Electric have $t_f = 5.0$ s as default. On processes with very fast dynamics, as motors, t_f should probably be set to a fraction of a second.

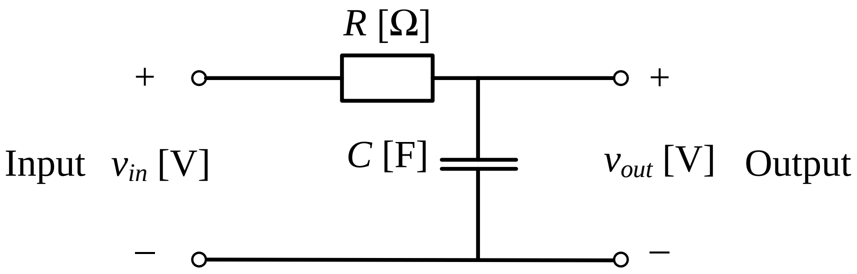


Figure 3.43: RC filter.

3.4.6.3 Analog RC circuit filter

Figure 3.43 shows an analog time constant filter implemented with a resistor and capacitor, often denoted an RC filter.

It can be shown, cf. Example 4.5, that the filter model is

$$v_{\text{out}}{}' = (v_{\text{in}} - v_{\text{out}}) \, / \, (RC) \tag{3.52}$$

Comparing with (3.46) we see that the filter time constant is

$$t_f = RC \tag{3.53}$$

So, you can obtain a specified filter time constant by chosing appropriate sizes of the resistor and the capacitor.

3.5 Actuators

3.5.1 Introduction

The automation equipment uses actuators to operate or manipulate the process to be controlled. In the following subsections, we will look at some of the most commonly used actuators:

- Control valve
- Pumps
- Motors
- Heaters

3.5.2 Valves

3.5.2.1 Structure and operation

Control valves control valves are used to manipulate liquid flow rates and gas flow rates. Figure 3.44 shows a pneumatically controlled control valve. By manipulating the air pressure, the diaphragm and thus the valve stem and plug are moved up or down. In this way, the liquid or gas flow rate can be manipulated using air pressure.

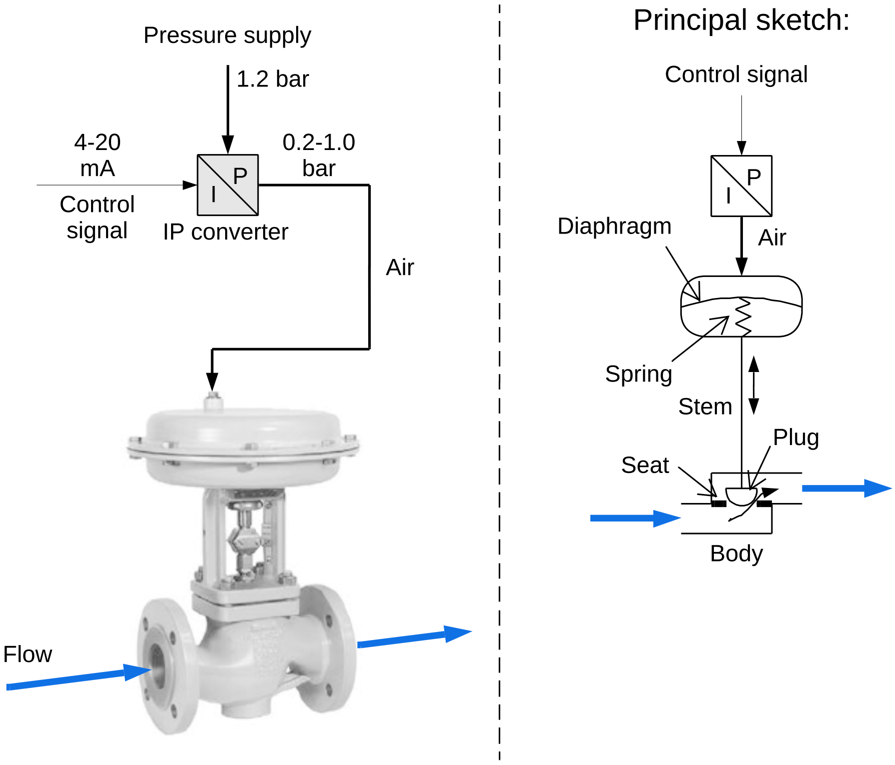


Figure 3.44: Pneumatically controlled control valve. (Valve image: Samson-Matek AS)

3.5.2.2 Valve equation and valve characteristics

Valve equation

The valve equation expresses the relationship between liquid flow rate Q, valve opening z, and pressure drop p_v across a valve. (In this presentation, liquid is assumed; we do not go into gas valves.) The valve equation can be written as

$$Q = K_v f(z) \sqrt{\frac{p_v}{G}} \tag{3.54}$$

where:

- Q [m^3/h] is the liquid flow rate.
- p_v [bar] is the pressure drop across the valve.
- G is specific (relative) density in relation to water. For water, $G = 1$. For oil, $G \approx 0.85$.
- K_v is the valve constant or the capacity index.
- z is normalized valve opening. $z = 0$ means fully closed valve. $z = 1$ means fully open valve.
- $f(z)$ is the valve function which has a value between 0 and 1. $f = 1$ means maximum flow rate for a given pressure drop p_v. The valve function constitutes the valve characteristics, see below.

Let's assume $f = 1$, i.e. maximum flow rate. The valve equation (3.54) is then

$$Q_{\max} = K_v \sqrt{\frac{p_v}{G}} \tag{3.55}$$

Therefore, the valve equation (3.54) can be written as

$$Q = Q_{\max} f(z) \tag{3.56}$$

which gives

$$f(z) = \frac{Q}{Q_{\max}} \tag{3.57}$$

In other words, f is the normalized flow rate.

Definition (or meaning) of valve constant K_v

From (3.55) we see that

$$K_v = Q_{\max} \tag{3.58}$$

assuming the pressure drop p_v is 1 bar, and water as medium. (3.58) defines K_v. It is assumed that $Q_{\max}$ is observed at temperature 15.5 °C = 60 °F. In other words, K_v is the water flow through a fully open valve at the specific temperature 15.5 °C = 60 °F.

Valve size selection

When you are to select the valve size for a given application, you select $K_v = Q_{\max}$. A capacity margin of e.g. 50 % should be taken into account.

Inherent valve characteristics

The valve function (3.57) defines the inherent valve characteristics, which is the valve characteristics when the valve is not connected to a process, i.e. when the valve "stands alone". For control valves, there are three relevant inherent characteristics, see below. They are plotted in Figure 3.45.

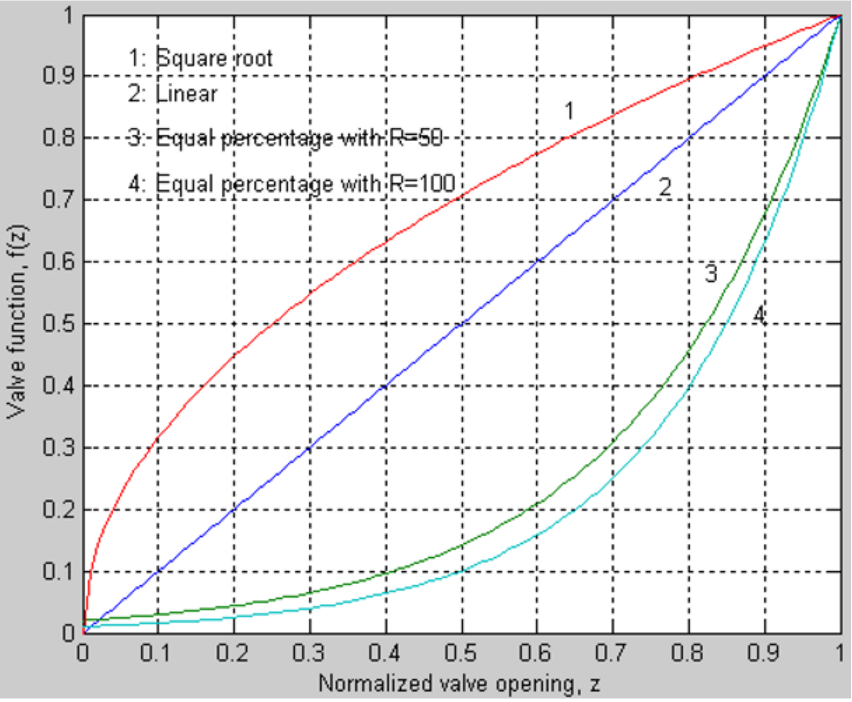


Figure 3.45: Various inherent valve characteristics (valve function).

Linear valve characteristics

The linear characteristic is given by

$$f(z) = z \tag{3.59}$$

Equal percentage valve characteristic

This is a logarithmic characteristic, and hence a nonlinear characteristic. The equal percentage characteristic is given by

$$f(z) = R^{1-z} \tag{3.60}$$

where R is the rangeability, which typically has a value of 50.

Square root valve characteristic

This is also a nonlinear characteristic, given by the square root function:

$$f(z) = \sqrt{z} \tag{3.61}$$

Valves with this valve characteristic are denoted “quick opening valves”. They are particularly relevant for safety operations, and not so relevant for usual feedback control.

Installed valve characteristic

Figure 3.46 shows a feedback control system of a process with valve as actuator. The control system may be e.g. a temperature control system or a level control system. The *installed* valve characteristic, $f_{\text{inst}}(z)$, that expresses the relationship between valve opening z and flow rate Q when the valve is connected to the process:

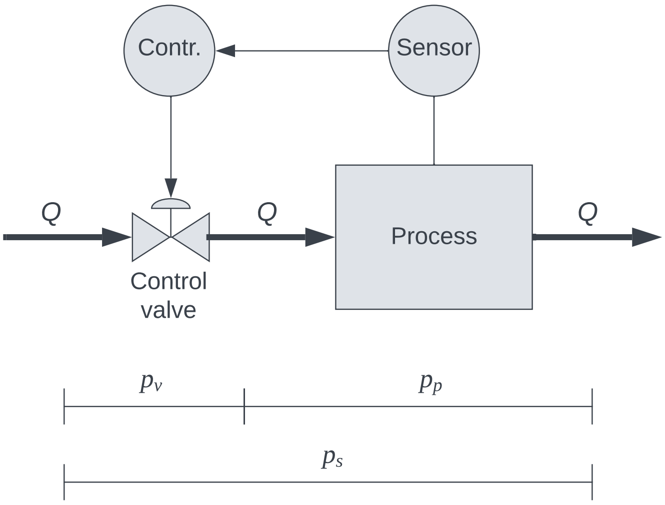


Figure 3.46: Feedback control system of a process with valve as actuator.

$$Q = Q_{\max} f_{\text{inst}}(z)$$

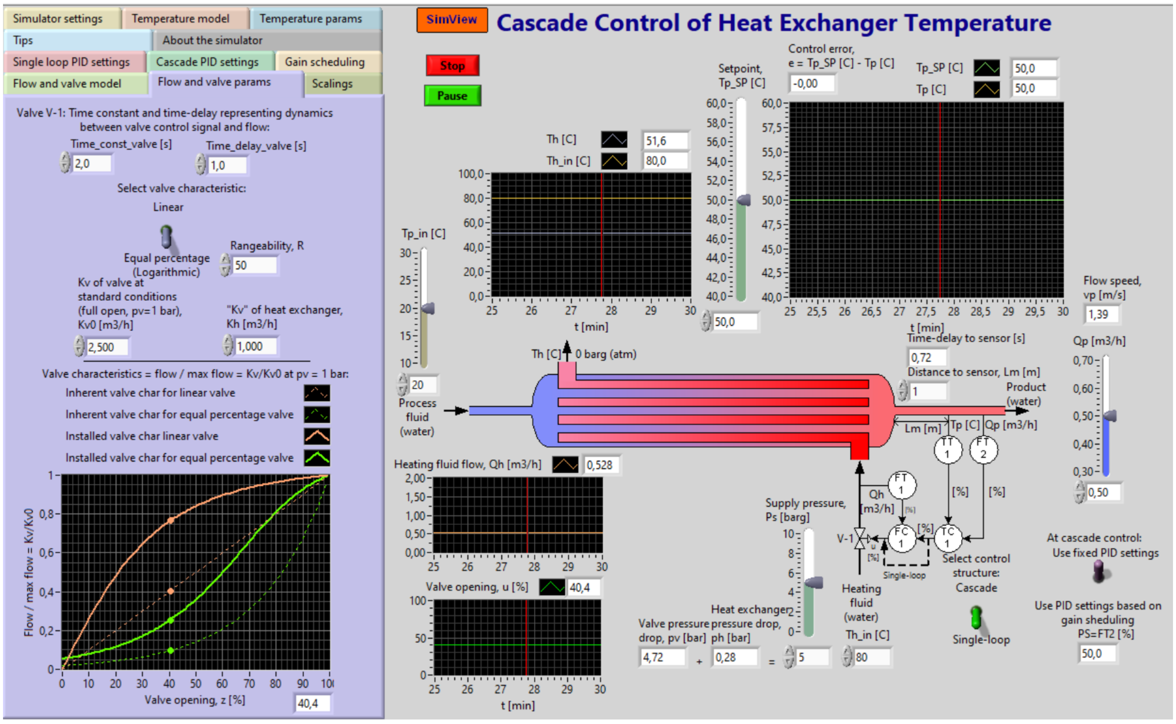
or

$$f_{\text{inst}}(z) = \frac{Q}{Q_{\max}} \tag{3.62}$$

$f_{\text{inst}}(z)$ may be different from $f(z)$ if the pressure drop p_v in the valve equation (3.54) depends on the flow rate Q.

Example 3.8 *Inherent and installed valve characterics*

The difference between $f_{\text{inst}}(z)$ and $f(z)$ is illustrated with the application depicted in Figure 3.47 where the temperature of a simulated heat exchanger is controlled with a control valve (V-1 in the figure) as an actuator. The valve adjusts the hot water flow rate used to heat the cold process fluid. Both the inherent characteristic and the installed characteristic for both the linear valve and the equal percentage valve are shown in the valve characteric diagram at the left side of the figure. This diagram is shown enlarged in Figure 3.48.



Valve V-1: Time constant and time-delay representing dynamics between valve control signal and flow:

Time_const_valve [s] 2,0

Time_delay_valve [s] 1,0

Select valve characteristic: Linear / Equal percentage (Logarithmic)

Rangeability, R 50

Kv of valve at standard conditions (full open, pv=1 bar), Kv0 [m3/h] 2,500

"Kv" of heat exchanger, Kh [m3/h] 1,000

Valve characteristics = flow / max flow = Kv/Kv0 at pv = 1 bar:

- Inherent valve char for linear valve
- Inherent valve char for equal percentage valve
- Installed valve char linear valve
- Installed valve char for equal percentage valve

Valve opening, z [%] 40,4

Th [C] 51,6

Th_in [C] 80,0

Setpoint, Tp_SP [C] 50,0

Control error, e = Tp_SP [C] - Tp [C] -0,00

Tp_SP [C] 50,0

Tp [C] 50,0

Flow speed, vp [m/s] 1,39

Time-delay to sensor [s] 0,72

Distance to sensor, Lm [m] 1

Qp [m3/h] 0,50

Tp_in [C] 20

Heating fluid flow, Qh [m3/h] 0,528

Valve opening, u [%] 40,4

Valve pressure drop, pv [bar] 4,72 + Heat exchanger pressure drop, ph [bar] 0,28 = Supply pressure, Ps [barg] 5

Th_in [C] 80

Select control structure: Cascade / Single-loop

At cascade control: Use fixed PID settings / Use PID settings based on gain sheduling PS=FT2 [%] 50,0

Figure 3.47: Heat exchanger where a control valve manipulates the hot water flow rate into the heat exchanger.

- The dashed curves are the inherent valve characteristics for both types of valves.
- The solid curves are the installed valve characteristics for both types of valves.
- The dots in the diagram represents the present operating point.

From the curves in Figure 3.48 we see that, in this application, the installed characteristic of the nonlinear, equal percentage valve is more linear than the installed characteristic of the linear valve! This is because the pressure drop over the heat exchanger – the process – depends largely on the flow through the process. This is explained further after this example.

[End of Example 3.8]

Selection of valve characteristic

I will here not go into a further, detailed analysis of how the installed versus differs from the inherent valve characteristic, but present guidelines for selecting between a linear valve and an equal percentage valve.

In general, it is advantageous for the control system that the installed characteristic is as linear as possible. Below are some guidelines for selecting a proper valve characteristic:

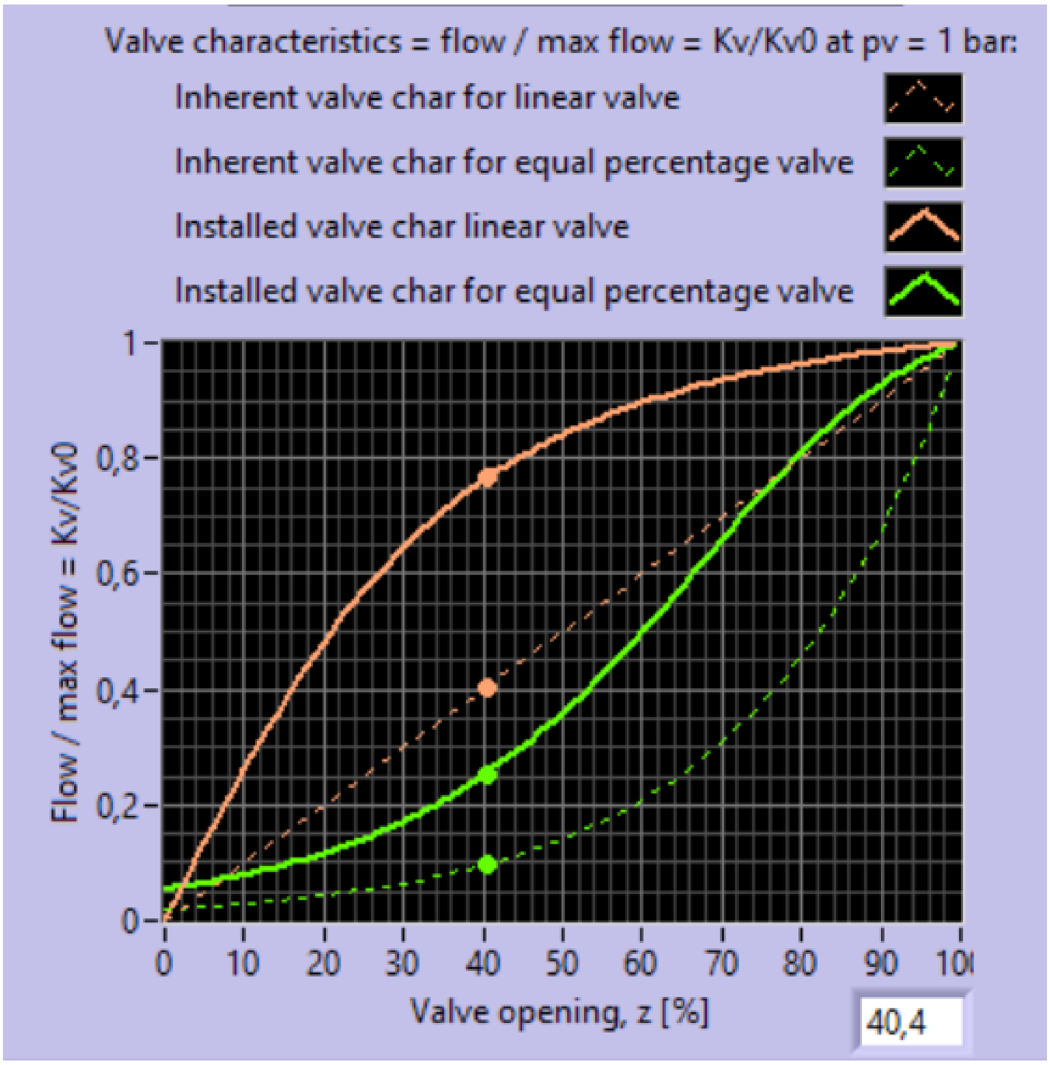


Figure 3.48: Inherent characteristic and the installed characteristic for both the linear valve and the equal percentage valve used in a simulated heat exchanger.

- *A valve with linear characteristic* should be selected if the pressure drop across the valve is approximately constant even if the flow rate varies. This is the case when the pressures on each side of the valve are approximately constant. An example is when a valve in the inlet is used to fill a tank with liquid.

- *A valve with equal percentage characteristic* should be selected if the pressure drop across the valve varies significantly with the flow. This is the case when the pressure after (downstream) the valve, or the "resistance pressure", varies significantly with the flow rate. Example 3.8 illustrates this case. There, flow resistance varies with flow rate due to the pipes and/or plates inside the heat exchanger, and concequently, an equal percentage characteristic should be chosen.

3.5.3 Pumps

3.5.3.1 Centrifugal pumps

Centrifugal pumps, see Figure 3.49, are the most common pump type in industry, for both liquids and gases. The pump is driven by a motor, typically an alternating current (AC) motor. The flow through the pump is given by both the pump speed (pump engine speed), and the pressure drop across the pump. This pressure drop again depends on the pressure (flow resistance) from the plant itself. This pressure drop will increase with the flow through the system and depend on e.g. of the valve opening etc. in the plant.

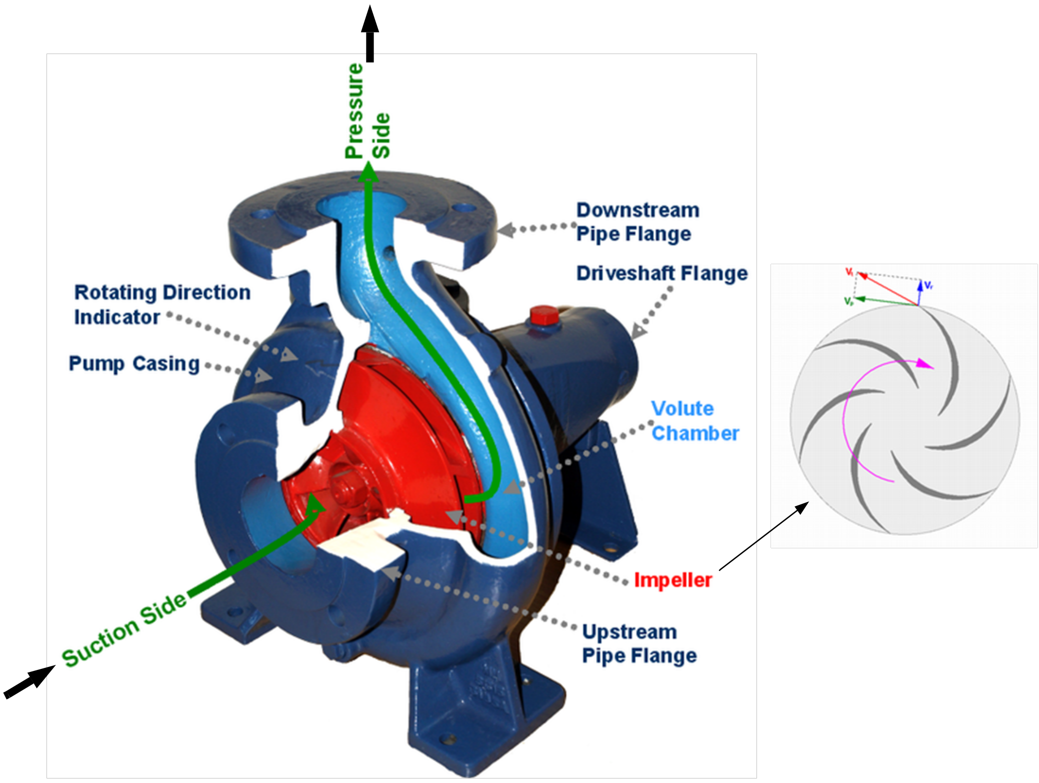


Figure 3.49: Centrifugal pump. (http://en.wikipedia.org/wiki/Pump).

In pump operations, pressure is often expressed in terms of head H, which is the hydrostatic pressure due to water column height, H:

$$p = \rho g H$$

where p [Pa] is pressure, H [mH_2O = meter water column] is head, ρ [kg/m^3] is density, and g [m/s^2] is gravity.

Flow control

The flow through centrifugal pumps is a function of both the speed of the impeller and the pressure difference across the pump. The pressure difference changes if the pressure (flow resistance) from the plant changes. Centrifugal pumps are therefore often flow controlled to ensure that the pump delivers the desired flow despite such pressure variations, see Figure 3.50.

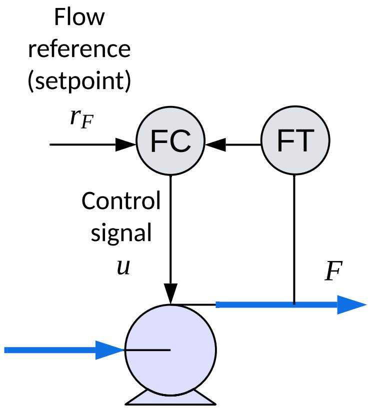


Figure 3.50: Centrifugal pump with flow rate control.

Select pump or valve?

Both control valves and pumps are actuators that are be used to manipulate the rate of liquid or gas flow to a plant. Should you use valve or pump? The company Siemens has made calculations that show that it is more economical to use a (speed-controlled) pump rather than a control valve (a supply pump is assumed to provide a suitable supply pressure for the valve). Using a pump can provide 50 % energy savings compared to using a valve. This is not so surprising. Using a pump corresponds to controlling the speed of a car with gas and brake, but never at the same time, while a supply pump plus a control valve is like pressing the gas pedal fully in to have a high fuel supply to the engine, and using the brake to obtain the desired speed, i.e.using gas and brake at the same time – which is obviously not an optimal way to drive a car.

3.5.3.2 Displacement pumps

Displacement pumps provide a liquid flow that is proportional to the pump speed – regardless of the pressure drop across the pump. Figure 3.51 shows a displacement pump – a so-called peristaltic pump – for small liquid flows. A roller pushes the liquid forward through a flexible plastic tube. The speed can be controlled with a voltage or current signal.

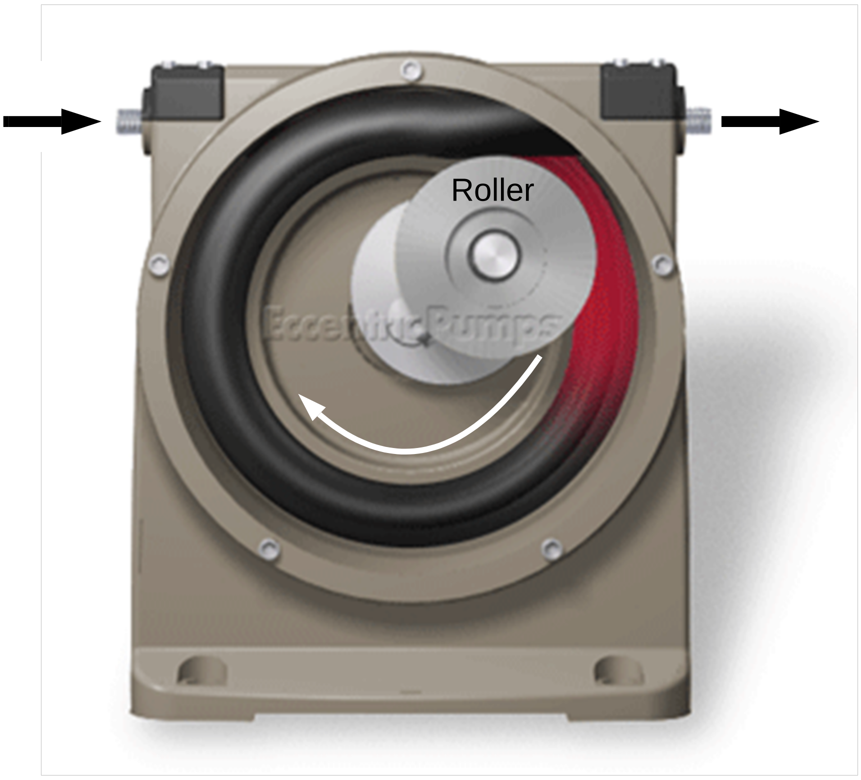


Figure 3.51: Peristaltic pump, which is a type of displacement pump. (http://en.wikipedia.org/wiki/Pump)

Other examples of displacement pumps are gear pumps and piston pumps.

Particles in the fluid can cause greater problems for positive displacement pumps than for centrifugal pumps. In general, positive displacement pumps can work at greater load pressure (resistance pressure) than centrifugal pumps can.

3.5.4 Electrical motors

There are several types of electrical motors. Here, only AC motors (AC = alternating current) are described, briefly. AC motors is the most used motor type in industrial applications. AC motors are used for the operation of fans, pumps, screws, compressors, etc.

AC motors are manipulated by frequency converters, see Figure 3.52.

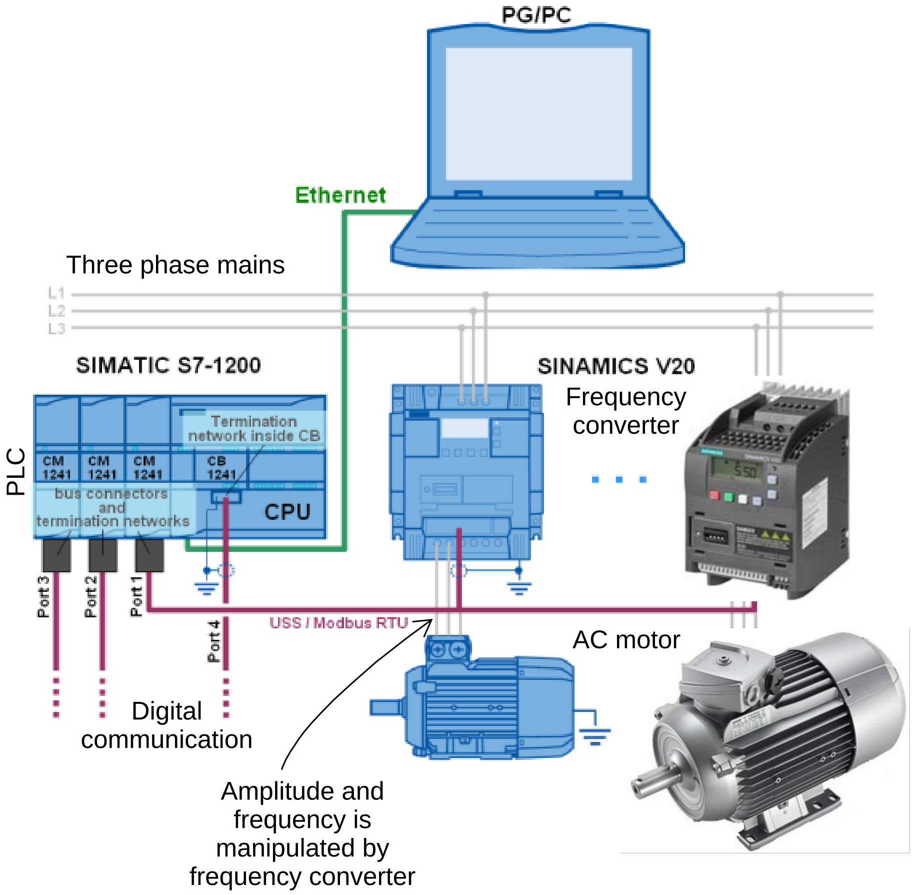


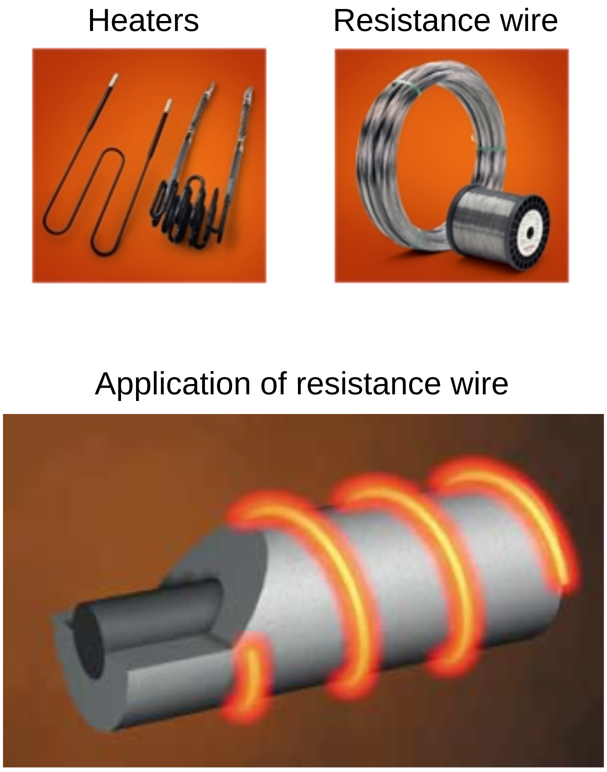
Figure 3.52: Sinamics V-20 frequency converter and Simotics GP AC motor. (Simatic S7-1200 PLC. Siemens AS.)

The frequency converter converts the alternating voltage from the fixed network (1- or 3-phase network, 230 V, 50 Hz) into an alternating voltage with variable voltage and frequency suitable for controlling the speed of the AC motor. The frequency converters are controlled by automation equipment such as PLCs.

3.5.5 Electrical heaters

Electrical heating elements in the form of heaters and resistance wires are widely used to manipulate the temperature in various types of processes. The heat is generated when an electric current passes through the resistor in the heating element. Figure 3.53 shows some commercial products.

If the heating element is to be included as an actuator in a temperature control system, the mains voltage may be switched on/off over the heating element using pulse width modulation, cf. Section 3.6.3.



Heaters

Resistance wire

Application of resistance wire

Figure 3.53: Heating elements. (Kanthal products. Sandvik AB, Sweden).

To select a proper heating element in terms of resistance value, you must first decide the average power P_{average} [W] the component is required to emit, and then calculate the resistance value R [Ω]. If you are going to use the AC voltage from the mains as a voltage source, you can use the effective value U_{eff} [V] of the AC voltage to calculate the resistance value. The following fundamental formulas of electrical power apply:

$$P_{\text{average}} = \frac{{U_{\text{eff}}}^2}{R} \tag{3.63}$$

which gives

$$R = \frac{{U_{\text{eff}}}^2}{P_{\text{average}}} \tag{3.64}$$

For mains voltage in e.g. Norway, $U_{\text{eff}} \approx 226$ V.

Resistance wires are characterized with their length-specific resistance value:

$$R_s = \frac{R}{L} \; [\Omega/\text{m}] \tag{3.65}$$

R_s can be used to select the appropriate resistance wire, as demonstrated in Example 3.9.

Example 3.9 *Calculating a proper resistance value*

Given a biogas reactor which is to be heated with a heating wire wound around the reactor. The heater must be able to emit 200 W when the mains voltage is switched on. Assume that a length of $L = 10$ m is appropriate. What value of R_s meets the specifications?

From (3.64) we get

$$R = \frac{{U_{\text{eff}}}^2}{P_{\text{mean}}} = \frac{(226 \text{ V})^2}{200 \text{ W}} = 255 \ \Omega$$

which gives

$$R_s = \frac{R}{L} = \frac{255 \ \Omega}{10 \text{ m}} = 25.5 \ \frac{\Omega}{\text{m}}$$

[End of Example 3.9]

3.6 Signal conditioning of control signals

3.6.1 Scaling of control signals

We will now focus on the control signal scaling function (block) in Figure 3.1. This function scales or transforms the control signal calculated by the controller into a control signal to be applied to the actuator – typically a milliampere (mA) current signal.

Figure 3.54 plots a linear scaling function. The units and values used in the figure are just examples. Other units and values may apply in other cases.

The linear scaling function in Figure 3.54 can be expressed mathematically as

$$v = a \cdot u + b \tag{3.66}$$

where the slope is

$$a = \frac{v_2 - v_1}{u_2 - u_1} \tag{3.67}$$

and the constant (based on point 1; alternatively point 2 could have been used) is

$$b = v_1 - a \cdot u_1 \tag{3.68}$$

Example 3.10 *Control signal scaling*

Assume a control system with control signal scaling as shown in Figure 3.54.

What are the values of a and b?

What is the control signal, v, to the actuator if the control signal, u, from the controller is 50 %?

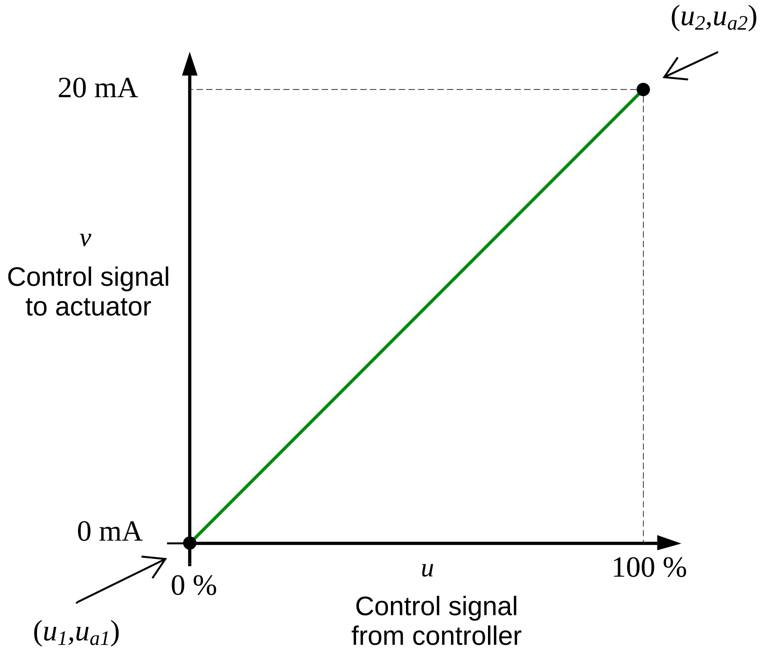


Figure 3.54: Control signal scaling.

(3.67) becomes

$$a = \frac{v_2 - v_1}{u_2 - u_1} = \frac{20 \text{ mA} - 0 \text{ mA}}{100 \ \% - 0 \ \%} = \frac{20}{100} \ \frac{\text{mA}}{\%} = 0.2 \ \frac{\text{mA}}{\%} \tag{3.69}$$

(3.68) becomes

$$b = v_1 - a \cdot u_1 = 0 \text{ mA} - 0.2 \ \frac{\text{mA}}{\%} \cdot 0 \ \% = 0 \text{ mA} \tag{3.70}$$

If $u = 50$ %, (3.66) gives

$$v = a \cdot u + b = 0.2 \ \frac{\text{mA}}{\%} \cdot 50 \ \% + 0 \text{ mA} = 10 \text{ mA} \tag{3.71}$$

[End of Example 3.10]

3.6.2 Digital-analog (DA) conversion

Figure 3.1 shows the place of the DA converter (DA = digital-analog) in a feedback control system. The DA converter is an electronic component that converts the digital control signal into an analog (physical) control signal, which is typically a current signal in the range of 0-20 mA, or 4-20 mA, or a voltage signal, e.g. 0-5 V, acting on the actuator.

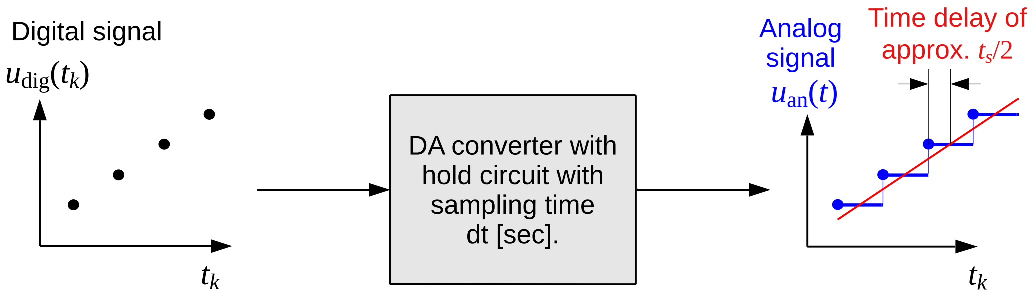

Digital signal
$u_{\text{dig}}(t_k)$
t_k
DA converter with hold circuit with sampling time dt [sec].
Analog signal
$u_{\text{an}}(t)$
Time delay of approx. $t_s/2$
t_k

Figure 3.55: DA (digital-analog) conversion.

Figure 3.55 illustrates DA conversion. The DA converter operates with a fixed time step, or sampling time, t_s [sec], which is typically set equal to the sampling interval in the AD converter, cf. Figure 3.1. The digital control signal is converted to an analog current or voltage signal, and is held in an electronic component denoted the hold circuit until the next digital control signal is available. This means that the analog output signal from the DA converter is actually a staircase-shaped signal, see Figure 3.1. In practice, the actuator controlled by this staircase-shaped signal is a slow system compared with the sampling time, and the actuator can hardly "feel" the steps but instead an almost smooth control signal.

As illustrated in Figure 3.1, the holding results in the average analog signal being approximately time-delayed by half the sampling time, i.e.

$$t_d \approx \frac{t_s}{2} \tag{3.72}$$

If t_s, and therefore t_d, is very small compared to the time constant, T (the time constant is defined precisely in Ch. 9.3), of the actuator, this time delay makes no impact on the performance of the control system. "Very small" may be interpreted as $t_s \lesssim T/10$.

3.6.3 Pulse-width modulation

Pulse-width modulation (PWM) is an "actuator" technique to obtain *in average* a specified control signal with just an *on/off signal*, which we can denote u_{pwm}. This signal is typically the closed/open state of a mechanical relay or a solid state relay (SSR), which is a semiconductor-based relay without any mechanical parts.

PWM is typically an option in industrial process controllers (Section 3.2.2). In microcomputer systems as Raspberry Pi and Arduino, PWM is used implement "analog output".

The principle of PWM is to keep u_{pwm} at constant values U_{on} and U_{off} for such a duration that the resulting average control signal, u_{mean}, is as specified. PWM elements operates with a given fixed period, t_p, e.g. 1.0 sec, see Figure 3.56. (In the figure, U_{off} is assumed 0,

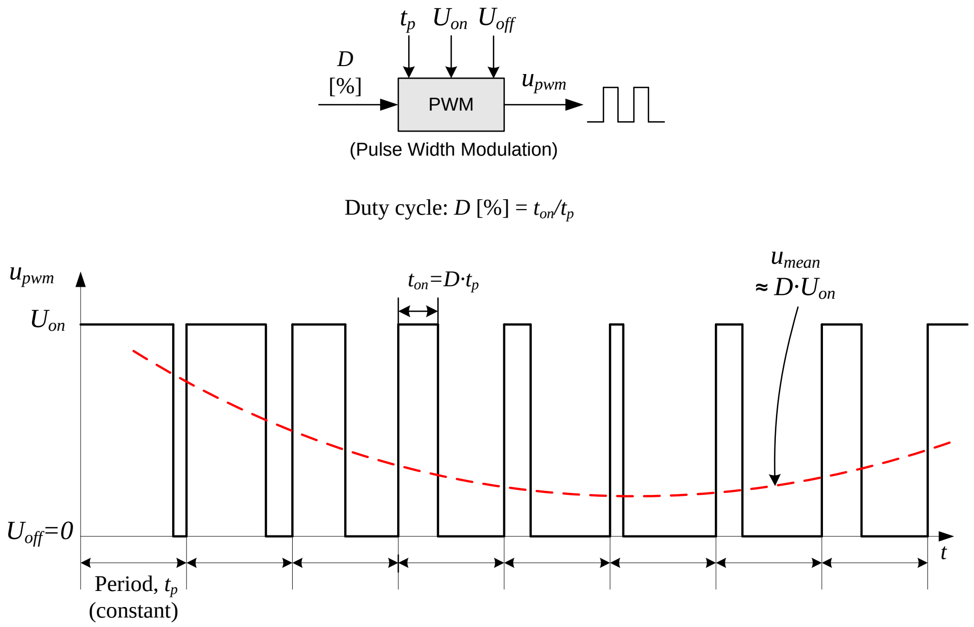


Figure 3.56: The principle of pulse-width modulation (PWM).

but it can be set to a nonzero value.) The part of the period where $u_{\text{pwm}} = U_{\text{on}}$ is denoted the duty cycle, D, which is given in percent:

$$D = \frac{t_{\text{on}}}{t_{\text{p}}} \ [\%] \tag{3.73}$$

For a given (specified) D, the on-time is then

$$t_{\text{on}} = D \cdot t_{\text{p}} \tag{3.74}$$

Note that D can have *any* value in terms of percentage, e.g. 12.3 %, or 45.6 %.

Figure 3.57 shows how PWM can be used in a temperature control system. In the figure, the PID controller and the PWM element are shown as individual blocks. However, a PID controller (as an automation components) may have the PWM and a mechanical relay or an SSR integrated.

Example 3.11 *Simulation of control system with PWM*

Figure 3.58 shows the front panel of a SimView simulator of a control system with PWM.

The simulator is available on:

http://techteach.no/simview/pwm_control

[End of Example 3.11]

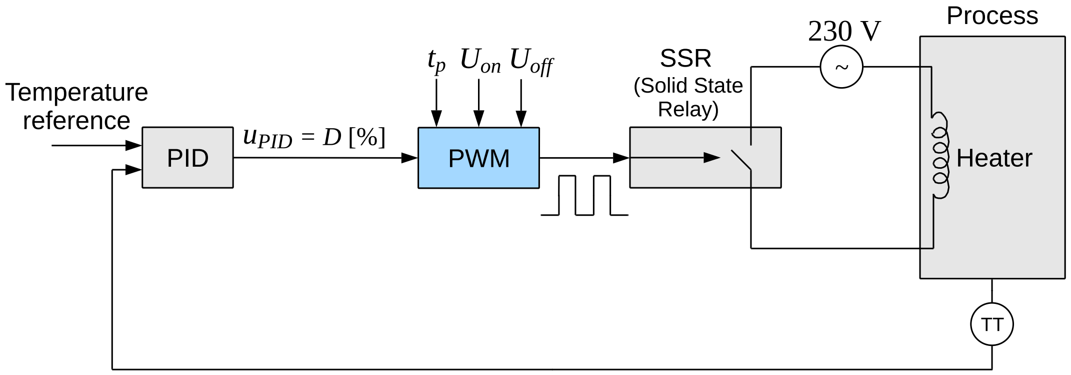


Figure 3.57: PWM in temperature control.

3.6.4 Converting current to voltage

Suppose you have a PID controller which generates a control signal in the form of a current signal, and the controller will be used to control an actuator which requires a voltage control signal. Then you need to convert the current signal into a voltage signal. How? By letting the current pass through a resistor, and using the voltage drop across the resistor as control signal, see Figure 3.59. You can find the resistance R with Ohm's Law:

$$R = \frac{U}{I}\ [\Omega] \tag{3.75}$$

For example, the resistance to convert 0-20 mA into 0-5 V is 5 V/20 mA = 250 Ω.

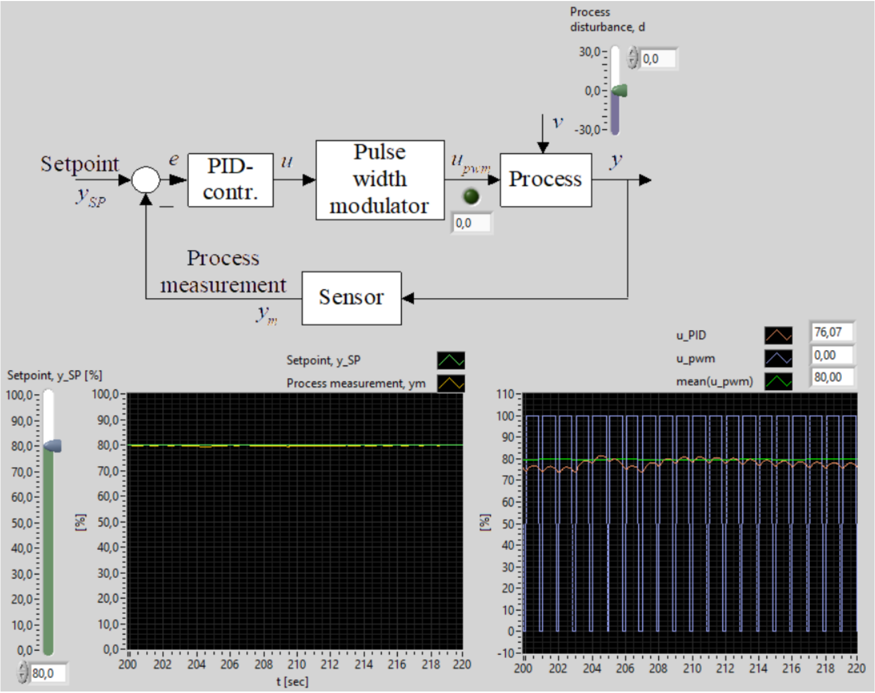


Figure 3.58: Control system with PWM.

3.7 *Problems for Chapter 3*

Problem 3.1 *Control and monitoring equipment*

A number of different control and motitoring applications are indicated below. What type of equipment might be suitable for each?

1. A laboratory rig that requires an advanced program for control, measurement and analysis. Standard Windows software must be able to be used on the system. It is acceptable if the control system locks for a period as long as the last sent control signals (e.g. the voltage signals) are maintained ("frozen").

2. A relatively small facility that does not require particularly advanced control. There will be little need for changes to the control program after start-up.

3. A relatively large plant that does not require particularly advanced monitoring and control. It is neccessary to have effective data communication between different levels in the company.

4. A small facility that requires fairly advanced, but still standard control. There is no need to implement advanced mathematical functions.

5. A relatively small plant that must be controlled in a safe way, i.e. that it is not acceptable for the control system to fail. There is a need to implement some advanced mathematical functions.

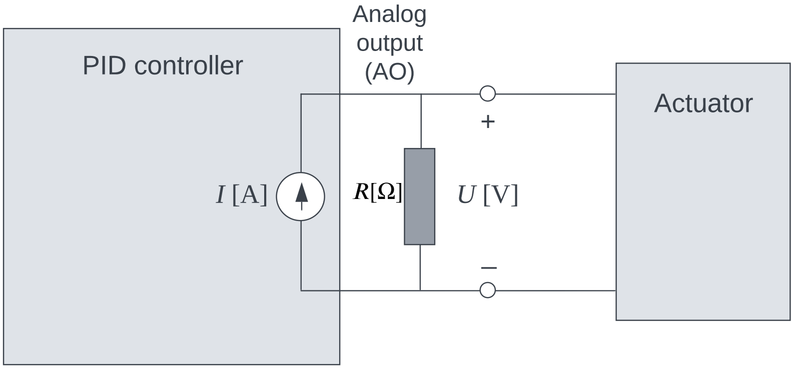


Figure 3.59: Converting current from a controller to voltage for an actuator.

6. A relatively large facility that requires advanced monitoring and control using pre-programmed functions. There is a need for effective data communication between different levels in the company.

Problem 3.2 *Pt100*

Assume that the resistance value of a Pt100 element is measured as $R = 161.4\Omega$ (with a Wheatstone's bridge circuit). What is the measured temperature T?

Problem 3.3 *Which thermometer?*

Assume that the reference of a given temperature control system is 80 °C. Should you then choose a thermocouple or resistance temperature sensor as sensor?

Problem 3.4 *Level measurement with dp-cell*

Assume a dp cell (differential pressure sensor) is used to measure the level of oil in an oil tank where there is atmospheric pressure above the oil surface. The sensor measures the static pressure at a point in a closed pipeline out of the bottom of the tank. The pressure measurement point is located 0.5 meters below the bottom level of the oil. The density of the oil is assumed to be 850 kg/m^3. Assume that the measurement signal is 0.1 bar. What is then the oil level, h?

Problem 3.5 *Alternative pressure units*

Pressure values can be represented in a number of alternative units. Assume that the gas pressure, p, in a vessel is 0.5 bar above atmospheric pressure. Express p in the following units (if you are unsure what some of the units below stand for, please try to find information on the internet):

1. bar, also denoted bara and bar(a)
2. barg, also denoted bar(g)
3. Pa (Pascal)
4. mmH_2O, i.e. hydrostatic pressure as mm water column pressure

Problem 3.6 *Selection of flow rate sensor*

Which type of sensor(s) (you may specify one or more types in the answer) for measuring liquid flow rate in a pipeline is (possibly) most relevant in each of these cases:

1. The volumetric flow rate is to be measured. It is not relevant \aa to make interventions in the pipeline for \aa f\aa m\aa lt the current.
2. The volumetric flow rate is to be measured. A very high accuracy in the measurement is required, and you have a lot of money.
3. The volumetric flow rate is to be measured. Intervention in the pipeline is acceptable. There is no requirement for very high accuracy.
4. The mass flow rate is to be measured (directly).
5. The density of the liquid is to be measured (directly), and also the mass flow is to be measured.

Problem 3.7 *Methane gas flow rate measurement*

For a given experimental biogas reactor, assume that the produced biogas flow rate is 258 L/d (measured with a thermal sensor). The methane gas concentration in the biogas is approximately 73 % (measured with a spectroscopic sensor). What is the methane gas flow rate in unit L/d?

Problem 3.8 *Encoder resolution*

Assume that an encoder is used to measure the rotation d [mm] of a measuring point at the end of a arm that can rotate (turn). The distance from the center of the arm – where the encoder is mounted – to the measuring point is $L = 1$ m. Assume that the encoder has 1024 bits.

1. What is the resolution, r_d [mm/pulse] in the measurement of d?
2. If the resolution found in part task 1 is not sufficiently small, how can you change the measurement system (but keep the encoder) to get an improved resolution?

Problem 3.9 *Calculation of speed from position measurements*

Given the following two values of a position measurement: $s(t_0) = 1.235$ m and $s(t_1) = 1.238$ m. The time interval between times t_0 and t_1 is 0.05 s. Calculate the speed $v(t_1)$ from these position measurements.

Problem 3.10 *Alternative sensors*

Briefly describe two or three measurement principles – in addition to those in the book – for the following process variables (you should use the internet if necessary):

- Temperature
- Pressure
- Level
- Liquid or gas flow
- Position

Problem 3.11 *Resistance for a current loop*

In a specific current loop including data acquisition device, the sensor outputs a current value in the range 4-20 mA. The data acquisition device can record voltage signals in the range 0-10 V. Find an appropriate resistance value R for the current loop.

Problem 3.12 *ADC resolution*

Given a 12 bit AD converter covering the analog signal range from $y_{a_{\min}} = 0$ V to $y_{a_{\max}} = 10$ V.

1. What is the resolution in volt?
2. Assume that the sensor is a level sensor with a range of 0.5-5 m which corresponds to a signal in the range of 0-10 V. What is the resolution in metres?
3. What is the relative resolution?
4. Assume that the 12 bit ADC is replaced by a 16 bit ADC. What is the relative resolution of the 16 bit ADC?

Problem 3.13 *Measurement parameters*

Assume that a level sensor of a water tank produces a measurement signal (m) in the range 4-20 mA, corresponding to a process value (p) in the range 5-15 m with a linear measurement characteristic.

1. What is the values of the following measurement parameters?
    - Upper process value
    - Lower process value
    - Zero
    - Span
2. What are the parameters a and b in the following linear measurement characteristic?
$$p = a \cdot m + b \tag{3.76}$$
3. What is p if $m = 12$ mA?

Problem 3.14 *Sensor accuracy*

Given a thermal gas flow sensor with measurement range 0-10 ml_n/min. (The subindex n means at the normal conditions, i.e. atmospheric pressure and temperature 20 °C. The sensor accuracy is specified as ±1 % FS. What is the accuracy in ml_n/min? What measurement error should you assume that every measurement reading has?

Problem 3.15 *From MA filter to time constant filter*

Assume that for a given filtering application you know that an MA filter with time constant $t_f = 2$ s operating with time step $t_s = 0.01$ s will work. But the hardware you are using implements a time constant filter. Which filter time constant should you use for the time constant filter?

Problem 3.16 *Changing the time step of an MA filter*

Assume that for a given application you have found out that an MA filter with filter time constant $t_f = 5$ sec operating with time step 0.05 s works fine. For some technical reason the time step must be increased to 0.1 s. Should you use $t_f = 5$ s also with this new time step, or some other value of t_f?

Problem 3.17 *Static response of MA filter*

Assume that the filter input of an MA filter is constant, say Y_m. We would like the corresponding static response, Y_{fm},to be equal to Y_m. Show that this is fortunately the case.

Problem 3.18 *Dimensioning a control valve*

Dimension a control valve for a pipeline with oil based on the following specifications:

- The pressure drop across the valve is mainly constant and equal to 0.5 bar.
- The temperature is about 15°C.
- The maximum oil flow rate through the valve is 15 L/min under normal conditions, but there should be a capacity margin of 50 %, i.e. the valve should be sized for 50 % greater flow than the assumed maximum flow rate.

Problem 3.19 *Dimensioning an electrical heater*

Assume that the water in a tank must be heated with a heating element. The heating element must be able to emit 1 kW when the mains voltage is switched on. Find the resistance value, R, of the heating element.

Problem 3.20 *Scaling of control signal*

Given a biogas reactor with a biogas flow rate control system. The biogas flow rate controller adjusts the biological feed to the reactor using a feeding pump. The controller calculates a flow rate in the range 0-80 L/d. The pump is controlled by a voltage signal in the range 0-5 V covering the aforementioned feed flow range, with a linear relationship between the ranges.

Assume that the controller at a given point of time demands a flow rate, u, of 43.5 L/d. What is the corresponding control signal value, v, in volts?

Problem 3.21 *PWM*

Given an electrical heater (resistor) controlled with PWM with period or cycle time period 0.5 s. When the PWM in the On state, the mains voltage is connected to heater which then delivers 1 kW. When the PWM is in the Off state, the heater delivers zero power.

Assume that at the PWM is operated so that the heater delivers 400 W (in average).

1. What is then the duty cycle, D?
2. And what is the On time, t_{on}?

Problem 3.22 *Pump control: Analog or PWM?*

Assume that a pump will used to generate a feed flow of biological material into a biogas reactor, and that there are the following two alternative ways to control the pump:

- Analog control: The pump is controlled with an analog milliampere signal. Unfortunately, it turns out to be practically difficult to obtain precise (repeatable) small feed flow rates.
- Approximately analog control with PWM with period time 60 s, which is a very short time compared to the slow response in biogas production in the biogas reactor. The biogas reactor will therefore hardly "feel" that it is actually receiving the feed flow in the form of flow pulses, but instead feel an approximately analog (smooth) flow.

The pump must be calibrated before it can be used. With analog control, it is necessary to calibrate at, say, 5 points within the operating flow rate range of the pump.

1. How many operating points is it necessary to calibrate the pump for with PWM?
2. For the pump, would you recommend analog control or PWM?

3.8 *Solutions to problems for Chapter 3*

Solution to Problem 3.1

1. PC-based control and monitoring system with a standard PC
2. PLC
3. DCS/SCADA
4. Process controller
5. PAC
6. DCS/DCS

Solution to Problem 3.2

From (3.4) we get

$$T = \frac{\frac{R}{R_0} - 1}{a} = \frac{\frac{161.4\ \Omega}{100\ \Omega} - 1}{3.9083 \cdot 10^{-3\circ}\mathrm{C}^{-1}} = 157.1^{\circ}\mathrm{C} \tag{3.77}$$

Solution to Problem 3.3

A resistance thermometer, because it is (far) more accurate than a thermocouple thermometer in the pertinent temperature range.

Solution to Problem 3.4

$$h = \frac{p}{\rho g} - h_0 = \frac{0.1 \cdot 100000\ \mathrm{N/m^2}}{850\ \mathrm{kg/m^3} \cdot 9.81\ \mathrm{m/s^2}} - 0.5\ \mathrm{m} = 0.70\ \mathrm{m} \tag{3.78}$$

Solution to Problem 3.5

1. bar is a unit of absolute pressure, i.e. zero pressure (as in vacuum) is the pressure reference. 1 bar (approx.) is the atmospheric pressure. So, $p = 1.513$ bar.
2. barg stands for “bar gauge” where gauge represents any measuring sensor (e.g. a Bourdon tube) which measures pressure with atmospheric pressure as pressure reference. So, $p = 0.5$ barg.
3. 1 bar is per definition equal to 100000 Pa. So, $p = 151300$ Pa.
4. mmH_2O (mm water column) is the pertinent water height in mm. The hydrostatic pressure is

$$p = \rho g h\ [\mathrm{Pa}] \tag{3.79}$$

where $\rho = 1000$ kg/m^3, $g = 9.81$ m/s^2 and h is in meters. We get

$$h\ [\text{mm}] = \frac{p}{\rho g} = \frac{151300\ \text{Pa}}{1000\ \text{kg/m}^3 \cdot 9,81\ \text{m/s}^2} \cdot 1000 = 15423\ \text{mm} \tag{3.80}$$

So, $p = 15423$ mmH$_2$O.

Solution to Problem 3.6

1. Ultrasound.
2. Coriolis.
3. Orifice with dp cell; Thermal; Ultrasound; Magnetic; Vortex.
4. Coriolis; Thermal.
5. Coriolis.

Solution to Problem 3.7

The methane gas flow rate is 0.73·251 L/d = 183 L/d.

Solution to Problem 3.8

1. The resolution is

$$r = \frac{360\ \text{deg}}{1024\ \text{pulse}} \tag{3.81}$$

The relationship between the rotation b [m] and the angle a [deg] is given by

$$b = a\frac{\pi}{180}L \tag{3.82}$$

which gives

$$\frac{b}{a}\frac{\text{m}}{\text{deg}} = \frac{\pi}{180\ \text{deg}}L\ \text{m}$$

The resolution r_d then becomes

$$r_d = r\frac{b}{a} = r\frac{\pi}{180^o}L = \frac{360\ \text{deg}}{1024\ \text{pulse}} \cdot \frac{\pi}{180\ \text{deg}} \cdot 1\ \text{m} = 0.0061359\ \frac{\text{m}}{\text{pulse}} = 6.1359\ \frac{\text{mm}}{\text{pulse}} \tag{3.83}$$

2. A gear can provide better resolution.

Solution to Problem 3.9

From (3.21) we get

$$v(t_1) \approx \frac{s(t_1) - s(t_0)}{T_s} = \frac{1.238\ \text{m} - 1.235\ \text{m}}{0.05\ \text{s}} = 0.06\ \text{m/s} \tag{3.84}$$

Solution to Problem 3.10

- **Temperature**:
  - *Thermistor*: A semiconductor component that has a temperature-sensitive resistance value. The resistance decreases with temperature. A measurement of the resistance indicates the temperature.
  - *Bimetal*: Two different metals with different temperature-dependent expansions are assembled (e.g. welded) together. The sensor will be bent more towards one of the sides as the temperature increases, or decreases. A measurement of the bend indicates the temperature.
  - *Pyrometer*: The higher the temperature in an object, the greater the heat radiation from that object. The pyrometer determines the temperature based on a measurement of the radiation.
- **Pressure**:
  - *Bourdon tube*: The expansion of a circular tube filled with gas or liquid varies with the pressure. A pointer connected to the tube indicates the pressure.
  - *Manometer*: The liquid level in each of the two tube legs of a U-shaped tube varies with the pressure difference in the tube legs. The difference between the level measurement of each of the levels indicates the pressure difference.
  - *Piezoelectric sensor*: In certain crystals, such as quartz, an electrical charge is generated that varies with the pressure the crystal is exposed to. (Piezoelectric means "electrical charge generated by pressure".) A voltage measurement indicates the pressure.
- **Level**:
  - *Buoyancy*: The buoyancy force of a partially submerged fixed body varies with the liquid level. A measurement of this force indicates the liquid level.
  - *Radioactive radiation*: An emitter on one side of a liquid tank emits radioactive radiation. The radiation detected by a rod-shaped receiver on the other side of the tank indicates the level of the liquid in the tank.
  - *Weight*: The total weight of a tank with material is measured. The material weight is equal to the measured total weight minus the weight of the tank itself. The material weight indicates the level of the material.
- **Liquid or gas flow rate**:
  - *Turbine*: A (small) turbine is mounted inside the pipe where the gas or liquid flows. A measurement of the rotational speed of the turbine indicates the flow rate.
  - *Rotameter*: A free-floating plug is placed in a vertical section of the pipe where the liquid to be read flows. The vertical position of the plug increases if the fluid flow rate increases. A meassurement of the position indicates the flow rate.
  - *Clock*: The time it takes to fill up a chamber of known volume is detected. When the chamber is full it is automatically emptied, and a new filling starts. The volumetric flow rate is calculated as the chamber volume divided by the time between each time the chamber is emptied.

- **Position**:
  - *Resolver*: The position of a shaft is measured from the induced current/voltage in electric coils which have a relative angular position depending on the angular rotation of the shaft.
  - *Potentiometer*: A variable resistor – a potensiometer – is fixed to the body whose position is to be measured. The resistance varies with the position of the body. The resistance can be detected with a small electrical circuit consisting of interconnected resistors called a voltage divider. The output voltage of the voltage divider indicates the position of the body.

Solution to Problem 3.11

From (3.23) we get

$$R = \frac{U}{I} = \frac{10\ \text{V}}{20\ \text{mA}} = 500\ \Omega \tag{3.85}$$

Solution to Problem 3.12

1. Resolution in volt:

$$R = \frac{y_{a_{\max}} - y_{a_{\min}}}{2^n - 1} = \frac{10\ \text{V} - 0\ \text{V}}{2^{12} - 1} = 0.0024\ \text{V} = 2.4\ \text{mV} \tag{3.86}$$

2. Resolution in meter:

$$R = \frac{5\ \text{m} - 0,5\ \text{m}}{2^{12} - 1} = 0.0011\ \text{m} = 1.1\ \text{mm} \tag{3.87}$$

3. Relative resolution (12 bits ADC):

$$R_{\text{rel}} = \frac{1}{2^{12} - 1} = 2.44 \cdot 10^{-4} \tag{3.88}$$

4. Relative resolution (16 bits ADC):

$$R_{\text{rel}} = \frac{1}{2^{16} - 1} = 1.53 \cdot 10^{-5} \tag{3.89}$$

Solution to Problem 3.13

1.

   - Upper process value: 15 m.
   - Lower process value: 5 m.
   - Zero: 5 m.
   - Span: 15 m − 5 m = 10 m.

2. Measurement characteristic parameters:

$$a = \frac{p_2 - p_1}{m_2 - m_1} = \frac{15 \text{ m} - 5 \text{ m}}{20 \text{ mA} - 4 \text{ mA}} = \frac{10}{16} \frac{\text{m}}{\text{mA}} = 0.625 \frac{\text{m}}{\text{mA}} \tag{3.90}$$

$$b = p_1 - a \cdot m_1 = 5 \text{ m} - 0.625 \frac{\text{m}}{\text{mA}} \cdot 4 \text{ mA} = 2.5 \text{ m} \tag{3.91}$$

3. With $m = 12$ mA:

$$p = a \cdot m + b = 0.625 \frac{\text{m}}{\text{mA}} \cdot 12 \text{ mA} + 2.5 \text{ m} = 10 \text{ m} \tag{3.92}$$

Solution to Problem 3.14

FS is 10 ml_n/min. The accuracy in ml_n/min then becomes ±1 % of 10 ml_n/min which is ±0.1, which is the measurement error you have to assume for any measurement reading.

Solution to Problem 3.15

The filter time constant of the time constant filter should be set to 2 s (the same as for the MA filter).

Solution to Problem 3.16

You should keep t_f unchanged, since the dynamic behaviour of the filter mainly depend on t_f. In other words, with unchanged t_f, the two filters will behave approximately the same.

Solution to Problem 3.17

In (3.49): We set $y_{m,k} = Y_m$. Assuming a the static response, both $y_{mf,k} = Y_{mf}$ and $y_{mf,k-1} = Y_{mf}$. This gives

$$Y_{mf} = (1-a)Y_{mf} + aY_m \tag{3.93}$$

which, fortunately, gives

$$Y_{mf} = Y_m \tag{3.94}$$

i.e., the static response is equal to the constant input.

Solution to Problem 3.18

The valve must be able to deliver a flow rate equal to $Q = 15$ L/min + 50 % of 15 L/min = 22.5 L/min. Oil has a relative density of $G = 0.85$. The valve equation (3.54) is solved with respect to the valve constant to give

$$K_v = \frac{Q}{\sqrt{\frac{p_v}{G}}} = \frac{22.5\ [\mathrm{L/min}]}{\sqrt{\frac{0.5\ \mathrm{bar}}{0.85}}} = \frac{22.5\ [0.001\ \mathrm{m}^3/(\mathrm{h}/60)]}{\sqrt{\frac{0.5\ \mathrm{bar}}{0.85}}} = 1.76\ \mathrm{m}^3/\mathrm{h} \tag{3.95}$$

Solution to Problem 3.19

From (3.64) we get

$$R = \frac{{U_{\mathrm{eff}}}^2}{P_{\mathrm{average}}} = \frac{(226\ \mathrm{V})^2}{1\ \mathrm{kW} = 1000\ \mathrm{W}} = 51.1\ \Omega \tag{3.96}$$

Solution to Problem 3.20

(3.67) becomes

$$a = \frac{v_2 - v_1}{u_2 - u_1} = \frac{5\ \mathrm{V} - 0\ \mathrm{V}}{80\ \mathrm{L/d} - 0\ \mathrm{L/d}} = 0.0625\ \frac{\mathrm{V}}{\mathrm{L/d}} \tag{3.97}$$

(3.68) becomes

$$b = v_1 - a \cdot u_1 = 0\ \mathrm{V} - 0.0625\ \frac{\mathrm{V}}{\mathrm{L/d}} \cdot 0\ \frac{\mathrm{L}}{\mathrm{d}} = 0\ \mathrm{V} \tag{3.98}$$

If $u = 43.5$ L/d, (3.66) gives

$$v = a \cdot u + b = 0.0625\ \frac{\mathrm{V}}{\mathrm{L/d}} \cdot 43.5\ \mathrm{L/d} + 0\ \mathrm{V} = 2.72\ \mathrm{V} \tag{3.99}$$

Solution to Problem 3.21

1. Duty cycle:

$$D = \frac{u_{\mathrm{mean}}}{U_{\mathrm{on}}} = \frac{400\ \mathrm{W}}{1000\ \mathrm{W}} = 0.4 = 40\ \% \tag{3.100}$$

2. On-time:

$$t_{\mathrm{on}} = Dt_p = 0.4 \cdot 0.5\ \mathrm{s} = 0.2\ \mathrm{s} \tag{3.101}$$

Solution to Problem 3.22

1. One operating point!
2. PWM. Reasons: (1) It is sufficient to calibrate the pump at only one operating point, namely the feed flow rate when the pump is in the On state. (2) With PWM, the pump may deliver more precise small feed flow rates (in average) than with analog control.

Part III

MODELING AND SIMULATION

Chapter 4

Mechanistic modeling of dynamic systems

4.1 Introduction

This chapter describes basic principles of mathematical modeling of dynamic systems. A mathematical model is the set of equations which describes the behavior of the system. The chapter focuses on how to develop dynamic models, and you will see that the models are differential equations. The differential equations can be represented on various forms, for example state space models (Ch. 6), mathematical block diagrams (Ch. 5), or transfer functions (Ch. 8). One very important application of models is creating simulators (Ch. 7).

Unfortunately we can never make a completely precise model of a physical system. There are always phenomena which we will not be able to model. Thus, there will always be model errors or model uncertainties. But even if a model describes just a part of the reality it can be very useful for analysis and design – if it describes the dominating dynamic properties of the system. A saying is "All models are wrong, but some are useful."

This chapter describes modeling based on physical principles. Such modeling has alternative names:

- Mechanistic modeling
- First principles modeling
- White-box modeling, as opposed to black-box modeling where the models are just mathematical expressions capable to represent the dynamic relation between the output and input of the system, like transfer functions.

4.2 What is a dynamic system?

Dynamic means "which has to do with the movement and change". Dynamic systems are systems where the variables can vary or develop with time. We say that dynamic systems have dynamic responses. Figure 4.1

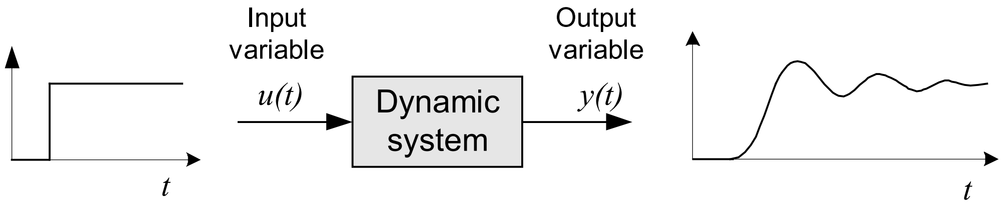


Figure 4.1: Dynamic systems are systems where the variables can vary or develop as functions of time.

gives an illustration. The figure shows a block diagram of a dynamic system. The input variable is here a step function, and the response in the output variable is dynamic since it changes with time. In general, dynamic systems may have more than one input variable and more than one output variable.

Here are some examples of dynamic systems:

- **A liquid tank**.
  - Input (variable): Inflow.
  - Output (variable): Level.
- **A motor**.
  - Input: Motor control voltage.
  - Output: Speed.
- **A heated water tank**.
  - *Input*: Supplied heat.
  - *Output*: Temperature in the water in the tank.
- **A robot manipulator**.
  - *Input*: Control signal to motor.
  - *Output*: Arm position.
- **A ship**.
  - *Input*: Thruster force.
  - *Output*: Ship position.
- **A signal filter**:

  - *Input*: Filter input to be filtered (smoothed).
  - *Output*: Filter output signal.

- **A control system for a physical process**:
  - *Input*: Reference.
  - *Output*: Process output variable.

4.3 A procedure for mathematical modeling

Below is described a procedure for developing dynamic mathematical models for physical systems:

1. **Define systems boundaries.** All physical systems works in interaction with other systems. Therefore it is necessary to define the boundaries of the system before we can begin developing a mathematical model for the system, but in most cases defining the boundaries is done quite naturally.

2. **Make simplifying assumptions.** One example is to assume that the temperature in a tank is the same everywhere in the tank, that is, there are homogeneous conditions in the tank.

3. **Use the Balance Law for the physical balances in the system.** See Figure 4.2.

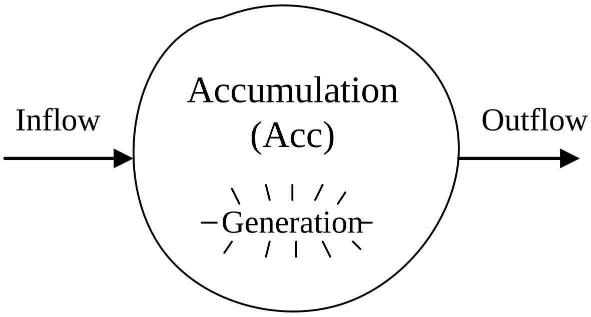


Figure 4.2: Illustration of the Balance Law.

In the figure, “Accumulation” is a general term. Specifically, it can be accumulated mass, mole, energy, momentum, or electric charge in a physical system. “Inflow” represents possibly several inflows, so it is total inflow. “Outflow” represents possibly several outflows, so it is total outflow. “Generation” can be e.g. material generated by chemical reactions, or generated energy in an exothermal reactor.

The accumulation in the system at time t is given by the following integral, where the term Acc is used for short:

$$\mathrm{Acc}(t) = \mathrm{Acc}(0) + \int_0^t \mathrm{Acc'}(\theta)\, d\theta \tag{4.1}$$

where $t = 0$ is the initial time. The integrand, Acc', is the rate of change, or time derivative, of the Acc, and is given by the following differential equation, which is often termed the *Balance Law* of the system:

$$\text{Acc}' = \text{Inflow} - \text{Outflow} + \text{Generation} \tag{4.2}$$

Usually, only the (4.2) is said to be the *mathematical model* of the system, although (4.1) is also essential to calculate the Accumulation.

4. **Draw an overall block diagram showing inputs, outputs and parameters.** A block diagram makes the model appear more clearly. Figure 4.3 shows a general overall block diagram. In the figure, the single lines can represent a number of variables. The output variables are typically the accumulations, or accumulations multiplied by some constants. The input variables are of two kinds: (1) *Manipulating (adjustable) variables* which you can use to manipulate or control the system (like power from a heater), and (2) *environmental variables* which you can not manipulate (like environmental temperature). In the context of control systems, environmental variables are often denoted disturbance variables.

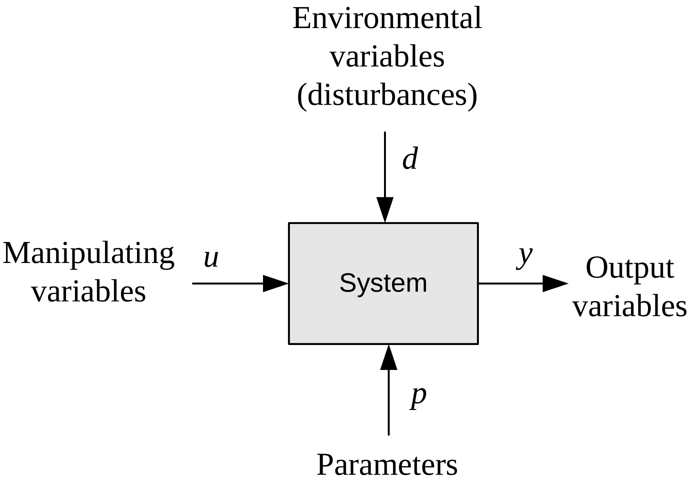


Figure 4.3: Overall block diagram.

   Mathematically, the input variables are *independent variables*, and the output variables are *dependent variables*.

   The parameters of the model are quantities in the model which typically (but not necessarily) have constant values, like liquid density and spring constant. If a parameter have a varying value, it may alternatively be regarded as an environmental variable.

5. **Present the model on a proper form.** The choice of model form depends on the application of the model. The most common model forms are:

   (a) *Differential equation* with the highest order of the time derivative alone on the left side of the equation. This is the model form used in examples and problems in the present chapter.

(b) *State space models* (Ch. 6.2), which are just the differential equations written with the time derivates alone on the left side of the equation. (If the model is a differential equation of order two or higher, the state space model is an equivalent set of first order differential equation.)

(c) *Mathematical block diagrams* (Ch. 5).

(d) *Transfer functions* (Ch. 8) which applies only for linear differential equation models.

The following sections contain several examples of mathematical modeling. In those examples, items 1 and 2 above are applied more or less implicitly.

Comments on notation

For simplicity, I write the time-varying variables without the time as argument. For example, I write m instead of $m(t)$. However, in cases where time delay is involved, it is important to show the time argument explicitly, for example $u(t - t_d)$ where t_d is the time delay. Although there is not an example with time delay in this chapter, there is one in Appendix 39.2 where the model of a wood chips tank including a conveyor belt is presented. The transportation time on the belt is a time delay.

4.4 Mathematical modeling of material systems

In a system where the mass may vary, mass is the “accumulation” in the (4.2) which now becomes a *mass balance*:

$$m' = \sum_i F_i \tag{4.3}$$

where m [kg] is the mass, and F_i [kg/s] is mass inflow (no. i).

Example 4.1 *Mass balance of a liquid tank*

Figure 4.4 shows a liquid tank with inflow and outflow. Assume that the inflow can be manipulated with e.g. a pump. The outflow is due to some consumption which here is regarded as an environmental variable.

The tank has straight, vertical walls. F_{in} and F_{out} are mass flows. h is liquid level. V is liquid volume. m is mass. A is cross sectional area. The density of the liquid, ρ, is assumed the same in the inlet, the outlet, and the tank.

The mass balance for the mass in the tank is:

$$m' = F_{\text{in}} - F_{\text{out}} \tag{4.4}$$

which is a differential equation for m. An additional condition for the differential equation is $m \geq 0$.

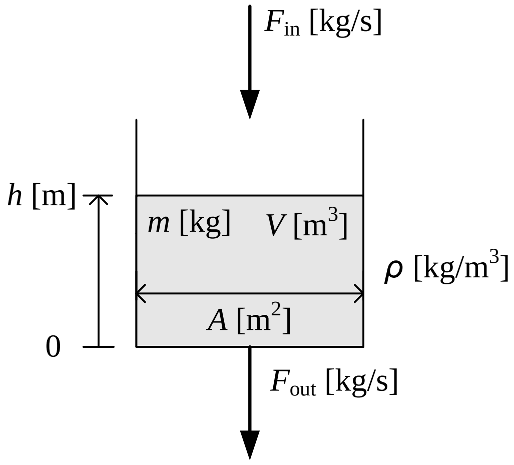


Figure 4.4: Example 4.1: Liquid tank.

Now, m is the integral of m':

$$m(t) = m(0) + \int_0^t m'(\theta)\, d\theta \tag{4.5}$$

with the condition $0 \leq h \leq m_{\text{max}}$ where m_{max} is the maximum liquid mass in the tank.

A model of the level

Maybe you are more interested in how level h varies than how m varies? The relation between h and m is a given by

$$m = \rho V = \rho A h \tag{4.6}$$

We insert this into the mass balance (4.4), which then becomes

$$m' = (\rho V)' = (\rho A h)' = \rho A h' = F_{\text{in}} - F_{\text{out}} \tag{4.7}$$

where parameters ρ and A have been moved outside the derivation (they are assumed constant).

By dividing by ρA we get the following differential equation of h:

$$h' = \frac{1}{\rho A}\left(F_{\text{in}} - F_{\text{out}}\right) \tag{4.8}$$

The level h is the integral of h':

$$h(t) = h(0) + \int_0^t h'(\theta)\, d\theta \tag{4.9}$$

with the condition $0 \leq h \leq h_{\text{max}}$.

Figure 4.5 shows an overall block diagram of the model (4.8). Note that F_{out} is an *input* variable despite it represents a physical *outflow* from the tank!

[End of Example 4.1]

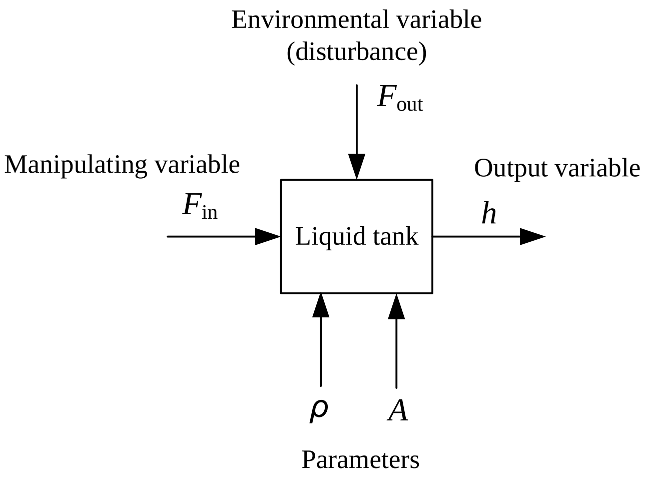


Figure 4.5: Example 4.1: Overall block diagram of the liquid tank.

4.5 Mathematical modeling of thermal systems

Mathematical modeling of thermal systems is based on the to set up energy balances. The term *energy* covers temperature-dependent energy, which we can call thermal energy, and kinetic and potential energy. In general we must assume that there is a transformation from one energy form to another within a given system. For example, kinetic energy can be transformed to thermal energy via friction. For many thermal systems we can assume that the energy consists of only thermal energy and we can neglect the transformation from kinetic and potential energy to thermal energy.

For thermal systems, the "accumulation" in the Balance Law (4.2) is thermal energy. Thus, the Balance Law becomes an *energy balance*:

$$E' = \sum_i Q_i \tag{4.10}$$

where E [J] is the thermal energy, and Q_i [J/s] is energy inflow no. i. The energy E is often assumed to be proportional to the temperature and the mass (or volume):

$$E = cmT = c\rho VT = CT \tag{4.11}$$

where T [K] is the temperature, c [J/(kg K)] is specific heat capacity, m [kg] is mass, V [m^3] volume, ρ [kg/m^3] is density, C [J/K] is total heat capacity.

Example 4.2 *Mathematical modeling of a heated liquid tank*

Figure 4.6 shows a liquid tank with continuous liquid inflow and outflow. There is heat transfer with the environment through the walls. The liquid receives power through a heating element. P is power from the heating element. T is temperature in the tank and in the outlet flow. T_{in} is the temperature in the inlet flow. T_a is the ambient temperature. F

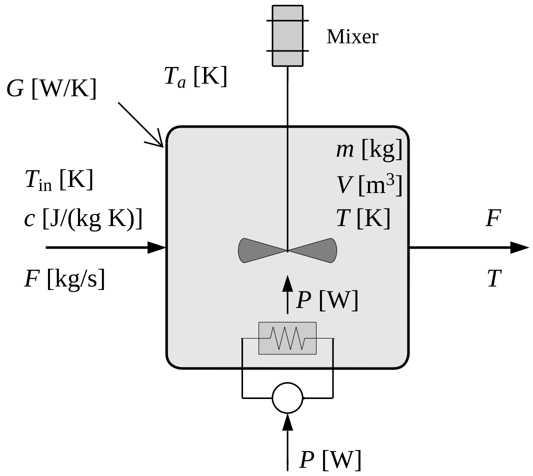


Figure 4.6: Example 4.2: Heated liquid tank.

is mass flow. m is mass of liquid (constant). c is specific heat capacity. G is heat transfer coefficient.

We will now set up an energy balance for the liquid in the tank to find the differential equation which describes the temperature $T(t)$. We will then make the following assumptions:

- The temperature in the liquid in the tank is homogeneous (due to the stirring machine).
- The inflow and in the outflow are equal, and the tank is filled by liquid.
- There is no storage of thermal energy in the heating element itself. This means that all of the supplied power to the heating element is supplied (immediately) to the liquid. (Thus, we do not write an energy balance for the heating element.)

The energy balance is based on the following energy transports (power):

1. Power from the heating element:
$$P = Q_1 \tag{4.12}$$
2. Power from the inflow:
$$cFT_{\text{in}} = Q_2 \tag{4.13}$$
3. Power removed via the outflow:
$$-cFT = Q_3 \tag{4.14}$$
4. Power via heat transfer from (or to) the environment:
$$G\left(T_e - T\right) = Q_4 \tag{4.15}$$

The energy balance is

$$E' = Q_1 + Q_2 + Q_3 + Q_4 \tag{4.16}$$

where the energy is given by

$$E = cmT$$

The energy balance can then be written as

$$(cmT)' = P + cFT_{\text{in}} - cFT + G\,(T_a - T) \tag{4.17}$$

If we assume that c and m are constant, we can move cm outside the derivative term. Furthermore, we can combine the terms on the right side. The result is

$$cmT' = P + cF\,(T_{\text{in}} - T) + G\,(T_a - T) \tag{4.18}$$

or:

$$T' = \frac{1}{cm}\left[P + cF\,(T_{\text{in}} - T) + G\,(T_a - T)\right] \tag{4.19}$$

The temperature T is the integral of T':

$$T(t) = T(0) + \int_0^t T'(\theta)\,d\theta \tag{4.20}$$

Figure 4.7 shows an overall block diagram of the model (4.19).

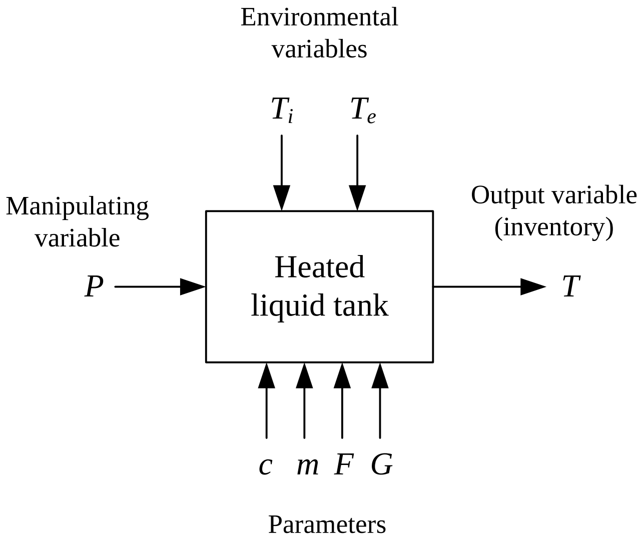


Figure 4.7: Example 4.2: Overall block diagram of heated tank.

[End of Example 4.2]

4.6 Mathematical modeling of kinetic systems

4.6.1 Systems with linear motion

For kinetic systems in the form of a body with linear motion (we will soon study rotational motion), the "accumulation" term in the Balance Law (4.2) is momentum. Thus, the Balance Law becomes a *momentum balance*, which is often denoted *force balance*:

$$I' = (mv)' = \sum_i F_i \tag{4.21}$$

where I [(kg·m/s)/s] is the momentum (mass times speed), and F_i is force (no. i). I is

$$I = mv = my' \tag{4.22}$$

where m [kg] is mass, v [m/s] is speed, and y [m] is position.

If m is constant, m can be moved outside the derivative term in (4.21), which then becomes

$$mv' = my'' = ma = \sum_i F_i \tag{4.23}$$

where $v' = y'' = a$ is acceleration. (4.23) is the well-known Newton's second law (the sum of forces is equal to mass times acceleration).

Often the mass m is constant. Then (4.23) can be used for mathematical modeling. But if m is time-varying, (4.21) must be used. One example of a system with *varying* mass is a conveyor belt where the mass on the belt is varying.

Example 4.3 *Mathematical modeling of a mass-spring-damper system*

Figure 4.8 shows a mass-spring-damper-system.[1] y is position. F is applied force. d is damping constant. K is spring constant. It is assumed that the damping force F_d is proportional to the speed:

$$F_d = Dy' \tag{4.24}$$

and that the spring force F_s is proportional to the position of the mass:

$$F_s = Ky \tag{4.25}$$

The spring force is assumed to be zero when y is zero. Force balance (Newton's 2. Law) yields[2]

$$\begin{aligned} my'' &= F - F_d - F_s \\ &= F - D\dot{y} - Ky \end{aligned} \tag{4.26}$$

[1] The mass-spring-damper system is not a typical system found in process control. It is chosen here because it is easy to develop a mathematical model using well known physical principles, here the Newton's second law. Examples of more relevance to process control are described in Chapter 4.

[2] Double-dot represents second order time-derivative: $\ddot{y}(t) \equiv d^2y(t)/dt^2$

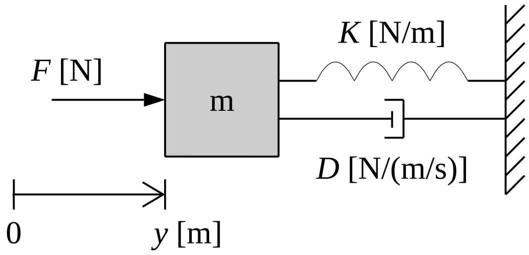


Figure 4.8: Mass-spring-damper.

which is a second order differential equation, which we can write as

$$y'' = (u - D\dot{y} - Ky)/m \tag{4.27}$$

Speed y' is the integral of acceleration y'':

$$y'(t) = y'(0) + \int_0^t y''(\theta)\, d\theta \tag{4.28}$$

And position y is the integral of speed y':

$$y(t) = y(0) + \int_0^t y'(\theta)\, d\theta \tag{4.29}$$

In other words, position is double-integral of the acceleration.

Figure 4.9 shows an overall block diagram for the model (4.26). The applied force u is the input variable, and the position y is the output variable.

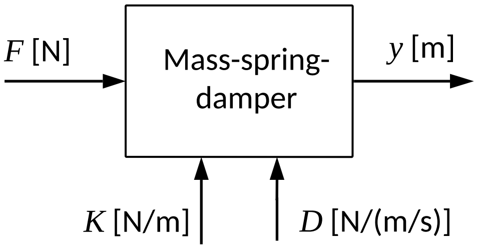


Figure 4.9: Overall block diagram of mass-spring-damper system.

[End of Example 4.3]

4.6.2 Systems with rotational motion

4.6.2.1 Momentum balance

Systems with rotational motion can be modelled in the same way as systems with linear motion (see above), but we must use *momentum balance*, which is often denoted *torque balance* for rotational systems:

$$S' = (J\omega)' = \sum_i T_i \tag{4.30}$$

Here, S [Nms] is momentum, J [kgm^2] is inertia, ω [rad/s] is rotational speed, and T_i is torque (no. i). If J is constant, (4.30) can be written

$$J\omega' = Ja'' = \sum_i T_i \tag{4.31}$$

where $\omega' = a''$ is angular acceleration, and a [rad] is angular position.

(4.31) can be written as:

$$a'' = \left(\sum_i T_i\right)/J \tag{4.32}$$

Angular speed a' is the integral of angular acceleration a'':

$$a'(t) = a'(0) + \int_0^t a''(\theta)\, d\theta \tag{4.33}$$

And angular position a is the integral of angular speed a':

$$a(t) = a(0) + \int_0^t a'(\theta)\, d\theta \tag{4.34}$$

In other words, angular position is double-integral of the angular acceleration.

4.6.2.2 Relations between rotational and linear motion

In mathematical modeling of mechanical systems which consists of a combination of rotational and linear systems, the following relations are useful: Torque T is force F times arm L:

$$T = FL \tag{4.35}$$

Arc b is angle a (in radians) times radius r:

$$b = ar \tag{4.36}$$

4.6.2.3 Coupled mechanical systems

Mechanical systems often consist of coupled (sub)systems. Each system can have linear and/or rotational motion. Some examples: (1) A robot manipulator where the arms are coupled. (2) A traverse crane where a wagon moves a pending load. (3) A motor which moves a load with linear motion, as in a lathe machine.

A procedure for mathematical modeling of such coupled systems is as follows:

1. The force or torque balance is put up for each of the (sub)systems, and internal forces and torques acting between the systems are defined.
2. The final model is derived by eliminating the internal forces and torques.

This procedure is demonstrated in Example 4.4. An alternative way of modeling coupled systems is to use *Lagrange mechanics* where the model (the equations of motion) are derived from an expression which contains kinetic and potential energy for the whole system (this method is not described here).

Example 4.4 *Mathematical modeling of coupled rotational and linear motion systems*

Figure 4.10 shows an electric motor (which can be a current-controlled DC-motor) which moves a load with linear motion via a gear and a rod.

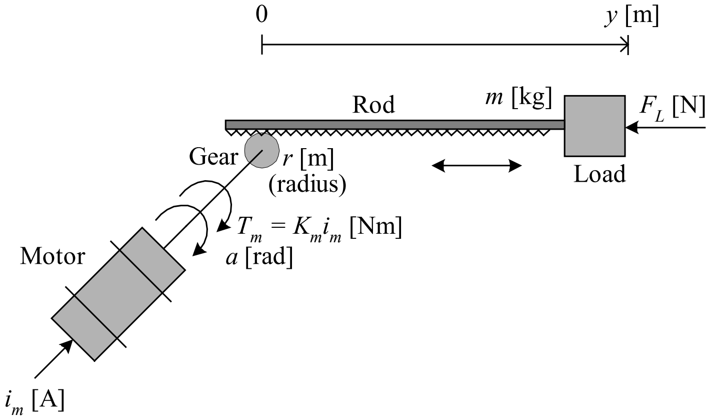


Figure 4.10: Example 4.4: Motor moving a linear load via a gear and a rod.

We set up a torque balance for the rotational part of the system and a force balance for the linear part, and then combines the derived equations. We shall finally have model which expresses the position y of the tool as a function of the applied motor current i_m. F_L is a load force acting on the load by the environment. (For simplicity the time argument t is excluded in the expressions below.)

1. **Torque and force balance:** The torque balance for the motor becomes

$$Ja'' = K_m i_m - T_1 \tag{4.37}$$

where T_1 is the torque which acts on the motor from the rod and the load via the gear. The force balance for rod and load becomes

$$my'' = F_1 - F_L \tag{4.38}$$

where F_1 is the force which acts on the rod and the load from the motor via the gear. The relation between T_1 and F_1 is given by

$$T_1 = F_1 r \tag{4.39}$$

The relation between y and θ is given by

$$y = ar \tag{4.40}$$

which yields

$$a'' = \frac{y''}{r} \tag{4.41}$$

By setting (4.41) and (4.39) into (4.37), (4.37) can be written

$$J\frac{y''}{r} = K_m i_m - F_1 r \tag{4.42}$$

2. **Elimination of internal force:** By eliminating the internal force F_1 between (4.38) and (4.42), we get

$$\left(m + \frac{J}{r^2}\right) y'' = \frac{K_m}{r} i_m - F_L \tag{4.43}$$

or:

$$y'' = \left(\frac{K_m}{r} i_m - F_L\right) \Big/ \left(m + \frac{J}{r^2}\right) \tag{4.44}$$

which is a mathematical model for the coupled system.

Speed y' is the integral of acceleration y'':

$$y'(t) = y'(0) + \int_0^t y''(\theta)\, d\theta \tag{4.45}$$

And position y is the integral of speed y':

$$y(t) = y(0) + \int_0^t y'(\theta)\, d\theta \tag{4.46}$$

Figure 4.11 shows an overall block diagram for the model (4.43). i_m and F_L are input variables, and y is the output variable.

[End of Example 4.4]

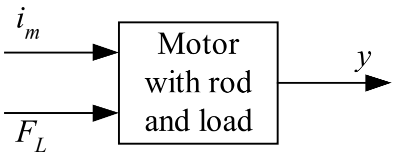


Figure 4.11: Overall block diagram of motor with rod and load.

4.7 Mathematical modeling of electric systems

This section gives a summary of some fundamental formulas for electric systems which you will probably use in mathematical modeling of electric systems.

4.7.1 Kirchhoff's law

4.7.1.1 Kirchhoff's Current Law

See the left part of Figure 4.12. The sum of currents into a junction in an electric circuit is

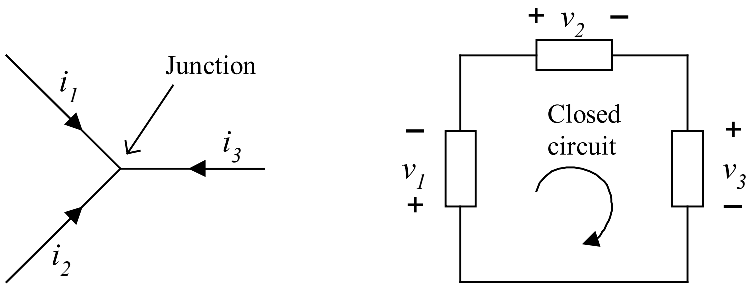


Figure 4.12: Kirchhoff's laws.

zero:

$$i_1 + i_2 + i_3 + \cdots = 0 \tag{4.47}$$

4.7.1.2 Kirchhoff's Voltage Law

: See the right part of Figure 4.12. The sum of voltage drops over the components on a closed electric loop is equal to zero:

$$v_1 + v_2 + v_3 + \cdots = 0 \tag{4.48}$$

4.7.2 Resulting resistance

Figure 4.13 shows series and parallel combination of resistors.

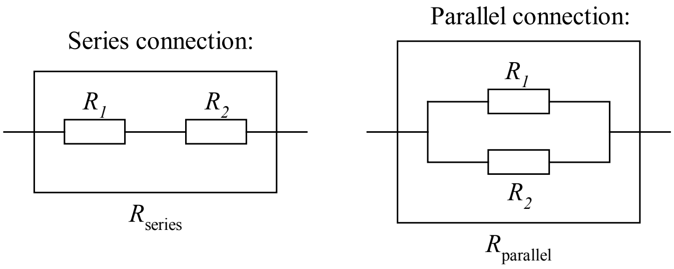


Figure 4.13: Series and parallel combination of resistors.

4.7.2.1 Resistors in series

Given two resistors R_1 and R_2 [Ω] in a series combination. The resulting resistance is

$$R_{\text{series}} = R_1 + R_2 \tag{4.49}$$

4.7.2.2 Resistors in parallel

Given two resistors R_1 and R_2 [Ω] in a parallel combination. The resulting resistance is

$$R_{\text{parallel}} = \frac{R_1 R_2}{R_1 + R_2} \tag{4.50}$$

4.7.3 Models of resistor, capacitor, and inductor

See Figure 4.14.

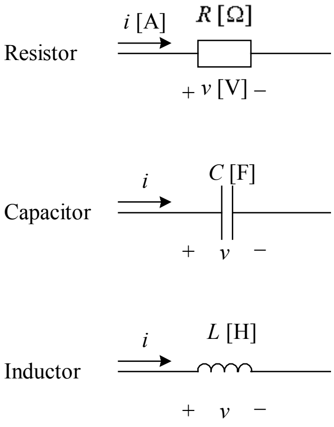


Figure 4.14: Resistor, capasitor and inductor.

Suppose that the current through a component is i [A] and that the corresponding voltage drop over the component v [V]. Current and voltage are then related as follows.

Resistor

$$v = Ri \quad \textbf{(Ohm's law)} \tag{4.51}$$

Capacitor

$$i = Cv' \tag{4.52}$$

Inductor

$$v = Li' \tag{4.53}$$

Example 4.5 *Mathematical modeling of an RC-circuit*

Figure 4.15 shows an RC-circuit (the circuit contains the resistor R and the capacitor C).

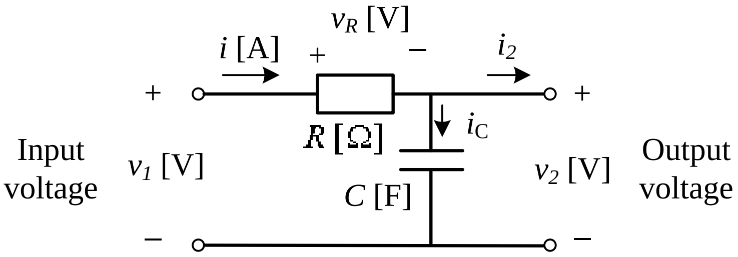


Figure 4.15: RC-circuit.

The RC-circuit is frequently used as an analog lowpass filter: Signals of *low* frequencies *passes* approximately unchanged through the filter, while signals of high frequencies are approximately filtered out (stopped).

We will now find a mathematical model relating v_{out} to v_{in}. First we apply the Kirchhoff's voltage law in the circuit which consists the input voltage terminals, the resistor, and the capacitor (we consider the voltage drops to be positive clockwise direction):

$$-v_{\text{in}} + v_R + v_{\text{out}} = 0 \tag{4.54}$$

(v_{out} equals the voltage drop over the capacitor.) In (4.54) v_R is given by

$$v_R = Ri \tag{4.55}$$

We assume that there is no current going through the output terminals. (This is a common assumption, and not unrealistic, since it it typical that the output terminals are connected to a subsequent circuit which has approximately infinite input impedance, causing the current into it to be approximately zero. An operational amplifier is an example of such a load-circuit.) Thus, jf. (4.52),

$$i = i_C = Cv'_{\text{out}} \tag{4.56}$$

The final model is achieved by using i as given by (4.56) in (4.55) and then using v_R as given by (4.55) for v_R in (4.54). The model becomes

$$RCv'_{\text{out}} = v_{\text{in}} - v_{\text{out}} \tag{4.57}$$

or:

$$v_{\text{out}}{}' = (v_{\text{in}} - v_{\text{out}}) \,/\, (RC) \tag{4.58}$$

v_{out} is is the integral of v'_{out}:

$$v_{\text{out}}(t) = v_{\text{out}}(0) + \int_0^t v_{\text{out}}(\theta)' \, d\theta \tag{4.59}$$

Figure 4.16 shows a block diagram for the model (4.57). v_{in} is the input variable, and v_{out} is the output variable.

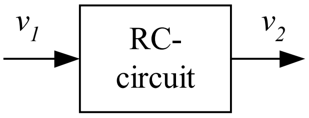


Figure 4.16: Overall block diagram of an RC-circuit.

[End of Example 4.5]

4.7.4 Power

4.7.4.1 Instantaneous power

When a current i flows through a resistor R, the instantaneous power delivered to the resistor is

$$P = ui \tag{4.60}$$

where $u = Ri$ is the voltage drop across the resistor.

4.7.4.2 Mean power

When an alternating (sinusoidal) current of amplitude I flows through a resistor R (for example a heating element), the mean or average value of the power delivered to the resistor is

$$\bar{P} = \frac{1}{2}UI = \frac{1}{2}RI^2 = \frac{1}{2}\frac{U^2}{R} \tag{4.61}$$

where U is the amplitude of the alternating voltage drop across the resistor. (4.61) is independent of the frequency.

4.8 *Problems for Chapter 4*

Problem 4.1 *Mass balance with volumetric flows*

In Example 4.1 the inflow and outflow are mass flows. Now, assume that the flows are instead volumetric flows, $q_{\text{in}}[\text{m}^3/\text{s}]$ and $q_{\text{out}}[\text{m}^3/\text{s}]$, respectively. Derive the differential equation of h under this assumption.

Problem 4.2 *Mole balance*

Figure 4.17 shows a stirred blending tank where the material A is fed into a tank for blending with a raw material. The symbols in Figure 4.17 are as follows: V is the liquid volume in the tank. q is the volumetric inflow of the raw material. q is also the volumetric outflow. c_A is the mole density or concentration of material A in the tank. w_A is the mole flow of material A.

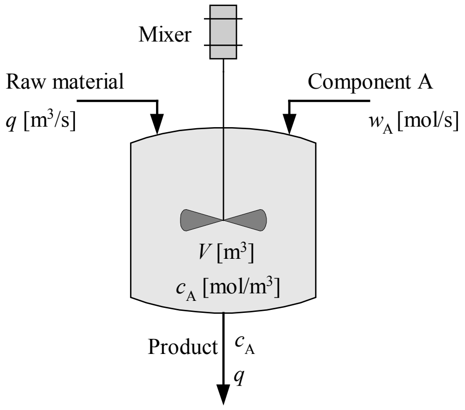


Figure 4.17: Problem 4.2: Blending tank.

Assumptions:

- The contents of the tank has constant volume.[3]
- The volumetric flow of material A is very small (negligible) compared to the volumetric flow of the raw material.
- There are homogeneous conditions (perfect stirring) in the tank.
- The raw material does not contain A.

1. Develop a mathematical model which expresses how the concentration c_A varies.
2. Draw an overall block diagram of the system.

[3] This can be accomplished with for example a level control system.

Problem 4.3 *Modeling of blending tank*

Figure 4.18 shows a tank with cold water inflow and heated (blended) water outflow. The tank is constantly full, and the volumetric flow is thus equal to the sum of the inflows.

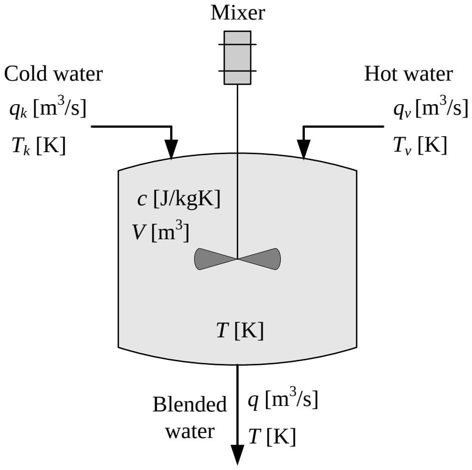


Figure 4.18: Tank with cold water inflow and heated (blended) water outflow.

Assume homogeneous conditions in the tank. Develop a mathematical model of the water temperature T in the tank.

Problem 4.4 *Modeling of a ship*

Figure 4.19 shows a ship.

In this problem we concentrate on the so-called surge (forward) direction, i.e., the movements in the other directions are disregarded. The wind acts on the ship with the force F_w. The absolute value of the hydrodynamic force F_h (force from water acting on the ship) is proportional to the square of the difference between the ship speed u and the water current speed u_c.[4] Assume that the proportionality constant is D (a positive number).

1. What is the mathematical relation between speed u and position y?
2. Develop a mathematical model of the ship expressing the motion (the position y) in the surge direction.
   Note: It is important to get the direction of the hydrodynamic force correct. Let us assume all speeds are positive in the positive surge direction (forwards). If the water current speed is larger than the ship speed, the hydrodynamic force acts on the ship in the forward direction. If the water current speed is smaller than the ship speed the hydrodynamic force acts on the ship in the backward direction. Your model must express this correctly.
3. Draw an input-output block diagram of the system. Assume that the ship position is the variable of particular interest.

[4] In the context of ship modeling, it is usual to use the symbol u for speed. In the control theory, however, u often represents the control signal, but control is not a topic in this problem.

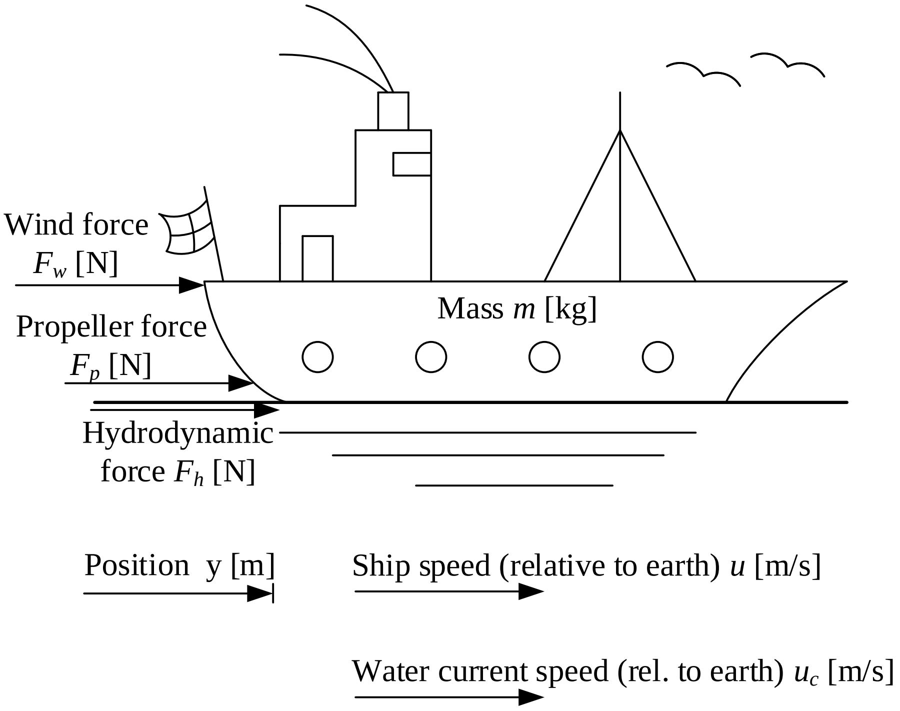


Figure 4.19: Ship.

Problem 4.5 *Modeling of a satellite*

Figure 4.20 shows a satellite with manoeuvering motors.

Develop a model of the angular motion of the satellite.

Problem 4.6 *Modeling of a pendulum*

Figure 4.21 shows a cart with the pendulum. A motor (in the cart) acts on the cart with a force F.[5]

You can use the following variables and parameters in the model to be derived in this problem:

- I – the moment of inertia of the pendulum about it's center of gravity. For the pendulum shown in Figure 1,

$$I = \frac{mL^2}{12} \tag{4.62}$$

- V and H – vertical and horizontal forces, respectively, in the pivot.

[5] This force can be manipulated by the controller to stabilize the pendulum in a standing position or in a hanging position at a specified position of the cart, but this problem is not about control. The system can be well controlled with model-based control, for example optimal control based on state-variable feedback (cf. e.g. Lecture notes on Modes, Estimation and Control, TechTeach/F. Haugen).

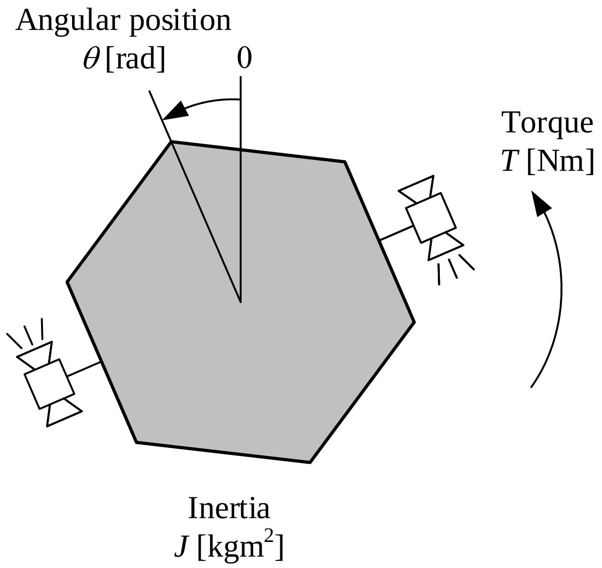


Figure 4.20: Satellite.

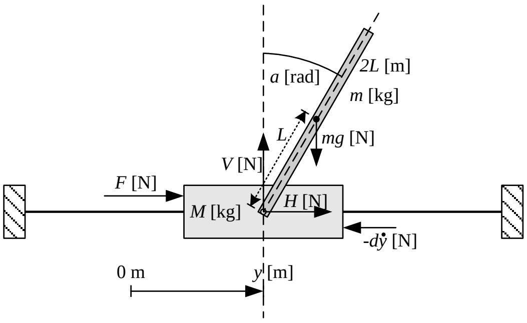


Figure 4.21: Pendulum.

- d – a damping coefficient.

Derive a mathematical model of the motion of the system based on the following principles:

1. Force balance (Newton's Second Law) applied to the horizontal movement of the center of gravity of the pendulum.
2. Force balance applied to the vertical movement of the center of gravity of the pendulum.
3. Torque balance (the rotational version of the Newton's Second Law applied to the center of gravity of the pendulum.
4. Force balance applied to the cart.

(When using the model for developing a simulator or design of a stabilizing controller, it will probably be necessary to eliminate the internal forces V and H, but this elimination is not a part of this problem. Hence, it is ok that the resulting model in this problem contains V and H.)

Problem 4.7 *Modeling of resistors*

Figure 4.22 shows a combination of resistors.

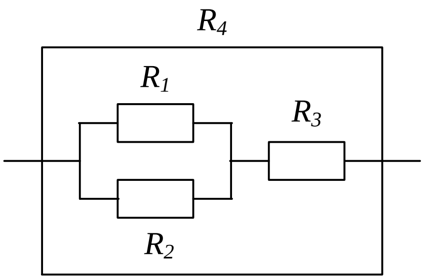


Figure 4.22: Combination of resistors.

What is the resulting resistance R_4?

Problem 4.8 *Calculation of resistance*

Given a lamp which receives $P = 100$ W mean (average) power when it is connected to the mains, which is an alternate voltage of amplitude $U = 220$ V. Calculate the lamp restistance R.

Problem 4.9 *Modeling of electric circuit (highpass filter)*

Figure 4.23 shows an (analog) highpass filter. (It attenuates low-frequent signals, while high-frequent signals pass through the filter.)

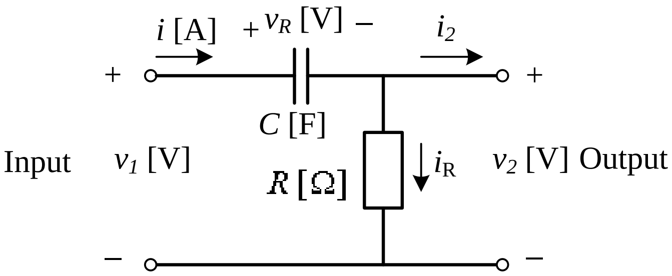


Figure 4.23: High-pass filter.

Find a mathematical model describing the behaviour of the output voltage v_2.

4.9 *Solutions to problems for Chapter 4*

Solution to Problem 4.1

With volumetric flows, the mass balance (4.4) becomes:

$$m' = (\rho A h)' = \rho A h' = \rho q_{\text{in}} - \rho q_{\text{out}} \tag{4.63}$$

which gives

$$h' = \frac{1}{A}\left(q_{\text{in}} - q_{\text{out}}\right) \tag{4.64}$$

$h(t)$ is given by the integral of h':

$$h(t) = h(0) + \int_0^t h'(\theta)\, d\theta \tag{4.65}$$

Solution to Problem 4.2

1. The "accumulation" is the total mole number is Vc_A. The Balance Law (4.2) is in terms of a mole balance:

   $$(Vc_A)' = w_A - c_A q \tag{4.66}$$

   A state space model of c_A:

   $$c_A' = \frac{1}{V}\left(w_A - c_A q\right) \tag{4.67}$$

   $c_A(t)$ is given by the integral of $c_A{}'$:

   $$c_A(t) = c_A(0) + \int_0^t c_A{}'(\theta)\, d\theta \tag{4.68}$$

2. Figure 4.24 shows an overall block diagram of the model (4.67). w_A and q are input variables, and c_A is the output variable.

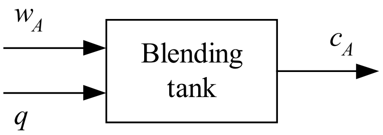


Figure 4.24: Problem 4.2: Overall block diagram for stirred blending tank.

Solution to Problem 4.3

Energy balance of the liquid in the tank:

$$(c\rho VT)' = c\rho VT' = c\rho q_k T_k + c\rho q_v T_v - c\rho q T \tag{4.69}$$

Cancelling ρ:

$$cVT' = cq_kT_k + cq_vT_v - cqT \tag{4.70}$$

Diving by cV gives a state space model of T:

$$T' = (cq_kT_k + cq_vT_v - cqT)/(cV) \tag{4.71}$$

Here, q is given by

$$q = q_k + q_v \tag{4.72}$$

$T(t)$ is given by the integral of T':

$$T(t) = T(0) + \int_0^t T'(\theta)\, d\theta \tag{4.73}$$

Solution to Problem 4.4

1. The relation between position y and speed u is

$$y' = u \tag{4.74}$$

2. Force balance:

$$\begin{aligned} mu' &= F_p + F_h + F_w && (4.75) \\ &= F_p - D|u - u_c|\,(u - u_c) + F_w && (4.76) \end{aligned}$$

(4.74) and (4.76) constitutes the model.

Alternatively, since

$$u' = y'' \tag{4.77}$$

the model can be expressed as

$$my'' = F_p - D|y' - u_c|\,(y' - u_c) + F_w \tag{4.78}$$

or:

$$y'' = \left(F_p - D|y' - u_c|\,(y' - u_c) + F_w\right)/m \tag{4.79}$$

Speed y' is the integral of acceleration y'':

$$y'(t) = y'(0) + \int_0^t y''(\theta)\, d\theta \tag{4.80}$$

And position y is the integral of speed y':

$$y(t) = y(0) + \int_0^t y'(\theta)\, d\theta \tag{4.81}$$

3. We can regard F_p, F_w and u_c as input variables, and y as the output variable. Figure 4.25 shows the block diagram.

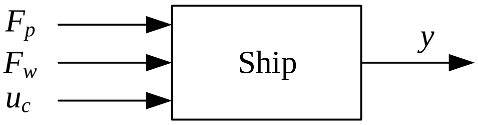


Figure 4.25: Overall block diagram of the ship model.

Solution to Problem 4.5

Torque balance:

$$J\theta'' = T \tag{4.82}$$

or:

$$\theta'' = T/J \tag{4.83}$$

Angular speed θ' is the integral of angular acceleration θ'':

$$\theta'(t) = \theta'(0) + \int_0^t \theta''(\theta)\, d\theta \tag{4.84}$$

And angular position θ is the integral of angular speed θ':

$$\theta(t) = \theta(0) + \int_0^t \theta'(\theta)\, d\theta \tag{4.85}$$

Solution to Problem 4.6

1. Force balance (Newton's Second Law) applied to the horizontal movement of the center of gravity of the pendulum.

$$m\,(y + L\sin a)'' = H \tag{4.86}$$

   (The differentiation of the additive term $(y + L\sin a)$ must be carried out in applications of this model, but it is not shown here.)

2. Force balance applied to the vertical movement of the center of gravity of the pendulum:

$$m\,(L\cos a)'' = V - mg \tag{4.87}$$

   (The differentiation of the additive term $(L\cos a)$ is not shown here.)

3. Torque balance (the rotational version of the Newton's Second Law applied to the center of gravity of the pendulum:

$$Ia'' = V\text{'}L\sin a - HL\cos a \tag{4.88}$$

4. Force balance applied to the cart:

$$My'' = F - H - dy' \tag{4.89}$$

(From Eq. (4.86) – (4.89), the internal forces V and H can be eliminated, resulting in two differential equations not containing V and H.)

Solution to Problem 4.7

The circuit consists of two resistors in parallel in series with the third resistor. The resulting resistance is

$$R_4 = \frac{R_1 R_2}{R_1 + R_2} + R_3 \tag{4.90}$$

Solution to Problem 4.8

Mean power is

$$P = \frac{1}{2}\frac{U^2}{R} \tag{4.91}$$

which solved for R gives

$$R = \frac{1}{2}\frac{U^2}{P} = \frac{1}{2}\frac{220^2}{100} = 242\ \Omega \tag{4.92}$$

Solution to Problem 4.9

There are many ways to find a mathematical model. Here is one: Kirchhoff's voltage law gives

$$-v_1 + v_C + v_2 = 0 \tag{4.93}$$

or

$$v_C = v_1 - v_2 \tag{4.94}$$

Kirchhoff's current law applied to the upper node gives

$$\begin{aligned} 0 &= i_C - i_R + \overbrace{i_2}^{=0} && (4.95)\\ &= C\, v_C{}' - \frac{v_2}{R} && (4.96)\\ &= C\,(v_1 - v_2)' - \frac{v_2}{R} && (4.97)\\ &= C\left(v_1{}' - v_2{}'\right) - \frac{v_2}{R} && (4.98) \end{aligned}$$

Getting v_2' alone on the left side:

$$v_2{}' = v_1{}' - v_2/\left(RC\right) \tag{4.99}$$

$v_2(t)$ is given by the integral of $v_2{}'$:

$$v_2(t) = v_2(0) + \int_0^t v_2(\theta)'\, d\theta \tag{4.100}$$

Chapter 5

Block diagram models

5.1 Introduction

The differential equation models in Ch. 6 can be represented with *block diagrams* which display graphically the structure of the model. Therefore, a block diagram may enhance your understanding of the model. And you can simulate with block diagram models in tools as OpenModelica, LabVIEW and Simulink, cf. Section 5.3. My experience is that block diagram simulation is an effective way of simulation as in tools as mentioned have libraries of ready-made model blocks that you can use to build the total model.

Figure 5.1 gives an overview over model blocks that you can use. Actually, there is no standard about how to design model blocks, and you may actually invent blocks yourself as long as you define their properties accurately.

5.2 How to draw block diagrams

Let's look at an example right away.

Example 5.1 *Block diagram of the mathematical model of a kettle*

Appendix 39.6 describes a kettle including a mathematical model – a differential equation – describing the temperature of the water. I repeat the model here for easy reference:

$$[CT(t)]' = P(t - t_d) + G\,[T_a(t) - T(t)] \tag{5.1}$$

where T is water temperature, C is heat capacity, Parameters and variables in (5.1) defined in Table 39.5.

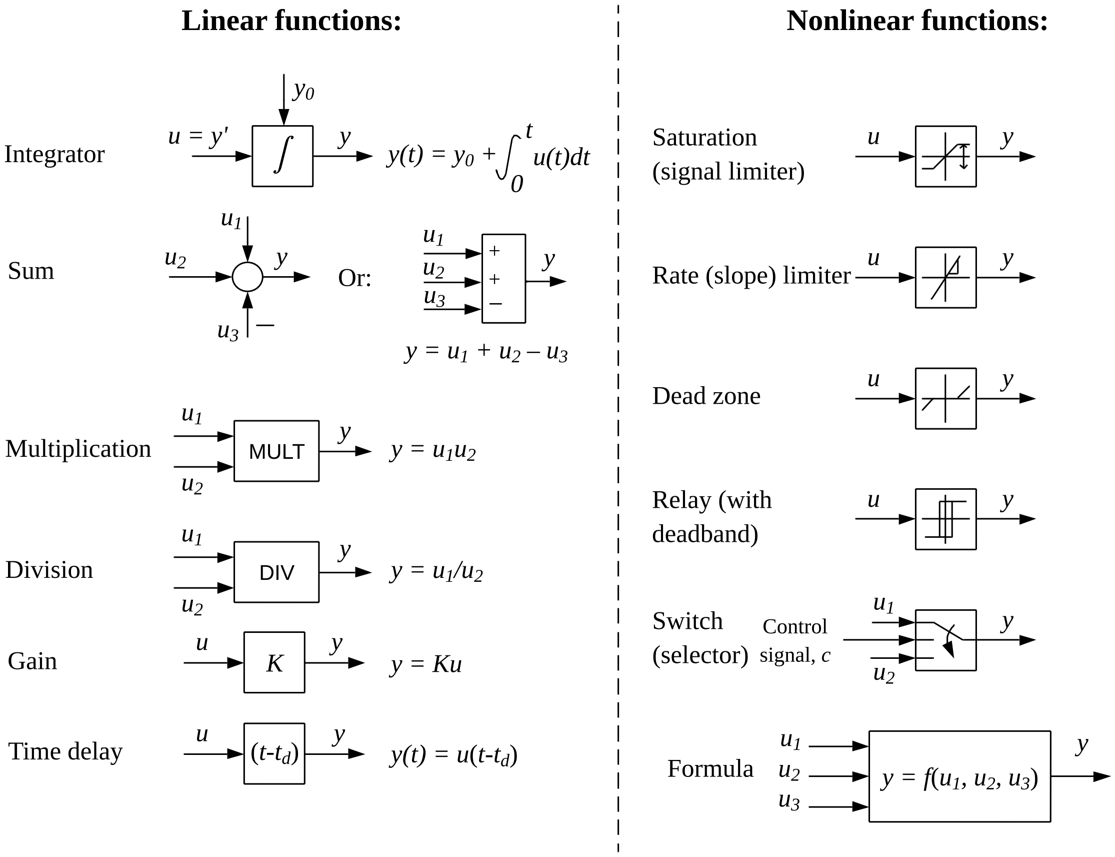


Figure 5.1: Elementary model blocks.

Step 1: Write the model as a state space model

Dividing by C in (5.1) gives

$$T(t)' = \{P(t - t_d) + G\,[T_{\rm a}(t) - T(t)]\}\,/C \tag{5.2}$$

which is a state space model.

Step 2: Write the integral that integrates the time derivative

$T(t)$ is the integral of $T'(t)$:

$$T(t) = T(0) + \int_0^t T'(\theta)\,d\theta \tag{5.3}$$

with initial condition $T(0) = T_{\rm init}$.

Step 3: Add an integrator block to the block diagram

See Figure 5.2.

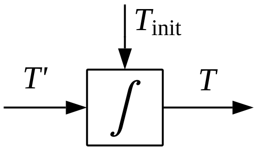


Figure 5.2: Integrator block.

Note: The symbol in the integrator block in Figure 5.2 is the common integral symbol. The symbol is not standardized. Other symbols are used in OpenModelica, LabVIEW and Simulink. For example, the symbol $\frac{1}{s}$ is used in Simulink since $\frac{1}{s}$ is the s-transfer function of an integrator (cf. Section 9.2.1.3).

Step 4: Add blocks, and input signals, needed to build the time derivative

The point is to add blocks, and input signals, to make up the time derivative, T'. In other words, you build the block diagram towards the left, starting at the line representing T'.

There are many ways to construct the block diagram. Figure 5.3 shows one way.

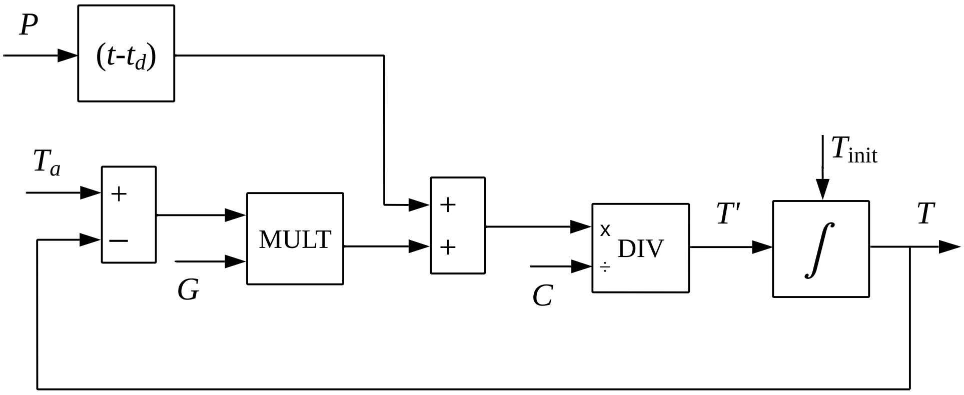


Figure 5.3: Block diagram of the mathematical model of the kettle.

Comments to the block diagram:

- The variable T in the term $G(T_a - T)$ is fetched from the output of the integrator.
- Just an interesting observation: As is clear from the block diagram, the model contains a "natural" negative feedback (loop). This negative feedback helps stabilizing the temperature of the kettle, just like a feedback controller can do for any physical process. The natural feedback works as follows: If T increase, the heat conductance term $G(T_a - T)$ increase, and therefore T' decrease, helping stabilizing T. If the kettle had perfect isolation, i.e. $G = 0$, there would not be such a natural negative feedback, and the temperature would not be stabilized; it would keep on increasing (for a nonzero power input).[1]

[End of Example 5.1]

Block diagram using formula block to define the time derivative

For complex models, the block diagram may contain lots of blocks. It may then be cumbersome, and prone to errors, to make changes to the block diagram. An alternative to using blocks all over the block diagram, is using a formula block, see Figure 5.1, to define the time derivative. You simply write with text the right side of the differential equation defining the time derivative inside the formula block. Or in mathematical terms: The formula block calculates $f()$ in the state space model:

$$x' = f(\cdot) \tag{5.4}$$

The output of the block is the time derivative, and you connect it to the integrator block.

Example 5.2 *Block diagram of kettle model using formula block*

See Figure 5.4.

A comment to the block diagram:

- I have represented the time delay of P in a separate block outside the formula block. Alternatively, you can drop the time delay block and just write P(t-td) instead of Pd in the formula block. But, if you write P(t-td) in the formula block there a chance that someone may interprete it as P times (t-td), which is seriously wrong.

[End of Example 5.2]

[1]

– In general, if a system has no negative feedback loops – natural or constructed, it is not stable.

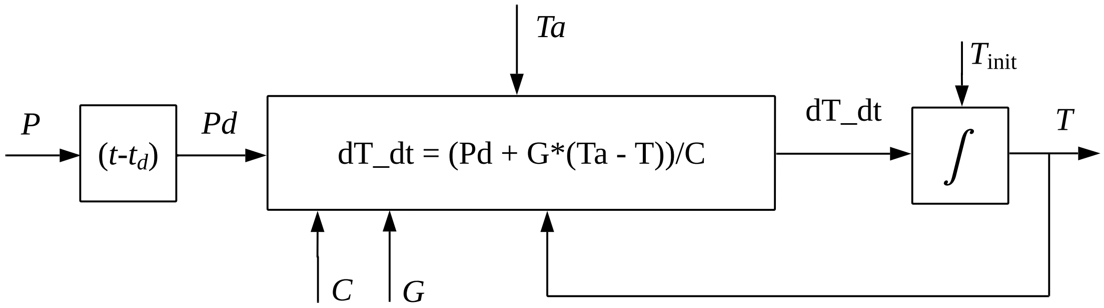


Figure 5.4: Block diagram of the kettle model using a formula block to calculate the time derivative.

5.3 Simulation with block diagram models

Earlier in this chapter, you have seen how mathematical models can be illustrated with block diagrams. Here are some relatively well-known simulation tools using block diagram models:

- OpenModelica
- Simulink
- LabVIEW

Some characteristics of such tools are:

- They come with a library of blocks which you connect to construct the model. The blocks comprise blocks for elementary mathematical functions, and blocks to represent state space models and transfer functions.
- They offer alternative simulation algorithms.

Block diagram-based simulation tools are user-friendly since the model to be simulated has a graphical representation, as opposed to the textual representation you are using when you program simulators in native textual programming code like Python, cf. Ch. 7.

This book concentrates on using OpenModelica (because this is a free tool). A tutorial to OpenModelica is in Appendix 42.

5.4 *Problems for Chapter 5*

Problem 5.1 *Mathematical block diagram of an accumulation*

In Ch. 4, the general model of dynamic systems are presented as (4.1)-(4.2), which are repeated here for convenience:

$$\text{Acc}(t) = \text{Acc}(0) + \int_0^t \text{Acc'}\, d\theta \tag{5.5}$$

where:

$$\text{Acc}' = \text{Inflow} - \text{Outflow} + \text{Generation} \tag{5.6}$$

Draw a mathematical block diagram of (5.5)-(5.6).

Problem 5.2 *Mathematical block diagram of water tank*

Figure 5.5 shows a water tank with pump inflow and valve outflow.

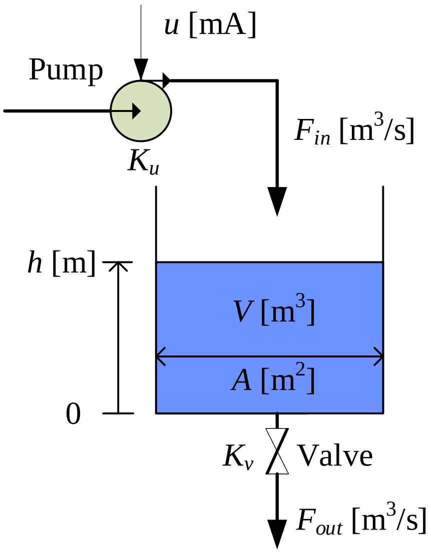


Figure 5.5: Water tank with pump inflow and valve outflow.

Here is a mathematical model of the level in a water tank:

$$Ah' = F_{\text{in}} - F_{\text{out}} \tag{5.7}$$

where:

$$F_{\text{in}} = K_u u \tag{5.8}$$

$$F_{\text{out}} = K_v \sqrt{\rho g h} \tag{5.9}$$

You are to draw alternative block diagrams with u as input variable, and h as output variable.

1. Draw a mathematical block diagram of the model using only mathematical blocks in addition to the integrator.

2. Then draw a block diagram using a formula block (see Figure 5.1) in addition to the integrator.

5.5 *Solutions to problems for Chapter 5*

Solution to Problem 5.1

See Figure 5.6.

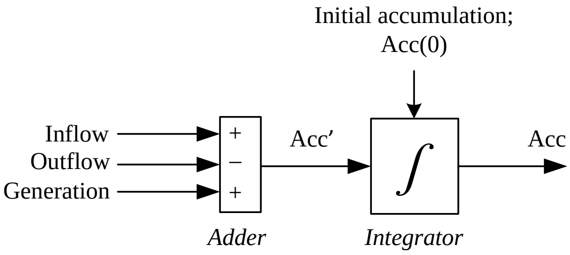


Figure 5.6: Mathematical block diagram of accumulation.

Solution to Problem 5.2

1. See Figure 5.8.

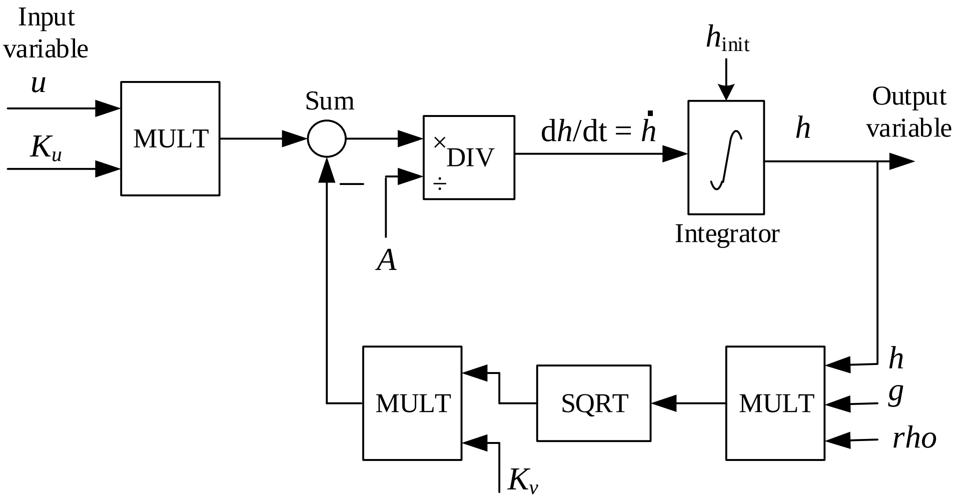


Figure 5.7: A mathematical block diagram of the model using elementary mathematical blocks for representing h'.

2. See Figure 5.7.

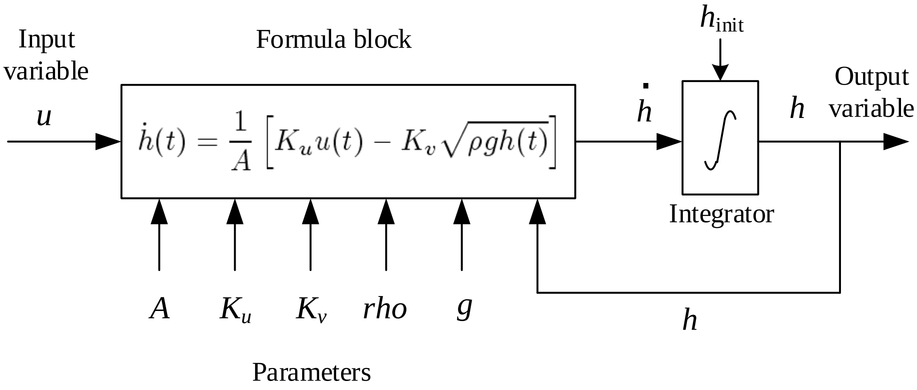


Figure 5.8: A mathematical block diagram of the model using one block with textual mathematical expressions for representing h'.

Chapter 6

State space models

6.1 Introduction

A *state space model* is just a structured form or representation of the differential equations for a system. Typically, the differential equations stem from mechanistic modeling of dynamic systems as explained in Ch. 4.

State space models are useful in a number of situations:

- Linearization of non-linear models
- Calculation of time-responses – both analytically and numerically
- Using simulation tools: Tools as Python, MATLAB and LabVIEW have simulation functions that assumes state space models.
- Analysis of dynamic systems, e.g. stability analysis
- Analysis and design of advanced controllers and estimators: Controllability and observability analysis; Design of LQ optimal controllers, Model-based predictive control; Design of state estimators (Kalman Filters).

6.2 The state space model

Most dynamic models can be represented with the following two equations:

- *The state space model*, which is a set of first order differential equations of the state variables[1]:

$$\begin{aligned} {x_1}' &= f_1(x, u, d, p) \\ \vdots \quad &\vdots \quad \vdots \\ {x_n}' &= f_n(x, u, d, p) \end{aligned} \tag{6.1}$$

- *The output model*, which is a set of algebraic equations defining the output variables:

$$\begin{aligned} y_1 &= g_1(x, u, d, p) \\ \vdots \quad &\vdots \quad \vdots \\ y_m &= g_m(x, u, d, p) \end{aligned} \tag{6.2}$$

The variables are (the indexes are dropped here, for simplicity):

- x is the state variable.
- y is the output variable.
- u is the input variable. In context of control, u represents the control variable (or signal).
- d is the disturbance, which also may be denoted the environmental variable, or the load variable.
- p is the parameter.

(6.1) and (6.2) can be written compactly on a vector form as:

$$x' = f(x, u, d, p) \equiv f(\cdot) \tag{6.3}$$

$$y = g(x, u, d, p) \equiv g(\cdot) \tag{6.4}$$

In (6.3) and (6.4), x, u, d, p, y are vectors. For example,

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{6.5}$$

which also can be written as $x = [x_1, \cdots, x_n]^{\mathrm{T}}$ where super-index T means transpose.

In (6.3) and (6.4), f and g are vector functions:

$$f = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} \tag{6.6}$$

[1] Alternative symbols to x' are $\dot{x}$ and $\frac{dx}{dt}$.

$$g = \begin{bmatrix} g_1 \\ \vdots \\ g_m \end{bmatrix} \tag{6.7}$$

The model (6.3) and (6.4) is often referred to as a *nonlinear* state space model because the vector functions $f(\cdot)$ and $g(\cdot)$ may contain nonlinear functions.

In many cases the mathematical modelling results in one or more first order differential equations. In such cases it is straighforward to write the model as a state space model; All you have to do is to ensure that the time derivatives appear alone on the left-hand side of the differential equations. However, when modeling kinetic systems, the model may consist of second order differential equations due to Newton's Second Law (since accelaration is the second order derivative of position). Example 6.1 demonstrates how to write a second order differential equation as a state space model.

Example 6.1 *Mass-spring-damper-model written as a state space model*

Figure 6.1 shows a mass-spring-damper-system. z is position. F is applied force. D is damping constant. K is spring constant. It is assumed that the damping force F_d is proportional to the speed, and that the spring force F_s is proportional to the position of the mass. The spring force is assumed to be zero when z is zero. Force balance (Newton's Second Law) yields

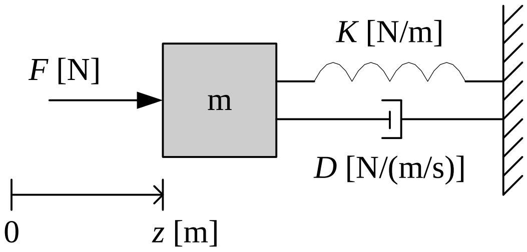


Figure 6.1: Mass-spring-damper.

$$\begin{aligned} mz'' &= F - F_d - F_s \\ &= F - Dz' - Kz \end{aligned} \tag{6.8}$$

which is a second order differential equation.

We *define the following new variables*: x_1 for position z, x_2 for speed z', and u for force F. Then the model (6.8) can be written as the following equivalent set of two first order differential equations:

$$x_1{}' = x_2 \tag{6.9}$$

$$m\, x_2{}' = -Dx_2 - Kx_1 + u \tag{6.10}$$

which can be written on the standard form (6.1) as follows:

$$x_1{}' = \underbrace{x_2}_{f_1} \tag{6.11}$$

$$x_2{}' = \underbrace{\left(-Dx_2 - Kx_1 + u\right)/m}_{f_2} \tag{6.12}$$

Let us regard the position x_1 as the output variable y:

$$y = \underbrace{x_1}_{g} \tag{6.13}$$

The initial position, $x_1(0)$, and the initial speed, $x_2(0)$, define the initial state of the system.

(6.11) and (6.12) and (6.13) constitute a second order state space model which is equivalent to the original second order differential equation (6.8).

[End of Example 6.1]

6.3 The response of a state space model

6.3.1 Dynamic response

The output equation, (6.4), or (6.2), expresses that y is a function of x. So, x must exist. But x does not appear explicitly in the model (6.3) – (6.4)! So how is x is obtained? It is obtained by solving the differential equations, i.e. by integrating the differential equations. Let's asssume that the state space model on the vector form (6.3). The response is:

$$x(t) \quad = \quad x(0) + \int_0^t x'(\theta)\, d\theta \tag{6.14}$$

where x' is known from the differential equations:

$$x'(t) = f\left(x(t), u(t), d(t), p\right) \tag{6.15}$$

The response in the output variable, $y(t)$, is:

$$y(t) = g\left(x(t), u(t), d(t), p\right) \tag{6.16}$$

See Figure 6.2. The state vector $x(t) = [x_1(t), \cdots, x_n(t)]^{\mathrm{T}}$ defines or spans the *state space* of the system, with $x_1(t), \cdots, x_n(t)$ as the coordinates of the space. $x(t)$ is the *state* of the system as a function of time. As time evolves, $x(t)$ creates a *state trajectory.* $x(t_1)$ is the state of the system at point of time t_1, and $x(t_2)$ is the state of the system at point of time t_2, etc.

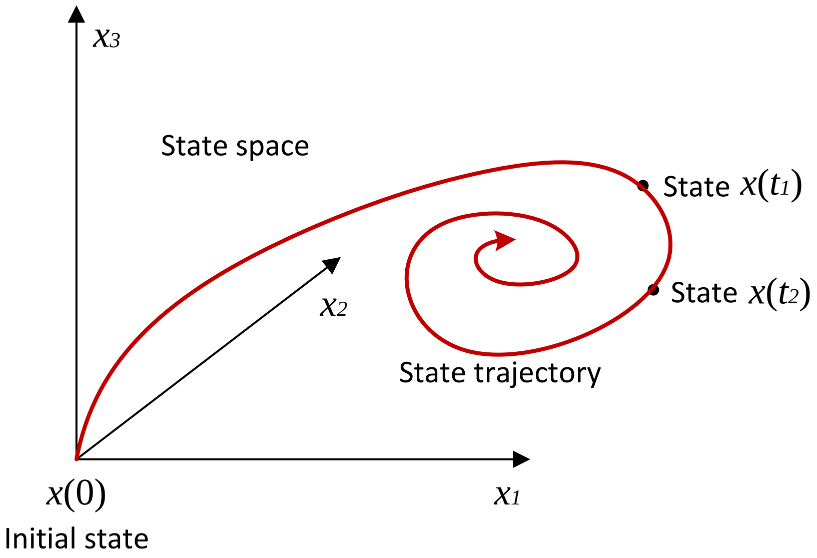


Figure 6.2: The notion of *state*.

6.3.2 Static response

In some situations it is useful to calculate the *static response* of a dynamic system, e.g. to check that the simulated response is correct. The static response is the steady state value of the output variable of the model when the input variables have constant values. This response can be calculated directly from the model after the *time-derivatives have been set equal to zero*, since then the variables have constant values, their time-derivatives equal zero, i.e.

$$x_s{}' = 0 = f(x_s, u_s, d_s, p) \tag{6.17}$$

which is an algebraic equation relating the static values of the variables x, u, and d, and the model parameters p.

Example 6.2 *Calculation of static response for mass-spring-damper*

The mass-spring-damper system described in Example 4.3 has the following model:

$$my'' = -Dy' - Ky + F \tag{6.18}$$

Suppose the force F is constant of value F_s. The corresponding static response in the position y can be found by setting the time-derivatives equal to zero and solving with respect to y. The result is

$$y_s = \frac{F_s}{K} \tag{6.19}$$

[End of Example 6.2]

Can you calculate the static response in the process variable (the process output) for any differential equation model? No! Because there may be models for which there is no static response, as demonstrated in Example 6.3.

Example 6.3 *Failed attempt to calculate static response*

See Example 4.1 where (4.8) is the model of the liquid tank, repeated here for convenience:

$$h' = (F_{\text{in}} - F_{\text{out}})/A \tag{6.20}$$

Assume that both F_{in} and F_{out} have constant values, $F_{\text{in},0}$ and $F_{\text{out},0}$, respectively. Assuming there is a static response in h, called h_s. Its time derivative is $h_s' = 0$. Then, (6.20) gives

$$0 = (F_{\text{in},0} - F_{\text{out},0})/A \neq 0$$

which make no sense unless $F_{\text{in},0} = F_{\text{out},0}$, and even in that case, no value of h_s is given. We know from physical insight that there is no static response in h (unless $F_{\text{in},0} = F_{\text{out},0}$); h will change all the time (unless $F_{\text{in},0} = F_{\text{out},0}$).

Only asymptotically stable systems have well-defined steady state responses which can be calculated from the static version of the mathematical model, as in Example 6.2. Asymptotic stability is defined in Ch. 19.

[End of Example 6.3]

6.4 Linear state space models

Linear state space models are a special case of the general state space model (6.3)-(6.4). Many methods for analysis of differential equation models, as stability analysis, response calculation and model transformations, are based on linear state space models. Let us study a general second order linear state space model to see how linear state space models are defined. The model has two state-variables, x_1 and x_2, and two input variables, u_1 and u_2:

$$x_1{}' = a_{11}x_1 + a_{12}x_2 + b_{11}u_1 + b_{12}u_2 \tag{6.21}$$

$$x_2{}' = a_{21}x_1 + a_{22}x_2 + b_{21}u_1 + b_{22}u_2 \tag{6.22}$$

where the a and b coefficients are parameters (constants).

(6.21)-(6.22) can written on matrix-vector form as follows:

$$\underbrace{\begin{bmatrix} x_1{}' \\ x_2' \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{u} \tag{6.23}$$

or, more compact:

$$x' = Ax + Bu \tag{6.24}$$

where x is the state vector and u is the input vector. A is called the system-matrix, and is square in all cases.

Let us assume that the system has two output variables, which generally can be functions of both the state variables and the input variables. The output function can then be written on the form

$$y_1 = c_{11}x_1 + c_{12}x_2 + d_{11}u_1 + d_{12}u_2 \tag{6.25}$$

$$y_2 = c_{21}x_1 + c_{22}x_2 + d_{21}u_1 + d_{22}u_2 \tag{6.26}$$

which can be written on matrix-vector form as follows:

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{u} \tag{6.27}$$

or, more compact:

$$y = Cx + Du \tag{6.28}$$

Example 6.4 *Mass-spring-damper model as a state space model on matrix-vector form*

The state space model (6.11), (6.12), (6.13) is linear. We get

$$\underbrace{\begin{bmatrix} {x_1}' \\ {x_2}' \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\frac{K}{m} & -\frac{D}{m} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}}_{B} u \tag{6.29}$$

$$y = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \end{bmatrix}}_{D_1} u \tag{6.30}$$

In (6.30) I have used symbol $D_1$for the matrix since I have already used D for the damping coefficient.

[End of Example 6.4]

6.5 Linearization of non-linear models

6.5.1 When do we have to linearize?

In many cases the mathematical model contains one or more non-linear differential equations. If the mathematical model is non-linear, there may be good reasons to *linearize* it, which means to develop a local linear model which approximates the original model about a given operating point. The reasons may be the following:

- We want to study the behavior of the system *about an operating point*, which is one specified state where the system can be. It is then the *deviations* from this operating point we study. Examples of such operating points are the level of 8.7 m in a tank, the temperature of 50 degrees Celcius in a heat exchanger, etc. It can be shown (and we will do it soon) that a model which describes the behavior of the deviations about the operating point, is approximately linear.
- We can use the large number of the methods which are available for analysis and design of linear systems, e.g. for stability analysis, frequency response, controller design and signal filter design. The number of methods for linear models are much larger than for non-linear models.

If you are to build a simulator, I recommend using the non-linear model – and not the linearized model – for the simulator since it gives a more accurate representation of the system.

Figure 6.3 shows the relation between the original non-linear system and the local linear system (model).

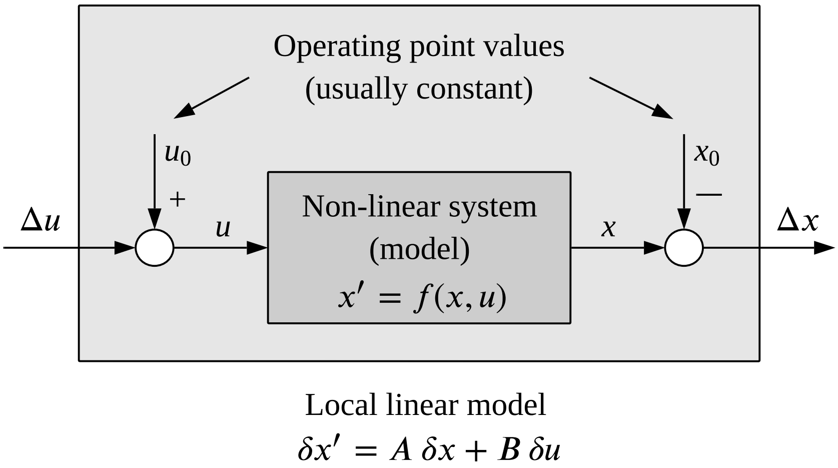


Figure 6.3: Illustration of the relation between the original non-linear system and the local linear system (model)

The input variable which excites the non-linear system is assumed to be given by

$$u = u_0 + \Delta u \tag{6.31}$$

where u_0 is the value in the operating point and Δu is the deviation from u_0. Similarly,

$$x = x_0 + \Delta x \tag{6.32}$$

If you are going to experiment with the system to develop or adjust a linear model about the operating point, you must adjust Δu and observe the corresponding response in Δx (or in the output variable Δy).

Figure 6.4 illustrates the variables in (6.32).

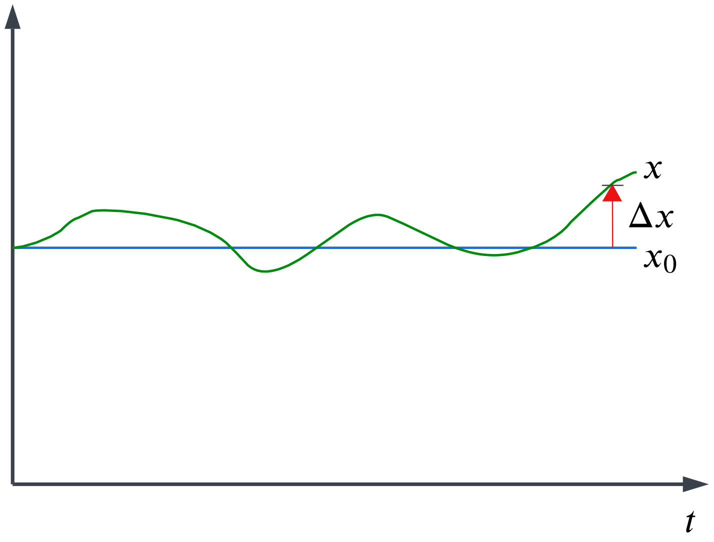


Figure 6.4: Illustration of the variables in $x = x_0 + \Delta x$.

6.5.2 Deriving the linearization formulas

We assume that the model is a non-linear state space model:

$$x' = f(x, u) \tag{6.33}$$

Assume that the system is in some operating point represented with

$$x_0{}' = f(x_0, u_0) \tag{6.34}$$

If the input variable u is changed by Δu from the operating point value u_0, the state-variable x will change from x_0. Let's denote that change by Δx. (6.33) can then be written

$$(x_0 + \Delta x)' = x_0{}' + \Delta x' = f(x_0 + \Delta x,\ u_0 + \Delta u) \tag{6.35}$$

We now use a first order Taylor series expansion of $f(\cdot)$ in (6.35):

$$x_0{}' + \Delta x' \approx f(x_0, u_0) + \left.\frac{\partial f}{\partial x}\right|_0 \Delta x + \left.\frac{\partial f}{\partial u}\right|_0 \Delta u \tag{6.36}$$

The expression $\left.\frac{\partial f}{\partial x}\right|_0$ means the partial time-derivative of f with respect to x, calculated at the operating point, that is, with x_0 and u_0 inserted into $\frac{\partial f}{\partial x}$. The same applies to $\left.\frac{\partial f}{\partial u}\right|_0$. Now we will exploit the fact that ${x_0}'$ is equal $f(x_0, u_0)$, cf. (6.34). This implies that these two terms can be cancelled against each other in (6.36), which then becomes

$$\Delta x' \approx \left.\frac{\partial f}{\partial x}\right|_0 \Delta x + \left.\frac{\partial f}{\partial u}\right|_0 \Delta u \tag{6.37}$$

Eq. (6.37) has the form of a linear state space model. Let's define new variables, δx and δu to be used in the place of Δx and Δu in (6.37) where we also replace the approximative sign with an equality sign. We have then obtained the following linear state space model:

$$\delta x' = \underbrace{\left.\frac{\partial f}{\partial x}\right|_0}_{A} \delta x + \underbrace{\left.\frac{\partial f}{\partial u}\right|_0}_{B} \Delta u \tag{6.38}$$

$$= A\,\delta x + B\,\delta u \tag{6.39}$$

which is a *local linear model* describing approximately the changes or deviations from the operating point.

In more detail (6.38) is

$$\begin{bmatrix} \delta x_1' \\ \delta x_2' \\ \vdots \end{bmatrix} = \underbrace{\left.\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}\right|_0}_{A} \cdot \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \vdots \end{bmatrix} + \underbrace{\left.\begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \cdots \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}\right|_0}_{B} \cdot \begin{bmatrix} \delta u_1 \\ \delta u_2 \\ \vdots \end{bmatrix} \tag{6.40}$$

A and B are Jacobian matrices, or just Jacobians. They are matrices containing partial derivatives which generally are functions of the operating point. If the operating point is *constant*, A and B will be constant matrices, which can be calculated once and for all.

Similarly, linearization of the output equation

$$y = g(x, u) \tag{6.41}$$

gives

$$\delta y = \underbrace{\left.\frac{\partial g}{\partial x}\right|_0}_{C} \delta x + \underbrace{\left.\frac{\partial g}{\partial u}\right|_0}_{D} \delta u = C\,\delta x + D\,\delta u \tag{6.42}$$

or

$$\begin{bmatrix} \delta y_1 \\ \delta y_2 \\ \vdots \end{bmatrix} = \underbrace{\left.\begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}\right|_0}_{C} \cdot \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \vdots \end{bmatrix} + \underbrace{\left.\begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \frac{\partial g_1}{\partial u_2} & \cdots \\ \frac{\partial g_2}{\partial u_1} & \frac{\partial g_2}{\partial u_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}\right|_0}_{D} \cdot \begin{bmatrix} \delta u_1 \\ \delta u_2 \\ \vdots \end{bmatrix} \tag{6.43}$$

If the operating point is a static *equilibrium* point, all variables have constant values and all time-derivatives are zero. Thus,

$${x_0}' = 0 = f(x_0, u_0) \tag{6.44}$$

The values of the model variables in the static operating point can be found by solving the algebraic equation (6.44) with respect to the unknown variables.

Example 6.5 *Linearization of a non-linear tank model*

The nonlinear model

Figure 6.5 shows a liquid tank with inlet via a pump and outlet via a valve with fixed opening.

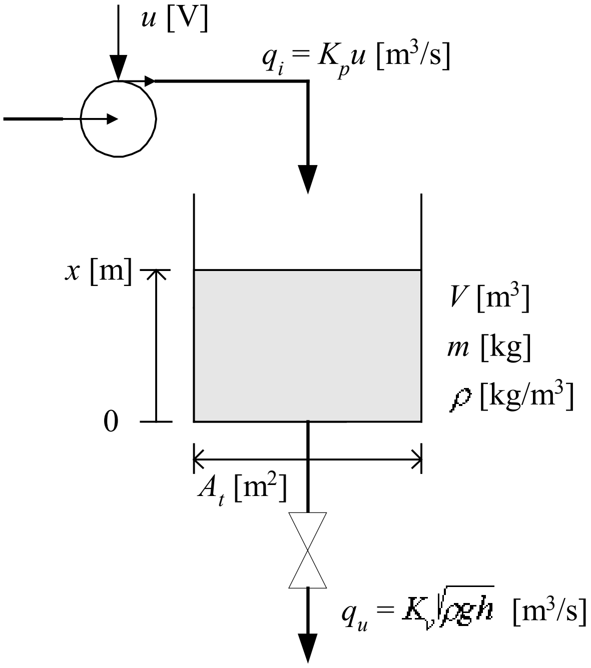


Figure 6.5: Liquid tank with non-linear mathematical model

The outflow is assumed to be proportional to the square root of the pressure drop over the valve, and this pressure is assumed to be equal to the hydrostatic pressure $\rho g h$ at the outlet. The mass balance becomes:

$$\begin{aligned}\rho A_t x' &= \rho q_i - \rho q_u \\ &= \rho K_p u - \rho K_v \sqrt{\rho g x}\end{aligned}$$

which can written:

$$x' = \frac{K_p}{A_t}u - \frac{K_v}{A_t}\sqrt{\rho g x} \equiv f(x,u) \tag{6.45}$$

Static operating point

Before we linearize the original model, we will find the static operating point. Let us assume that the pump control signal is constant: $u = u_0$ (constant). We calculate the corresponding static level, $x = x_0$ (constant), by setting $x' = 0$ in the dynamic model (6.45):

$$0 = \frac{K_p}{A_t}u_0 - \frac{K_v}{A_t}\sqrt{\rho g x_0} \equiv f(x_0, u_0) \tag{6.46}$$

Solving (6.46) with respect to x_0 gives:

$$x_0 = \frac{1}{\rho g}\left(\frac{K_p u_0}{K_v}\right)^2 \tag{6.47}$$

Linearization

Now that we have found the static operating point (u_0, x_0), we can derive the local linear model by linearizing (6.45):

$$\begin{aligned} \delta x' &= \left.\frac{\partial f}{\partial x}\right|_0 \delta x + \left.\frac{\partial f}{\partial u}\right|_0 \delta u \\ &= \underbrace{-\frac{K_v}{A_t}\frac{\rho g}{2\sqrt{\rho g x_0}}}_{A}\delta x + \underbrace{\frac{K_p}{A_t}}_{B}\delta u \\ &= A\,\delta x + B\,\delta u \end{aligned} \tag{6.48}$$

[End of Example 6.5]

6.6 *Problems for Chapter 6*

Problem 6.1 *State space model of a system of tanks*

Figure 6.6 shows two coupled liquid tanks. u_1 and u_2 are control signals.

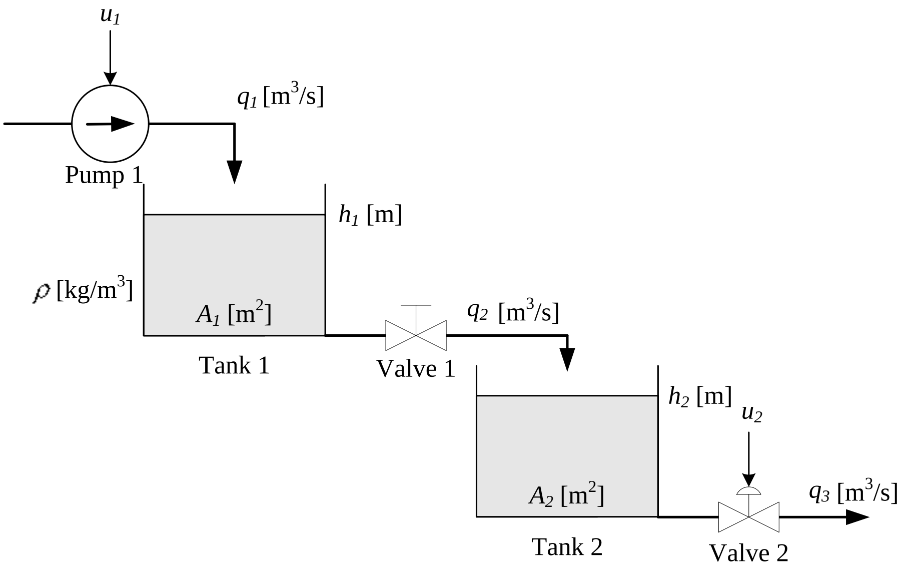


Figure 6.6: Two coupled liquid tanks

Material balance of the liquid in tank 1 gives:

$$\rho A_1\, h_1{}' = \underbrace{\rho K_p u_1}_{q_1} - \underbrace{\rho K_{v_1}\sqrt{\frac{\rho g h_1}{G}}}_{q_2} \tag{6.49}$$

Material balance of the liquid in tank 2 gives:

$$\rho A_2\, h_2{}' = \underbrace{\rho K_{v_1}\sqrt{\frac{\rho g h_1}{G}}}_{q2} - \underbrace{\rho K_{v_2} u_2\sqrt{\frac{\rho g h_2}{G}}}_{q_3} \tag{6.50}$$

Valve 1 has fixed opening. Valve 2 is a control valve with control signal u between 0 and 1. The square root functions stems from the common valve characteristic which expresses that the flow is proportional to the square root of the pressures drop across the valve. Here, the pressure drops are assumed to be equal to the hydrostatic pressures at the bottom the tanks. For example, for tank 1 the hydrostatic pressure is $\rho g h_1$. The parameter G is the relative density of the liquid.[2]

Assume that the input variables are u_1 and u_2, and that the output variables are $y_1 = h_1$ and $y_2 = h_2$. Write the model (6.49) – (6.50) as a state space model. Is the state space model linear or nonlinear?

[2] $G = \rho/\rho_{water}$.

Problem 6.2 *Calculation of static response of thermal system*

Calculate the static response in the temperature T from the thermal model (4.19).

Problem 6.3 *State space model on matrix-vector form*

Write the following model as a state space model on matrix-vector form:

$$\begin{aligned} {x_1}' &= x_2 \\ 2\,{x_2}' &= 8u_2 - 6x_2 - 2x_1 + 4u_1 \\ y &= 5x_1 + 7u_1 + 6x_2 \end{aligned} \tag{6.51}$$

Problem 6.4 *Linearization*

Given the state space model (6.52) – (6.54) which are solutions to Problem (6.1). Linearize the model.

6.7 *Solutions to problems for Chapter 6*

Solution to Problem 6.1

Density ρ can be cancelled. (6.49) becomes

$$h_1{}' = \overbrace{\frac{1}{A_1}\left(K_p u_1 - K_{v_1}\sqrt{\frac{\rho g h_1}{G}}\right)}^{f_1(\cdot)} \tag{6.52}$$

(6.50) becomes

$$h_2{}' = \overbrace{\frac{1}{A_2}\left(K_{v_1}\sqrt{\frac{\rho g h_1}{G}} - K_{v_2} u_2\sqrt{\frac{\rho g h_2}{G}}\right)}^{f_2(\cdot)} \tag{6.53}$$

The measurement equations become

$$y_1 = \overbrace{h_1}^{g_1(\cdot)} \tag{6.54}$$

$$y_2 = \overbrace{h_2}^{g_2(\cdot)} \tag{6.55}$$

The state space model is non-linear due to the square root functions.

Solution to Problem 6.2

We start by setting

$$T' = 0 \tag{6.56}$$

in the model (4.19), which then becomes:

$$0 = \frac{1}{cm}\left[P + cF\left(T_i - T\right) + U_h\left(T_e - T\right)\right] \tag{6.57}$$

Solving for T gives the static response:

$$T_s = \frac{P + cFT_i + U_h T_e}{cF + U_h} \tag{6.58}$$

Solution to Problem 6.3

Firstly, we isolate the first order derivatives on the left side, and list the variables in the proper order to prepare for the matrix-vector form:

$$\begin{array}{rcl} x_1{}' & = & x_2 \\ x_2{}' & = & -x_1 - 3x_2 + 2u_1 + 4u_2 \\ y & = & 5x_1 + 6x_2 + 7u_1 \end{array} \tag{6.59}$$

Finally,

$$\underbrace{\begin{bmatrix} {x_1}' \\ {x_2}' \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{u} \tag{6.60}$$

and

$$y = \underbrace{\begin{bmatrix} 5 & 6 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 7 & 0 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{u} \tag{6.61}$$

or, compactly:

$$x' = Ax + Bu \tag{6.62}$$

$$y = Cx + Du \tag{6.63}$$

Solution to Problem 6.4

Linearization of the differential equations:

$$\begin{bmatrix} \delta\, {h_1}' \\ \delta\, {h_2}' \end{bmatrix} = \left.\begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} \\ \frac{\partial f_2}{\partial h_1} & \frac{\partial f_2}{\partial h_2} \end{bmatrix}\right|_0 \cdot \begin{bmatrix} \delta h_1 \\ \delta h_2 \end{bmatrix} \tag{6.64}$$

$$+ \left.\begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} \end{bmatrix}\right|_0 \cdot \begin{bmatrix} \delta u_1 \\ \delta u_1 \end{bmatrix} \tag{6.65}$$

$$= \underbrace{\left.\begin{bmatrix} -\frac{K_{v_1}}{A_1}\sqrt{\frac{\rho g}{G}}\frac{1}{2\sqrt{h_1}} & 0 \\ \frac{K_{v_1}}{A_2}\sqrt{\frac{\rho g}{G}}\frac{1}{2\sqrt{h_1}} & -\frac{K_{v_2}u_2}{A_2}\sqrt{\frac{\rho g}{G}}\frac{1}{2\sqrt{h_2}} \end{bmatrix}\right|_0}_{A} \cdot \begin{bmatrix} \delta h_1 \\ \delta h_2 \end{bmatrix} \tag{6.66}$$

$$+ \underbrace{\left.\begin{bmatrix} \frac{K_p}{A_1} & 0 \\ 0 & -\frac{K_{v_2}}{A_2}\sqrt{\frac{\rho g h_2}{G}} \end{bmatrix}\right|_0}_{B} \cdot \begin{bmatrix} \delta u_1 \\ \delta u_2 \end{bmatrix} \tag{6.67}$$

Linearization of the output equation:

$$\begin{bmatrix} \delta y_1 \\ \delta y_2 \end{bmatrix} = \left.\begin{bmatrix} \frac{\partial g_1}{\partial h_1} & \frac{\partial g_1}{\partial h_2} \\ \frac{\partial g_2}{\partial h_1} & \frac{\partial g_2}{\partial h_2} \end{bmatrix}\right|_0 \cdot \begin{bmatrix} \delta h_1 \\ \delta h_2 \end{bmatrix} \tag{6.68}$$

$$+ \left.\begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \frac{\partial g_1}{\partial u_2} \\ \frac{\partial g_2}{\partial u_1} & \frac{\partial g_2}{\partial u_2} \end{bmatrix}\right|_0 \cdot \begin{bmatrix} \delta u_1 \\ \delta u_2 \end{bmatrix} \tag{6.69}$$

$$= \underbrace{\left.\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right|_0}_{C} \cdot \begin{bmatrix} \delta h_1 \\ \delta h_2 \end{bmatrix} \tag{6.70}$$

$$+ \underbrace{\left.\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\right|_0}_{D} \cdot \begin{bmatrix} \delta u_1 \\ \delta u_2 \end{bmatrix} \tag{6.71}$$

Chapter 7

Simulation algorithms of state space models

7.1 Why simulate?

Why simulate? Some good reasons are:

- ***Learning and training:*** A simulator can be a very good learning and training tool! Simulations can give students, operators and other people an experience of how systems behave, without running physical experiments. While running physical experiments is often the dream scenario, it may simply not be possible to run physical experiments – it can be too expensive or risky, and perhaps the physical system does not even exist!
- ***Testing:*** Different Scenarios – like "if-then" scenarios - may be tested without any risk to the environment. Just think about the benefits of being able to check whether an oil platform or aircraft management system will work without actually doing experiments with the platform or plane.
- ***Analyze:*** With simulations, you can observe how variables which you may not actually measure will develop in reality. This can give new deep insight into, and understanding of the system.
- ***Design:*** You can use a simulator to design – or redesign – a real system. For example, you can find out if a wastewater magazine in a so-called combined drainage system, where rain and sewage are mixed, is large enough to be able to collect the wastewater due to heavy rainfall.

Is simulation important and useful? You can think about it...

A modern term used for simulators is digital twin. (Maybe a buzzword, but quite a descriptive one.)

7.2 Simulation algorithm for state space models

7.2.1 Introduction

A textual simulation algorithm is an algorithm that you can program in a textual programming tool as Python, MATLAB, Octave, JavaScript, C, etc. Although there are simulation tools (functions) – i.e. differential equation solvers – in Python and MATLAB etc., I will here show how you can develop a simulation algorithm completely from scratch. Your own implementation may run faster than with the simulation tools, and your implementation will become completely transparent, which may be useful for testing and documentation. (Personally, I usually develop simulation code from scratch in my projects.)

We will focus on developing a simulation algorithm ready for programming based on the state space model (6.3). Including the output model (6.4) in the simulator algorithm is straightforward since that model comprises only algebraic expressions.

Thus, we assume that the model of the system to be simulated has the form of(6.3), which is repeated here for convenience:

$$x' = f(x, u, d, p) \tag{7.1}$$

The simulation algorithm which we will develop, and eventually implement in a program in Python, will calculate the state, $x(t_k)$, at any discrete point of time, t_k. as illustrated in Figure 7.1. A simulation algorithm is based on some method of discretization of the given continuous-time model, (7.1).

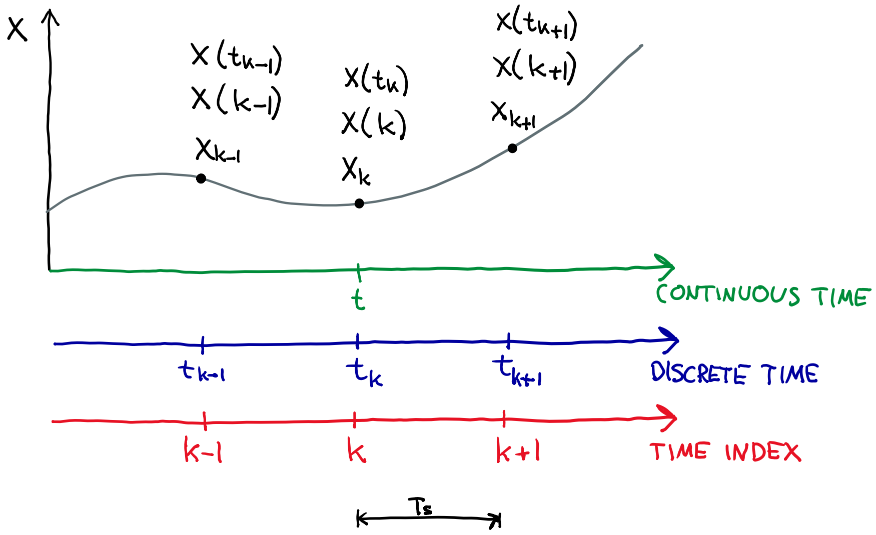


Figure 7.1: Discrete-time values of x with three alternative symbols.

Some comments about the symbols shown in Figure 7.1:

- t_s [s] is the time step, which defines the resolution along the simulation time axis.

- k is the time index, which is an integer, which counts the number of time steps corresponding to a given point of time, t_k. Example: Assume that $t_s = 0.1$ s. How many time steps are there from $t = 0$ to 5.0 s? Answer: 50. Then, we have the following alternative ways to represent the point of time $t = 5.0$ s:

$$t = 5.0 \Leftrightarrow t_{50} \Leftrightarrow k = 50$$

- Often, t_k means "now", the present point of time. Then, t_{k-1} means the previous point of time, and t_{k+1} means the next (future) point of time.
- The following symbols are equivalent:

$$x(t_k) \Leftrightarrow x(k) \Leftrightarrow x_k$$

Among these, I will mainly use the symbol x_k in the following.

7.2.2 The simulation algorithm

Figure 7.2 illustrates the simulation algorithm.

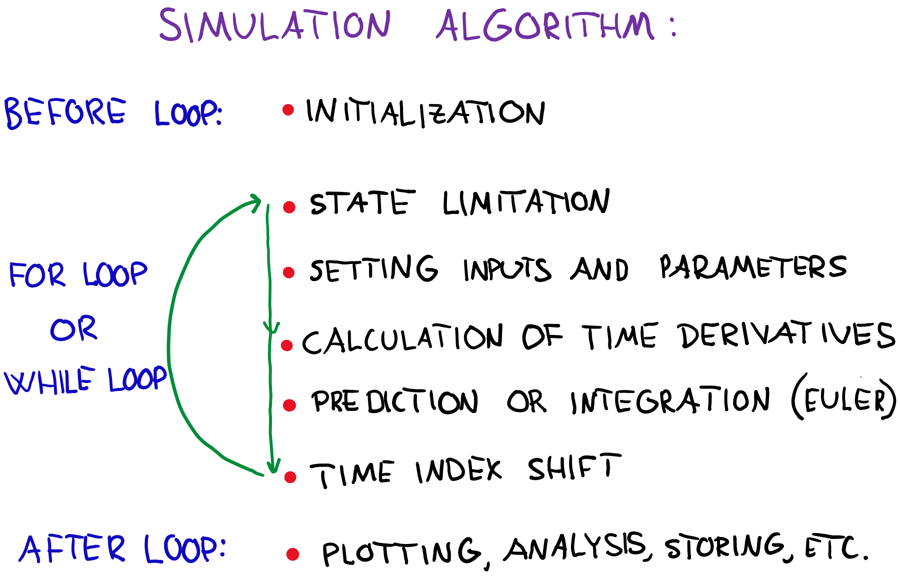

SIMULATION ALGORITHM:
BEFORE LOOP:
• INITIALIZATION
FOR LOOP
OR
WHILE LOOP
• STATE LIMITATION
• SETTING INPUTS AND PARAMETERS
• CALCULATION OF TIME DERIVATIVES
• PREDICTION OR INTEGRATION (EULER)
• TIME INDEX SHIFT
AFTER LOOP:
• PLOTTING, ANALYSIS, STORING, ETC.

Figure 7.2: Simulation algorithm.

The following assumptions are made for the simulation algorithm based on (7.1):

1. The time-step of the algorithm is specified as t_s [s]. (How to select t_s will be discussed below.)
2. At the present time t_k, the following have known values:
   (a) The state, x_k.

(b) The input, u_k.

(c) The disturbance, d_k.

(d) The parameter, p_k.

With these assumptions, we can predict x_{k+1} by integrating x'_k given by (7.1) between t_k and t_{k+1}:

$$x_{k+1} = x_k + \int_{t_k}^{t_{k+1}} {x_k}'\, dt \tag{7.2}$$

where ${x_k}'$ is

$${x_k}' = f\left(x_k, u_k, d_k, p_k\right) \tag{7.3}$$

The simulation program running on a computer must calculate (7.2) *numerically*. There are many ways to do that. It is my experience from a long life with models and simulations that the Euler Forward method of calculating (7.2) numerically is sufficient in, by far, the most cases, and I will therefore concentrate on that method here.

The Euler Forward regarded as an integration method

The Euler Forward method implies that the integrand of (7.2) is kept constant during the integration time interval. Thus,

$$x_{k+1} = x_k + t_s x'_k \tag{7.4}$$

where ${x_k}'$ is given by (7.3). This corresponds to approximating the integral with rectangle integration, see Figure 7.3.

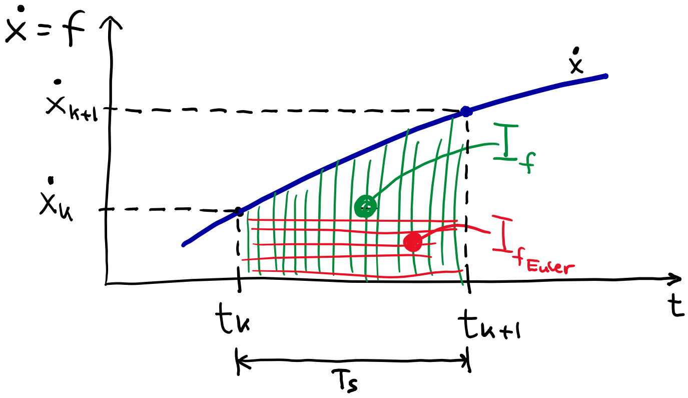


Figure 7.3: Euler Forward approximation, or rectangle approximation, of the integral of x' between t_k and t_{k+1}.

The Euler method regarded as a prediction method

Another graphical interpretation of the Euler Forward method is shown in Figure 7.4. x_{k+1} is predicted by assuming that the slope at time t_k is constant until t_{k+1}.

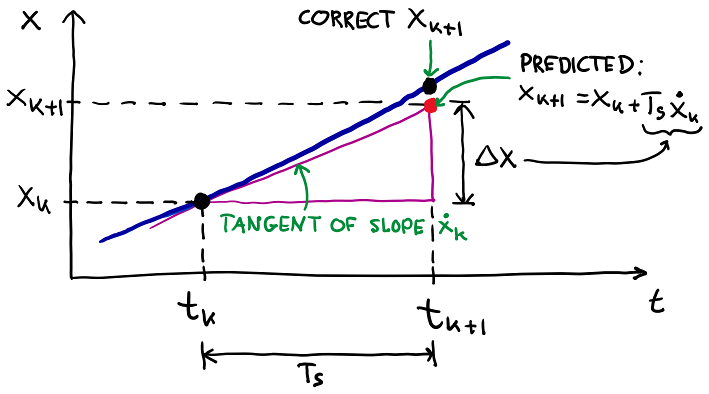


Figure 7.4: Graphical interpretation of the Euler Forward method: x_{k+1} is predicted by assuming that the slope at time t_k is constant until t_{k+1}.

Summary of the similation algorithm:

Before the simulation loop:

- Initializations: Time settings, parameter values, preallocation of arrays for storing data.
- In particular, initialization of the state variable:

$$x_k = x_{\text{init}} \tag{7.5}$$

***Inside the simulation loop*:**

- Assuming state variable x_k is known from initialization or from previous iteration of simulation loop:
  - State limitation: x_k must be limited between $x_{\text{min}}$and x_{max} to avoid unrealistic values (e.g. a negative liquid level):

$$x_k \in [x_{\text{min}},\ x_{\text{max}}] \tag{7.6}$$

- Assuming that the following quantities have known values at time t_k:
  - Set the inputs:
    - Control variable (or manipulating variable) u_k.
    - Disturbance (or load or environmental variable) d_k.
  - Set the parameter p_k.
- Apply x_k for storing in an array[1] for later plotting or analysis, signal processing, file saving, etc.

[1] which should be preallocated to save computational time

- Calculate the time derivative at time t_k:

$$x_k{}' = f\left(x_k, u_k, d_k, p_k\right) \tag{7.7}$$

- Calculate the state prediction for the next time step, t_{k+1}, i.e. the Euler integration step:

$$x_{k+1} = x_k + t_s\, x_k{}' \tag{7.8}$$

- Make a time index shift of x (to prepare for the next iteration of the algorithm):

$$x_k = x_{k+1} \tag{7.9}$$

After the simulation loop:

- Plotting, analysis, saving data to file, etc.

When presenting a simulation algorithm briefly, you may present it in terms of (7.5), (7.7), (7.8), and (7.9).

Example 7.1 *Simulation algorithm of temperature of a kettle*

We will now develop a simulation algorithm of the kettle presented in Ch. 39.6. For your convenience, some of the information is repeated below.

Figure 7.5 shows a sketch of the kettle.

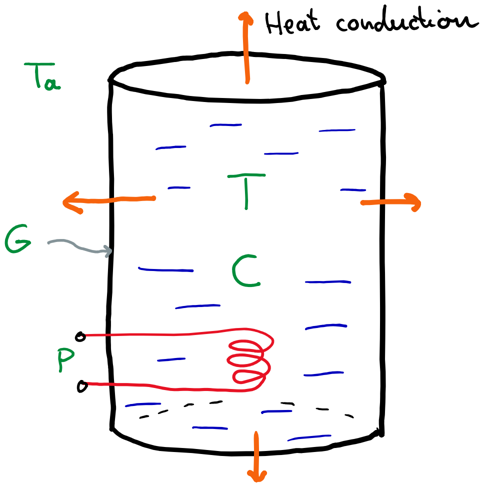


Figure 7.5: Sketch of a kettle.

The mathematical model to be used in the simulator is the following differential equation which stems from thermal energy balance of the water in the kettle:

$$T'(t) = \{P(t - t_d) + G[T_a(t) - T(t)]\}/C \tag{7.10}$$

where:

- $C = 2101$ J/K is the heat capacity of water in kettle.
- $G = 2.34$ W/K is the thermal conductivity of plastic jacket.
- $T_a = 20$ °C is the ambient (room) temperature.

Now, I will change the model a little to make the presentation in the following sections simpler. The time delayed power, $P(t - t_d)$, will be represented with $P_d(t)$ where subindex d means "time delayed". Now, (7.10) becomes

$$T' = [P_d + G(T_a - T)]/C \tag{7.11}$$

where I have dropped the time argument t for simplicity. So, now the time delay is included in the model implicitly as out input variable is $P_d(t)$. However, in Problem 7.2 the time delay of P should be included in the model explicitly, using the method of a "moving array" to represent the time delay.

Figure 7.6 shows an overall block diagram with main variables and parameters of the kettle.

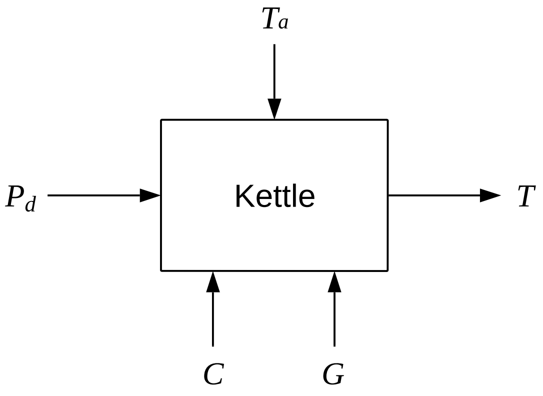


Figure 7.6: Overall block diagram of the kettle.

Although we will not deal with the time delay explicitly, In Problem 7.2 However, the time delay is not ne

The initial temperature is

$$T_{\text{init}} = 20°\text{C} \tag{7.12}$$

The supplied power will be changed as a step from 0 to 700 W at time $t = 0$ s.

The core of the simulation algorithm is:

- The calculation of the time derivative at time index k:

$$T_k{}' = \left[P_d + G\left(T_a - T\right)\right]/C \tag{7.13}$$

- The state prediction (the Euler step) at time index $k + 1$:

$$T_{k+1} = T_k + t_s \, T_k{}' \tag{7.14}$$

The time step will be set as

$$t_s = 1.0 \text{ s}$$

but later in this chapter we will observe what happens if it is set too large.

Python code

Python program 7.1 implements the simulator.

http://techteach.no/control/python/sim_kettle.py

Listing 7.1: sim_kettle.py

```python
"""
Simulation of kettle
Finn Aakre Haugen, TechTeach. finn@techteach.no
2022 12 31
"""

# %% Import of packages:

import matplotlib.pyplot as plt
import numpy as np

# %% Model parameters:

Ta = 20  # [oC]

# %% Derived parameters:

C = 2101  # [J/K] Heat capacity of water in kettle
G = 2.34  # [W/K]  Thermal conductivity

# %% Simulation time settings:

ts = 1.0  # [s]
t_start = 0  # [s]
t_stop = 400  # [s]
N_sim = int((t_stop - t_start)/ts) + 1  # Num time-steps
```

```
# %%  Params of input signals:

Pd_on = 700  # [W]
Pd_off = 0  # [W]

# %% Preallocation of arrays for plotting:

t_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
Ta_array = np.zeros(N_sim)
Pd_array = np.zeros(N_sim)

# %% State limits:

T_min = 0  # [oC]
T_max = 100  # [oC]

# %% Initialization:

T_k = T_init = 20.0  # [oC]

# %% Simulation loop:

for k in range(0, N_sim):

    # Limitation of state:
    T_k = np.clip(T_k, T_min, T_max)

    # Time:
    t_k = k*ts

    # Setting input:
    if (0 <= t_k <= 0):
        Pd_k = Pd_off
    else:
        Pd_k = Pd_on

    Ta_k = Ta

    # Time derivative:
    dT_dt_k = (Pd_k + G*(Ta - T_k))/C

    # Euler step (prediction):
    T_kp1 = T_k + ts*dT_dt_k

    # Updating arrays for plotting:
    t_array[k] = t_k
    T_array[k] = T_k
    Ta_array[k] = Ta_k
    Pd_array[k] = Pd_k

    # Time index shift:
    T_k = T_kp1
```

```
# %% Plotting:

plt.close(’all’)
plt.figure(1)

plt.subplot(2, 1, 1)
plt.plot(t_array, T_array,’b’, label=’T’)
plt.plot(t_array, Ta_array,’g’, label=’Ta’)
plt.legend()
plt.grid()
plt.ylabel(’[deg C]’)

plt.subplot(2, 1, 2)
plt.plot(t_array, Pd_array, ’r’, label=’Pd’)
plt.legend()
plt.grid()
plt.xlabel(’t [s]’)
plt.ylabel(’[W]’)

plt.savefig(’plot_sim_kettle.pdf’)
plt.show()
```

Figure 7.7 shows a plot from the simulation.

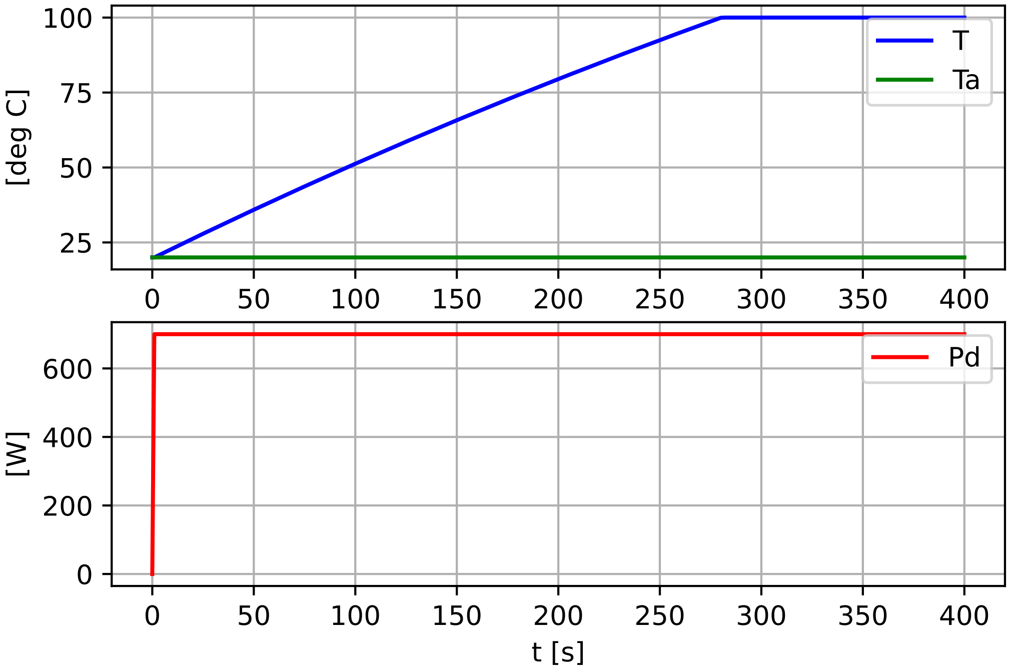


Figure 7.7: Plots from the simulation

[End of Example 7.1]

7.2.3 How to test the simulator

The simulations that we run in the previous sections seems reasonable. If the responses had looked strange, we would of course have tried to search for the error and made sure to fix it. But regardless of whether the responses look correct or incorrect, we should *test* the simulator with our own test scenarios, possibly by checking whether the simulations we have already carried out are in accordance with manual calculations from the model. In general, we should test both static simulations and dynamic simulations.

7.2.3.1 Static test of the simulator

In static testing, we assume that the system is influenced by constant input signals and that all variables in the model have constant values, which means that time derivatives are zero. We manually calculate the static response. Hopefully this this value is equal to the static value from a simulation.

Manual calculation

Let's calculate T_{static} manually. We set the supplied power P_d so low that T does not reach the limit of 100 degrees C. $P_d = 100$ W is ok. We set $T'_{\text{static}} = 0$ in the model (7.11), which then becomes

$$0 = P_d + G\,(T_a - T_{\text{static}}) \tag{7.15}$$

which gives

$$T_{\text{static}} = T_a + \frac{P_d}{G} = 20 + \frac{100}{2.34} = 62.7\ ^\circ\text{C} \tag{7.16}$$

Simulation

Figure 7.8 shows the simulated response with $t_{\text{stop}} = 8000$ s. T is approximately static at the end of the simulation. The simulation is run with the following Python program.

http://techteach.no/control/python/sim_kettle_static_test.py

In the program, the final value of T, which we assume is the static value, is found with the Python code T_array[–1]. The result is 62.7°C, the same as the manually calculated value. We can therefore conclude that it seems that the simulator is correctly implemented as far as static simulations are concerned.

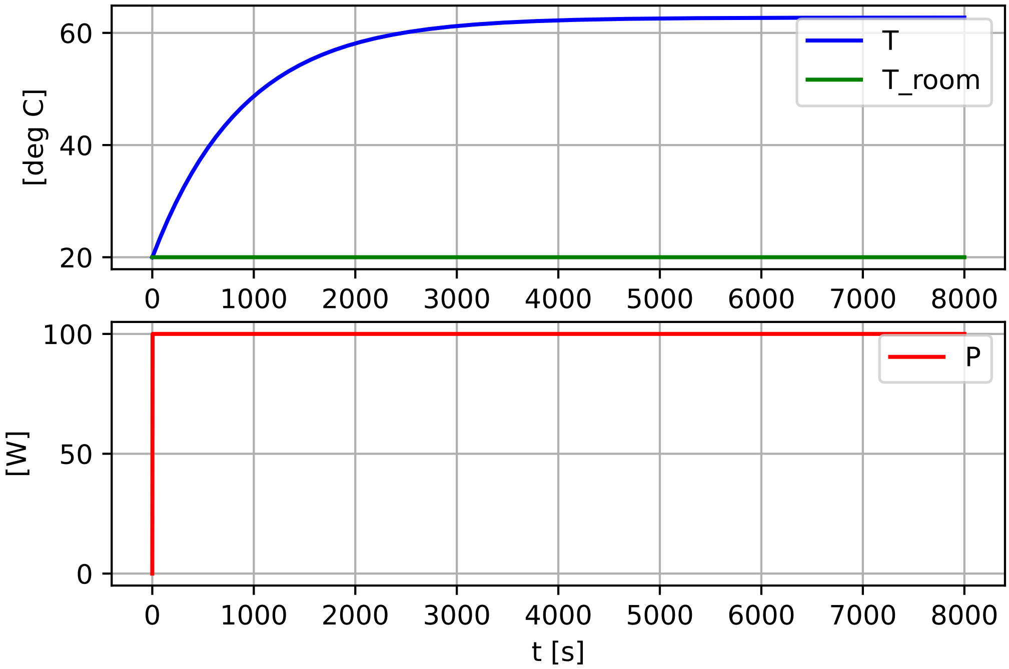


Figure 7.8: Simulation of the kettle until an approximate static state is obtained.

7.2.3.2 Dynamic test of the simulator

In a dynamic test of the simulator, we ensure that T' is different from zero, and then we check whether the simulated T' agrees with manually calculated T'.

Manual calculation

Let's manually calculate T' just after $t = 0$, i.e. $T'(0)$. From model (7.11) we get:

$$T'(0) = \frac{1}{C}\left[P_d(0) + G\left(T_a - T(0)\right)\right] \tag{7.17}$$

Let us assume that $T(0) = T_a$ and that the supplied power, P_d, is a step from 0 to 100 W. We then get:

$$T'(0) = \frac{P_d(0)}{C} = \frac{100}{2101} = 0.0476\,^{\circ}\mathrm{C/s} \tag{7.18}$$

Simulation

Due to the heat loss through the plastic jacket, $T(t)$ starts to flatten immediately after $t = 0$. If we are to read off $T'(0)$ graphically, we should therefore simulate with a very small

t_{stop}. Let's set $t_{\text{stop}} = 1$ s, and use s very small simulation time step, namely $dt = 0.01$ s. The simulator is implemented in the following Python program.

http://techteach.no/control/python/sim_kettle_dynamic_test.py

Figure 7.9 shows the simulated step response. We can now read off $T'(0)$ manually from the plot. However, it is more accurate to use Python code to find $T'(0)$. You may use the following code (which is already in the program above):

(np.diff(T_array)/dt)[1]

The result of the above expression is 0.0476 °C/s, which is the same as with manual calculations. We can therefore conclude that it seems that the simulator is correctly implemented as far as dynamic simulations are concerned.

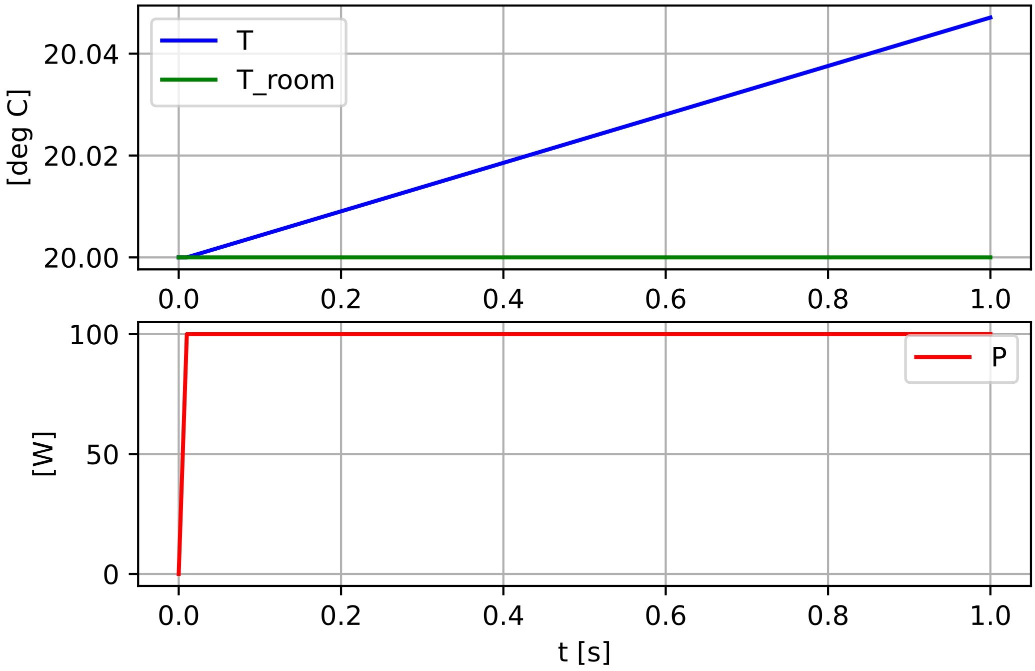


Figure 7.9: Simulation of the kettle with a very short stop time (1 sec) to find the initial slope, $T'(0)$.

7.2.3.3 Are static and dynamic tests sufficient to conclude?

We have now successfully accomplished both a static and a dynamic test of the simulator of the kettle. However, these test does *not* prove that the simulator is correctly implemented, although there are strong indications that it is. Maybe we just had luck with our test scenarios!

7.2.4 How to choose the simulation time step?

As part of the discretization, we must select an appropriate value of the time step, t_s, which specifies the resolution along the simulation time axis. As illustrated in Figure 7.10, a "small" t_s provides greater (better) resolution along the simulation axis – which is a benefit. On the other hand, the computational demand on the computer is larger since it has to perform more calculations (more frequently), and also produce more simulated data.

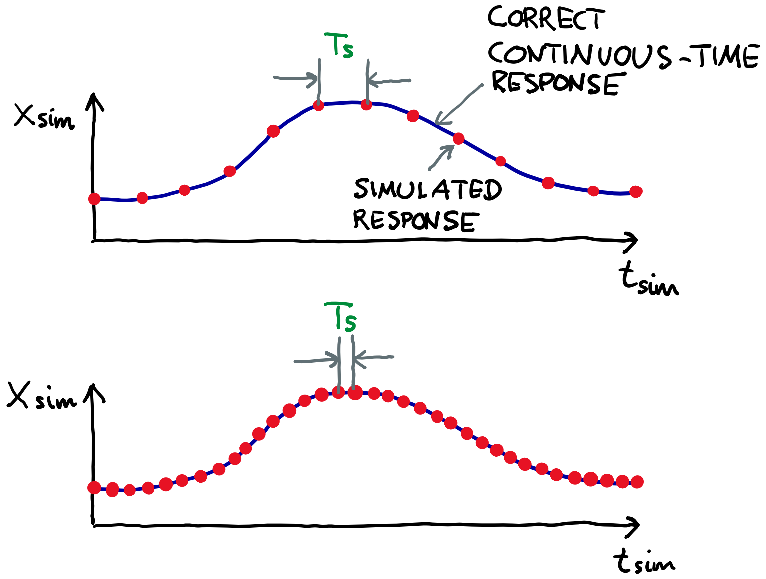


Figure 7.10: The importance of the time step t_s in simulations. Top: Relatively large t_s. Bottom: Relatively small t_s.

If we are so unfortunate to choose t_s too large, the simulation algorithm itself may become unstable, so that the simulated responses "take off" and become completely different from the correct response. This is illustrated in Figure 7.11.

A general rule is to select t_s as large as possible (to minimize both the computation demand and the data generated during the simulation), but the simulation must be *insensitive* to t_s. The simulation is insensitive to t_s is the simulated response is approximately the same with a somewhat larger t_s and with a somewhat smaller t_s.

You must expect some trial and error to find a suitable t_s. If you have some insight into how fast or slow the simulated system is, I think you can get a reasonable value of t_s by

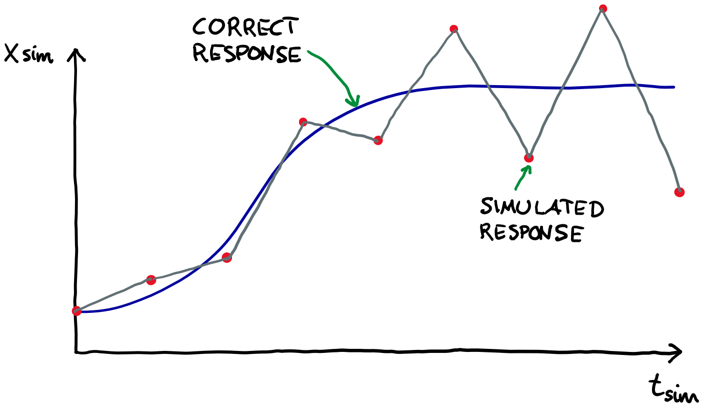


Figure 7.11: With a large dt, the simulation algorithm may become unstable.

following the following guidelines:

- Select t_s so that the simulation gives an accurate expression of the responses as they would actually have been in the original physical system (to be simulated).
- For a fast system, select a relatively small t_s.
- For a slow system, you can choose a relatively large t_s, but a small t_s will also work well, except that the computer will have a larger computational demand during the simulation, and more simulation data will be produced for plotting and/or file storage.

If you know the approximate time constant, t_c, of the system to be simulated, cf. Ch. 9.3, you can try

$$t_s = t_c/100 \tag{7.19}$$

If you have no idea what to choose as t_s, you can start with

$$t_s = 0.01 \text{ s} \tag{7.20}$$

and then make some adjustment of t_s if necessary.

Example 7.2 *How the simulation depends on the time step*

It can be shown that the time constant of the kettle is approx. 900 s. From (7.19), a reasonable t_s is then $t_s = 900/100 = 9$ s. A simulation with $t_s = 9.0$ s gives a response which is very similar to the response with $t_s = 1.0$ s which is already shown in Figure 7.7. Instead of showing the response with $t_s = 9.0$ s, let's do some more exciting changes, namely setting t_s to considerably larger values.

In the program linked to below the simulation time step is set relatively large, namely $t_s = 1200$ s. To avoid saturations and the consequently clipping of the temperature response, the supplied power is set as 100 W instead of 700 W as in earlier simulations.

Inaccurate simulation

http://techteach.no/control/python/sim_kettle_dt_inaccurate.py

Figure 7.12 shows the step response in the temperature, T. The temperature oscillates someehat, which is a result of numerical inaccuracy in the simulation algorithm due to the large t_s. With large t_s the prediction of T_{k+1} based on T'_k becomes inaccurate.

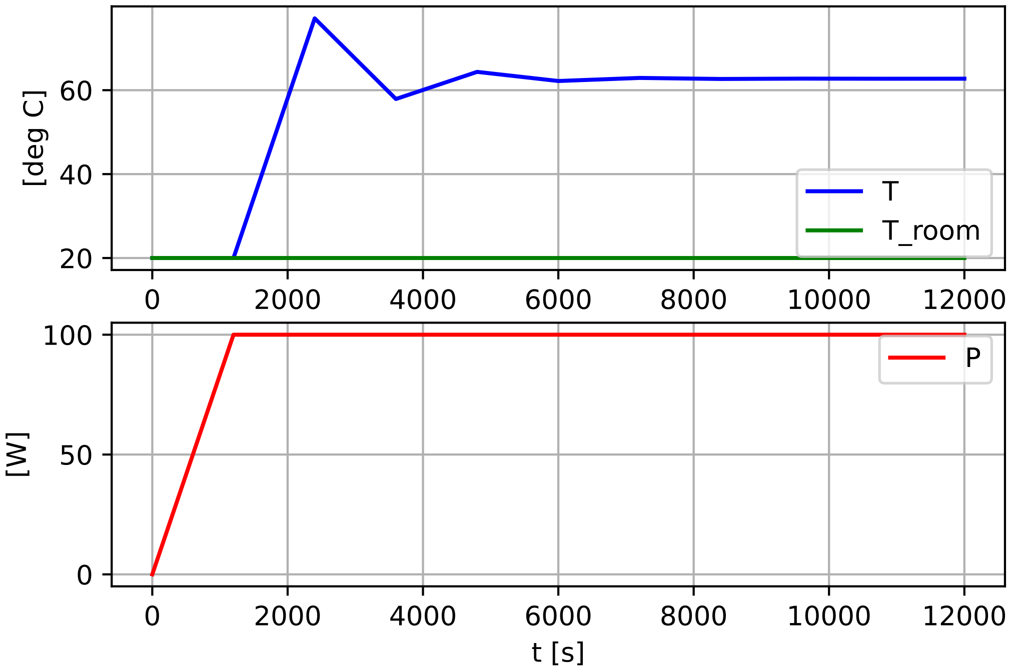


Figure 7.12: Step response in temperature, T, when simulation time step is $t_s = 1200$ s. The simulation is numerically inaccurate.

Unstable simulation

Let's see what happens with an even larger time step, namely $t_s = 2000$ s. The following program runs the simulation.

http://techteach.no/control/python/sim_kettle_dt_unstable.py

Figure 7.13 shows the step response in T. The simulation oscillates with larger and larger amplitude – the simulation is numerically unstable, while the kettle itself is stable, of course!

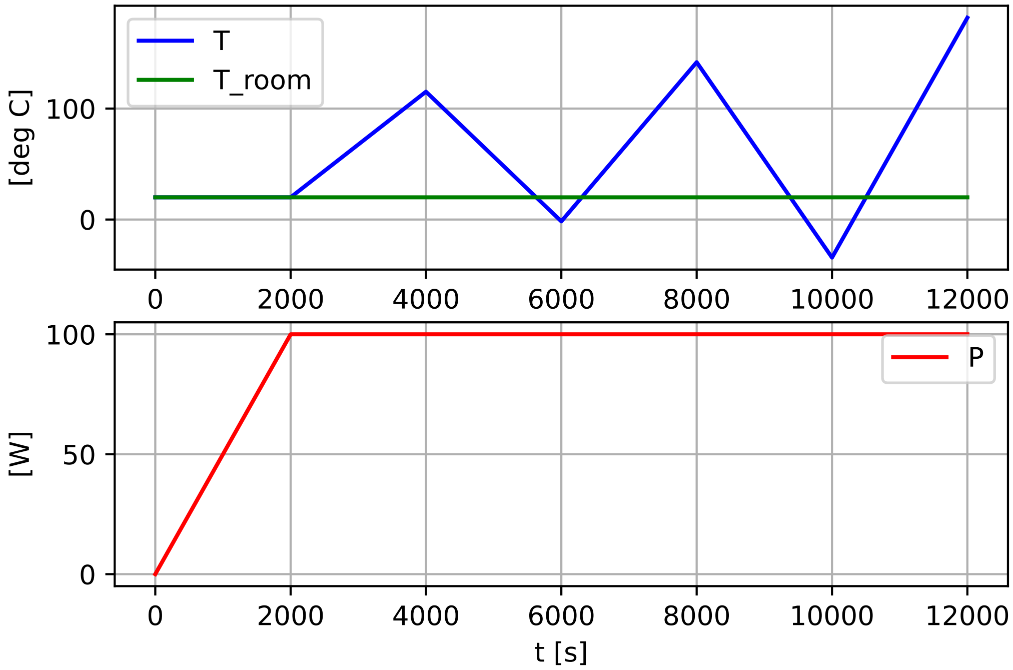


Figure 7.13: Step response in temperature T with time step is $t_s = 2000$ s. The simulation is numerically unstable!

[End of Example 7.2]

7.2.5 Simulation along real time or scaled real time?

If you want a "live" simulation – which can be very effective educationally, you need to run the simulation in real time so you can see that the simulation progresses as the simulated time runs. Also, if you want the simulator to run in parallel with the real system, maybe for monitoring purposes, you need to run the simulator in real time.

However:

- For systems that are very slow, real-time simulation will hardly be popular. Then it is better with scaled real time, so that the simulation runs eg. 100 times faster than real time.

- For systems that are very fast, real-time system is of little uses since there will be no time to experience the live simulation. In this case it may be suitable to run the simulation for example 100 times slower than real time.

Figure 7.14 illustrates the above (it is assumed 2 times faster and 2 times slower than real time, respectively).

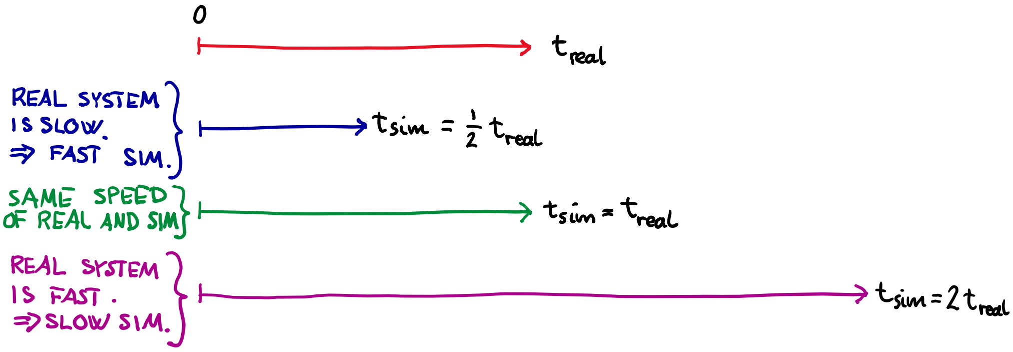

0
t_{real}
REAL SYSTEM IS SLOW. ⇒ FAST SIM.
$t_{sim} = \frac{1}{2} t_{real}$
SAME SPEED OF REAL AND SIM
$t_{sim} = t_{real}$
REAL SYSTEM IS FAST. ⇒ SLOW SIM.
$t_{sim} = 2 t_{real}$

Figure 7.14: Real-time, or scaled real-time simulation

How do you implement real-time, or possibly, scaled real-time simulation? Here is a procedure:

1. Select a proper value of the simulation time step, t_s. Note: The selection of dt shall be independent of the speed of the simulator!

2. If you want the simulator to run k times faster than real time, then ensure to implement the real cycle time, or iteration time, of your simulation algorithm, $t_{\text{cycle,real}}$, according to
$$t_{\text{cycle,real}} = \frac{t_s}{k} \tag{7.21}$$
How to implement $t_{\text{cycle,real}}$, depends on the programming tool you are using, and I will not discuss this further here.

7.2.6 Why predict?

Why calculate x_{k+1} when we actually need x_k? After all, it is x_k we need. So why calculate the predicted value x_{k+1}? Why not just subtract 1 from all of the time indexes in the algorithm above to get a formula for x_k instead of x_{k+1}? Let us try. Then (7.7) becomes

$$x_{k-1}{}' = f\left(x_{k-1}, u_{k-1}, d_{k-1}, p_{k-1}\right) \tag{7.22}$$

and (7.7) becomes

$$x_k = x_{k-1} + t_s \, x_{k-1}{}' \tag{7.23}$$

While this may look ok as the basis of a simulation algorithm, a drawback is that the delayed values u_{k-1} and d_{k-1} and p_{k-1} are required. It is much more common to use the "prediction" algorithm (7.7) – (7.8).

7.2.7 Euler Forward vs. Euler Backward

The approximation (7.8), on which we have based the simulation algorithm, is the Euler Forward approximation of the time-derivative of x. As you may have guessed, or know already, there is also the Euler Backward method of calculating (approximating) the integral of (7.2):

$$x_k = x_{k-1} + t_s \, {x_k}' = x_{k-1} + t_s f\left(x_k, u_k, d_k, p_k\right) \tag{7.24}$$

Actually, if the function $f()$ is *linear* in x_k we will certainly be able to obtain a formula for x_k by solving (7.24) for x_k. This is actually the Euler Backward method. However, often process models are nonlinear, i.e. $f()$ is a nonlinear function of x_k, and then we may get problems. As an example, assume the following model:[2]

$$x' = -K_v\sqrt{x} \tag{7.25}$$

Then, (7.24) becomes:

$$x_k = x_{k-1} + t_s \, {x_k}' = x_{k-1} - t_s K_v\sqrt{x_k} \tag{7.26}$$

Do you see the problem? It is that x_k appears at both sides of the nonlinear equation (7.26), making it difficult (but not impossible) to calculate x_k, which we need in the simulation algorithm! With the Euler Forward method, the problem does not even appear – not for any function $f()$.

It can be shown that the Euler Forward method is somewhat less accurate than the Euler Backward method. But if we choose a small enough time step, t_s, the two methods give almost identical results.

Above, we have actually got a demonstration of why the Euler Backward method is also called the Euler Implicit method: (7.26) gives an implicit solution of x_k. And the Euler Forward method is also denoted the Euler Explicit method.

7.3 Simulation of second order differential equation models

In Section 7.2, the simulation algorithm assumed a first order differential equation. What if the model consists of a second order differential equation, like when you apply the Newton's Second Law to model the motion of a mechanical system?[3] One common way to obtain a simulation algorithm for second order differential equation is to represent this differential equation by two first order differential equations, and applying the Euler method to each of them. This procedure is demonstrated in Example 7.3 where we develop a simulation

[2] A model of a water tank with a valve in the outlet may be modelled like this. The outflow through the valve is the cause of the square root function.

[3] Differential equation of higher order than two are very rare. You may meet them in e.g. signal processing and state estimator design, but such applications are not relevant here.

algorithm for a mass-spring-damper system. Both the speed and the position of the system will be simulated.

Example 7.3 *Simulator of a mass-spring-damper*

Figure 7.15 shows a mass-spring-damper system. The damping force F_d is assumed proprtional with the speed (i.e. so-called viscous damping is assumed):

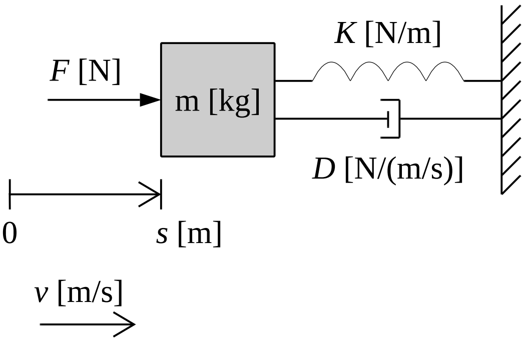


Figure 7.15: Mass-spring-damper system.

$$F_d = -Dv \tag{7.27}$$

The spring force is assumed proportional with the position of the body relative to position $s = 0$:

$$F_s = -Ks \tag{7.28}$$

The Newton's 2nd law (force balance) is:

$$ms'' = mv' = F - F_d - F_s = F - Dv - Ks \tag{7.29}$$

The state space model is:

$$s' = v \tag{7.30}$$

$$v' = \frac{1}{m}\left[F - Dv - Ks\right] \tag{7.31}$$

The simulation algorithm is:

Before the simulation loop:

- Initialization:

$$s_k = s_{\text{init}} \tag{7.32}$$

$$v_k = v_{\text{init}} \tag{7.33}$$

Inside the simulation loop:

- State limitation:

$$s_k \in [s_{\min},\ s_{\max}] \tag{7.34}$$

$$v_k \in [v_{\min},\ v_{\max}] \tag{7.35}$$

- Time derivatives (the differential equations):

$${s_k}' = v_k \tag{7.36}$$

$${v_k}' = \frac{1}{m}\left[F_k - Dv_k - Ks_k\right] \tag{7.37}$$

- Euler integration:

$$s_{k+1} = s_k + t_s \cdot {s_k}' \tag{7.38}$$

$$v_{k+1} = v_k + t_s \cdot {v_k}' \tag{7.39}$$

- Time index shift:

$$s_k = s_{k+1} \tag{7.40}$$

$$v_k = v_{k+1} \tag{7.41}$$

Program 7.2 shows an implementation of the simulator, including various numerical settings (I do not show these values in the text). Figure 7.16 shows simulated responses due to a step change of the applied force F.

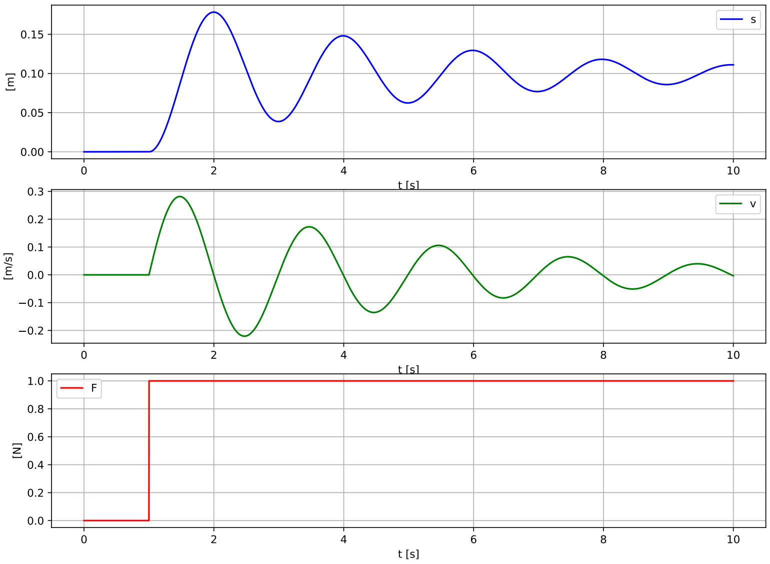


Figure 7.16: Simulation of a mass-spring-damper.

http://techteach.no/control/python/prog_sim_mfd.py

Listing 7.2: prog_sim_mfd.py

```
import matplotlib.pyplot as plt
import numpy as np

m = 1  # [kg]
K = 10  # [N/m]
D = 0.5  # [N/(m/s)]

# For state limitation, but not effective here.
s_min = -np.inf; s_max = np.inf,
v_min = -np.inf; v_max = np.inf

Ts = 0.001  # [s]
t_start = 0  # [s]
t_stop = 10  # [s]
N_sim = int((t_stop - t_start)/Ts) + 1

t_array = np.zeros(N_sim)
s_array = np.zeros(N_sim)
v_array = np.zeros(N_sim)
F_array = np.zeros(N_sim)

s_k = s_init = 0  # [m]
v_k = s_init = 0  # [m/s]

for k in range(0, N_sim):

    t_k = k*Ts

    if 0 <= t_k <= 1: F_k = 0  # [N]
    else: F_k = 1  # [N]

    s_k = np.clip(s_k, s_min, s_max)
    v_k = np.clip(v_k, v_min, v_max)

    ds_dt_k = v_k
    dv_dt_k = (1/m)*(F_k - D*v_k - K*s_k)

    s_kp1 = s_k + Ts*ds_dt_k
    v_kp1 = v_k + Ts*dv_dt_k

    t_array[k] = t_k
    s_array[k] = s_k
    v_array[k] = v_k
    F_array[k] = F_k

    s_k = s_kp1
    v_k = v_kp1

plt.close(’all’)
plt.figure(1, figsize=(12, 9))
plt.subplot(3, 1, 1)
plt.plot(t_array, s_array, 'b', label='s')
plt.xlabel('t [s]')
plt.ylabel('[m]')
plt.grid()
plt.legend()

plt.subplot(3, 1, 2)
plt.plot(t_array, v_array, 'g', label='v')
plt.grid()
plt.xlabel('t [s]')
plt.ylabel('[m/s]')
plt.legend()

plt.subplot(3, 1, 3)
plt.plot(t_array, F_array, 'r', label='F')
plt.grid()
plt.xlabel('t [s]')
plt.ylabel('[N]')
plt.legend()

# plt.savefig('plot_sim_mfd.pdf')
plt.show()
```

[Slutt på eksempel 7.3]

7.4 Simulation algorithm of time delays

Time delays – also denoted transportation time and time delay – appear in various systems:

- Transportation of material on conveyor belt or through pipelines.
- Apparent time delay due to imperfect mixing in tanks, e.g. a delayed response in the measured temperature of liquid in a tank after some change in the supplied power to the liquid.
- Delay of information through communication channels.

How to make a simulation algorithm of a time delay? Let us make a simulator of the transportation taking place on the conveyor belt shown in Figure 7.17. The belt runs with fixed speed.

There is a time delay from inflow F_{in} to the belt to the outflow F_{out} from the belt. Example: Suppose the time delay is 45 seconds. A change in the inflow at 12:30:00 will then give the same change in the outflow at 12:30:45. Mathematically, the relationship between F_{in} and F_{out} can be expressed as:

$$F_{\text{out}}(t) = F_{\text{in}}(t - t_d) \tag{7.42}$$

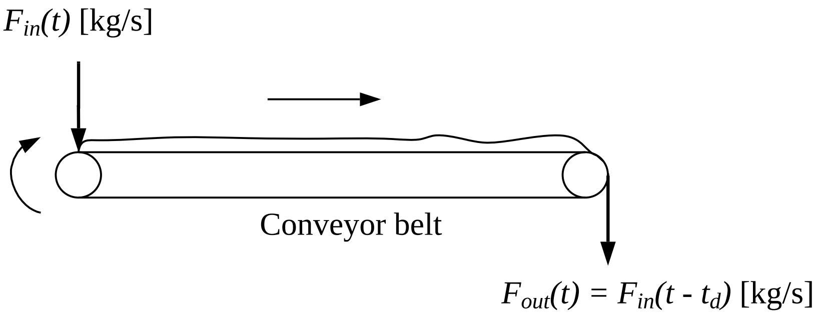


Figure 7.17: Conveyor belt.

where $t_d = 45$ s is the time delay (transportation time).

Let us imagine that the conveyor belt in Figure 7.17 is represented by an *array*, where the input signal to the array is put into the first element of the array, and the time delayed signal is taken out from from the last element of the array. At each discrete time in the simulation, the value of each element in the array is moved one step toward the end of the array. This is illustrated in Figure 7.18 where y_{in} is the input to the array and y_{out} is the output of the array. The mathematical relationship between y_{out} and y_{in} is

$$y_{\text{out}}(t) = y_{\text{in}}(t - t_d) \tag{7.43}$$

What should be the length of the array? If the time delay is t_d [s] and the simulation time step is dt [s], the length of the array should be

$$N_f = \frac{t_d}{t_s} \tag{7.44}$$

or rounded upwards to the nearest integer if this fraction is not an integer (rounding upwards is safer than rounding downwards assuming it is safer to overstate the time delay than to understate it).

We can now state a simulation algorithm as follows. The algorithm is executed at each point of simulation time.

1. The time delayed signal, $y_{\text{out}}(t) = y_{\text{in}}(t - t_d)$, is the value of the last element of the array.
2. In principle, all the elements of the array are moved one step towards the end of the array. (The program example below shows a way to implement the move, but there are other ways.)
3. The value of the variable to be delayed, $y_{\text{in}}(t)$, is entered into the first element of the array.

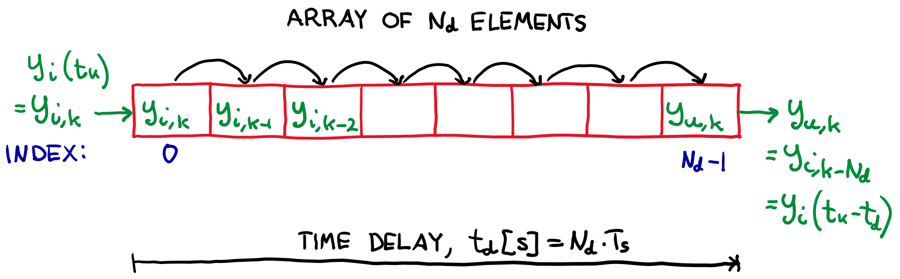


Figure 7.18: Realization of a time delay using an array. At each simulation point of time, all the elements of the array are moved one step toward the end of the array.

Example 7.4 *A Python program for simulation of a time delay*

Python program 7.3 implements the simulator of a time delay.

http://techteach.no/control/python/prog_sim_time_delay.py

Listing 7.3: prog_sim_time_delay.py

```
# %% Import of packages:

import matplotlib.pyplot as plt
import numpy as np

# %% Model parameters:

t_delay = 1.0   # [s]

# %% Simulation time settings:

Ts = 0.01   # [s]
t_start = 0   # [s]
t_stop = 5   # [s]
N_sim = int((t_stop - t_start)/Ts) + 1

# %% Preallocation of arrays for plotting:

t_array = np.zeros(N_sim)
y_in_array = np.zeros(N_sim)
y_out_array = np.zeros(N_sim)

# %% Initialization:

Y_out_init = 0.5

# %% Preallocation of array for time-delay:

Nd = int(round(t_delay/Ts)) + 1
delay_array = np.zeros(Nd) + Y_out_init
```

```
# %% Simulation loop:

for k in range(0, N_sim):

    t_k = k*Ts

    # Excitation:
    if (t_start <= t_k < 2.0):
        y_in_k = 0
    if (2.0 <= t_k):
        y_in_k = 1.0

    # Time delay:
    y_out_k = delay_array[-1]
    delay_array[1:] = delay_array[0:-1]
    delay_array[0] = y_in_k

    # Writing values to arrays for plotting:
    t_array[k] = t_k
    y_in_array[k] = y_in_k
    y_out_array[k] = y_out_k

# %% Plotting:

plt.close(’all’)
plt.figure(1)
plt.grid()
plt.plot(t_array, y_out_array, ’b’)
plt.plot(t_array, y_in_array, ’r’)
plt.xlabel(’t [s]’)
plt.legend(labels=(’y_out’, ’y_in’))

plt.savefig(’plot_sim_time_delay.pdf’)
plt.show()
```

The specifications of the simulator are as follows:

- Time step: $t_s = 0.01$ s
- Time delay: $t_d = 1$ s
- The input signal, y_{in}, is changed as a step at time $t_0 = 2$ s.
- The array representing the time delay is initially filled with values Y_out_init = 0.5. This causes the time delay value, y_{out}, to have value Y_out_init until the simulation time has become greater than the time delay.

Figure 7.19 shows the results of a simulation with the program.

Comments to the above program:

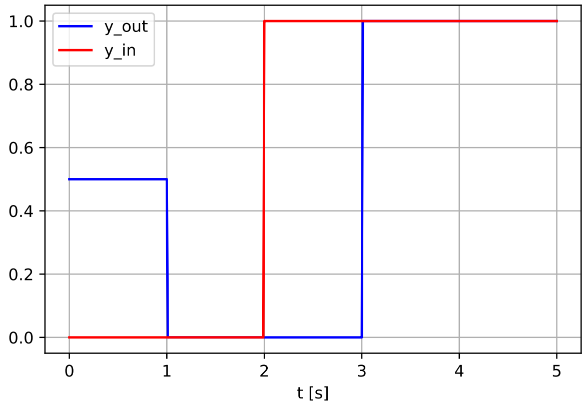


Figure 7.19: Simulation of a time delay.

- The code Y_out_init = 0.5 defines a constant to fill the array with an initial value.
- The code Nd = int(round(td/Ts)) + 1 calculates the number of elements of the array needed to represent the time delay.
- The code delay_array = np.zeros(Nd) + y_out_init creates the array, and gives all the elements the initial value of y_out_init.
- The code beneath # Excitation: generates a step change of the input signal.
- The code beneath # Time delay: realizes the time delay accordiing to the algorithm presented above:
  - *Step 1 of the algorithm:* The code y_out_k = delay_array[-1] assigns the time delayed variable, y_out_k, the value of the last element of the array. Index -1 addresses the last element.
  - *Step 2 of the algorithm:* The code delay_array[1:] = delay_array[0:-1] moves each of the elements one step towards the right. The access code 0:-1 means "all elements but the last one". The access code 1: means "all elements starting with element of index 1 (i.e. the element of index 0 is excluded)".
  - *Step 3 of the algorithm:* The code delay_array[0] = y_in_k assigns the element of index 0 (the leftmost element) of delay_array the value of the input variable, y_in_k.
- The code beneath # Writing values to arrays for plotting: writes values to the arrays to be used for plotting (but the code for plotting is not shown here).

[End of Example 7.4]

7.5 *Problems for Chapter 7*

Problem 7.1 *Simulator of a heated water tank*

A mathematical model of a heated water tank with initial model parameter values is presented in Ch. 39.4. The model (39.7) includes a time delay of the supplied power P. However, in the problems below you can (for simplicity) just represent the time delayed P with P_d, i.e. $P(t - t_d) \equiv P_d(t)$, so you do not have to implement the time delay explicitly in the simulator.

1. "Pseudo" simulation algorithm

Derive a "pseudo" simulation algorithm for the temperature (more or less ready to be implemented in a program).

2. Calculation of static operating point

Calculate from the model the constant power, P_0, needed to bring the temperature to a constant value of 25 deg C.

3. Programming and simulation

Program a simulator in Python of the tank heater. The simulator must be implemented with "native" code in a For loop based on the Euler Forward discretization of the model (a built-in simulation function of Python should not be used). You can set the time-step to 1 s. The following variables should be plotted: T, T_{in}, and T_{env} in one subplot, and P in another subplot. Assume that the initial temperature is $T_{\text{init}} = 20$ deg C. Run a simulation with $P = P_0$ as calculated above. Is the static T the same as specified in Problem 2 above?

4. Stability of the simulator

Demonstrate that the simulator becomes numerically inaccurate if you select a (too) large simulator time step. Also, run a simulation with such a large time step that the simulation becomes numerically unstable.

Problem 7.2 *Simulation of kettle including time delay*

In Example 7.1, the kettle simulator program, http://techteach.no/control/python/sim_kettle.py, does not include any time delay. Now, modify that program to include a time delay of 20 sec in supplied power, P.

Verify with a simulation that you have implemented the time delay correctly.

Problem 7.3 *Simulator of a ship*

A mathematical model of the surge or longitudal motion of a ship is presented in Ch. 39.9. Program a simulator of the ship according to the following specifications:

- Time step is 1 s.
- Stop time is 1000 s.
- Position is initially 0 m, and speed is initially 0 m/s.
- The propeller force is initially 0 kN. It is changed as a step from 0 to 200 kN at time 200 s, and back to 0 kN at time 400 s.
- Water current can be assumed zero.
- Wind speed can be assumed zero.
- Position, speed, propeller force, and hydrodynamic force together with wind force are plotted in individual plots (four plots, altogether).

Hint: Write the model as a second order state space model with the following state variables: Position x_1, and speed x_2.

7.6 *Solutions to problems for Chapter 7*

Solution to Problem 7.1

1. “Pseudo” simulation algorithm

The model is:

$$c\rho VT'(t) = P_d(t) + c\rho F_v\left[T_{\text{in}}(t) - T(t)\right] + G\left[T_a(t) - T(t)\right] \tag{7.45}$$

We write this model as a state space model, and drop the time argument for simplicity:

$$T' = \left[P_d + c\rho F_v\left(T_{\text{in}} - T\right) + G\left(T_a - T\right)\right]/(c\rho V) \tag{7.46}$$

The model simulation algorithm:

- Before the simulation loop:
  - Initialization: T_k = T_init
- Inside the simulation loop:
  - Limitation of T_k between T_min = 0 deg C and T_max = 100 deg C (using e.g. the numpy.clip function)
  - Setting input signals Pd_k, and T_in_k, and T_env_k
  - Any use of T_k, e.g. storing in an array for later plotting
  - Time derivative: dT_dt_k = (1/(c*rho*V))*(Pd_k + c*rho*Fv*(T_in_k - T_k) + G*(T_a_k - T_k))
  - Prediction or integration (Euler forward): T_kp1 = T_k + dt*dT_dt_k
  - Time index shift: T_k = T_kp1
- After the simulation loop:
  - Plotting, analysis, saving simulation data to file, etc.

2. Calculation of static operating point:

Python program 7.4 implements a solution.

http://techteach.no/control/python/sim_heated_tank_1.py

Listing 7.4: sim_heated_tank_1.py

```
#%% Model params:

c = 4200 # [J/(kg*K)]
rho = 1000 # [kg/m3]
V = 0.2 # [m3]
```

```
G = 1000 # [W/K]
Fv = 0.25e-3  # [m3/s]
T_in = 20  # [deg C]
T_a = 20  # [deg C]

#%% Calculation of power giving specified static temp:

T_static = 25  # [deg C]  Static temp

# From model after t’ is set to zero (static value):
Pd_0 = - (c*rho*Fv*(T_in-T_static) + G*(T_a-T_static))  # [W]
print(’Pd_0 [W] =’, Pd_0)
```

The result is:

```
Pd_0 [W] = 10250.0
```

3. Programming and simulation:

Python program 7.5 implements a solution.

http://techteach.no/control/python/sim_heated_tank_2.py

Listing 7.5: sim_heated_tank_2.py

```
#%% Imports:

import numpy as np
import matplotlib.pyplot as plt

#%% Model params:

c = 4200 # [J/(kg*K)]
rho = 1000 # [kg/m3]
V = 0.2 # [m3]
G = 1000 # [W/K]
Fv = 0.25e-3  # [m3/s]
T_in = 20  # [deg C]
T_a = 20  # [deg C]

T_min = 0
T_max = 100

#%% Calculation of power giving specified static temp:

T_static = 25  # [deg C]  Static temp

# From model after T’ is set to zero (static value) in the model:
Pd_0 = - (c*rho*Fv*(T_in-T_static) + G*(T_a-T_static))  # [W]

#%% Sim time settings:

dt = 1  # [s]
```

```
t_start = 0  # [s]
t_stop = 6000  # [s]
N_sim = int((t_stop - t_start)/dt) + 1

#%% Preallocation of arrays for storing:

t_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
T_in_array = np.zeros(N_sim)
T_a_array = np.zeros(N_sim)
Pd_array = np.zeros(N_sim)

#%% Sim loop:

T_k = T_init = 20  # [deg C] Initial temp

for k in range(0, N_sim):

    # State limitation:
    T_k = np.clip(T_k, T_min, T_max)

    t_k = k*dt

    Pd_k = Pd_0
    T_in_k = T_in
    T_a_k = T_a

    t_array[k] = t_k
    T_array[k] = T_k
    T_in_array[k] = T_in_k
    T_a_array[k] = T_a_k
    Pd_array[k] = Pd_k

    # Time derivative:
    dT_dt_k = (Pd_k + (c*rho*Fv)*(T_in-T_k) + G*(T_a_k-T_k))/(c*rho*V)
    T_kp1 = T_k + dt*dT_dt_k

    # Time index shift:
    T_k = T_kp1

# %% Plotting:

plt.close(’all’)
plt.figure(1)

plt.subplot(2, 1, 1)
plt.plot(t_array, T_array, ’r’, label=’T’)
plt.plot(t_array, T_in_array, ’b’, label=’T_in’)
plt.plot(t_array, T_a_array, ’g’, label=’T_a’)
plt.legend()
plt.grid()
plt.xlabel(’t [s]’)
plt.ylabel(’[deg C]’)
plt.subplot(2, 1, 2)
plt.plot(t_array, Pd_array, 'm', label='Pd')
plt.legend()
plt.grid()
plt.xlabel('t [s]')
plt.ylabel('[W]')

plt.savefig('plot_sim_heated_tank_2.pdf')
plt.show()
```

Figure 7.20 shows the simulated responses.

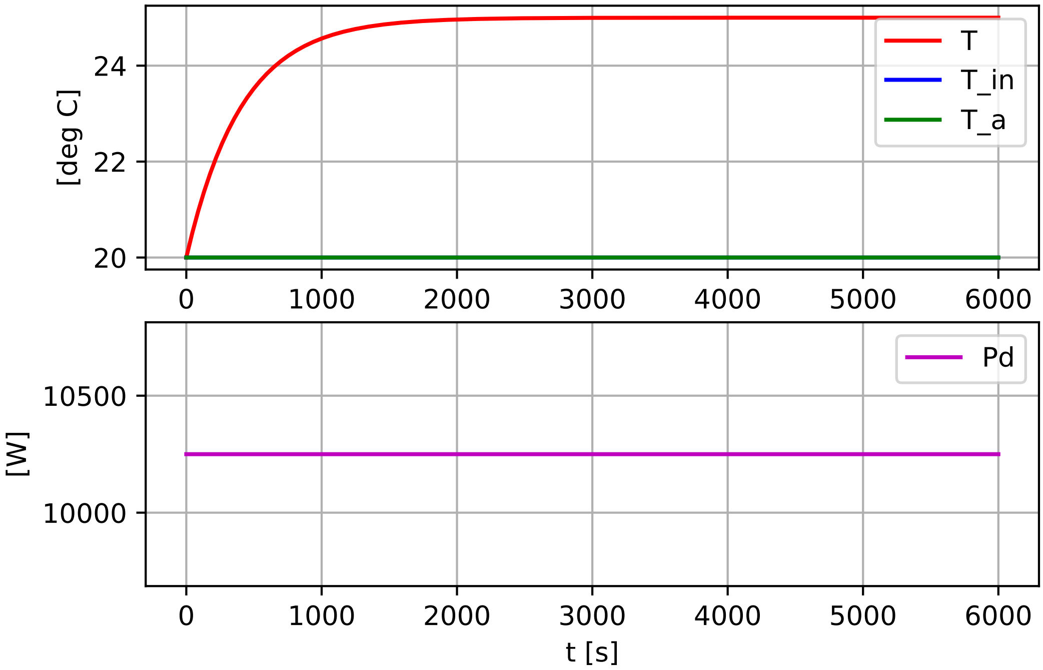


Figure 7.20: Problem 7.1: Simulated responses.

Comments:

- The static value of T can be read off from the plot in Figure 7.20, and can be found more precisely with the code T_array[-1], and is 25.00, which is the same as the specified value, cf. Problem 2.

4. Stability of the simulator:

Python program 7.6 runs a simulation with time step

$$t_s = 700 \text{ s}$$

http://techteach.no/control/python/sim_heated_tank_2.py

Listing 7.6: sim_heated_tank_3.py

```
#%% Imports:

import numpy as np
import matplotlib.pyplot as plt

#%% Model params:

c = 4200 # [J/(kg*K)]
rho = 1000 # [kg/m3]
V = 0.2 # [m3]
G= 1000 # [W/K]
Fv = 0.25e-3  # [m3/s]
T_in = 20  # [deg C]
T_a = 20  # [deg C]

T_min = 0
T_max = 100

#%% Calculation of power giving specified static temp:

T_static = 25  # [deg C]  Static temp

# From model after t’ is set to zero (static value):
Pd_0 = - (c*rho*Fv*(T_in-T_static) + G*(T_a-T_static))  # [W]

#%% Sim time settings:

dt = 700  # [s]
t_start = 0  # [s]
t_stop = 10000  # [s]
N_sim = int((t_stop - t_start)/dt) + 1

#%% Preallocation of arrays for storing:

t_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
T_in_array = np.zeros(N_sim)
T_a_array = np.zeros(N_sim)
Pd_array = np.zeros(N_sim)

#%% Sim loop:

T_k = T_init = 20  # [deg C] Initial temp

for k in range(0, N_sim):

    # State limitation:
    T_k = np.clip(T_k, T_min, T_max)

    t_k = k*dt
    Pd_k = Pd_0
    T_in_k = T_in
    T_a_k = T_a

    t_array[k] = t_k
    T_array[k] = T_k
    T_in_array[k] = T_in_k
    T_a_array[k] = T_a_k
    Pd_array[k] = Pd_k

    # Time derivative:
    dT_dt_k = (Pd_k + (c*rho*Fv)*(T_in-T_k) + G*(T_a_k-T_k))/(c*rho*V)
    T_kp1 = T_k + dt*dT_dt_k

    # Time index shift:
    T_k = T_kp1

# %% Plotting:

plt.close(’all’)
plt.figure(1)

plt.subplot(2, 1, 1)
plt.plot(t_array, T_array, ’r’, label=’T’)
plt.plot(t_array, T_in_array, ’b’, label=’T_in’)
plt.plot(t_array, T_a_array, ’g’, label=’T_a’)
plt.legend()
plt.grid()
plt.xlabel(’t [s]’)
plt.ylabel(’[deg C]’)

plt.subplot(2, 1, 2)
plt.plot(t_array, Pd_array, ’m’, label=’Pd’)
plt.legend()
plt.grid()
plt.xlabel(’t [s]’)
plt.ylabel(’[W]’)

plt.savefig(’plot_sim_heated_tank_3.pdf’)
# plt.savefig(’plot_sim_heated_tank_3_unstable.pdf’)
plt.show()
```

Figure 7.21 shows the simulated response. The simulation is (numerically) inaccurate.

Figure 7.22 shows the simulated response with time step $t_s = 900$ s. The simulation is (numerically) unstable.

Solution to Problem 7.2

Python program 7.7 implements a solution.

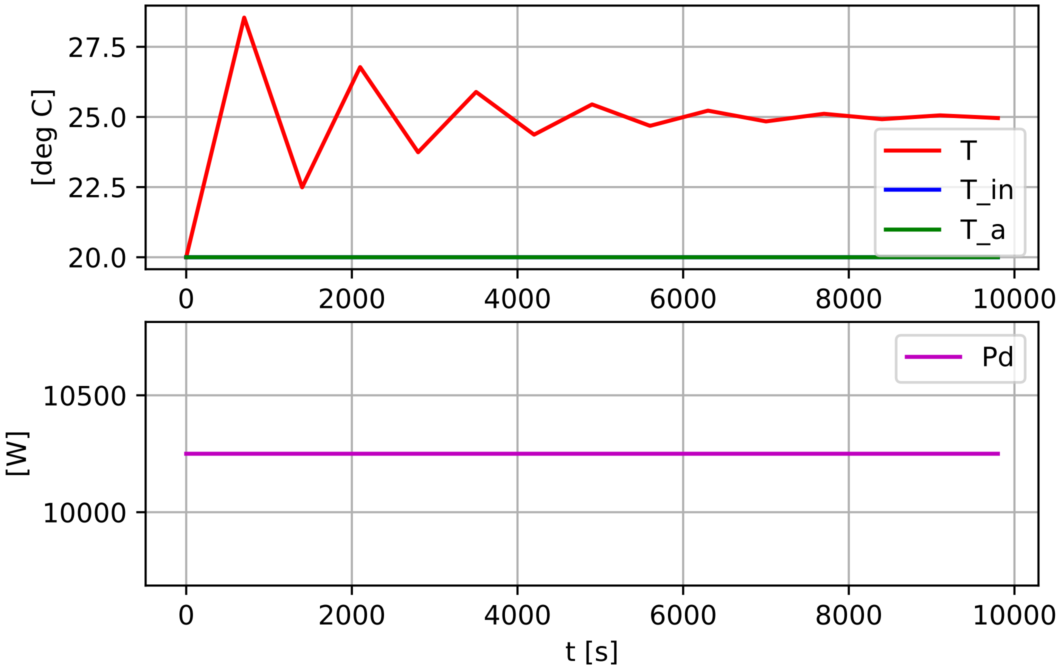


Figure 7.21: Problem 7.1: Simulated response with time step $t_s = 700$ s.

http://techteach.no/control/python/sim_kettle_incl_time_delay.py

Listing 7.7: sim_kettle_incl_time_delay.py

```
"""
Simulation of kettle
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 12
"""

# %% Import of packages:

import matplotlib.pyplot as plt
import numpy as np

# %% Model parameters:

Ta = 20  # [oC]

# %% Derived parameters:

C = 2101  # [J/K] Heat capacity of water in kettle
G = 2.34  # [W/K]  Thermal conductivity

# %% Simulation time settings:

ts = 1.0  # [s]
```

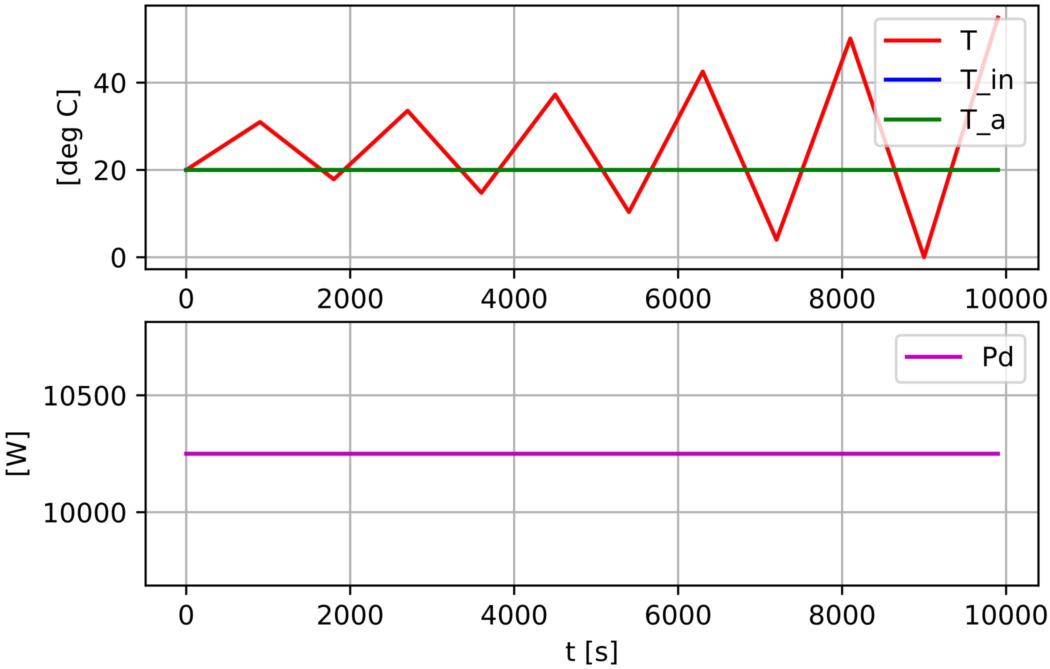


Figure 7.22: Problem 7.1: Simulated response with time step $dt = 900$ s.

```
t_start = 0  # [s]
t_stop = 400  # [s]
N_sim = int((t_stop - t_start)/ts) + 1  # Num time-steps

# %%  Params of input signals:

P_on = 700  # [W]
P_off = 0  # [W]

#%% Array for transport delay of P:

td = 20  # [s]
P_delayed_init = P_off  # [W]
N_delay = int(round(td/ts)) + 1
P_delay_array = np.zeros(N_delay) + P_delayed_init

# %% Preallocation of arrays for plotting:

t_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
Ta_array = np.zeros(N_sim)
P_array = np.zeros(N_sim)

# %% State limits:

T_min = 0  # [oC]
```

```
T_max = 100  # [oC]

# %% Initialization:

T_k = T_init = 20.0  # [oC]

# %% Simulation loop:

for k in range(0, N_sim):

    # Limitation of state:
    T_k = np.clip(T_k, T_min, T_max)

    # Time:
    t_k = k*ts

    # Setting input:
    if (0 <= t_k <= 0):
        P_k = P_off
    else:
        P_k = P_on

    Ta_k = Ta

    # Moving delay array elements one step:
    Pd_k = P_delay_array[-1]  # Take out delayed value
    P_delay_array[1:] = P_delay_array[0:-1]  # Shift array elements
    P_delay_array[0] = P_k  # Put in non-delayed value

    # Time derivative:
    dT_dt_k = (Pd_k + G*(Ta_k - T_k))/C

    # Euler step (prediction):
    T_kp1 = T_k + ts*dT_dt_k

    # Updating arrays for plotting:
    t_array[k] = t_k
    T_array[k] = T_k
    Ta_array[k] = Ta
    P_array[k] = P_k

    # Time index shift:
    T_k = T_kp1

# %% Plotting:

plt.close(’all’)
plt.figure(1)

plt.subplot(2, 1, 1)
plt.plot(t_array, T_array,’b’, label=’T’)
plt.plot(t_array, Ta_array,’g’, label=’Ta’)
plt.legend()
plt.grid()
plt.ylabel('[deg C]')

plt.subplot(2, 1, 2)
plt.plot(t_array, P_array, 'r', label='P')
plt.legend()
plt.grid()
plt.xlabel('t [s]')
plt.ylabel('[W]')

plt.savefig('plot_sim_kettle_time_delay.pdf')
plt.show()
```

Figure 7.23 shows the simulated response. The time delay of 20 s is apparent in the plot.

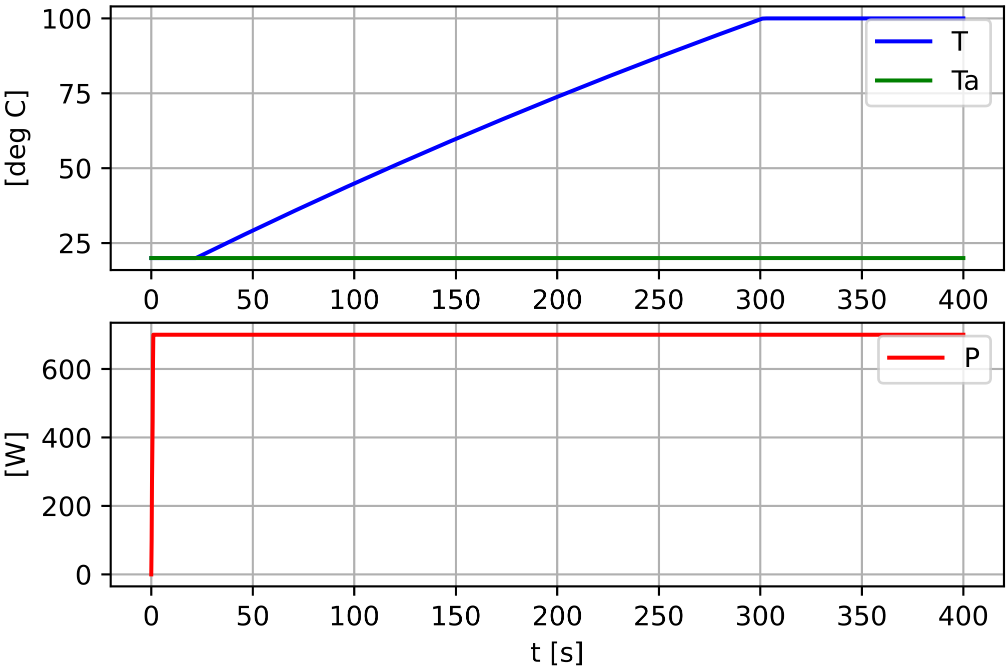


Figure 7.23: Problem 7.2: Simulated response.

Solution to Problem 7.3

With the following definitions:

- Position

$$y = x_1 \tag{7.47}$$

- Speed:

$$y' = x_2 \tag{7.48}$$

the original model, (39.28), can be represented with the following two first order differential equations:

$$x_1{}' = x_2 \tag{7.49}$$

$$x_2{}' = \left(F_p + F_h + F_w\right)/m \tag{7.50}$$

where:

$$F_h = D_h\left(u_c - x_2\right)\left|u_c - x_2\right| \tag{7.51}$$

$$F_w = D_w\left(V_w - x_2\right)\left|V_w - x_2\right| \tag{7.52}$$

Python program 7.8 implements a simulator based on (7.49)-(7.50).

http://techteach.no/control/python/sim_ship.py

Listing 7.8: sim_ship.py

```
#%% Importing packages:

import matplotlib.pyplot as plt
import numpy as np

#%% Model parameters:

m = 71164*1000  # [kg]
Dh = 8.4*1000  # [N/(m/s)^2]
Dw = 0.177*1000  # [N/(m/s)^2]

#%% State limits:

x1_min = -np.inf
x1_max = np.inf
x2_min = -np.inf
x2_max = np.inf

#%% Simulation time settings:

dt = 1  # [s]
t_start = 0  # [s]
t_stop = 1000  # [s]
N_sim = int((t_stop - t_start)/dt) + 1

#%% Preallocation of arrays for plotting:

t_array = np.zeros(N_sim)
x1_array = np.zeros(N_sim)
x2_array = np.zeros(N_sim)
Fp_array = np.zeros(N_sim)
Fh_array = np.zeros(N_sim)
```

```
Fw_array = np.zeros(N_sim)

#%% Initialization:

x1_init = 0
x2_init = 0
x1_k = x1_init
x2_k = x2_init

#%% Simulation loop:

for k in range(0, N_sim):

    # State limitation:
    x1_k = np.clip(x1_k, x1_min, x1_max)
    x2_k = np.clip(x2_k, x2_min, x2_max)

    # Time:
    t_k = k*dt

    # Excitations:
    if t_start <= t_k < 200:
        Fp_k = 0  # [N]
        uc_k = 0
        Vw_k = 0
    if 200 <= t_k:
        Fp_k = 200*1000
        uc_k = 0
        Vw_k = 0
    if 400 <= t_k:
        Fp_k = 0
        uc_k = 0
        Vw_k = 0

    # Forces from water and wind:
    Fh_k = Dh*(uc_k - x2_k)*np.abs(uc_k - x2_k)
    Fw_k = Dw*(Vw_k - x2_k)*np.abs(Vw_k - x2_k)

    # Time derivatives:
    dx1_k = x2_k
    dx2_k = (1/m)*(Fp_k + Fh_k + Fw_k)

    # State prediction (Euler step):
    x1_kp1 = x1_k + dx1_k*dt
    x2_kp1 = x2_k + dx2_k*dt

    # Arrays for plotting:
    t_array[k] = t_k
    x1_array[k] = x1_k
    x2_array[k] = x2_k
    Fp_array[k] = Fp_k
    Fh_array[k] = Fh_k
    Fw_array[k] = Fw_k
    # Time shift:
    x1_k = x1_kp1
    x2_k = x2_kp1

#%% Plotting:

plt.close(’all’)  # Closes all figures before plotting
plt.figure(1)

plt.subplot(4, 1, 1)
plt.plot(t_array, x1_array, ’r’, label=’y’)
plt.legend()
plt.grid()
plt.ylabel(’[m]’)

plt.subplot(4, 1, 2)
plt.plot(t_array, x2_array,  ’b’, label=’dydt’)
plt.legend()
plt.grid()
plt.ylabel(’[m/s]’)

plt.subplot(4, 1, 3)
plt.plot(t_array, Fp_array/1000, ’g’, label=’Fp’)
plt.legend()
plt.grid()
plt.ylabel(’[kN]’)

plt.subplot(4, 1, 4)
plt.plot(t_array, Fh_array/1000, label=’Fh’)
plt.plot(t_array, Fw_array/1000, label=’Fw’)
plt.legend()
plt.grid()
plt.ylabel(’[kN]’)
plt.xlabel(’t [s]’)

#%% Saving the plot figure as a pdf file:

plt.savefig(’plot_simulation_ship.pdf’)
plt.show()
```

Figure 7.24 shows the simulated responses.

Comments to the simulation:

- The speed increases relatively fast between $t = 200$ and 400 s, which is due to the relatively large propeller force.
- The speed decreases relatively slowly after $t = 400$ s, which is due to the relatively small forces from water and wind.

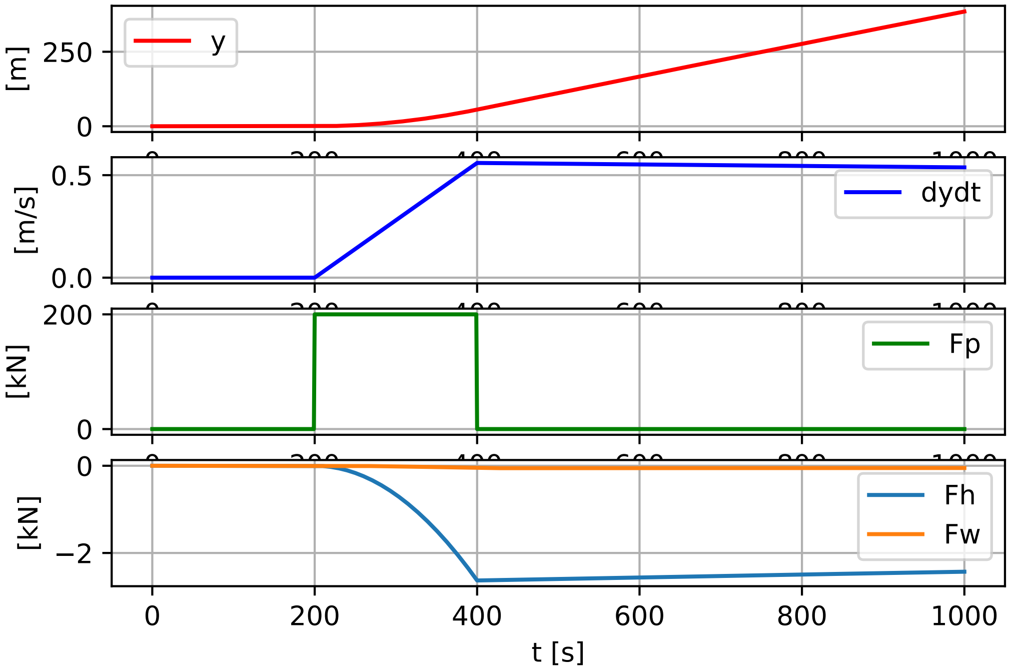


Figure 7.24: Problem 7.3: Simulated responses.

Chapter 8

Transfer functions

8.1 Introduction

Transfer functions is a model form based on the Laplace transform. (The Laplace transform is presented briefly in Ch. 44.) Transfer functions can be derived from a *linear* higher order differential equation, or from a *linear* state space model.

Transfer functions are useful in analysis and design of dynamic systems, typically i.e. process models, control systems, and signal filters. Here is why:

- **Compact model form**: If the original model is a higher order differential equation, or a set of first order differential equations, the relation between the input variable and the output variable can be described by one transfer function, which is a rational function of the Laplace variable s, without any time-derivatives.
- **Representation of standard models**: Transfer functions are often used to represent standard models of controllers and signal filters.
- **Simple to combine systems**: For example, the transfer function for a combined system which consists of two systems in a series combination, is just the product of the transfer functions of each system.
- **Simple to calculate time responses**: The calculations will be made using the Laplace transform, and the necessary mathematical operations are usually much simpler than solving differential equations. Calculation of time-responses for transfer function models is described in Chapter 8.5.
- **The basis of frequency response analysis and design**: The frequency response is a function which expresses how sinusoidal signals are transfered through a dynamic system, assumed to have a linear mathematical model (e.g. a transfer function model). The frequency response can be found from the transfer function of the system. Frequency response is an important tool particularly in analysis and design of signal filters, and is the basis of some traditional methods of analysis and design of control systems, cf. Chapters 21 and 22.

Information about the mathematical notation of Laplace transformed variables and transfer functions used in this book is in Appendix 37.

8.2 Definition of the transfer function

The first step in deriving the transfer function of a system is taking the Laplace transform of the differential equation (which must be linear). Example shows how to derive a transfer function for a specific differential equation, but the procedure is general.

Example 8.1 *Deriving the transfer function from a first order differential equation*

Given the following differential equation:

$$y'(t) = ay(t) + bu(t) \tag{8.1}$$

u is the input (the independent) variable, y is the output (the depend) variable, and a and b are parameters (coefficients). Assume that the initial state, i.e. the value of y at time $t = 0$, is y_0. We start by taking the Laplace transform of both sides of the differential equation:

$$\mathcal{L}\left\{y'(t)\right\} = \mathcal{L}\left\{ay(t) + bu(t)\right\} \tag{8.2}$$

The left side of (8.2) is Laplace transformed using the differentiation rule (44.20) to $\mathcal{L}\left\{y'(t)\right\}$:

$$\mathcal{L}\left\{y'(t)\right\} = sY(s) - y_0 \tag{8.3}$$

The right side of (8.2) is transformed using the linearity property of the Laplace transform, cf. (44.17):

$$\begin{aligned}
&\mathcal{L}\left\{ay(t)\right\} + \mathcal{L}\left\{bu(t)\right\} && (8.4)\\
&= a\mathcal{L}\left\{y(t)\right\} + b\mathcal{L}\left\{u(t)\right\} && (8.5)\\
&= aY(s) + bU(s) && (8.6)
\end{aligned}$$

Thus, the Laplace transform of the differential equation (8.2) is

$$sY(s) - y_0 = aY(s) + bU(s) \tag{8.7}$$

Solving for the output variable $Y(s)$:

$$Y(s) = \overbrace{\frac{1}{s-a}y_0}^{Y_{\text{init}}(s)} + \overbrace{\frac{b}{s-a}U(s)}^{Y_u(s)} \tag{8.8}$$

In (8.8), the total response $Y(s)$ is made up by the following additive contributions:

- $Y_{\text{init}}(s)$ is the contribution from initial state y_0:

$$Y_{\text{init}}(s) = \frac{1}{s-a}y_0 \tag{8.9}$$

- $Y_u(s)$ is the contribution from input u:

$$Y_u(s) = \underbrace{\frac{b}{s-a}}_{H(s)} U(s) \tag{8.10}$$

In (8.10),

$$H(s) = \frac{b}{s-a} \tag{8.11}$$

is *the transfer function* from input $U(s)$ to output $Y(s)$. With a somewhat relaxed terminology, we can also say that $H(s)$ is the transfer function from input u to output y (although y and u exist only in the time domain).

In words, the transfer function is the s-valued function that the input $U(s)$ is multiplied with to produce its contribution to the output $Y(s)$.

We may use any symbol for the transfer function. A common symbol is $H(s)$, as used above. One alternative symbol is

$$H_{Y,U}(s) = \frac{b}{s-a} \tag{8.12}$$

We may also express the transfer function as

$$H(s) = \frac{Y(s)}{U(s)} \tag{8.13}$$

We can illustrate the transfer function (8.11) with a block diagram, see Figure 8.1.

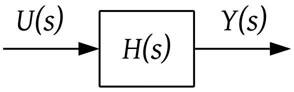


Figure 8.1: The transfer function (8.11) illustrated with a block diagram.

[End of Example 8.1]

Note: A transfer function is a type a mathematical model of a dynamic system. (A differential equation is another type of model.) As for any model, the transfer function is independent of how the input signal to the system varies.

A system may have several transfer functions

In Example 8.1, the system has one transfer function. However, a system may have several transfer functions, as demonstrated in the following example.

Example 8.2 *Transfer functions of a differential equation with two input variables*

Given the following differential equation:

$$y'(t) = ay(t) + b_1u_1(t) + b_2u_2(t) \tag{8.14}$$

u_1 and u_2 are input variables, y is the output variable, and a, b_1 and b_2 are parameters. Assume that the initial state, i.e. the value of y at time $t = 0$, is y_0. Taking the Laplace transform of both sides of the differential equation:

$$\mathcal{L}\left\{y'(t)\right\} = \mathcal{L}\left\{ay(t) + b_1u_1(t) + b_2u_2(t)\right\} \tag{8.15}$$

gives

$$sY(s) - y_0 = aY(s) + b_1U_1(s) + b_2U_2(s) \tag{8.16}$$

Solving for the output variable $Y(s)$:

$$Y(s) = \overbrace{\frac{1}{s-a}y_0}^{\mathrm{Y_{init}}(s)} + \overbrace{\frac{b_1}{s-a}U_1(s)}^{\mathrm{Y}_1(s)} + \overbrace{\frac{b_2}{s-a}U_2(s)}^{\mathrm{Y}_2(s)} \tag{8.17}$$

In (8.17), we have the following two transfer functions:

$$H_1(s) = \frac{b_1}{s-a_1} \tag{8.18}$$

and

$$H_2(s) = \frac{b_2}{s-a_2} \tag{8.19}$$

The transfer functions derived above is illustrated with the block diagram shown in Figure 8.2

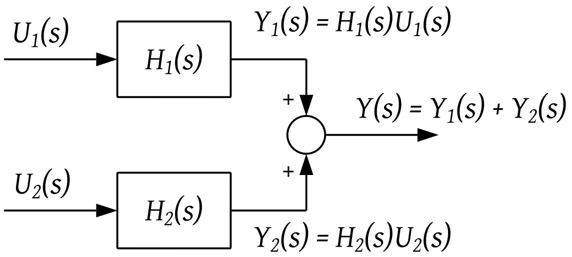


Figure 8.2: Block diagram representing the transfer functions $H_1(s)$ and $H_2(s)$ in (8.17).

[End of Example 8.2]

8.3 Characteristics of transfer functions

Typically, transfer functions are rational functions (of the Laplace variable s). Such transfer functions may be written on a factorized form – often called a *zero-pole form*:

$$H(s) = K\frac{(s - z_1)(s - z_2) \cdots (s - z_r)}{(s - p_1)(s - p_2) \cdots (s - p_n)} = \frac{b(s)}{a(s)} \tag{8.20}$$

Here $z_1, \ldots, z_r$ are the *zeros* and $p_1, \ldots, p_n$ are the *poles* of the transfer function.

The s-polynomial in the denominator of $H(s)$, which is $a(s)$ in (8.20), is denoted the *characteristic polynomial.*

The poles are the roots of the characteristic polynomial, that is

$$a(s) = 0 \text{ for } s = s_1,\ s_2,\ ...,\ s_n \text{ (the poles)} \tag{8.21}$$

The *order* of a transfer function is the order of the characteristic polynomial.

As shown in Ch. 19, the values of the poles determines the stability property of the system. The system is stable only if all the poles have negative real parts, or, in other words, if all the poles lie in the left half complex plane. If at least one pole is in the right half plane, the system is unstable.

Example 8.3 *Characteristics of a transfer function*

Given the transfer function

$$H(s) = \frac{4s - 4}{s^2 + 5s + 6} = 4\frac{(s - 1)}{(s + 3)(s + 2)} \tag{8.22}$$

It has two poles: -3 and -2, and one zero: 1. Its order is 2.

[End of Example 8.3]

8.4 Combining transfer functions blocks in block diagrams

Transfer function blocks may be combined in a block diagram according to the rules shown in Figure 8.3.

One possible purpose of such a combination is to simplify the block diagram, or to calculate the resulting or overall transfer function. For example, the combined transfer function of two transfer functions connected in series is equal to the product of the individual transfer functions, jc. the Series connection rule in Figure 8.3.

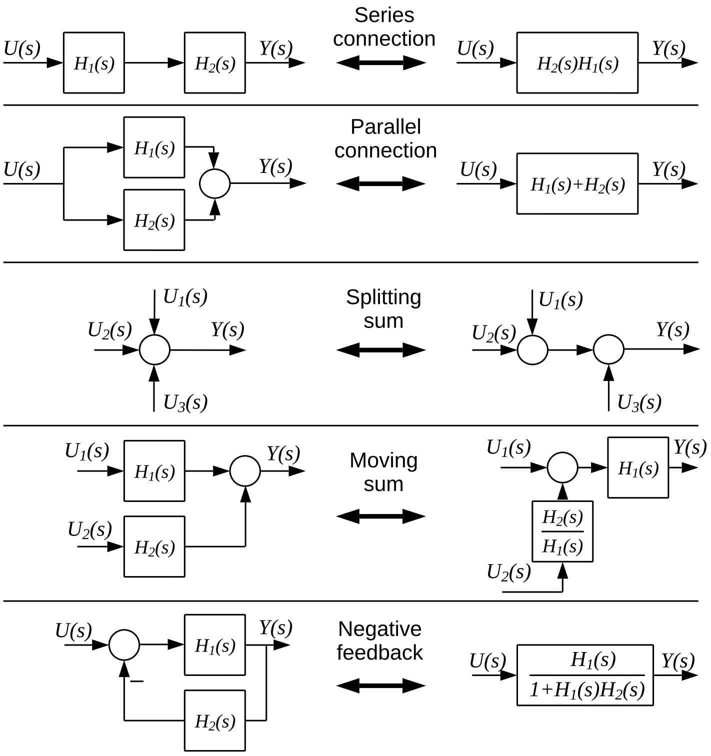


Figure 8.3: Rules for combining transfer function blocks.

8.5 How to calculate responses with transfer functions

Assume given the following transfer function:

$$H(s) = \frac{Y(s)}{U(s)} \tag{8.23}$$

To calculate the time-response $y(t)$ for a given input signal $u(t)$, we can do as follows:

1. First, find $U(s)$ – the Laplace transform of the input signal. $U(s)$ can be found from precalculated Laplace transform pairs, cf. Section 44.3, possibly combined with one or more of the Laplace transform properties, cf. Section 44.4, where particularly the linearity property (44.17) is useful.

2. The Laplace transform of the output signal, $Y(s)$, is calculated from (8.23), that is,

$$Y(s) = H(s)U(s) \tag{8.24}$$

where $U(s)$ is found as explained above.

3. The time-function $y(t)$ is calculated as the inverse Laplace transform of $Y(s)$, cf. Ch. 44.

Example 8.4 *Calculation of time-response for transfer function model*

Given the transfer function model

$$Y(s) = \underbrace{\frac{3}{s+0.5}}_{H(s)} U(s) \tag{8.25}$$

Suppose that $u(t)$ is a step from 0 to 2 at $t = 0$. We shall find an expression for the time-response $y(t)$. The Laplace transform of $u(t)$ is, cf. (44.7),

$$U(s) = \frac{2}{s} \tag{8.26}$$

Inserting this into (8.25) gives

$$Y(s) = \frac{3}{s+0.5} \cdot \frac{2}{s} = \frac{6}{(s+0.5)s} = \frac{12}{(2s+1)s} \tag{8.27}$$

(8.27) has the same form as the Laplace transform pair (44.11) which is repeated here:

$$\frac{k}{(Ts+1)s} \iff k\left[1 - e^{-t/T}\right] \tag{8.28}$$

Here $k = 12$ and $T = 2$. The time-response becomes

$$y(t) = 12\left[1 - e^{-t/2}\right] \tag{8.29}$$

Figure 8.4 shows $y(t)$. The steady state response is 12, which can be calculated from $y(t)$ by setting $t = \infty$.

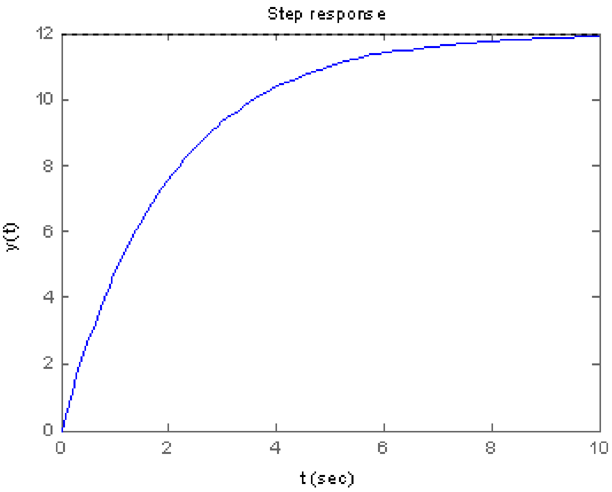


Figure 8.4: Example 8.4: The time-response $y(t)$ given by (8.86).

[End of Example 8.4]

Why not calculate the response by solving the differential equation?

For a given system, its transfer function and its differential equation can be regarded as two alternative model types of the system. To calculate the response in the ouput for some input signal, you may either do the calculation with the transfer function and the Laplace transform, or do the calculation by solving differential equation. I think it is generally much easier to use the transfer function and the Laplace transform.[1]

8.6 Static transfer function and static response

Suppose that the input signal to a system is a step of amplitude u_s, which has the following Laplace transform, cf. (44.7):

$$U(s) = \frac{u_s}{s} \tag{8.30}$$

The corresponding static time-response can found from the Final Value Theorem:

$$y_s = \lim_{s\to 0} s \cdot Y(s) = \lim_{s\to 0} s \cdot H(s)U(s) = \lim_{s\to 0} s \cdot H(s)\frac{u_s}{s} = \underbrace{\lim_{s\to 0} H(s)}_{H_s} u_s \tag{8.31}$$

where H_s is the *static transfer function.* That is,

$$H_s = \lim_{s\to 0} H(s) \tag{8.32}$$

Thus, the static transfer function, H_s, is found by letting s approach zero in the transfer function, $H(s)$.

Once we know the static transfer function H_s the static (steady state) response y_s due to a constant input of value u_s, is

$$y_s = H_s u_s \tag{8.33}$$

Example 8.5 *Static transfer function and static response*

See Example 8.4. The transfer function is

$$H(s) = \frac{3}{s + 0.5} \tag{8.34}$$

The corresponding static transfer function becomes

$$H_s = \lim_{s\to 0} H(s) = \lim_{s\to 0} \frac{3}{s + 0.5} = 6 \tag{8.35}$$

Assume that the input u has the constant value of $u_s = 2$. What is the corresponding static response y_s in the output? It can be calculated from the static transfer function as

$$y_s = H_s u_s = 6 \cdot 2 = 12 \tag{8.36}$$

which is the same result as found in Example 8.4.

[End of Example 8.5]

[1] A classmate of mine once said after he had learnt about the Laplace transform: “I shall never again solve a differential equation!”

8.7 Simulation with transfer functions

8.7.1 Introduction

In the following sections, two alternative tools for simulation of responses in the output of transfer functions are presented:

- Python Control Package, using Python code.
- OpenModelica, based on a block diagram representation of the transfer function.

There are simulation tools also in LabVIEW, Matlab and Simulink, but they are not presented in this book.

8.7.2 Simulation with Python Control Package

This is described in Ch. 41.2.4.

8.7.3 Simulation with OpenModelica

This is described in Ch. 42.

8.8 From transfer function to differential equation

Assume we have the transfer function model

$$Y(s) = H(s)U(s) \tag{8.37}$$

where $H(s)$ is the transfer function from input $U(s)$ to output $Y(s)$. We can find an equivalent differential equation relating $y(t)$ and $u(t)$ in the time domain as explained in Example 8.6.

Example 8.6 *From transfer function to differential equation*

Given the following transfer function $H(s)$, which is the same as (8.25) in Example 8.4:

$$Y(s) = \underbrace{\frac{3}{s+0.5}}_{H(s)} U(s) \tag{8.38}$$

Here is how to find an equivalent differential equation relating $y(t)$ and $u(t)$:

1. Cross-multipy (8.38) to get:

$$(s + 0.5)Y(s) = 3U(s)$$

2. Dissolve the parenthesis to get:

$$sY(s) + 0.5Y(s) = 3U(s)$$

3. Apply the pertinent Laplace transform properties, cf. Section 44.4, to get:

$$y' + 0.5y = 3u$$

which is the differential equation.

[End of Example 8.6]

8.9 From transfer function to state space model

It may happen that we want to transform a given transfer function into an equivalent state space model.

See Figure 8.5. It shows a block diagram of the linear state space model (8.39)-(8.41).

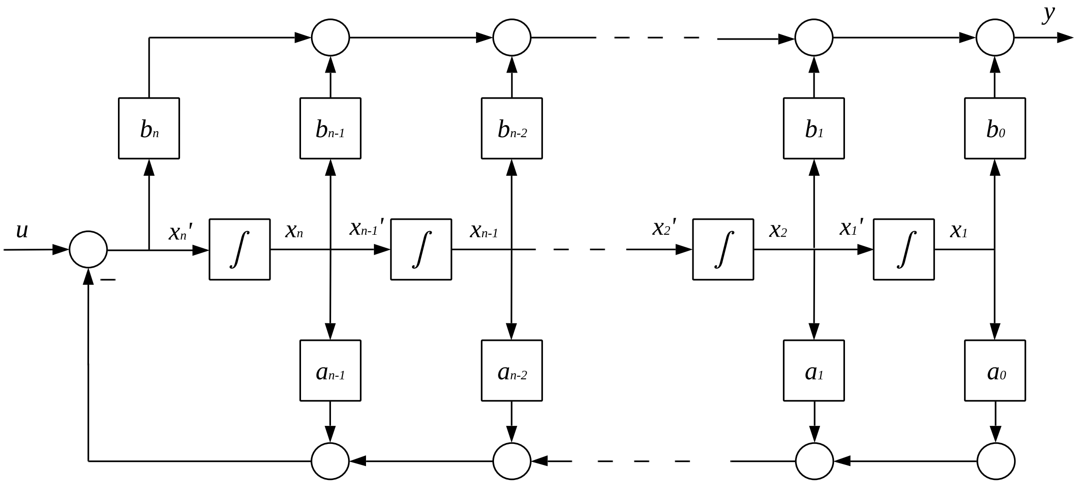


Figure 8.5: Canonical block diagram

$$x_1{}' = x_2 \tag{8.39}$$

$$x_2{}' = x_3$$

$$\vdots$$

$$x_n{}' = -a_0 x_1 - a_1 x_2 - \cdots - a_{n-1} x_n + u \tag{8.40}$$

$$y = b_0 x_1 + b_1 x_2 + \cdots + b_n(-a_0 x_1 - a_1 x_2 - \cdots - a_{n-1} x_n) + b_n u \tag{8.41}$$

This block diagram, and the corresponding state space model, has a special form denoted controller canonical form. (Canonical means "according to the rules".) The block diagram represents the system in the time domain.

Now, assume that we replace each of the integrators in the block diagram with blocks containing $1/s$, which is the transfer function of an integrator, cf. Ch. 9.2.1.3. The resulting block diagram is then a corresponding block diagram in the Laplace domain.[2] It can be shown, for example by using pertinent rules for combining transfer function blocks, cf. Section 8.4, that the transfer function from input $U(s)$ to output $Y(s)$ is

$$\frac{Y(s)}{U(s)} = H(s) = \frac{b_n s^n + b_{n-1} s^{n-1} + \cdots b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \cdots a_1 s + a_0} \tag{8.42}$$

So, we can transform a given transfer function into a state space model by drawing a canonical block diagram!

Note that the block diagram in Figure (8.5) is actually only one of an infinitely number of possible block diagrams with (8.42) as the resulting transfer function. Among these block diagrams are other forms of canonical block diagrams (and corresponding state space models). I have selected the controller canonical form here as I think it is the most convenient for the present application.

Example 8.7 *From transfer function to state space model*

Given the following transfer function model:

$$Y(s) = H(s)U(s) \tag{8.43}$$

with transfer function

$$H(s) = \frac{4s + 5}{s^2 + 2s + 3} = \frac{0s^2 + 4s + 5}{s^2 + 2s + 3} \tag{8.44}$$

Figure 8.6 shows a corresponding block diagram in the time domain.

From the block diagram, we can write the following state space model:

$$x_1' = x_2 \tag{8.45}$$

$$x_2' = -3x_1 - 2x_2 + u \tag{8.46}$$

$$y = 5x_1 + 4x_2 \tag{8.47}$$

[2] I will not draw that block diagram as it has exactly the same structure as the block diagram in Figure 8.5.

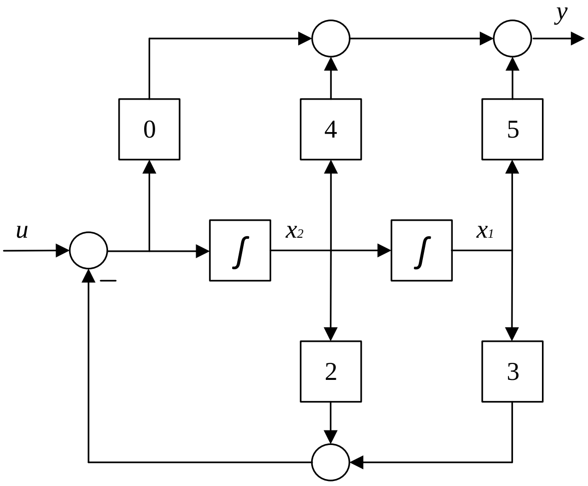


Figure 8.6: Block diagram in th time domain corresponding to the transfer function (8.43).

Above, we derived a state space model from the transfer function, which is a set of first order differential equations. Alternatively, we can obtain a higher order differential equation using the technique presented in Section 8.8: From (8.43) and (8.44) we get

$$\left(s^2 + 2s + 3\right) Y(s) = (4s + 5)\, U(s) \tag{8.48}$$

Dissolving the parentheses:

$$s^2 Y(s) + 2sY(s) + 3Y(s) = 4sU(s) + 5U(s)$$

Taking the inverse Laplace transform gives the following second order differential equation:

$$y''(t) + 2y'(t) + 3y(t) = 4u'(t) + 5u(t) \tag{8.49}$$

which is an alternative to the state space model derived earlier in this example.

[End of Example 8.7]

8.10 From state space model to transfer function

Given a state space model:

$$x' = Ax + Bu \tag{8.50}$$

$$y = Cx + Du \tag{8.51}$$

We can derive the transfer function from u to y as follows: Take the Laplace transform of (8.50) – (8.51) to get (I is the identity matrix of equal dimension as of A)

$$sIX(s) - x_0 = AX(s) + BU(s) \tag{8.52}$$

$$Y(s) = CX(s) + DU(s) \tag{8.53}$$

We neglect the initial state x_0, as we always can do when deriving transfer functions from differential equation models. Solving (8.52) for $X(s)$ gives

$$X(s) = (sI - A)^{-1}BU(s) \tag{8.54}$$

Inserting this $X(s)$ into (8.53) gives

$$Y(s) = \left[C(sI - A)^{-1}B + D\right]U(s) \tag{8.55}$$

from which we get the following transfer function from u to y:8.10

$$\begin{aligned} H(s) = \frac{Y(s)}{U(s)} &= C(sI - A)^{-1}B + D && (8.56)\\ &= C\frac{\text{adj}(sI - A)}{\det(sI - A)}B + D && (8.57)\\ &= \frac{1}{\det(sI - A)} \cdot C \cdot \text{adj}(sI - A) \cdot B + D && (8.58) \end{aligned}$$

One interesting observation: The *poles* of $H(s)$ are the roots of the denominator:

$$a(s) = \det(sI - A) = 0 \tag{8.59}$$

which is the characteristic equation. (8.59) defines the *eigenvalues* of A.[3] So, the poles of the transfer function are the same as the eigenvalues of A. Consequently, the stability of a dynamic system depends on the eigenvalues in the same way that it depends on the poles.

Example 8.8 *Calculating the transfer function from a state space model*

Given the following state space model:

$$\underbrace{\begin{bmatrix} x_1' \\ x_2' \end{bmatrix}}_{x'} = \underbrace{\begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B}u \tag{8.60}$$

$$y = \underbrace{\begin{bmatrix} 3 & 1 \end{bmatrix}}_{C}x + \underbrace{[0]}_{D}u \tag{8.61}$$

The transfer function from u to y using (8.58) is:

$$\begin{aligned} H(s) &= C(sI - A)^{-1}B + D \\ &= \frac{1}{\det(sI - A)} \cdot C \cdot \text{adj}(sI - A) \cdot B + D \\ &= \frac{1}{s^2 + 3s + 2} \cdot \begin{bmatrix} 3 & 1 \end{bmatrix} \cdot \text{adj}\left(sI - \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}\right) \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + [0] \\ &= \frac{1}{s^2 + 3s + 2} \cdot \begin{bmatrix} 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} s+3 & 1 \\ -2 & s \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + [0] \\ &= \frac{s+3}{s^2 + 3s + 2} \end{aligned}$$

[End of Example 8.8]

[3] In mathematics litterature it is more common to use the symbol λ instead of s for eigenvalues.

8.11 *Problems for Chapter 8*

Problem 8.1 *From second order differential equation to transfer function*

Given the following differential equation model:

$$y'' = -a_1 y' - a_0 y + b_0 u + c_0 d \tag{8.62}$$

where y is the output, u is the input, d is the disturbance, and a_1, a_0, b_0 and c_0 are parameters. Find the transfer function from u to y. What is the order of the transfer function?

Problem 8.2 *Transfer function of a wood chips tank*

Appendix 39.2 presents a mathematical model of a wood chips tank. Derive the transfer function $H_1(s)$ from screw control signal u to level h, and the transfer function $H_2(s)$ from outflow rate F_{out} to level h.

Problem 8.3 *Transfer function of a mass-spring-damper-system*

Figure 8.7 shows a mass-spring-damper-system.

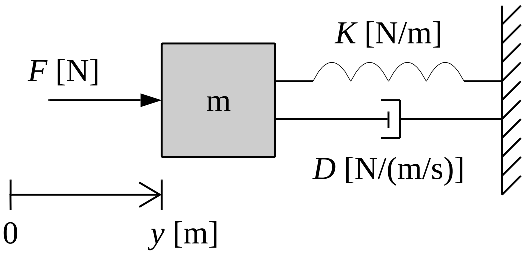


Figure 8.7: Mass-spring-damper-system.

y is position. F is applied force. D is damping constant. K is spring constant. It is assumed that the damping force F_d is proportional to the speed, and that the spring force F_s is proportional to the position of the mass (body). The spring force is assumed to be zero when y is zero. Force balance (Newton's 2. Law) yields

$$my'' = F - Dy' - Ky \tag{8.63}$$

Calculate the transfer function from force F to position y.

Problem 8.4 *Simulation of transfer function using Python Control Package*

In Problem 8.3, you are to derive the transfer function from force F to position y.

Assume the following model parameters: $m = 20$ kg, $D = 4$ N/(m/s), and $K = 2$ N/m.

Simulate the response in y due to a step of 5 N at time zero in F. The initial position and speed have zero values.

Problem 8.5 *Characteristics of transfer function*

Given the following transfer function:

$$H(s) = \frac{s+3}{s^2+3s+2} \tag{8.64}$$

1. What is the order?
2. What is the characteristic equation?
3. What is the characteristic polynomial?
4. What are the poles and the zeros?

Problem 8.6 *Transfer function block diagrams*

Given a thermal process with transfer function $H_p(s)$ from supplied power P to temperature T as follows:

$$T(s) = \underbrace{\frac{b_p}{s+a_p}}_{H_p(s)} P(s) \tag{8.65}$$

The transfer function from temperature T to temperature measurement T_m is as follows:

$$T_m(s) = \underbrace{\frac{b_m}{s+a_m}}_{H_m(s)} T(s) \tag{8.66}$$

a_p, b_p, a_m, and b_m are parameters.

1. Draw a transfer function block diagram of the system (process with sensor) with P as input variable and T_m as output variable.
2. What is the transfer function from P to T_m? (Derive it from the block diagram.)

Problem 8.7 *Calculation of step response using transfer function*

Given the transfer function model

$$Y(s) = \underbrace{\frac{5}{s}}_{H(s)} U(s) \tag{8.67}$$

Suppose that the input $u(t)$ is a step from 0 to 3 at $t = 0$. Calculate the response $y(t)$ due to this input.

Problem 8.8 *Static transfer function*

See Problem 8.3. The transfer function from force F to position y is

$$H(s) = \frac{Y(s)}{F(s)} = \frac{1}{ms^2 + Ds + K} \tag{8.68}$$

Calculate the static transfer function H_s. From H_s calculate the static response y_s corresponding to a constant force, F_s.

Problem 8.9 *From transfer function to state space model*

The transfer function of a first order high-pass filter can be written as:

$$H(s) = \frac{Y(s)}{U(s)} = \frac{\frac{s}{\omega_c}}{\frac{s}{\omega_c} + 1} = \frac{s}{s + \omega_c} \tag{8.69}$$

where ω_c [rad/s] is the crossover or corner frequency.

Find an equivalent state space model from a canonical block diagram corresponding to (8.69).

8.12 *Solutions to problems for Chapter* 8

Solution to Problem 8.1

Taking the Laplace transformation of the differential equation gives (here, any non-zero initial values of y and y' are neglected):

$$s^2Y(s) = -a_1sY(s) - a_0Y(s) + b_0U(s) + c_0D(s) \tag{8.70}$$

Solving for $y(s)$ gives:

$$Y(s) = \frac{b_0}{s^2 + a_1s + a_0}U(s) + \frac{c_0}{s^2 + a_1s + a_0}D(s) \tag{8.71}$$

Hence, the transfer function from u to y is:

$$H(s) = \frac{Y(s)}{U(s)} = \frac{b_0}{s^2 + a_1s + a_0} \tag{8.72}$$

Solution to Problem 8.2

The mathematical model based on mass balance of the wood chips in the tank is given by (8.73), and repeated below:

$$\rho Ah'(t) \quad = \quad u(t-\tau) - F_{\text{out}}(t) \tag{8.73}$$

The Laplace transform of (8.73) is

$$\rho A\left[sH(s) - h_0\right] = e^{-\tau s}U(s) - F_{\text{out}}(s) \tag{8.74}$$

Solving for output variable $H(s)$ gives

$$H(s) = \frac{1}{s}h_0 + \underbrace{\frac{1}{\rho As}e^{-\tau s}}_{H_1(s)}U(s) + \underbrace{\left(-\frac{1}{\rho As}\right)}_{H_2(s)}F_{\text{out}}(s) \tag{8.75}$$

Thus, the transfer functions are

$$H_1(s) = \frac{1}{\rho As}e^{-\tau s} \tag{8.76}$$

and

$$H_2(s) = -\frac{1}{\rho As} \tag{8.77}$$

Solution to Problem 8.3

Laplace transform of (19.27) gives

$$m\left[s^2Y(s) - sy_0' - y_0\right] = F(s) - D\left[sY(s) - y_0\right] - KY(s) \tag{8.78}$$

Setting initial values $y_0 = 0$ and $y_0' = 0$, and then solving for $Y(s)$:

$$Y(s) = \underbrace{\frac{1}{ms^2 + Ds + K}}_{H(s)} F(s) \tag{8.79}$$

The transfer function is

$$H(s) = \frac{Y(s)}{F(s)} = \frac{1}{ms^2 + Ds + K} \tag{8.80}$$

Solution to Problem 8.4

Python program 8.1 implements the simulator.

http://techteach.no/control/python/sim_msd_tf_python_control_package.py

Listing 8.1: sim_msd_tf_python_control_package.py

```
# %% Import:

import numpy as np
import matplotlib.pyplot as plt
import control

#%% Defining Laplace variable for def of transfer func:
s = control.tf(’s’)

#%% Model

m = 20  # [kg]
D = 4  # [N/(m/s)]
K = 2  # [N/m]

H = 1/(m*s**2 + D*s + K)

# %% Simulation of unit step response:

ampl_F = 5
t_final = 50
(t, y_unit_step) = control.step_response(H, t_final)
y = y_unit_step*ampl_F
F = np.zeros(len(t)) + ampl_F

# %% Plotting:

plt.close(’all’)
plt.figure(1)

plt.subplot(2,1,1)
plt.plot(t, F, color=’blue’, label=’F’)
plt.legend()
plt.grid()
plt.xlabel(’t [s]’)
plt.ylabel(’[N]’)

plt.subplot(2,1,2)
plt.plot(t, y, color=’red’, label=’y’)
plt.legend()
plt.grid()
plt.xlabel(’t [s]’)
plt.ylabel(’[m]’)

# plt.savefig(’plot_sim_msd.pdf’)
plt.show()
```

Figure 8.8 shows the simulation.

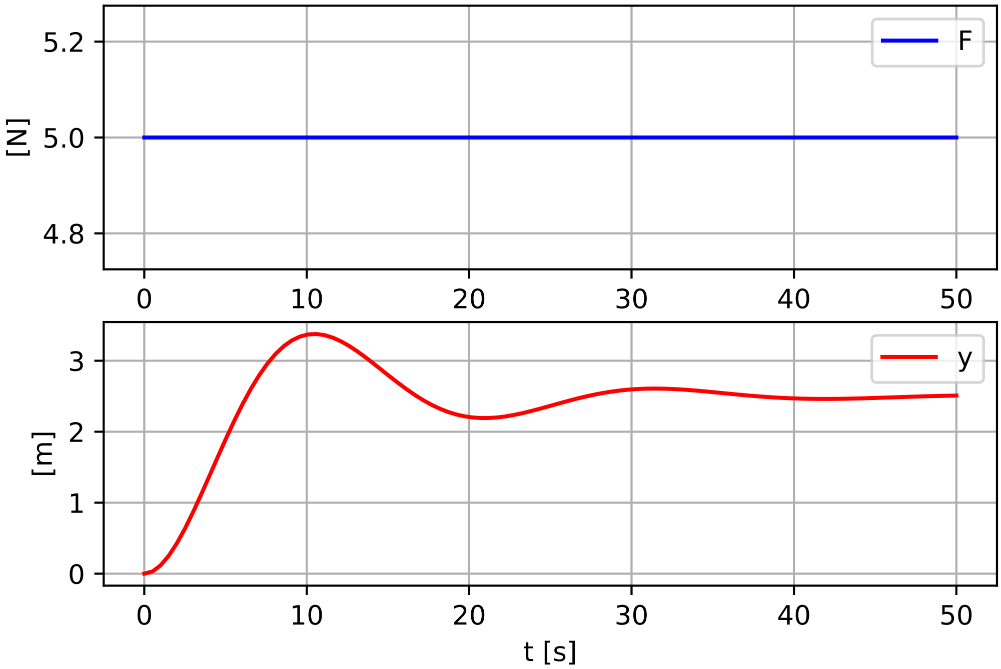


Figure 8.8: Problem 8.4: Simulation of mass-spring-damper.

Solution to Problem 8.5

1. Order: 2.
2. $s^2 + 3s + 2 = 0$
3. $s^2 + 3s + 2$

4. We write the transfer function on pole-zero-form:

$$H(s) = \frac{s+3}{s^2+3s+2} = \frac{s+3}{(s+1)(s+2)} \tag{8.81}$$

We see that the poles are -1 and -2, and the zero is -3.

Solution to Problem 8.6

1. Figure 8.9 shows the block diagram.
2. According to the *series combination rule* the transfer function becomes

$$H(s) = \frac{T_m(s)}{P(s)} = H_m(s)H_p(s) = \frac{b_m}{s+a_m}\frac{b_p}{s+a_p} \tag{8.82}$$

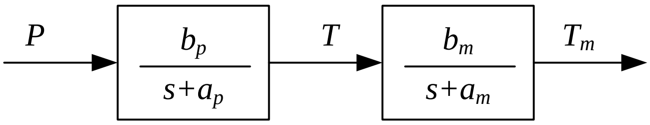


Figure 8.9: Block diagram of transfer functions in series.

Solution to Problem 8.7

The Laplace transform of $u(t)$ is (cf. the Laplace transform pair (44.7):

$$U(s) = \frac{3}{s} \tag{8.83}$$

Inserting this into (8.67) gives

$$Y(s) = \frac{5}{s} \cdot \frac{3}{s} = \frac{15}{s^2} \tag{8.84}$$

which has the same form as in the Laplace transform pair (44.8), which is repeated here:

$$\frac{k}{s^2} \quad \Longleftrightarrow \quad kt \tag{8.85}$$

We have $k = 15$, so the response is

$$y(t) = 15t \tag{8.86}$$

Solution to Problem 8.8

Setting $s = 0$ in the transfer function gives

$$H_s = H(0) = \frac{1}{K} \tag{8.87}$$

The static response y_s corresponding to a constant force, F_s, is

$$y_s = H_sF_s = \frac{F_s}{K} \tag{8.88}$$

Solution to Problem 8.9

Figure 8.10 shows a canonical block diagram corresponding to (8.69). From the block diagram, we find the following state space model:

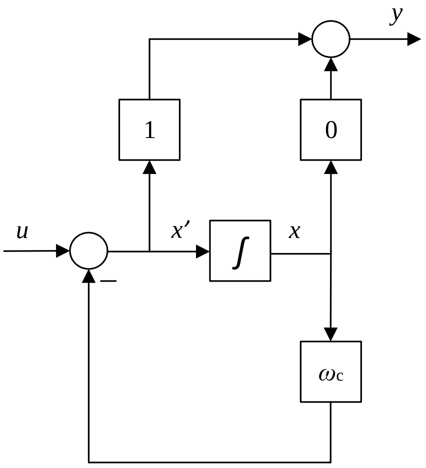


Figure 8.10: A canonical block diagram corresponding to (8.69).

$$x' = -\omega_c x + u = Ax + Bu \tag{8.89}$$

$$y = -\omega_c x + u = Cx + Du \tag{8.90}$$

Chapter 9

Process dynamics

9.1 Introduction

In this chapter a number of standard dynamic models in the form of transfer functions will be defined. With such standard models you can characterize the dynamic properties of a physical system in terms of for example gain, time constant, and time delay. These terms are also useful for controller tuning, as in the SIMC tuning method which is described in Section 14.8.

9.2 Integrators

9.2.1 Integrator model

9.2.1.1 Differential equation

An integrator is a system where *the output variable y is the time integral of the input variable u*, multiplied with the integrator gain K_i:

$$y(t) = K_i \int_0^t u(\theta)\, d\theta \tag{9.1}$$

Taking the time derivative of both sides of (9.1) yields the following differential equation describing an integrator:

$$y'(t) = K_i u(t) \tag{9.2}$$

9.2.1.2 Block diagram

Figure 9.1 shows a block diagram of the integrator model (9.2).

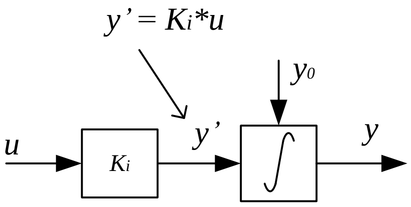


Figure 9.1: Block diagram of the integrator (9.17).

9.2.1.3 Transfer function

Taking the Laplace transform using (44.23) gives

$$sY(s) = K_i U(s) \tag{9.3}$$

which gives the following transfer function of an integrator:

$$\frac{Y(s)}{U(s)} = H(s) = \frac{K_i}{s} \tag{9.4}$$

9.2.1.4 Pole

From (9.4) we find that the pole of an integrator is

$$p = 0 \tag{9.5}$$

which is in the origin of the complex plane, see Figure 9.2.

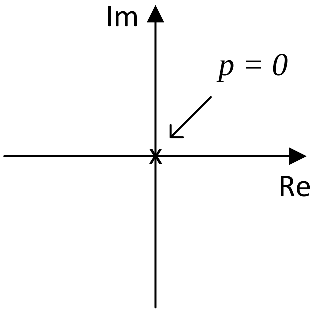


Figure 9.2: The pole of an integrator.

9.2.2 Dynamics in terms of step response

Let us now find the *step response* of the integrator. We assume that $u(t)$ is a step of amplitude U at $t = 0$. From (44.7) $U(s) = \frac{U}{s}$. Thus,

$$Y(s) = H(s)U(s) = \frac{K_i}{s} \cdot \frac{U}{s} = \frac{K_i U}{s^2} \tag{9.6}$$

which, inverse Laplace transformed using (44.8), is

$$y(t) = K_i U t \tag{9.7}$$

Thus, the step response of an integrator is *a ramp* with rate of change $K_i U$.

Figure 9.3 shows a simulated response of an integrator. The response is with the following settings:

$$K_i = 0.5 \tag{9.8}$$

$$U = 4 \tag{9.9}$$

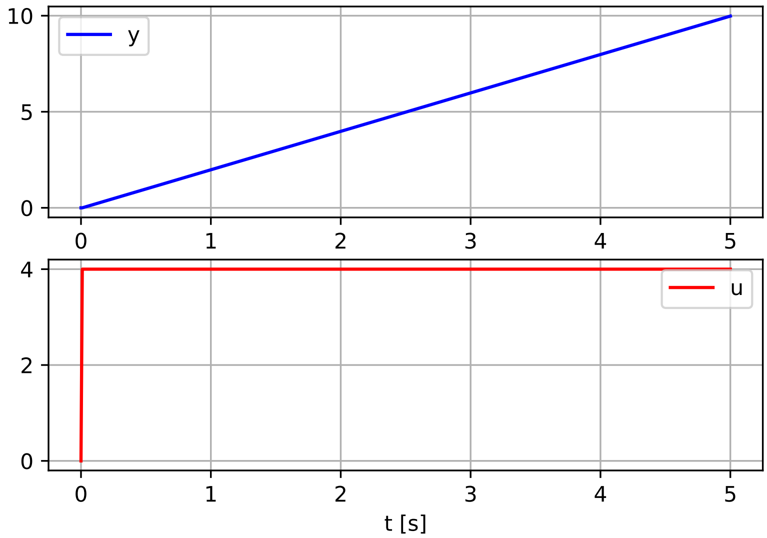


Figure 9.3: Simulated response of an integrator.

The following Python program implements the simulator which produces the response shown in Figure 9.3.

http://techteach.no/control/python/sim_integrator_elementary_code.py

The program above contains elementary programming code implementing an Euler forward simulation algorithm. Alternatively, you can simulate the step response with the forced_response() function of the Python Control package, as in the following program:

http://techteach.no/control/python/sim_integrator_forced_response_py_con.py

You can also use the step_response() function of the Python Control package.

A SimView simulator of an integrator is available on:

http://techteach.no/simview/integrator

Example 9.1 *An physical integrator: A liquid tank*

See Example 4.1 on page 154 which describes a liquid tank. Assume for simplicity that there is no outflow from the tank. The mathematical model of this system is then

$$h'(t) = \frac{1}{A} q_i(t) \tag{9.10}$$

Taking the Laplace transform of (9.10) gives

$$sH(s) - h_0 = \frac{1}{A} Q_i(s) \tag{9.11}$$

which gives

$$H(s) = \frac{h_0}{s} + \underbrace{\frac{1}{As}}_{G(s)} Q_i(s) \tag{9.12}$$

So, the transfer function is

$$G(s) = \frac{H(s)}{Q_i(s)} = \frac{1}{A} \cdot \frac{1}{s} \tag{9.13}$$

The system is an integrator!

It is actually quite naturally that the liquid tank is an integrator, since the level is proportional to the integral of the inflow. This can be seen by integrating (9.10), which gives

$$h(t) = h(0) + \int_0^t \frac{1}{A} q_i(\theta)\, d\theta \tag{9.14}$$

Whenever you need a concrete example of an integrator, recall the tank!

A SimView simulator of the tank is available on:

http://techteach.no/simview/liquid_tank

Figure 9.4 shows the result of a simulation where the inflow is changed step-wise.

[End of Example 9.1]

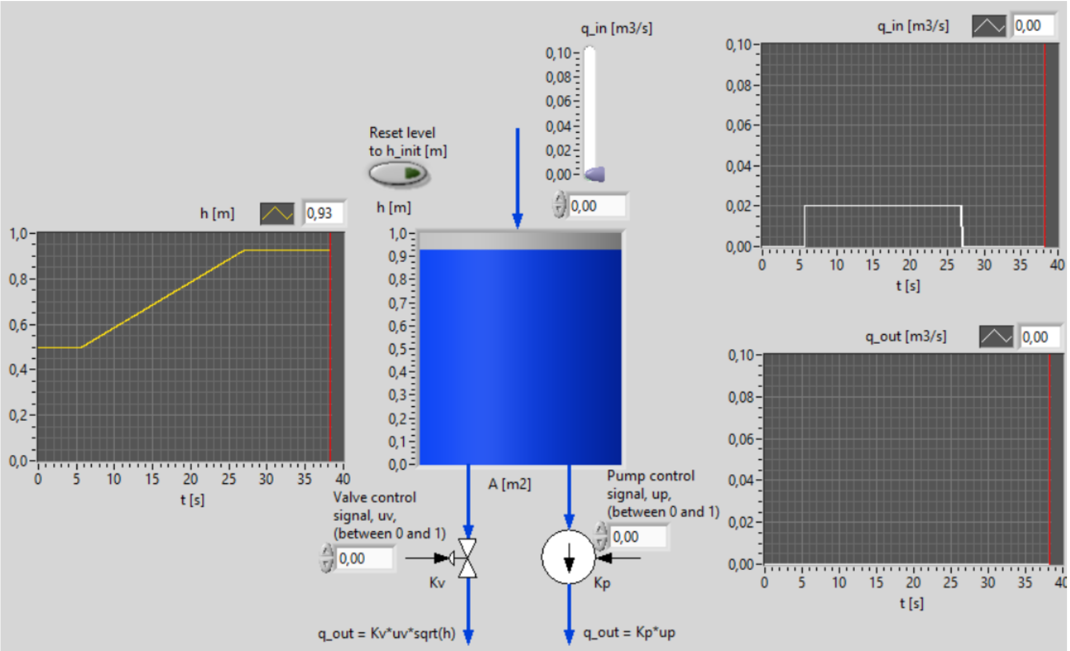


Figure 9.4: Example 9.1: Simulation where the inflow is changed step-wise.

9.3 Time constant systems

9.3.1 The standard model of time constant systems

9.3.1.1 Differential equation

Many mathematical models of material systems, thermal systems, mechanical systems, and electric systems, are in the form of a first order differential equation, which can be written on the following general form:

$$y' = ay + bu \tag{9.15}$$

where y is the output variable, and u is the input variable. a and b are parameters. It has become a tradition within the field of automatic control to write (9.15) on a different form when the purpose is to analyse the dynamic properties of (9.15). That form is

$$t_c y' = Ku - y \tag{9.16}$$

where K is the *gain* and t_c is the *time constant*.

An alternative but equivalent way of writing (9.16) is

$$y' = (Ku - y) / t_c \tag{9.17}$$

which is on a state space form, to be used in e.g. a simulation algorithm.

Both (9.16) and (9.17) can be regarded as a standard time constant model.

The relations between the parameters of (9.15) and (9.16) are:

$$K = -\frac{b}{a} \tag{9.18}$$

$$t_c = -\frac{1}{a} \tag{9.19}$$

The parameters K and t_c give useful information about the dynamic properties of first order differential equations, as you will see soon.

9.3.1.2 Block diagram

Figure 9.5 shows a block diagram of the time constant model (9.17). The block diagram contains an integrator which calculates y from y', while the integrator is not presented in Section 9.3.1.1. The feedback from y has a stabilizing effect on the system, as can be seen in the step response shown in Figure 9.7.

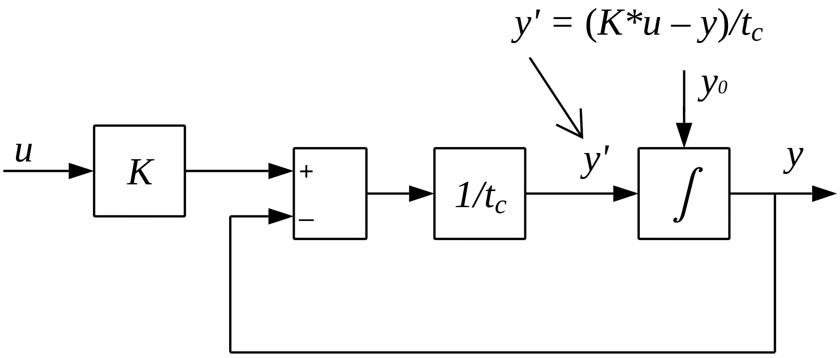


Figure 9.5: Block diagram of the time constant model (9.17).

9.3.1.3 Transfer function

Often the time constant model is in the form of a transfer function. To find the transfer function, we take the Laplace transform of both sides of (9.16), which gives

$$t_c sY(s) = KU(s) - Y(s) \tag{9.20}$$

The transfer function is

$$\frac{Y(s)}{U(s)} = H(s) = \frac{K}{t_c s + 1} \tag{9.21}$$

Here is an example showing how to find K and t_c from the transfer function:

$$H(s) = \frac{3}{4s + 2} = \frac{1.5}{2s + 1} \tag{9.22}$$

The gain is $K = 1.5$, and the time constant is $t_c = 2$ (in a proper time unit, e.g. seconds). The clue is to write the transfer function on the standard form of (9.21), i.e. with 1 as the constant in the denominator.

9.3.1.4 Pole

From (9.21) we find that the pole of a time constant system is

$$p = -\frac{1}{t_c} \tag{9.23}$$

If t_c is positive, as it is for stable systems, the pole is in the left half plane, see Figure 9.6.

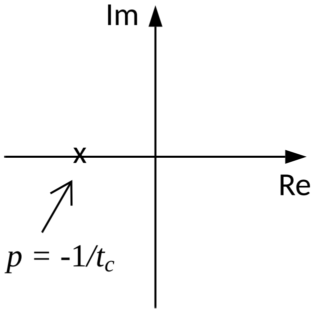


Figure 9.6: The pole of a a time constant system.

9.3.1.5 Dynamics in terms of step response

To analyse the dynamic properties of (9.16), or (9.21), it is common to study the *step response* of the system. We assume that the input signal $u(t)$ is changed as a step of amplitude U at time $t = 0$. From (44.7) $U(s) = \frac{U}{s}$. The Laplace transformed response becomes

$$Y(s) = H(s)U(s) = \frac{K}{t_c s + 1} \cdot \frac{U}{s} \tag{9.24}$$

Taking the inverse Laplace transform using (44.11) gives the system step response:

$$y(t) = KU(1 - e^{-t/t_c}) \tag{9.25}$$

This step response is shown in plotted in Figure 9.7. The annotations on the plot are explained in the following.

The response is actually generated from a simulation of the differential equation model (9.17) assuming the input u is a step, but the response is the same as (9.25). The response is with the following settings:

$$K = 0.5 \tag{9.26}$$

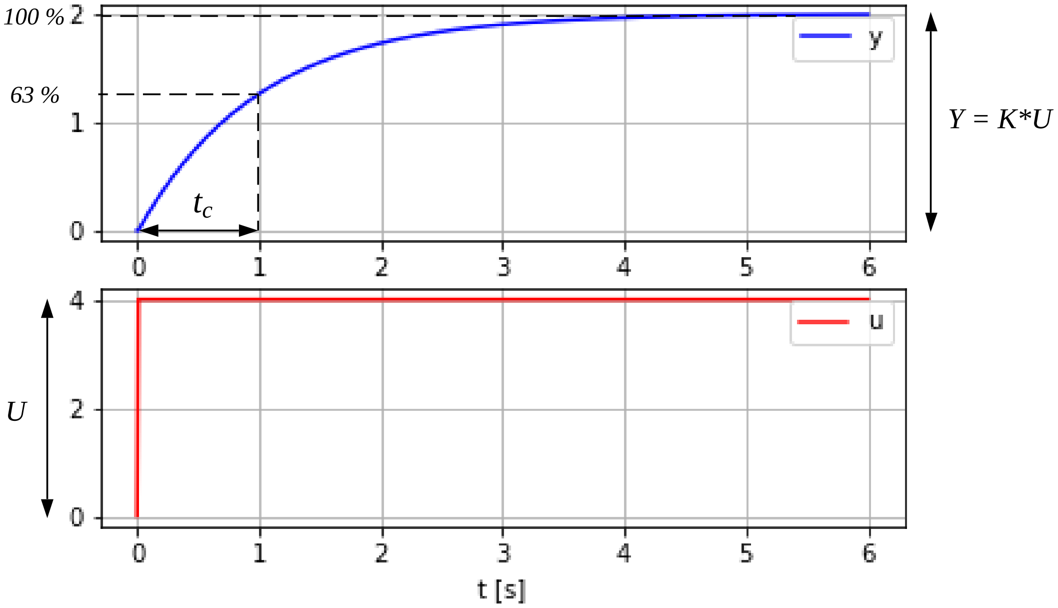


Figure 9.7: Step response of a time constant system.

$$t_c = 1 \text{ s} \tag{9.27}$$

$$U = 4 \tag{9.28}$$

The following Python program implements the simulator which produces the response shown in Figure 9.7.

http://techteach.no/control/python/sim_time_const_sys.py

The program above contains elementary programming code implementing an Euler forward simulation algorithm. Alternatively, you can simulate the step response with the forced_response() function of the Python Control package:

http://techteach.no/control/python/sim_time_const_forced_response_py_con.py

You can also use the step_response() function of the Python Control package.

A SimView simulator of a time constant system is available on:

http://techteach.no/simview/time_constant

The impact of K on the steady state step response

The steady state response due to the input step is

$$Y = \lim_{t\to\infty} y(t) = KU \tag{9.29}$$

which can be found from (9.25) with $t \to \infty$. The steady state response can also be found from the static version of (9.17), i.e. with the time derivative (y') assumed zero. Thus, the step U is amplified with the gain K to get the steady state output value. This can be seen in Figure 9.7, where

$$Y = KU = 0.5 \cdot 4 = 2 \tag{9.30}$$

In Section 8.6 the static transfer function H_s was defined. What is H_s of a time constant system? We get

$$H_s = \frac{Y}{U} = \frac{KU}{U} = K \tag{9.31}$$

So, H_s is equal to the gain K.

The impact of t_c on the transient step response

It has become common to analyse the impact of t_c on the step response by setting $t = t_c$ in (9.25):

$$\begin{aligned} y(T) &= KU(1 - e^{-t_c/t_c}) && (9.32)\\ &= KU(1 - e^{-1}) && (9.33)\\ &= 0.63 \cdot KU && (9.34)\\ &= 0.63 \cdot Y && (9.35)\end{aligned}$$

Thus, at time $t = t_c$ the step response has reached 63 % of the steady state response Y. This is shown in Figure 9.7 where $t_c = 1$ s. This suggests a practical way to read off the time constant from a step response curve, namely as the 63 % rise time of the step response.

Qualitatively, we can state the importance of the time constant as follows: *The less t_c, the faster the system.*

Does the steady state response depend on the time constant? No, because the steady state response is equal to $Y = KU$ which is independent of t_c.

Above, we saw that the time constant is the 63 % rise time of the step response. Here is another interpretation of the time constant: It is the 98 % rise time, and therefore, we may say it is the approximate "settling time" since 98 is very close to 100. This can be seen by setting $t = 4t_c$ in (9.36):

$$y(t = 4t_c) = KU(1 - e^{-4t_c/t_c}) = KU(1 - e^{-4}) = 0.98 \tag{9.36}$$

If you take a look at Figure 9.7, you will see that the step response is indeed very close to the steady state response at $t = 4t_c = 4 \cdot 1$ s.

The impact of K and t_c on the initial slope of the step response

The initial slope, $S_0 = y'(0^+)$, of the step response is indicated with the tangent shown in Figure 9.8.

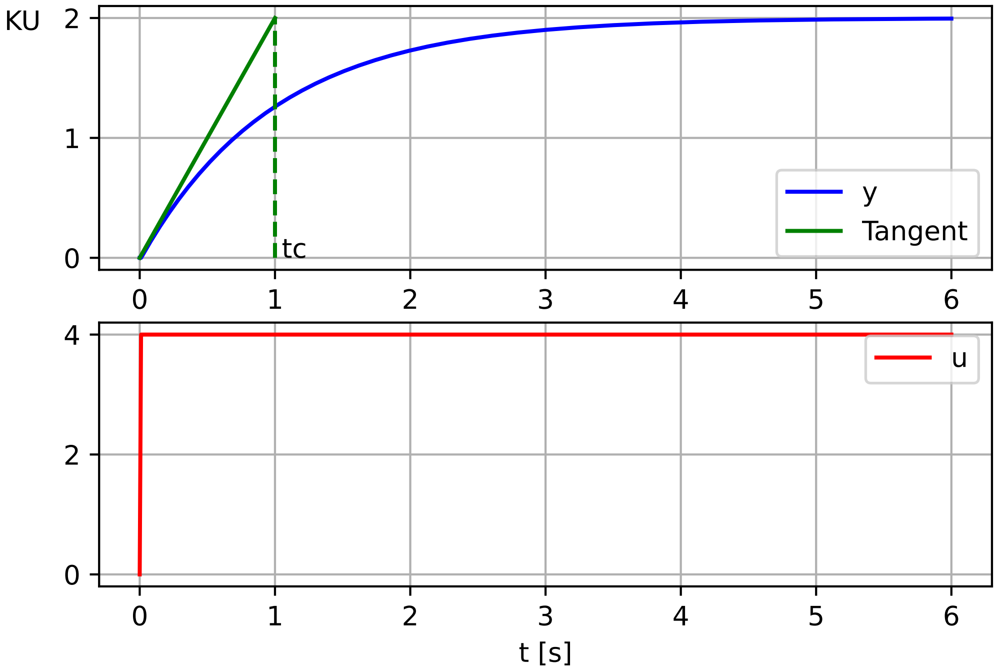


Figure 9.8: Step response with initial slope.

The value of S_0 is can be derived (9.37) from (9.17) as follows. The initial value of y is 0. Then, (9.17) becomes

$$S_0 = y'(0^+) = (KU - 0)/t_c = KU/t_c \tag{9.37}$$

As indicated in Figure 9.8, the tangent representing the initial slope crosses the steady state response, which is KU, at $t = t_c$.

9.3.1.6 Step response of time constant systems when initial state is non-zero

In Section 9.3.1.5 we assumed that the system initially was “at rest” with input u and output y having values zero before the step change in u. However, in practical situations u and y may initially be at rest but with non-zero values before the step change. Assume that u is nonzero:

$$u = u_0 \tag{9.38}$$

Then, the corresponding static response can be calculated from (9.17) with y' set to zero, giving

$$y_0 = Ku_0 \tag{9.39}$$

The model (9.17) can be used to express the dynamics of the deviations from the static operating point (u_0, y_0):

$$\delta y' = K \cdot \delta u - \delta y \tag{9.40}$$

where

$$\delta y = y - y_0 \tag{9.41}$$

and

$$\delta u = u - u_0 \tag{9.42}$$

Assume that u is changed from u_0 as a step of amplitude U, i.e.

$$\delta u(t) = U$$

Then the step response is

$$\delta y(t) = KU(1 - e^{-t/t_c})$$

or, by using (9.41),

$$y(t) = y_0 + KU(1 - e^{-t/t_c})$$

where y_0 is given by (9.39).

Figure 9.9 shows the simulated step response of a system with the following parameters:

$$K = 0.5 \tag{9.43}$$

$$t_c = 1 \text{ s} \tag{9.44}$$

$$u_0 = 2 \tag{9.45}$$

$$U = 4 \tag{9.46}$$

The initial, static value of y is

$$y_0 = Ku_0 = 0.5 \cdot 2 = 1 \tag{9.47}$$

The following Python program implements the simulator. (No program listing is presented here since it is almost identical to the program shown earlier in this chapter.)

http://techteach.no/control/python/sim_time_const_sys_nonzero_init_state.py

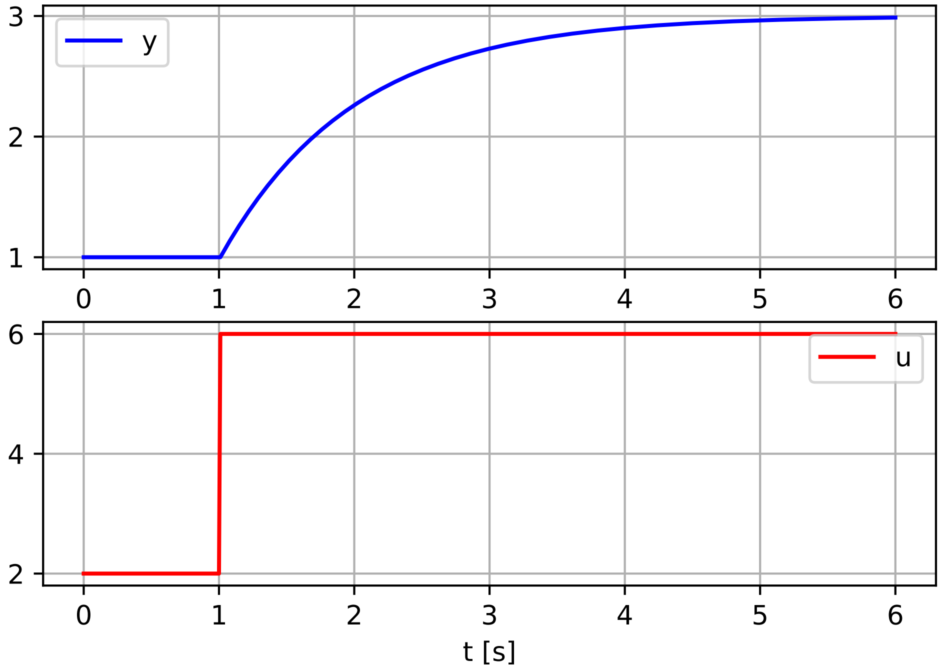


Figure 9.9: Simulation with a non-zero operating point.

9.3.2 Time constant model expanded with process disturbance as input

Above we assumed that the time constant system has one input variables, cf. (9.16). Let us assume that the system has two input signals, namely the control signal u and the process disturbance d, and that the model is

$$t_c y' = K_u u + K_d d - y \tag{9.48}$$

Figure 9.10 shows a block diagram of (9.48).

Taking the Laplace transform and gives

$$t_c sY(s) = -Y(s) + K_u U(s) + K_d D(s) \tag{9.49}$$

from which we find two transfer functions, $H_{y,u}(s)$ and $H_{y,d}(s)$:

$$Y(s) = \underbrace{\frac{K_u}{t_c s + 1}}_{H_{y,u}(s)} U(s) + \underbrace{\frac{K_d}{t_c s + 1}}_{H_{y,d}(s)} D(s) \tag{9.50}$$

The system has one time constant: t_c, and two gains: K_u and K_d.

Above the system has two inputs, u and d. If the system has more than two inputs, the model is expanded naturally (not described in detail here).

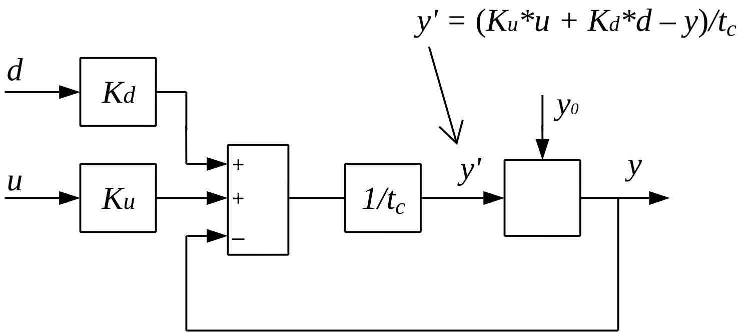


Figure 9.10: Block diagram of (9.48).

Example 9.2 *First order system: Heated liquid tank*

In Example 4.2 we developed a mathematical model of heated liquid tank (a thermal system). The model is repeated here:

$$cmT' = P + cF\,(T_{\text{in}} - T) + G(T_a - T) \tag{9.51}$$

Let us for simplicity assume that the tank is well isolated so that

$$G \approx 0 \tag{9.52}$$

We will now calculate the transfer functions from P to T and from T_{in} to T. Taking the Laplace transform of (9.51) while setting the initial value T_0 to zero gives[1]

$$cmsT^*(s) = P^*(s) + cF\,[T_{\text{in}}^*(s) - T^*(s)] \tag{9.53}$$

From (9.53) we will find

$$T^*(s) = \underbrace{\frac{K_1}{t_c s + 1}}_{H_1(s)} P^*(s) + \underbrace{\frac{K_2}{t_c s + 1}}_{H_2(s)} \theta_i^*(s) \tag{9.54}$$

The gains and the time constant of each of the two transfer functions are

$$K_1 = \frac{1}{cF} \tag{9.55}$$

$$K_2 = 1 \tag{9.56}$$

$$t_c = \frac{m}{F} = \frac{\text{Mass}}{\text{Flow}} \tag{9.57}$$

Comments:

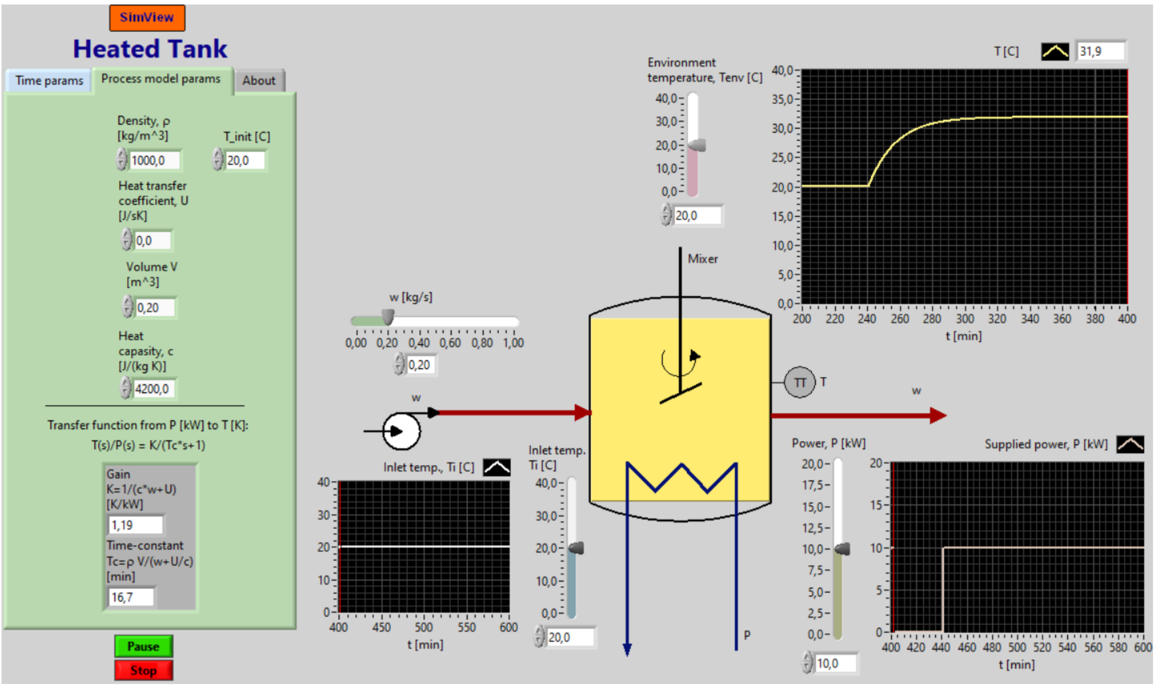
[1] I am here using superscript * to denote Laplace transformed variables, cf. comments about notation in Appendix 44.

- The time constant, which represents the “dynamics”, is the same for both transfer functions $H_1(s)$ and $H_2(s)$.
- In many applications the flow F may change. Assume that the flow is decreased. The dynamic properties of the system then change:
  - According to (9.55) the gain from P to T increases, and hence the T becomes more sensitive to P, giving higher value of T for a given change of P.
  - According to (9.57) the time constant increases, causing a more sluggish response in T to a change in P.
- In chemistry (incl. bioprocesses like biogas reactor technology), the ratio mass/flow in (9.57) is denoted the hydraulic retention time.

A SimView simulator of the heated tank is available on:

http://techteach.no/simview/heated_tank

Figure 9.11 shows the simulation of the step response of the heated tank.



Heated Tank

Density, ρ [kg/m^3]: 1000,0

T_init [C]: 20,0

Heat transfer coefficient, U [J/sK]: 0,0

Volume V [m^3]: 0,20

Heat capacity, c [J/(kg K)]: 4200,0

Transfer function from P [kW] to T [K]:
T(s)/P(s) = K/(Tc*s+1)

Gain K=1/(c*w+U) [K/kW]: 1,19

Time-constant Tc=ρ V/(w+U/c) [min]: 16,7

w [kg/s]: 0,20

Environment temperature, Tenv [C]: 20,0

T [C]: 31,9

Inlet temp. Ti [C]: 20,0

Power, P [kW]: 10,0

Figure 9.11: Example 9.2: Simulation of the step response of a heated tank.

[End of Example 9.2]

9.4 Second order systems

9.4.1 Mathematical model

9.4.1.1 Transfer function model

A standard second order transfer function model (with u as input variable and y as output variable) is

$$Y(s) = \frac{K{\omega_0}^2}{s^2 + 2\zeta\omega_0 s + {\omega_0}^2}U(s) \equiv \frac{K}{\left(\frac{s}{\omega_0}\right)^2 + 2\zeta\frac{s}{\omega_0} + 1}U(s) \tag{9.58}$$

where K is the *gain*, ζ (zeta) [dimension 1] is the *relative damping factor,* and ω_0 [rad/s] is the *undamped resonance frequency.*

9.4.1.2 Differential equation

A differential equation model which corresponds to the transfer function (9.58) can be found as follows: Cross-multiplying (9.58) by the denominator gives

$$\left(s^2 + 2\zeta\omega_0 s + {\omega_0}^2\right)Y(s) = K{\omega_0}^2 U(s)$$

Taking the inverse Laplace transform, and neglecting the initial values, gives the following differential equation:

$$y'' + 2\zeta\omega_0 y' + {\omega_0}^2 y = K{\omega_0}^2 u$$

or

$$y'' = K{\omega_0}^2 u - 2\zeta\omega_0 y' - {\omega_0}^2 y \tag{9.59}$$

9.4.1.3 State space model

A state space model corresponding to (9.59) can be found by defining the following two state variables:

$$x_1 = y \tag{9.60}$$

$$x_2 = y' \tag{9.61}$$

(9.59) can then be written as the following linear state space model:

$$\underbrace{\begin{bmatrix} {x_1}' \\ {x_2}' \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -{\omega_0}^2 & -2\zeta\omega_0 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ K{\omega_0}^2 \end{bmatrix}}_{B}u \tag{9.62}$$

$$y = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C}\underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \end{bmatrix}}_{D}u \tag{9.63}$$

Example 9.3 *Second order system: Mass-spring-damper*

In Problem 8.3 the transfer function from force F to position y of a mass-spring-damper system is to be found. The transfer function is repeated here:

$$H(s) = \frac{Y(s)}{F^*(s)} = \frac{1}{ms^2 + Ds + K} \tag{9.64}$$

To find the standard parameters, we must transform the transfer function to one of the equivalent standard forms given by (9.58). Let us here choose the first one:

$$H(s) = \frac{\overbrace{\frac{1}{m}}^{K_1{\omega_0}^2}}{\underbrace{s^2 + \frac{D}{m}s + \frac{K}{m}}_{s^2+2\zeta\omega_0 s+{\omega_0}^2}} \tag{9.65}$$

By equating the coefficients and using the following parameters values: $m = 20$ kg, $D = 4$ N/(m/s) and $K = 2$ N/m, we get (I use K_1 as symbol of the gain since K is already used for the spring coefficient):

$$K_1 = \frac{1}{K} = 0.5 \text{ [m/N]} \tag{9.66}$$

$$\omega_0 = \sqrt{\frac{K}{m}} = \sqrt{0.1} = 0.32 \text{ [rad/s]} \tag{9.67}$$

$$\zeta = \frac{D}{2\sqrt{mK}} = 0.32 \tag{9.68}$$

A SimView simulator of the mass-spring-damper system is available on:

http://techteach.no/simview/mass_spring_damper

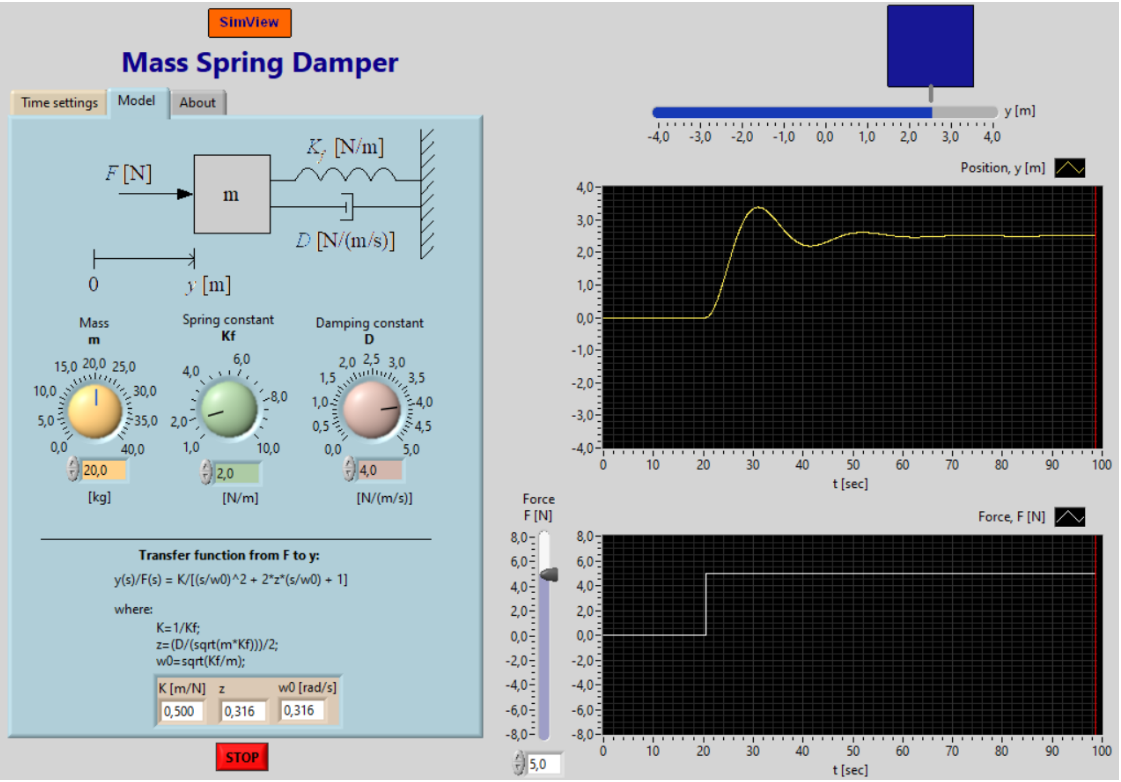
Figure 9.12 shows the result of a simulation.

[End of Example 9.3]

9.4.2 Classification of second order systems

We will classify second order systems from the shape of the step response. We assume that the input variable $u(t)$ is a step of amplitude U, which Laplace transformed is $U(s) = U/s$. Then the Laplace transformed time-response becomes

$$Y(s) = H(s)U(s) = \frac{K{\omega_0}^2}{s^2 + 2\zeta\omega_0 s + {\omega_0}^2}\frac{U}{s} \tag{9.69}$$



Mass Spring Damper

Mass m: 20,0 [kg]

Spring constant Kf: 2,0 [N/m]

Damping constant D: 4,0 [N/(m/s)]

Force F [N]: 5,0

Transfer function from F to y:

```
y(s)/F(s) = K/[(s/w0)^2 + 2*z*(s/w0) + 1]

where:
        K=1/Kf;
        z=(D/(sqrt(m*Kf)))/2;
        w0=sqrt(Kf/m);
```

K [m/N]	z	w0 [rad/s]
0,500	0,316	0,316

Figure 9.12: Example 9.3: Simulation of mass-spring-damper system.

The shape of the time-response $y(t)$, which is calculated as the inverse Laplace transform of $Y(s)$, depends on the poles, cf. Section 8.5. The poles are the roots of the characteristic equation $a(s)$:

$$a(s) = s^2 + 2\zeta\omega_0 s + {\omega_0}^2 = 0 \tag{9.70}$$

The poles p_1 and p_2 are the roots of $a(s)$ and becomes

$$p_1,\ p_2 = -\zeta\omega_0 \pm \sqrt{\zeta^2 - 1}\ \omega_0 \tag{9.71}$$

The value of ζ determines whether the poles are real or complex conjugate:

- If $\zeta \geqslant 1$, the poles are real and given by (9.71).
- If $0 \leq \zeta < 1$, the poles are complex conjugate:

$$p_1,\ p_2 = \underbrace{-\zeta\omega_0}_{\text{Re}} \pm j\underbrace{\sqrt{1-\zeta^2}\ \omega_0}_{\text{Im}} \tag{9.72}$$

Figure 9.13 shows the pole placement when the poles are complex conjugate.

Figure 9.14 classifies second order systems by the value of ζ. (This is a common way to do the classification.) The step responses referenced in the figure can be calculated by taking the inverse Laplace transform of (9.69), but the detailed calculations are not shown here.

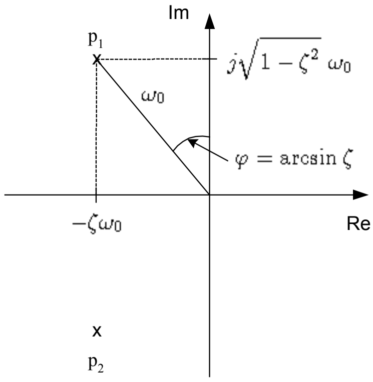


Figure 9.13: Pole placement of second order systems when the poles are complex conjugate. The poles are given by (9.72).

In the following are given simulated step responses and pole plots for representative examples of overdamped, underdamped, and undamped systems. The parameter values are shown on the front panels of the simulators in the respective figures.

In all cases the steady state value of the step response is

$$y_s = KU \tag{9.73}$$

because the static transfer function is K.

9.4.2.1 Overdamped systems

Figure 9.15 shows the step response and the poles for an example of an overdamped system. The shown step response is generated with the following Python program9.3 which simulates a second order system, with model parameter settings as shown in Figure 9.15.

http://techteach.no/control/python/sim_2order_sys_forced_response_py_con.py

Comments:

- The step response has no overshoot.
- The poles p_1 and p_2 are real and distinct:

$$p_1,\ p_2 = -\zeta\omega_0 \pm \sqrt{\zeta^2 - 1}\,\omega_0 \tag{9.74}$$

The transfer function can therefore be written on the form

$$H(s) = \frac{Kp_1p_2}{(s-p_1)(s-p_2)} = \frac{K}{(T_1s+1)(T_2s+1)} \tag{9.75}$$

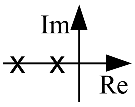
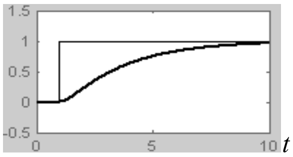
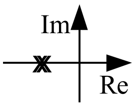
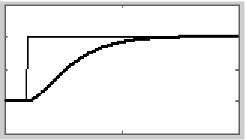
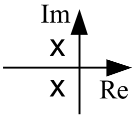
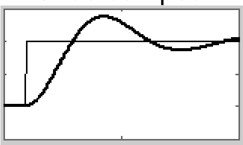
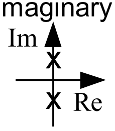
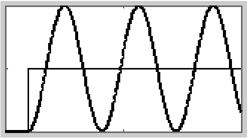
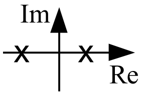
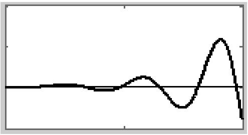
Value of ζ	**Poles p_1 and p_2**	**Type of step response $y(t)$**	**Reference to y(t) in Appendix**
$\zeta > 1$	Real and distinct 	Overdamped 	(B.17)
$\zeta = 1$	Real and multiple 	Critically damped 	(B.18)
$0 < \zeta < 1$	Complex conj. 	Underdamped 	(B.23)
$\zeta = 0$	Imaginary 	Undamped 	(B.23)
$\zeta < 0$	Pos. real part 	Unstable 	(B.17) or (B.18) or (B.23)

Figure 9.14: Classification of second order systems by the value of ζ.

This implies that the second order system can be split into two first order subsystems having time constants T_1 and T_2, respectively. The largest of these time constants can be denoted the *dominating time constant*.

9.4.2.2 Underdamped system

Figure 9.16 shows the step response and the poles for an example of an underdamped system. The shown step response is generated with the Python program9.3 given in Section 9.4.2.1, with the model parameter settings as shown in Figure 9.16.

Comments:

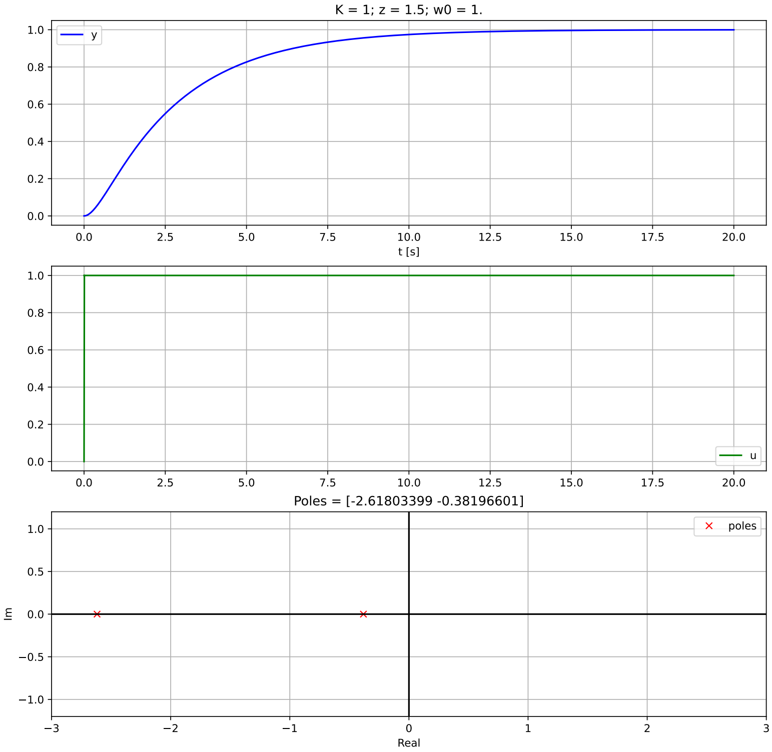


Figure 9.15: Step response and the poles of an example of an overdamped system.

- The poles are complex conjugate:

$$p_1,\ p_2 = -\zeta\omega_0 \pm j\sqrt{1-\zeta^2}\ \omega_0 \tag{9.76}$$

- The less ζ, the less damping in the step response. It can be shown that the less ζ, the more dominating imaginary part over the real part of the poles. This is a general property of poles: The larger imaginary part relative to the real part, the less damping in the time-response. Figure 9.17 shows the step-response for various values of ζ.

- The *overshoot factor* δ of the step response is defined as

$$\delta = \frac{y_{\max} - y_s}{y_s} \tag{9.77}$$

  where y_s is the steady state value of the step-response. It can be shown that δ is a function of the relative damping factor ζ, as follows:

$$\delta = e^{-\zeta\pi/\sqrt{1-\zeta^2}} \tag{9.78}$$

  δ is plotted as a function of ζ in Figure 9.18.

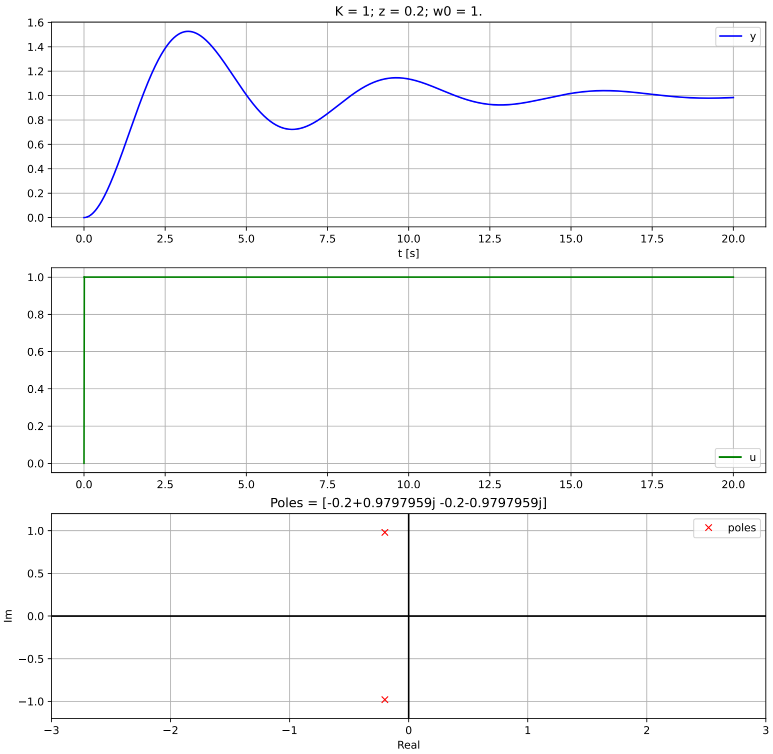


Figure 9.16: Step response and poles of an example of an underdamped system.

The inverse function of (9.78) is

$$\zeta = \frac{|\ln\delta|}{\sqrt{\pi^2 + (\ln\delta)^2}} \tag{9.79}$$

A few examples: Overshoot $\delta = 0.1$, that is, 10% overshoot, corresponds to $\zeta = 0.6$. Overshoot $\delta = 0$ (zero overshoot) corresponds to $\zeta = 1$ (critically damped system).

- Simulations shows that the 63% response-time of the step response is approximately

$$t_r \approx \frac{1.5}{\omega_0} \tag{9.80}$$

ω_0 expresses in a way how quick the system is. ω_0 is the distance from origin to the poles, see Figure 9.13. This is a general property of poles: The longer distance from origin, the faster the dynamics of the system.

- It can be shown that the frequency in the oscillations are

$$\beta = \sqrt{1-\zeta^2}\,\omega_0 \text{ [rad/s]} \tag{9.81}$$

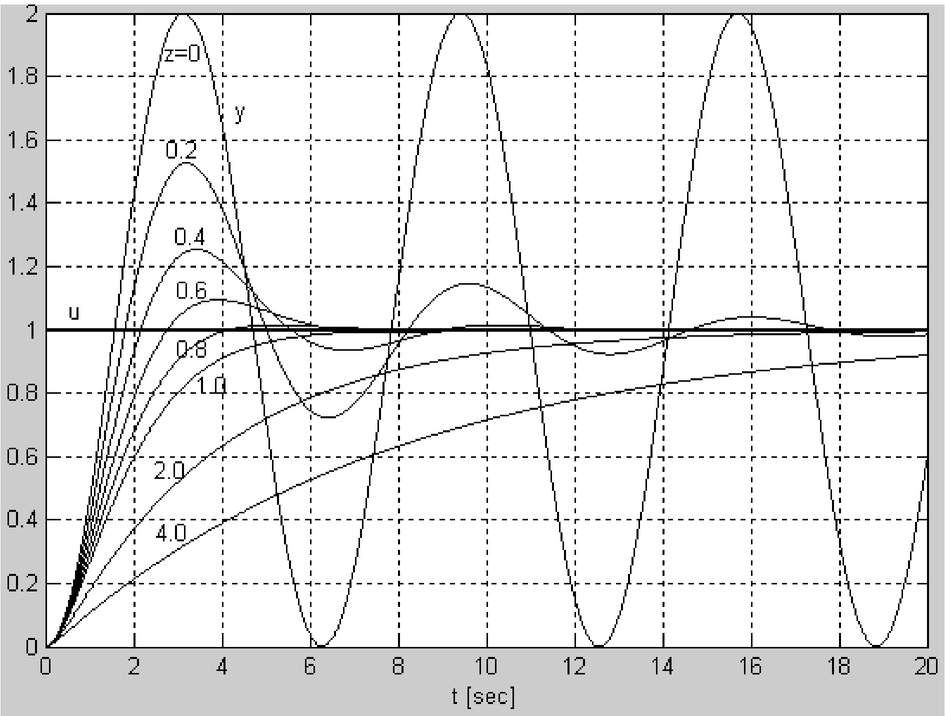


Figure 9.17: Step-response for various values of ζ of second order systems.

9.4.2.3 Undamped system

Figure 9.19 shows the step response and the poles for an example of an undamped system. The shown step response is generated with the Python program9.3 given in Section 9.4.2.1, with the model parameter settings as shown in Figure 9.19.

Comments:

- The step response is undamped, steady state oscillations:

$$y(t) = KU\,(1 - \cos\omega_0 t) \tag{9.82}$$

  The frequency of the oscillations in rad/s is ω_0 — therefore the name *undamped resonance frequency*.

- The poles are purely imaginary:

$$p_1,\ p_2 = \pm j\omega_0 \tag{9.83}$$

  The real part is zero, which is an explanation of why the step response is undamped. In general, damping is due non-zero real part of poles.

9.5 Time delays

In many systems there is a *time delay* or dead-time in the signal flow, for example with material transport on a conveyor belt, see Figure 9.20. In this application, the relation

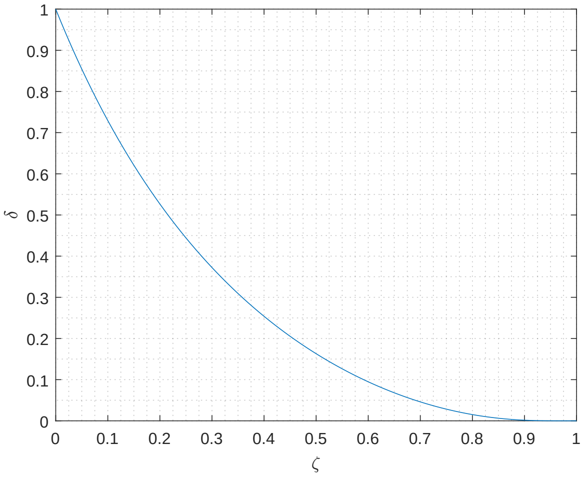


Figure 9.18: Overshoot factor δ plotted as a function of the relative damping factor ζ, cf. (9.78).

between the input variable F_{in} and the output variable F_{out} is

$$F_{\text{out}}(t) = F_{\text{in}}(t - t_d) \tag{9.84}$$

where t_d is the time delay which is the transportation time on the belt. In other words: The outflow at time t is equal to the inflow t_d time units ago.

What is the transfer function of a time delay? Taking the Laplace transform of (9.84) using (44.19):

$$F_{\text{out}}^*(s) = \underbrace{e^{-t_d s}}_{H(s)} F_{\text{in}}^*(s) \tag{9.85}$$

Thus, the transfer function of a time delay of t_d [time unit] is

$$H(s) = e^{-t_d s} \tag{9.86}$$

Figure 9.21 shows a simulation of a time delay. The time delay is $t_d = 1$ sec.

The simulation is made with the following SimView simulator:

http://techteach.no/simview/timedelay

9.5.1 Approximation of time delay by Padé approximation

Some times it is difficult to use the irrational transfer function $e^{-t_d s}$ in calculations and simulations with transfer functions. A Padé-approximation of $e^{-t_d s}$ can then help as it approximates $e^{-t_d s}$ with a rational transfer function, i.e. a transfer function with

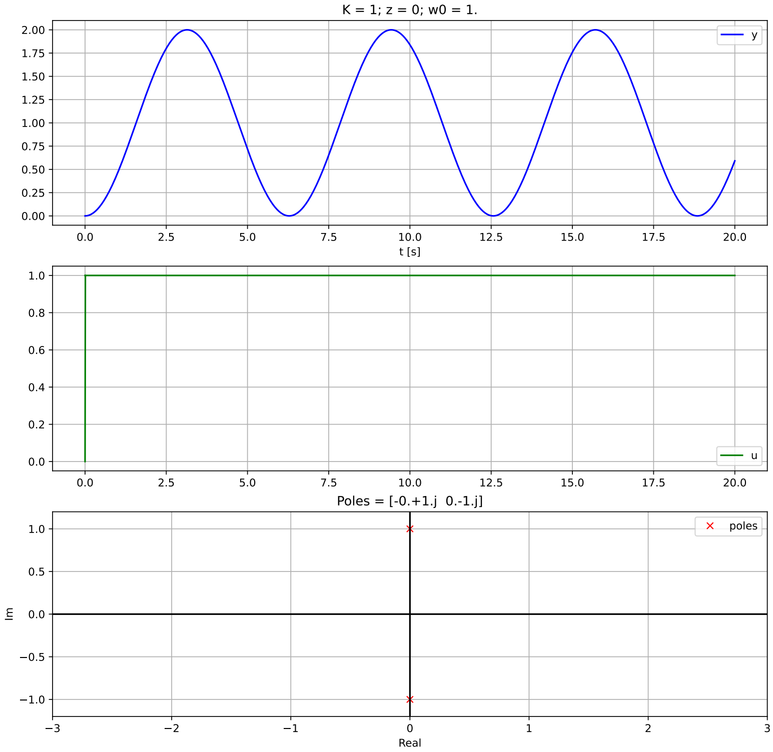


Figure 9.19: Step response and the poles of an example of an undamped system.

polynomials of s in both the numerator and the denominator. A Padé-approximation is a function of

- the time delay t_d
- the order n you select for the approximation

For example, a Padé-approximation of order $n = 2$ is

$$e^{-t_d s} \approx \frac{1 - \frac{t_d}{2}s + \frac{t_d^2}{12}s^2}{1 + \frac{t_d}{2}s + \frac{t_d^2}{12}s^2} \tag{9.87}$$

Ch. 41.2.6 shows how to create Padé-approximations with the Python Control package, and how to use them in simulations.

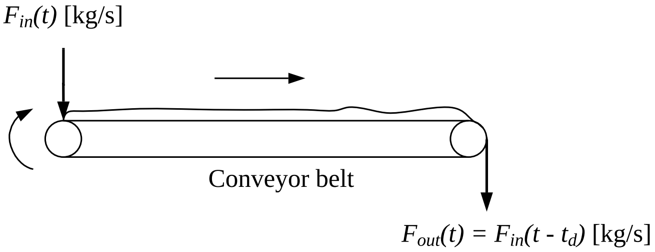


Figure 9.20: Time delay (t_d) of a conveyor belt.

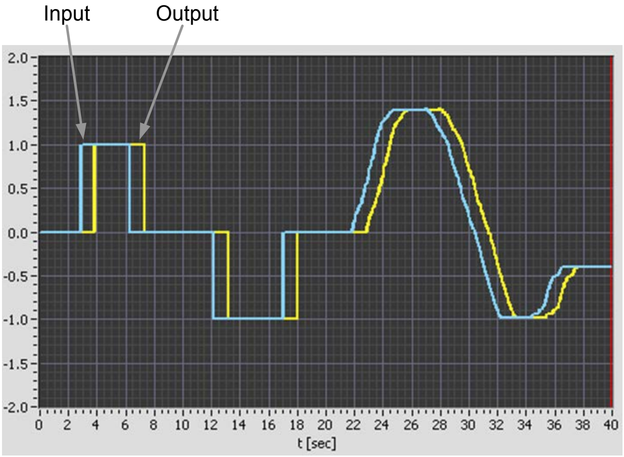


Figure 9.21: Output is equal to input, time delayed 1 sec.

9.6 Higher order systems

Systems having higher order of the denominator polynomial of the transfer function than one, are so-called higher order systems, or more specifically, second order systems, third order systems and so on. A serial connection of first order systems results in a higher order system. (But not all possible higher order systems can be constructed by serial connection of first order systems.) When transfer functions are connected in series, the resulting transfer function is the product of the individual transfer functions, cf. Figure 8.3. As an example, Figure 9.22 shows a second order system consisting of “two time constants” connected in series.

The combined transfer function becomes

$$H(s) = \frac{1}{(t_{c_1}s + 1)(t_{c_2}s + 1)} = \frac{Y_2(s)}{U(s)} \tag{9.88}$$

The figure also shows the step responses in the system. It is assumed that $t_{c_1} = 1$, $t_{c_2} = 1$ and $K = 1$. Observe that each first order systems makes the response become more sluggish, as it has a smoothing effect.

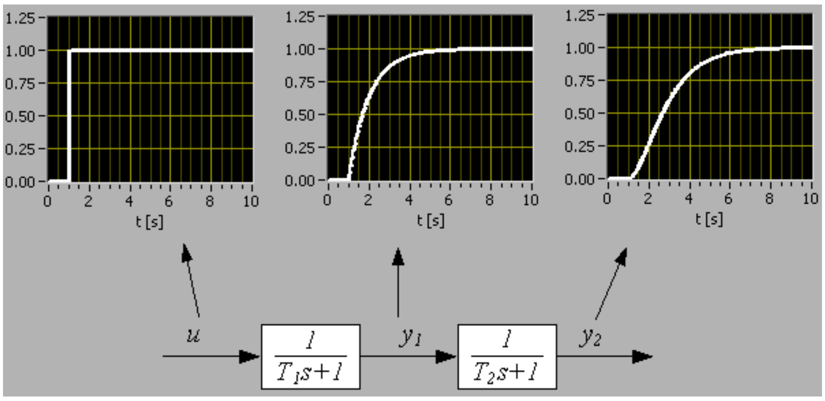


Figure 9.22: Step responses in a second order system. (In the figure, $T_1 = t_{c_1}$and $T_2 = t_{c_2}$.)

Let us define the *response-time* t_r as the *time it takes for a step response to reach 63% of its steady state value*. For time constant systems, the response-time is equal to the time constant:

$$t_r = t_c \tag{9.89}$$

For higher order systems (order larger than one) it turns out that the response-time can be roughly estimated as the sum of the time constants of the assumed serial subsystems that make up the higher order system:

$$t_r \approx \sum_i t_{c_i} \tag{9.90}$$

As an example, the response-time of the system shown in Figure 9.22 is

$$t_r \approx 1 + 1 = 2 \text{ s} \tag{9.91}$$

Does the simulation shown in Figure 9.22 confirm this?[2]

A SimView simulator of the following second order transfer function with a zero, i.e. a system with two poles and one zero,

$$H(s) = \frac{Y(s)}{U(s)} = \frac{b_1 s + b_0}{s^2 + a_1 s + a_0} \tag{9.92}$$

is available on:

http://techteach.no/simview/transferfunction

[2] Yes

9.7 *Problems for Chapter 9*

Problem 9.1 *Dynamic response in tank*

See Problem 8.2. The transfer function from $W_{out}(s)$ to $H(s)$ is

$$\frac{H(s)}{W_{\mathrm{out}}(s)} = -\frac{1}{\rho As} = H_2(s) \tag{9.93}$$

1. Does this transfer function represent integrator dynamics?
2. Assume that $w_{\mathrm{out}}(t)$ is a step from 0 to W at time $t = 0$. Calculate the response $h(t)$ that this excitation causes in the level h. You are required to base your calculations on the Laplace transform.

Problem 9.2 *Tank as integrator*

Figure 9.23 shows an isolated tank (having zero heat transfer through the walls).

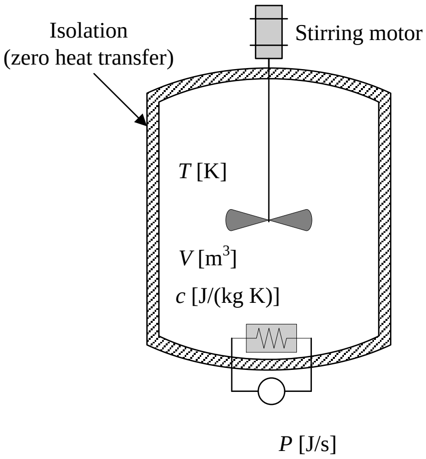


Figure 9.23: Isolated tank.

Show that the tank dynamically is an integrator with the power P as input variable and the temperature T as output variable. (Hint: Study the transfer function from P to T.)

Problem 9.3 *Time constant dynamics*

Given a system with gain K, time constant t_c, and time delay t_d. The system is initially in steady (static) state, with the value of the output variable, y, equal to y_0. At time t_0, the system is affected by a step change from u_0 to u_1 at the input, u. Plot (by hand is ok) principally $u(t)$ and $y(t)$ manually in their respective plots, showing how the given information appears in the plots. The origin (0, 0) shall appear in both plots.

Problem 9.4 *Time constant dynamics - 2*

Assume a system with gain −3, time constant 10 s, and time delay 2 s. Sketch (manually) principally the step response of the system assuming the input step has amplitude 4 appearing at time 50 s. Before the step appears, the system output is at rest with value 15.

Problem 9.5 *Time constant dynamics from transfer function*

Calculate the gain and the time constant of the following transfer function:

$$H(s) = \frac{Y(s)}{U(s)} = \frac{2}{4s+8} \tag{9.94}$$

Draw by hand roughly the step response of $y(t)$ due to a step of amplitude 6 in u from the following information:

- The steady state value of the step response
- The time constant
- The initial slope of the step response

Problem 9.6 *Deriving transfer function from step response*

Figure 9.24 shows the temperature response T of a thermal system due to a step of amplitude 1 kW in the supplied power P.

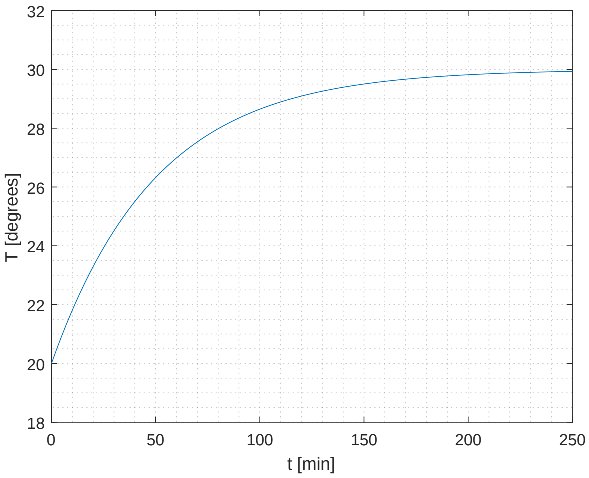


Figure 9.24: Problem 9.6: Temperature response due to step in supplied power.

Find the transfer function from δP (power) to δT (temperature) where δ indicates deviations from the steady state values. Assume that the system is of first order (a time constant system).

Problem 9.7 *Time constant dynamics of kettle*

A kettle and a mathematical model based on energy balance are presented in Ch. 39.6. The model is repeated here:

$$T'(t) = \{[P(t - t_d) + G(T_a - T(t))]\}/C \tag{9.95}$$

1. Derive the transfer function, $H(s)$, from power $P^*(s)$ to temperature $T^*(s)$. (Here, I use star for the Laplace transformed variables since the symbols in the time domain are capitals.)

2. What is the gain, K, and the time constant, t_c? What are their numerical values? (Parameter values are given in Table 39.5.)

Problem 9.8 *Dynamics of an RC circuit*

Figure 9.25 shows an RC-circuit (the circuit contains the resistor R and the capacitor C).

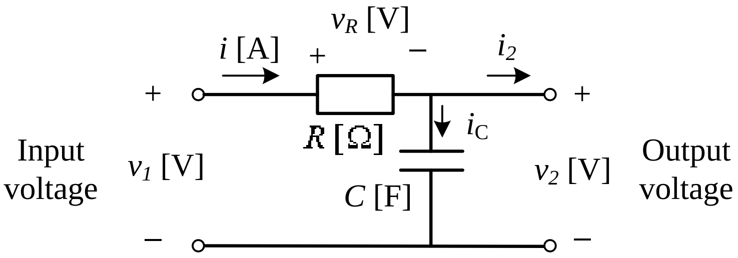


Figure 9.25: Problem 9.8: RC-circuit.

The RC-circuit is frequently used as an analog lowpass filter: Signals of *low* frequencies *passes* approximately unchanged through the filter, while signals of high frequencies are approximately filtered out (stopped). It can be shown that a mathematical model of the RC circuit is

$$RCv'_{\text{out}} = v_{\text{in}} - v_{\text{out}} \tag{9.96}$$

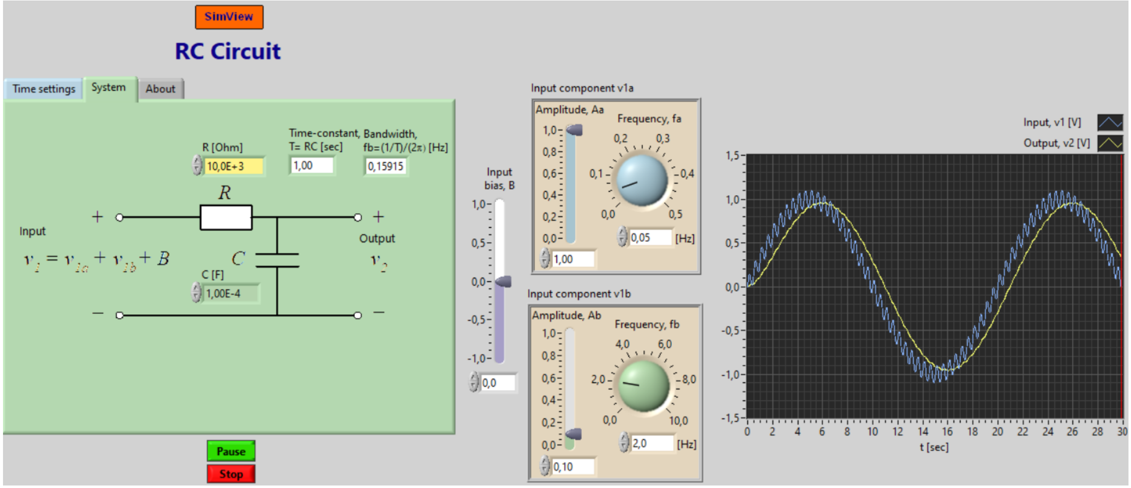
1. Calculate the transfer function $H(s)$ from $V_{\text{in}}(s)$ to $V_{\text{out}}(s)$, and calculate the gain and the time constant of $H(s)$.

2. Assume that the RC circuit is used as a signal filter. Assume that the capacitance C [F] is fixed. How can you adjust the resistance R (increase or descrease) so that the filter performs stronger filtering or, in other words: is more sluggish.

A SimView simulator of the RC circuit is available on:

http://techteach.no/simview/rc_circuit

Figure 9.26 shows results of a simulation with the simulator.


SimView

RC Circuit

R [Ohm]: 10,0E+3

Time-constant, T= RC [sec]: 1,00

Bandwidth, fb=(1/T)/(2π) [Hz]: 0,15915

C [F]: 1,00E-4

Input bias, B: 0,0

Input component v1a: Amplitude, Aa: 1,00; Frequency, fa: 0,05 [Hz]

Input component v1b: Amplitude, Ab: 0,10; Frequency, fb: 2,0 [Hz]

Figure 9.26: Problem 9.8: Simulation of an RC circuit.

Problem 9.9 *Second order system: Position control system*

Figure 9.27 shows an angular position control system of an electric motor.

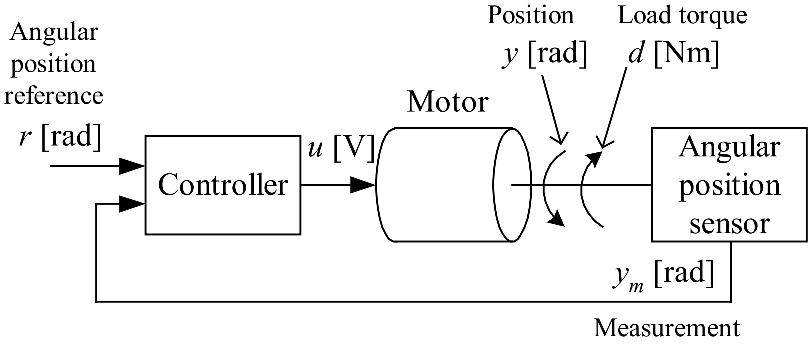


Figure 9.27: Problem 9.9: Angular position control system of an electric motor.

Figure 9.28 shows a block diagram with transfer functions of the control system.

Assume that the transfer functions are as follows:

$$H_p(s) = \frac{K_u}{(t_c s + 1)\, s} = \frac{1}{(s+1)\, s} \tag{9.97}$$

$$H_d(s) = \frac{K_d}{(t_c s + 1)\, s} = \frac{-1}{(s+1)\, s} \tag{9.98}$$

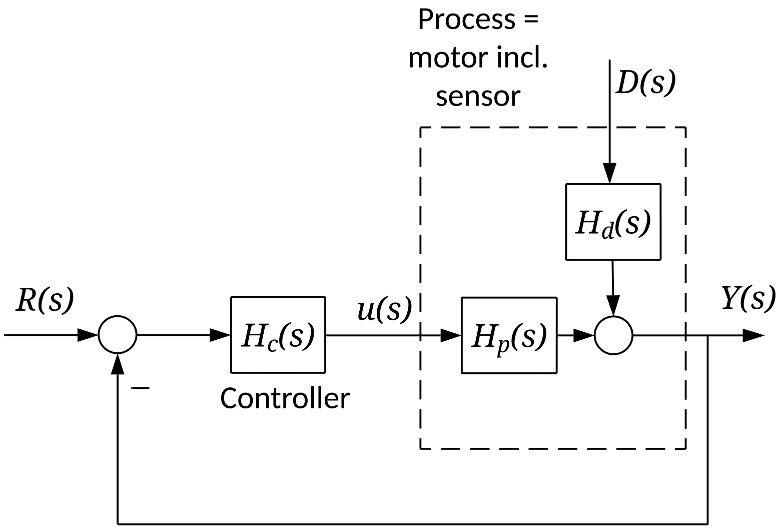


Figure 9.28: Problem 9.9: Block diagram of the positional control system.

$$H_c(s) = K_c \quad \text{(proportional controller)} \tag{9.99}$$

Comments about $H_p(s)$: The factor $K_u/(t_c s + 1)$, which represents a time constant system, is the transfer function from control signal u to speed, say v. The factor $1/s$, which is an integrator, is the transfer function from v to (measured) position y. The process to be controlled is thus a "time constant with integrator" process.

1. Derive the transfer function $H_t(s)$ ("t" for "tracking") from position reference r to position measurement y_m. You can use the numerical model parameters given in (9.97). (This transfer function is generally denoted the tracking transfer function.)

2. $H_t(s)$ turns out to be a second order transfer function. What are the parameters K, ζ and ω_0 of $H_t(s)$ (possibly) in terms of the controller gain K_c.

3. You will see that both ζ and ω_0 are functions of K_c, and you may calculate K_c from either a specified ζ or from a specified ω_0. Of these two parameters, it is ζ that determines the stability. Since it is necessary that a control system has acceptable stability, K_c should be calculated from a specified ζ rather from a specified ω_0. Let us say that $\zeta = 0.6$ is a good value. $\zeta = 0.6$ gives 10% overshoot ($\delta = 0.1$) in the step response, cf. (9.79). Calculate K_c so that this specification is obtained.

4. With the value of K_c that you have calculated above, what is the 63 % response time of the control system?

5. Make a Python program which simulates the solution above (you may simulate with the forced_response function of the Python Control package). Simulate the step response of the control system, i.e. r is a step, and you simulate the response in y_m. Are the overshoot and response time confirmed with the simulations?

Problem 9.10 *Time delay of pipeline*

For a pipeline of length 0.5 m and cross sectional area of 0.01 m^2 filled with liquid which flows with a volumetric flow 0.001 m^3/s, calculate the time delay (transport delay) from inlet to outlet of the pipe.

Problem 9.11 *Response time of compound system*

Assume that a system can be well described by 3 time constant systems in series, with the following time constants respectively: 0.5, 1.0, and 2.0 sec. What is the approximate response time of the system?

Problem 9.12 *Simulation of Padé approximation with Python Control Package*

Given a "time constant with time delay" system, with gain 2, time constant 4, and time delay 1 sec. Assume that the system input is a step of amplitude 5. Simulate the response of the system with the forced_response() function of the Python Control Package. The time delay should be represented by a Padé approximation of order 10.

9.8 *Solutions to problems for Chapter 9*

Solution to Problem 9.1

1. Yes! Because the transfer function has the form of K_i/s.
2. The Laplace transform of the response is

$$H(s) = H_2(s)W_{\text{out}}(s) = -\frac{1}{\rho As}W_{\text{out}}(s) \tag{9.100}$$

Since $w_{\text{out}}(t)$ is a step of amplitude W at $t = 0$, $w_{\text{out}}(s)$ becomes, according to (44.7),

$$W_{\text{out}}(s) = \frac{W}{s} \tag{9.101}$$

With this $W_{\text{out}}(s)$, (9.100) becomes

$$H(s) = -\frac{1}{\rho As}\frac{W}{s} \tag{9.102}$$

According to (44.8),

$$h(t) = -\frac{W}{\rho A}t \tag{9.103}$$

That is, the response is a ramp with negative slope.

Solution to Problem 9.2

Energy balance:

$$c\rho VT' = P \tag{9.104}$$

Laplace transformation:

$$c\rho V\left[sT^*(s) - T_0\right] = P^*(s) \tag{9.105}$$

which yields

$$T^*(s) = \frac{1}{s}T_0 + \underbrace{\frac{1}{c\rho Vs}}_{H(s)}P^*(s) \tag{9.106}$$

The transfer function is

$$H(s) = \frac{T^*(s)}{P^*(s)} = \frac{1}{c\rho Vs} = \frac{K}{s} \tag{9.107}$$

which is the transfer function of an integrator with gain $K = 1/(c\rho V)$.

Solution to Problem 9.3

See Figure 9.29.

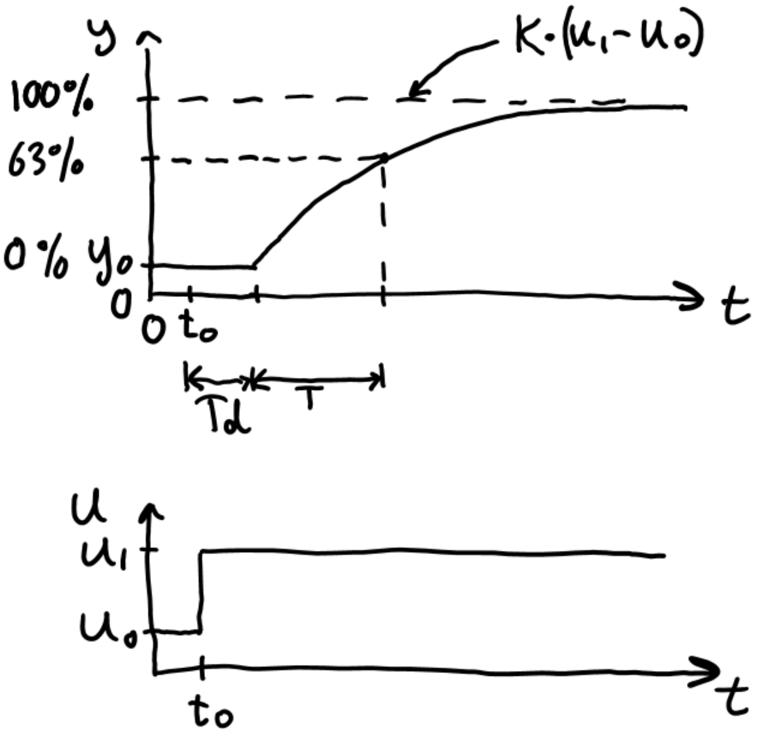


Figure 9.29: Problem 9.3: Time constant dynamics. (In the figure, T is t_c.)

Solution to Problem 9.4

See Figure 9.30.

Solution to Problem 9.5

We manipulate the transfer function so that the constant term of the denominator is 1:

$$H(s) = \frac{2}{4s+8} = \frac{2/8}{(4/8)\,s+8/8} = \frac{0.25}{0.5s+1} = \frac{K}{t_c s+1} \tag{9.108}$$

Hence,

$$K = 0.25$$

and

$$t_c = 0.5$$

We base the drawing of the step response on the following information:

- The steady state value of the step response:

$$y_s = KU = 0.25 \cdot 6 = 1.5 \tag{9.109}$$

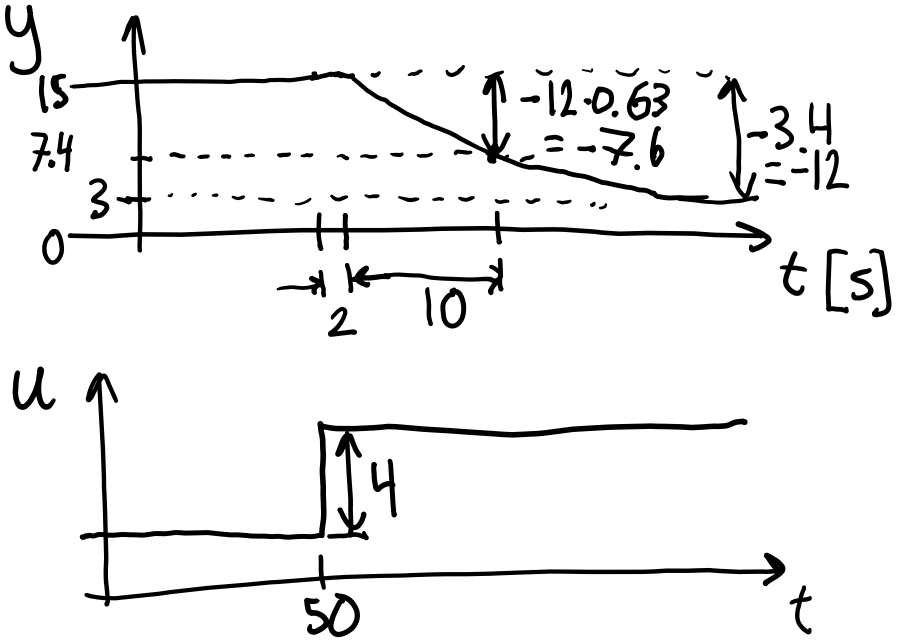


Figure 9.30: Problem 9.4: Time constant dynamics. (In the figure, T is t_c.)

- The time constant:

$$t_c = 0.5 \tag{9.110}$$

  which is the time when the step response has reached value

$$0.63 \cdot y_s = 0.63 \cdot 1.5 = 0.95 \tag{9.111}$$

- The initial slope of the step response:

$$S_0 = y'(0^+) = \frac{KU}{t_c} = \frac{0.25 \cdot 6}{0.5} = 3 \tag{9.112}$$

Figure 9.31 shows the step response.

Solution to Problem 9.6

From Figure 9.24 we see that the gain is

$$K = \frac{\Delta T}{\Delta P} = \frac{30 \text{ K} - 20 \text{ K}}{1 \text{ kW}} = 10 \frac{\text{K}}{\text{kW}} \tag{9.113}$$

and that the time constant (the 63% rise time) is

$$t_c = 50 \text{ min} \tag{9.114}$$

The transfer function becomes

$$\frac{\delta T(s)}{\delta P(s)} = \frac{10}{50s+1} \, \frac{\text{K}}{\text{kW}}$$

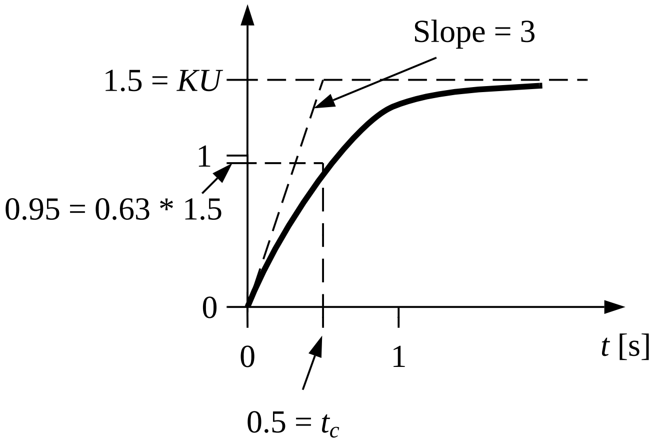


Figure 9.31: Problem 9.5: Step response.

Solution to Problem 9.7

1. We can neglect T_a in (9.95). We also neglect the initial temperature, $T(0)$. Taking the Laplace transform gives

$$sT^*(s) = \left[e^{-t_d s}P^*(s) - GT^*(s)\right]/C$$

where the term $e^{-t_d s}$ represents the time delay. Solving for $T^*(s)$ gives

$$T^*(s) = \frac{1}{Cs+G}e^{-t_d s}P^*(s) = \frac{\frac{1}{G}}{\frac{C}{G}s+1}e^{-t_d s}P^*(s) = \frac{K}{t_c s+1}e^{-t_d s}P^*(s) \tag{9.115}$$

2. The gain is

$$K = \frac{1}{G} = \frac{1}{2.34\ \text{W/K}} = 0.427\ \text{K/W} \tag{9.116}$$

The time constant is

$$t_\text{c} = \frac{C}{G} = \frac{2101\ \text{J/K}}{2.34\ \text{W/K}} = 898\ \text{s} \tag{9.117}$$

Solution to Problem 9.8

1. Laplace transformation of the differential equation (9.96) gives

$$RCsv_\text{out}(s) = v_\text{in}(s) - v_\text{out}(s) \tag{9.118}$$

Solving for $v_\text{out}(s)$ gives

$$v_\text{out}(s) = \frac{1}{RCs+1}v_\text{in}(s) \tag{9.119}$$

The transfer function is

$$H(s) = \frac{1}{RCs+1} = \frac{K}{t_c s + 1} \tag{9.120}$$

The gain is

$$K = 1 \tag{9.121}$$

The time constant is

$$t_c = RC \tag{9.122}$$

2. The filtering is stronger (time constant larger) if R is increased.

Solution to Problem 9.9

1. From Figure 9.28 (for simplicity dropping the Laplace variable as argument):

$$Y(s) = H_p(s)H_c(s) \cdot [R(s) - Y(s)] \tag{9.123}$$

Solving for $Y(s)$:

$$Y(s) = \frac{H_p(s)H_c(s)}{1 + H_p(s)H_c(s)} R(s) = H_t(s)R(s) \tag{9.124}$$

So,

$$H_t(s) = \frac{H_p(s)H_c(s)}{1 + H_p(s)H_c(s)} = \frac{K_c \cdot \frac{1}{(s+1)s}}{1 + K_c \cdot \frac{1}{(s+1)s}} = \frac{K_c}{s^2 + s + K_c} \tag{9.125}$$

2. (9.125) is on the standard form (9.58):

$$H_t(s) = \frac{K_c}{s^2 + s + K_c} = \frac{K{\omega_0}^2}{s^2 + 2\zeta\omega_0 s + {\omega_0}^2} \tag{9.126}$$

Equating coefficients in (9.126) gives

$$K = 1 \tag{9.127}$$

$$\omega_0 = \sqrt{K_c} \tag{9.128}$$

$$\zeta = \frac{1}{2\sqrt{K_c}} \tag{9.129}$$

3. From (9.129):

$$K_c = \frac{1}{4\zeta^2} = \frac{1}{4 \cdot 0.6^2} = 0.69 \tag{9.130}$$

4. The 63 % response time is given by (9.80):

$$t_r \approx \frac{1.5}{\omega_0} = \frac{1.5}{\sqrt{K_c}} = \frac{1.5}{\sqrt{0.69}} = 1.8 \text{ s} \tag{9.131}$$

5. The following Python program implements a simulator of the control system using the forced_response() function of the Python Control package. Figure 9.32 shows the simulated step response. From the step response we can read off the 63 % response time as approximately 1.9 s, which is in good accordance with the estimated value (9.131).

http://techteach.no/control/python/sim_control_dcmotor_2ord_forced_resp_py_con.py

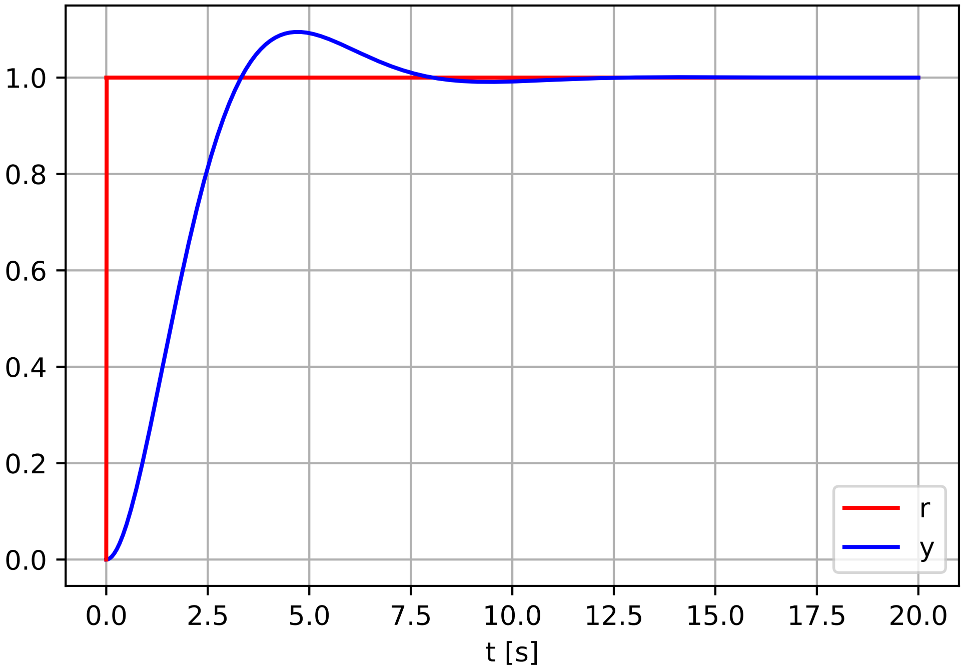


Figure 9.32: Problem 9.9: Simulated step responses of the control system.

Solution to Problem 9.10

Given: $L = 0.5$ m, $A = 0.01$ m^2, $F = 0.001$ m^3/s. The the time delay becomes

$$t_d = \frac{AL}{F} = \frac{0.01 \text{ m}^2 \cdot 0.5 \text{ m}}{0.001 \text{ m}^3/\text{s}} = 5 \text{ s} \tag{9.132}$$

Solution to Problem 9.11

The approximate response time is

$$t_r = 0.5 + 1 + 2 = 3.5 \text{ s} \tag{9.133}$$

Solution to Problem 9.12

Python program 9.1 implements the simulator.

http://techteach.no/control/python/sim_pade_forced_resp_python_control.py

Listing 9.1: sim_pade_forced_resp_python_control.py

```
# %% Imports:

import numpy as np
import control
import matplotlib.pyplot as plt

# %% Generating transfer function:

# Transfer function without time delay:
s = control.tf(’s’)

K = 2
T = 4
H_without_delay = K/(T*s + 1)

# Transfer function of Pade approx:
T_delay = 1
n_pade = 10
(num_pade, den_pade) = control.pade(T_delay, n_pade)
H_pade = control.tf(num_pade, den_pade)

# Transfer function with time delay:
H_with_delay = H_without_delay * H_pade

# %% Simulation setup:

t0 = 0
t1 = 20
dt = 0.01
nt = int(t1/dt) + 1 # Number of points of sim time
t = np.linspace(t0, t1, nt)

# %% Defining input signal:

U = 2
u = U * np.ones(nt)

# %% Simulation:

(t, y) = control.forced_response(H_with_delay, t, u)

# %% Plotting:

plt.close(’all’)
plt.figure(1)

plt.subplot(2, 1, 1)
plt.plot(t, y, color=’blue’, label=’y’)
plt.legend()
plt.xlabel(’t [s]’)
plt.grid()
plt.subplot(2, 1, 2)
plt.plot(t, u, color='green', label='u')
plt.plot([0, dt], [0, U], 'green')  # Plotting vertical line of step
plt.legend()
plt.xlabel('t [s]')
plt.grid()

plt.show
plt.savefig('sim_pade.pdf')
```

Figure 9.33 shows the simulation.

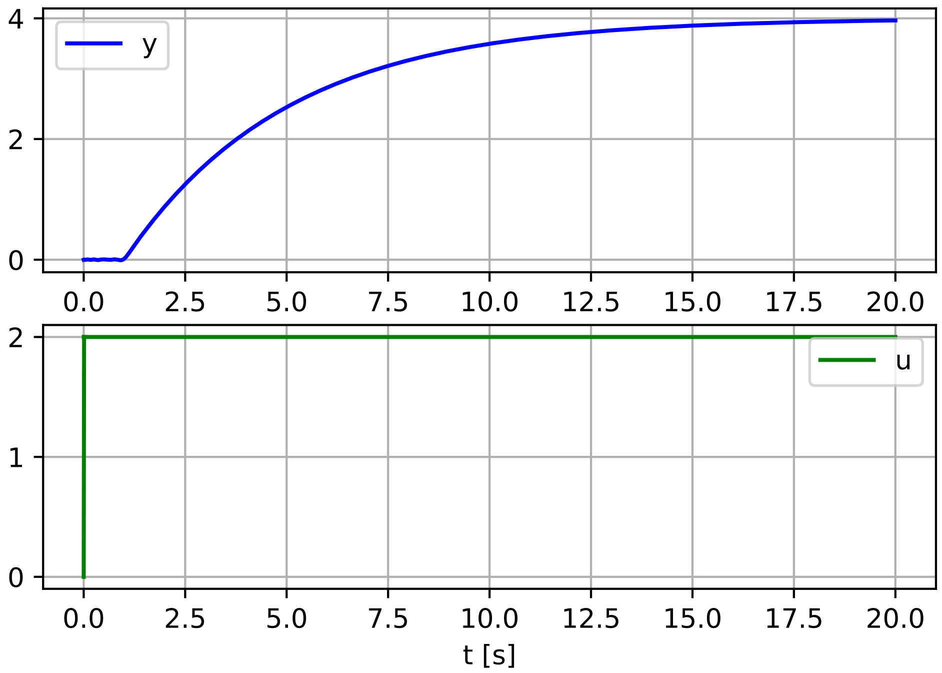


Figure 9.33: Problem 9.12: Simulation of a "time constant with time delay" ssystem.

Chapter 10

Adaptation of models to data

10.1 Introduction

Assume that you have a mathematical model of a system with parameters that characterizes the model. The model can be static or dynamic:

- **Static models** are models not containing time-derivatives. Thus, differential equations are not static models. One example of a static model is

  $$y = a_1 x_1 + a_2 x_2 \tag{10.1}$$

  where y is the output and x_j are inputs, and a_i are parameters to be estimated from known data of output and inputs.

- **Dynamic models** may be in the form of:
  - Differential equations (continuous-time models), linear or nonlinear, possibly in the form of a state space model
  - Difference equations (discrete-time models), linear or nonlinear, possibly in the form of a state space model
  - Transfer functions:
    - Laplace-transform based transfer functions (i.e. continuous-time transfer functions)
    - Z-transform based transfer functions (i.e. discrete-time transfer functions)

Model adaptation is calculation of values of model parameters so that the model behaves in as good accordance as possible to given observations or measurements from the real system. "as good as possible" indicates that the model adaptation problem can be solved with optimization! Indeed, we will do that in Section 10.2.

But why do we want a model when we already have data? Good question! And here are two good answers:

- **Prediction or simulation**: We can use the model for prediction, ie to predict (predict) or simulate what may happen in the future, which we have no observations from yet.
- **Analysis**: We can use the model to analyse the data we already have, e.g. calculate increments for a trend that is in the data.

10.2 Model adaptation as an optimization problem

10.2.1 How to find the best model

We assume given a time series of observations or measurements of the system. The observations may consist of a time series of the input variable or signal and a time series of the output variable or signal, see Figure 10.1. In some cases we have no input excitation signal; only “data” or measurements, e.g. statistical data.

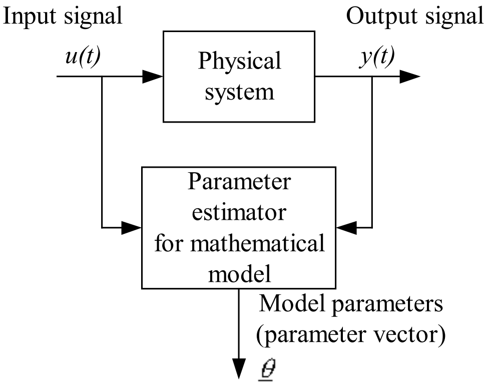

Input signal
u(t)
Physical system
Output signal
y(t)
Parameter estimator for mathematical model
Model parameters (parameter vector)
$\underline{\theta}$

Figure 10.1: Estimation of parameters of a mathematical model from time-series of the observations which may consist of the input variable or signal (u in the figure) and the output variable or signal (y).

The formulation of the parameter estimation problem as an optimization problem is typically as follows:

- The objective function to be minimized is the sum of squared prediction errors – also denoted a least squares criterion:

$$f = \text{SSPE} = \sum_{k=1}^{N} e(k)^2 \tag{10.2}$$

  where $e(i)$ is the prediction error which is the difference between the observations (measurements) and the model-based predicted or calculated observations:

$$e(i) = y_{\text{obs}}(i) - y_{\text{pred}}(i) \tag{10.3}$$

Figure 10.2 illustrates the prediction error. It is assumed that a linear model is to be adapted to the observerations. y_{pred} is calculated in *simulations*, using the model. Therefore, y_{pred} is a function of the parameters to be estimated, and SSPE is a also a function of the parameters.

- The parameters to be estimated are used as optimization variables. All the parameters may be collected in a parameter vector:

$$P = [p(1),\, p(2), ...,\, p(r)]^T \tag{10.4}$$

- In each iteration, the optimizer runs a simulation with parameter values that are adjusted based on previous iterations (simulations). The iterations stops when when the parameter values that minimizes the SSPE, are found. *These "best" values are then used as the parameter estimates.*

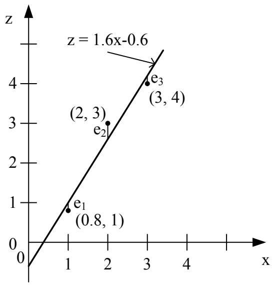


Figure 10.2: Illustration of the prediction error, $e(i)$. A linear model is assumed.

Mathematical formulation of the parameter estimation problem

Mathematically, the optimization problem can be stated as:

$$\min_{P=[p(1),p(2),...,p(r)]} \text{SSPE} \tag{10.5}$$

s.t. (subject to) the given mathematical model.

Figure 10.3 illustrates the principle of optimization-based parameter estimation.

Any nonlinear optimizer can be used to implement the parameter estimation, e.g. the slsqp optimizer in the Scipy package of Python or the fmincon optimizer in the Optimization toolbox of Matlab.

Selecting the best model among several model candidates

In some applications there are *several* candidates of models, for example, a first order differential equation and a second order differential equation that are both assumed to represent the real system. You can select the best model as the one that minimizes the SSPE index (10.2) *calculated from a validation data set.* The validation data set may be a part, e.g. a half of the original data set, while the data set used for parameter estimation is the other half of the original data set. This is illustrated in Figure 10.4.

Note that if you have only *one* (fixed) mathematical model that you want to adapt to the given data, there is no need to validate the model with any validation data set. In that

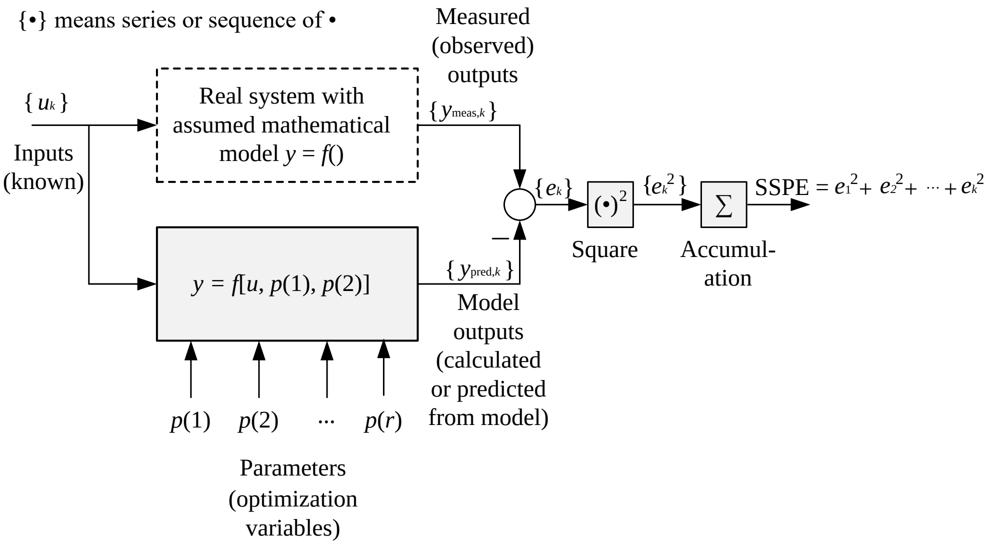

{•} means series or sequence of •
{ uk }
Inputs (known)
Real system with assumed mathematical model y = f()
Measured (observed) outputs
{ ymeas,k }
{ek}
(•)²
{ek²}
Σ
SSPE = e1² + e2² + ··· + ek²
Square
Accumul-ation
–
{ ypred,k }
y = f[u, p(1), p(2)]
Model outputs (calculated or predicted from model)
p(1)
p(2)
···
p(r)
Parameters (optimization variables)

Figure 10.3: The principle of parameter estimation using optimization. The ultimate (estimated) parameter values are those that minimize SSPE. (SSPE = sum of squared prediction errors.)

case, you can use the whole data set for estimation. An example: If you assume that the following transfer function model:

$$H(s) = \frac{Y(s)}{U(s)} = \frac{K}{Ts+1} \tag{10.6}$$

is appropriate to represent the dynamics of a given system, then you can use the whole data for estimation of parameters K and T. In other words, model validation is not necessary. But if you have *two* model candidates, say model (10.6) *and* the following second order transfer function:

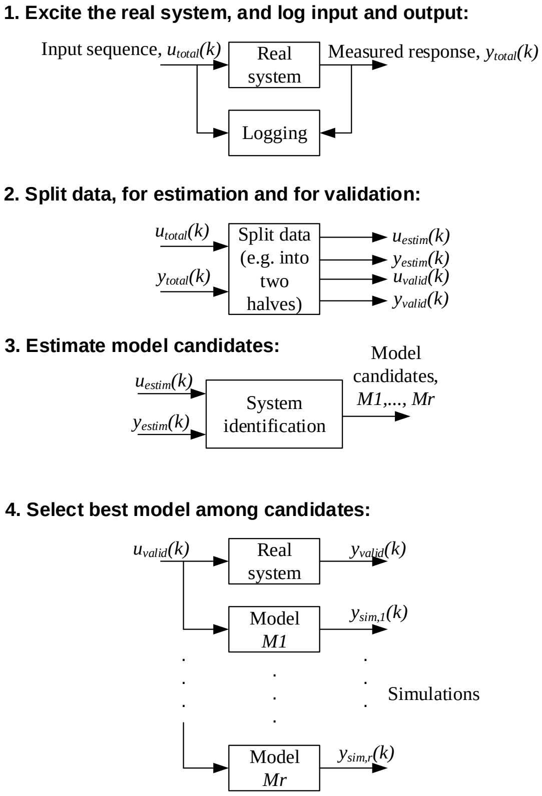
$$H_2(s) = \frac{Y(s)}{U(s)} = \frac{K}{(T_1 s+1)(T_2 s+1)} \tag{10.7}$$

then you must use a validation data set to select among the two model candidates.

10.2.2 Good excitation is necessary!

Assume, as an example, that you want to estimate the time constant T of a first order transfer function. You can not estimate T, which is related to the dynamics of the system, if y and u have constant values all the time. Thus, it is necessary that the excitation signal, $u(t)$, has sufficiently rich variation to give the LS-method enough information about the dynamics of the system to produce an accurate estimate of T.

The up-down-up signal is a good excition signal, see Figure 10.5. This signal is simple to generate manually during the experiment. This signal gives in many cases enough excitation for the estimator to calculate accurate parameter estimates, but the period of the signal shift must be large enough to give the system output a chance to approximately stabilize between the steps.



1. Excite the real system, and log input and output:
Input sequence, $u_{total}(k)$
Real system
Measured response, $y_{total}(k)$
Logging
2. Split data, for estimation and for validation:
$u_{total}(k)$
$y_{total}(k)$
Split data (e.g. into two halves)
$u_{estim}(k)$
$y_{estim}(k)$
$u_{valid}(k)$
$y_{valid}(k)$
3. Estimate model candidates:
$u_{estim}(k)$
$y_{estim}(k)$
System identification
Model candidates, $M1,..., Mr$
4. Select best model among candidates:
$u_{valid}(k)$
Real system
$y_{valid}(k)$
Model $M1$
$y_{sim,1}(k)$
Simulations
Model Mr
$y_{sim,r}(k)$

Figure 10.4: Procedure for selecting the best among several model candidates.

10.3 Adaptation of static models to data

10.3.1 Adaptation of static models using brute force optimization

10.3.1.1 Introduction

If the number of model parameters to be estimated is not so large, e.g. 5 or less, the straightforward brute force method (also called the grid method) of optimization can be used to find the estimates (the best values). As optimization criterion, we can use minimum of the squared sum of prediction error (sspe), cf. Section 10.5.

The following sections demonstrates how to adapt a static model to given data using the brute force method:

- Section 10.3.1.2: Adaptation of a model to global temperature data using the brute force method implemented from scratch with elementary code including nested for loops.
- Section 10.3.1.4: The same application as in Section 10.3.1.2, but using Python's scipy.optimize.brute() function which implements the brute force method. The brute() function has a very useful option to obtain an ultimate accurate optimal estimate with the simplex optimization method started from the brute force optimal solution.

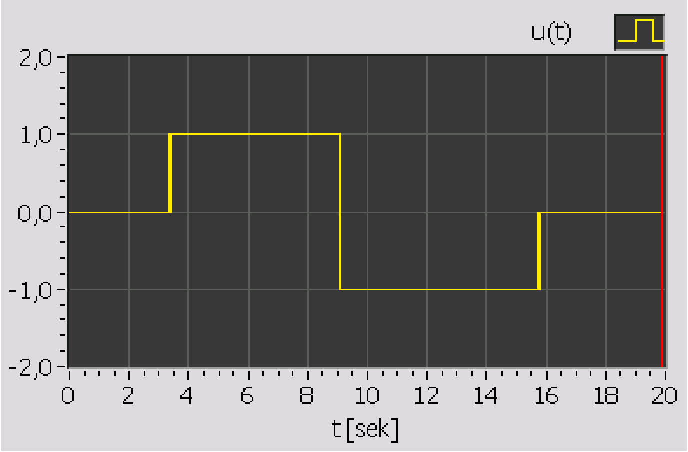


Figure 10.5: A good excition signal: Up-down-up signal.

Note that the brute force method is a global optimization method, which means it has the potential of finding the global minimum. While iterative optimization methods, e.g. the Nelder-Mead method or the Newton-Raphson method, are local optimization methods, which means they may only find – or get stuck at – a local optimum – not a global optimum.

10.3.1.2 Adaptation of static models using brute force optimization in elementary code

Example 10.1 *Adaptation of a static model to global temperature data using the brute force method*

NASA publishes the change (dT) of the Earth's annual average temperature (T) relative to a fixed reference temperature, which is the average temperature (T_r) over the period 1951-1980. In other words:

$$dT = T - T_r \tag{10.8}$$

T_r is approx. 14 °C. The actual value of T_r is not so important, but the change from this is an important environmental indicator. The value of T is based on more than 6000 measurements from weather stations around the world and from other observations.

Figure 10.6 shows a plot of dT for the period t = [1980–2022] years. The plot is generated with Program 10.1, which also contains the data:

http://techteach.no/control/python/global_temp_change.py

Listing 10.1: global_temp_change.py

```
# %% Import of packages

import numpy as np
import matplotlib.pyplot as plt

# %% Data:
```

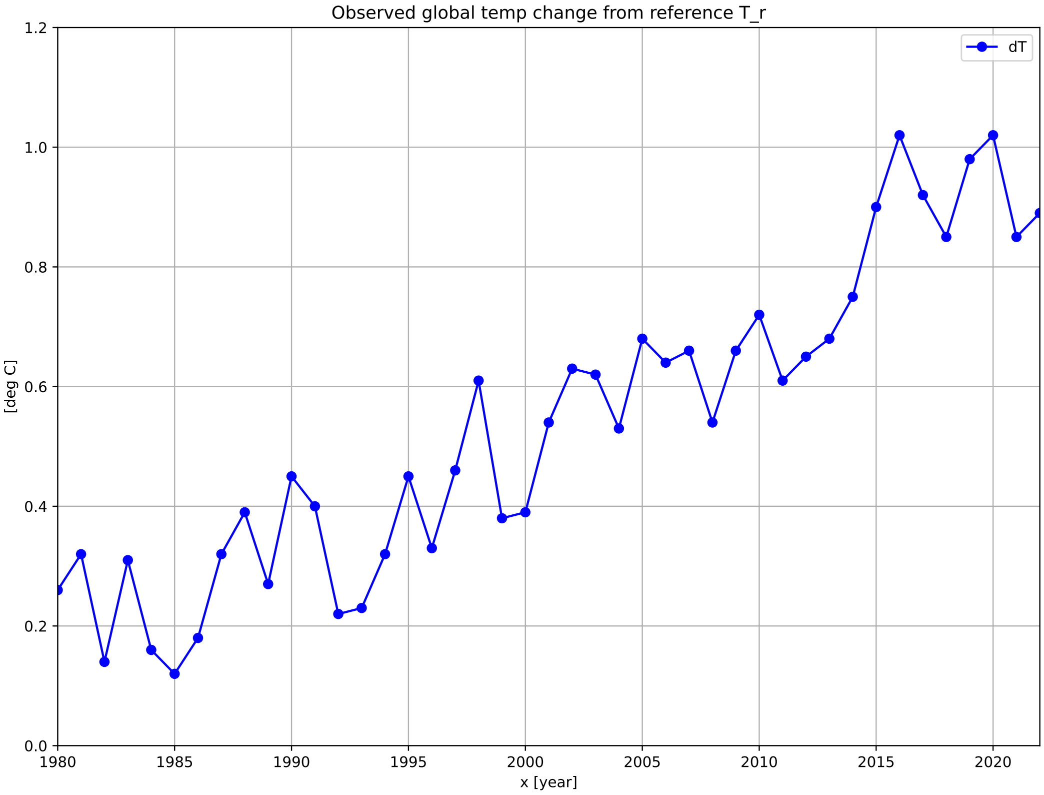


Figure 10.6: The change (dT) of the Earth's annual average temperature (T) in the period 1980-2022 relative to a fixed reference temperature. Source: NASA (https://climate.nasa.gov/vital-signs/global-temperature/).

```
x_array = np.arange(1980, 2023)
y_obs_array = np.array([0.26,0.32,0.14,0.31,0.16,0.12,
                        0.18,0.32,0.39,0.27,0.45,0.40,
                        0.22,0.23,0.32,0.45,0.33,0.46,
                        0.61,0.38,0.39,0.54,0.63,0.62,
                        0.53,0.68,0.64,0.66,0.54,0.66,
                        0.72,0.61,0.65,0.68,0.75,0.90,
                        1.02,0.92,0.85,0.98,1.02,0.85,0.89])

# %% Plotting:

plt.close('all')

plt.figure(1)

plt.plot(x_array, y_obs_array,'b-o', label='dT = y')
plt.legend()
plt.xlim(1980, 2022)
plt.ylim(0, 1.2)
plt.title('Observed global temp change from reference T_r')
plt.xlabel('t = x [year]')
plt.ylabel('[deg C]')
plt.grid()

plt.savefig('plot_global_temp_change.pdf')
plt.show()
```

We will fit a straight line to the data:

$$dT = at + b \tag{10.9}$$

where a and b are parameters to be estimated. Let's use the standard symbols $y = dT$ and $x = t$ in following. So, the model is

$$y = ax + b \tag{10.10}$$

As objective function we select the minimum SSPE, or least squares, function with a and b as optimization variables:

$$\min_{a,\, b} f(a,\, b) \tag{10.11}$$

where

$$f = \text{SSPE} = \sum_{k=0}^{N-1} (e_k)^2 \tag{10.12}$$

$N = 43$ is the number of observations. e_k is the prediction error:

$$e_k = y_{\text{obs,k}} - y_{\text{pred,k}} \tag{10.13}$$

$y_{\text{obs,k}}$ is given in Program 10.1. $y_{\text{pred,k}}$ is predicted from the model (10.10). Thus,

$$y_{\text{pred}} = ax + b \tag{10.14}$$

In detail, the objective function is:

$$\begin{aligned} f(a,\, b) &= \sum_{k=0}^{N-1} [e_k]^2 && (10.15) \\ &= \sum_{k=0}^{N-1} [y_{\text{obs,k}} - y_{\text{pred,k}}]^2 && (10.16) \\ &= \sum_{k=0}^{N-1} \{y_{\text{obs,k}} - [ax_k + b]\}^2 && (10.17) \end{aligned}$$

where $y_{\text{obs,k}} = dT_k$ og $x_k = t_k$ are given in Program 10.1.

In the brute force method, you must define ranges of candidate values of the optimization variables. The optimal solution is found among these candidate values. In the program, I have used the following ranges of candidate value of a and b:

- *Candidate value range of* a: Since a is the slope of the linear function (10.14), we can estimate a value of a as the slope between two appropriate data points in Program 10.1. I select
$$a \approx \frac{0.89 - 0.26}{2022 - 1980} = 0.015 \tag{10.18}$$
Now, the candidate value range of a may be set as
$$0 \le a \le 0.03 \tag{10.19}$$

- *Candidate value range of* b: Let us take e.g. the data point (1980, 0.26): By setting $y = 0.26$ and $x = 1980$ in (10.10) with $a = 0.015$ from (10.18), we get an estimate of b as
$$b = y - ax = 0.26 - 0.015 \cdot 1980 = -29.4 \tag{10.20}$$
Now, the value range of b may be set as
$$-50 \le b \le 0 \tag{10.21}$$

Since we have only two optimization variables (a and b), we can allow a fairly high resolution of the optimization variables. I set $N_{\text{resolution}} = 1000$ for both a and b, which gives 1,000,000 calculations.

Program 10.2 implements the model adaptation, and plotting both $y_{\text{obs}}$and y_{pred}. On my computer it takes approx. 6 sec to run the program.

The results are as follows.

```
a_est = 1.8979e-02
b_est = -3.7437e+01
f_min = 3.7409e-01
Elapsed time = 6.27e+00
```

Figure 10.7 shows plots of $y_{\text{obs}}$and y_{pred}.

When presenting the results, I have used several digits for a (and the same with b) since x (year) has several digits, otherwise we can get an inaccurate prediction of y using (10.14). To reduced such numeric inaccuracies, we can normalize the data before the model adaptation, cf. Section 10.3.1.3.

How accurate are the estimates of a and b found above? a and b can be estimated accurately using the (ordinary) least squares method, cf. Example 10.5. The results with that method are as follows.

```
a_opt = 1.8938e-02
b_opt = -3.7360e+01
```

So, we have found quite an accurate result with the brute force method!

http://techteach.no/control/python/model_adapt_temp_bf.py

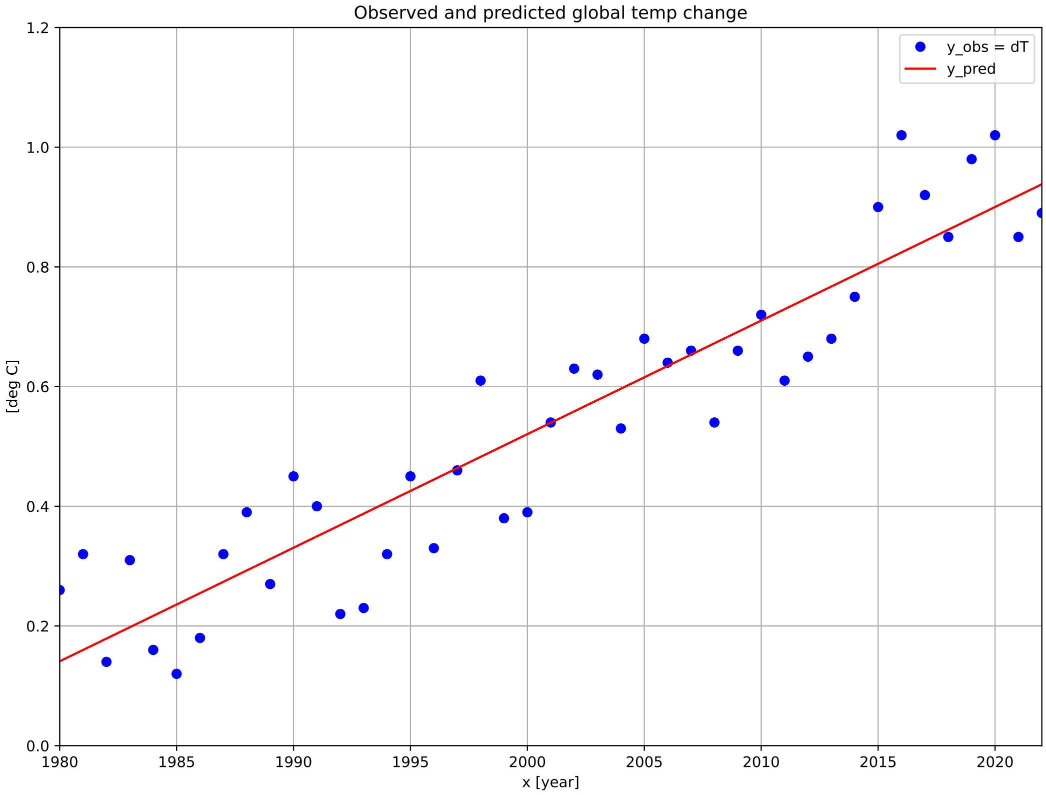


Figure 10.7: $y_{\text{obs}}$and y_{pred}.

Listing 10.2: model_adapt_temp_bf.py

```
"""
Model adaptation to global temp change data using brute force method
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""

#%% Import of packages

import numpy as np
import matplotlib.pyplot as plt
import time

#%% Definition of objective function

def fun_obj(params):
    a = params[0]
    b = params[1]
    y_pred_array = a*x_array + b
    e_array = y_obs_array - y_pred_array
    f_obj = sum(e_array*e_array)
    return f_obj

#%% Data

x_array = np.arange(1980, 2023)
y_obs_array = np.array([0.26,0.32,0.14,0.31,0.16,0.12,
                        0.18,0.32,0.39,0.27,0.45,0.40,
                        0.22,0.23,0.32,0.45,0.33,0.46,
                        0.61,0.38,0.39,0.54,0.63,0.62,
                        0.53,0.68,0.64,0.66,0.54,0.66,
                        0.72,0.61,0.65,0.68,0.75,0.90,
                        1.02,0.92,0.85,0.98,1.02,0.85,0.89])

#%% Initialization

N_resol_param = 1000

a_lb = 0.0
a_ub = 0.03
a_array = np.linspace(a_lb, a_ub, N_resol_param)

b_lb = -50
b_ub = 0
b_array = np.linspace(b_lb, b_ub, N_resol_param)

f_min = np.inf

#%% Starting timer to get execution time

tic = time.time()

#%% Brute force optimization

for a in a_array:

    for b in b_array:

        # Calc of objective function:
        params = np.array([a, b])
        f = fun_obj(params)

        # Improvement of solution:
        if (f < f_min):
            f_min = f
            a_est = a
            b_est = b

#%% Stopping timer

toc = time.time()
t_elapsed = toc-tic

#%% Presentation of result
```

```
print(f'a_est = {a_est:.4e}')
print(f'b_est = {b_est:.4e}')
print(f'f_min = {f_min:.4e}')
print(f'Elapsed time = {t_elapsed:.2e}')

#%% Prediction with adapted model

y_pred = a_est*x_array + b_est

#%% Plotting

plt.close('all')
plt.figure(1)

plt.plot(x_array, y_obs_array,'bo', label='y_obs = dT')
plt.plot(x_array, y_pred,'r-', label='y_pred')
plt.legend()
plt.xlim(1980, 2022)
plt.ylim(0, 1.2)
plt.title('Observed and predicted global temp change')
plt.xlabel('x [year]')
plt.ylabel('[deg C]')
plt.grid()

plt.savefig('temp_model_adapt_bf_N_1000.pdf')
# plt.savefig('temp_model_adapt_bf_N_10.pdf')
plt.show()
```

What is bad with selecting a small N?

Above, we used $N = 1000$ candidate values for both a and b. What if we try a smaller N? Obviously, that would make out program complete in a shorter time, which sounds good. Let's try $N = 10$. The results are then:

```
a_opt = 1.6667e-02
b_opt = -3.3333e+01
```

which differs quite much from the values found with $N = 1000$.

So, it is important to select a sufficiently large N to avoid inaccurate estimates. How large? Find it with trial-and-error! You can try different values of N, and select one for which the estimates are almost insensitive to N, i.e., the estimates do not change except for, say, the 3rd decimal digit.

[End of Example 10.1]

10.3.1.3 Normalization of data before model adaptation

To avoid inaccurate estimate values due to rounding errors, we can normalize the data before we carry out the model adaptation. Usually, it is only the x-values (not the y-values) that are normalized. (Within machine learning theory, this is called “feature scaling”.)

There are alternative ways to normalize. Here are two common ways:

- *Standardization*:
$$x_{\text{norm}} = \frac{x - m_x}{\sigma_x} \tag{10.22}$$
where $m_x$is the mean value and σ_x is the standard deviation of the x-values.
- *Min-max normalization*:
$$x_{\text{norm}} = \frac{x - m_x}{x_{\text{max}} - x_{\text{min}}} \tag{10.23}$$

After the normalization, the data series x_{norm} will, roughly said, have values around 0 and with a numerical value not too far from 1. Then you use x_{norm} when you carry out the parameter estimation. The estimated parameters, as a and b in our example, may get values that are completely different from the estimated values obtained without normalization of the x-values.

Note: When predictions are to be calculated with the adapted model found using normalized data, you must use x_{norm} for x in the model.

Example 10.2 *Normalization before model adaptation*

This example is similar to Example 10.1, but now with normalized x-values from standardization.

Program 10.3 implements the model adaptation.

The results of Program 10.3 are:

```
a_opt = 2.3524e-01
b_opt = 5.3604e-0
```

A plot of y_{pred} and y_{obs} (with normalization) will be the same as the plot shown in Figure 10.7 (without normalization), so I do not show a new plot here.

http://techteach.no/control/python/model_adapt_temp_norm_bf.py

Listing 10.3: model_adapt_temp_norm_bf.py

```
"""
Model adaptation to global temp change data using brute force method
with x-values normalized with standardization.
Finn Aakre Haugen, TechTeach. finn@techteach.no
```

```
2023 12 10
"""

#%% Import of packages

import numpy as np
import matplotlib.pyplot as plt
import time

#%% Definition of objective function

def fun_obj(params):
    a = params[0]
    b = params[1]
    y_pred_array = a*x_norm_array + b
    e_array = y_obs_array - y_pred_array
    f = sum(e_array*e_array)
    return f

#%% Data

x_array = np.arange(1980, 2023)

y_obs_array = np.array([0.26,0.32,0.14,0.31,0.16,0.12,
                        0.18,0.32,0.39,0.27,0.45,0.40,
                        0.22,0.23,0.32,0.45,0.33,0.46,
                        0.61,0.38,0.39,0.54,0.63,0.62,
                        0.53,0.68,0.64,0.66,0.54,0.66,
                        0.72,0.61,0.65,0.68,0.75,0.90,
                        1.02,0.92,0.85,0.98,1.02,0.85,0.89])

#%% Normalization (standardization)

x_norm_array = (x_array - np.mean(x_array))/np.std(x_array)

#%% Initialization

N_resol_param = 1000

a_lb = 0.0
a_ub = 0.5
a_array = np.linspace(a_lb, a_ub, N_resol_param)

b_lb = 0
b_ub = 1.5
b_array = np.linspace(b_lb, b_ub, N_resol_param)

f_min = np.inf

#%% Starting timer to get execution time

tic = time.time()

#%% Brute force optimization
```

```
for a in a_array:

    for b in b_array:

        # Calc of objective function:
        params = np.array([a, b])
        f = fun_obj(params)

        # Improvement of solution:
        if (f < f_min):
            f_min = f
            a_est = a
            b_est = b

#%% Stopping timer

toc = time.time()
t_elapsed = toc-tic

#%% Presentation of result

print(f’a_est = {a_est:.4e}’)
print(f’b_est = {b_est:.4e}’)
print(f’f_min = {f_min:.4e}’)
print(f’Elapsed time = {t_elapsed:.2e}’)

#%% Prediction with adapted model

y_pred = a_est*x_norm_array + b_est

#%% Plotting

plt.close(’all’)
plt.figure(1, figsize=(12, 9))

plt.plot(x_array, y_obs_array,’bo’, label=’y_obs = dT’)
plt.plot(x_array, y_pred,’r-’, label=’y_pred’)
plt.legend()
plt.xlim(1980, 2022)
plt.ylim(0, 1.2)
plt.title(’Observed and predicted global temp change’)
plt.xlabel(’x [year]’)
plt.ylabel(’[deg C]’)
plt.grid()

# plt.savefig(’temp_model_adapt_norm_bf.pdf’)
plt.show()
```

Comments to program 10.3:

- x_{norm} is used in the object function and when calculating y_{pred}.

- x_{norm} has values between x_norm_array[0] = –1.692 and x_norm_array[-1] = 1.692. For comparison, x (not normalized) has values between x_array[0] = 1980 and x_array[-1] = 2022. x_{norm} thus has values around 0 and absolute values not very different from 1.
- a and b have completely different values than when when estimated without normalization of x.
- The ranges of candidate values of a and b must be chosen different from the case without normalization, but I do not show these ranges here.
- The object function, f, has value 3.7357e-01, which is very similar to the case without normalization, 3.7409e-01.

[End of Example 10.2]

10.3.1.4 Adaptation of static models using Python's brute() function

As alternative to implementing the brute force method from scratch, Python has a built-in optimization function based on the brute force method, namely scipy.optimize.brute(). You may be able to save some programming time using this feature compared to programming the brute force method from scratch, but probably not much. And the execution time is almost the same.

However, scipy.optimize.brute() has one strong feature which is offered as an option: It can invoke an accurate iterative optimization method, namely a Nelder-Mead algorithm (also denoted downhill simplex algorithm), following after the brute force optimization, to find an accurate optimal solution (i.e. a minimum). The starting point of this Nelder-Mead algorithm is the optimal solution that brute() function found from its built-in brute force method. So, this option gives a final improvement of the optimal solution. Figure 10.8 illustrates this.

Example 10.3 *Adaptation of a static model to data using Python's brute() function*

This example is based on the data and model used in Example 10.1.

In the present example, I will use Python's scipy.optimize.brute() function which implements the brute force method, cf. Section 10.3.1.1. The standard use of brute() gives exactly the same result as a brute force method implemented from scratch, so using brute() in the standard way, has no substantial benefit to implementing brute force yourself, as in Example 10.1. Also, the execution time with brute() takes almost the same time as our own the brute force method. But the brute() function has a great feature: It has an option to obtain an accurate ultimate optimal estimate with the iterative Nelder-Mead optimization algorithm started from the brute force optimal solution. These two options – without and with final optimization – can be selected as follows:

- *Standard use of brute()*: With input argument finish=None to the brute() function, plain brute force optimization is carried out.

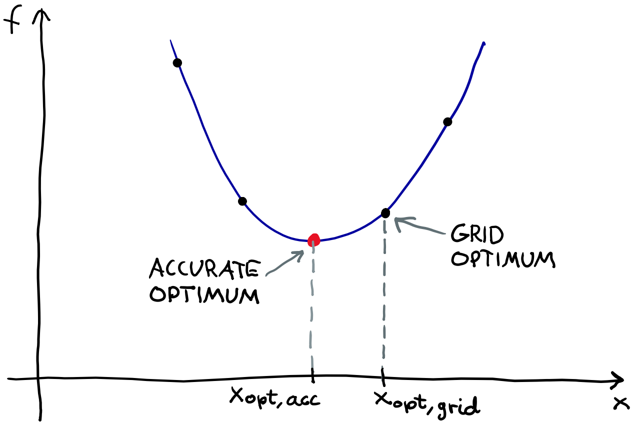


Figure 10.8: The starting point of the Nelder-Meads algorithm is the optimal solution that brute() function found from its built-in brute force method.

- *brute() with ultimate optimazation*: With input argument finish=optimize.fmin to the brute() function, the Nelder-Mead algorithm is run with the brute force solution as starting point.

I have tried both options in this example.

Program 10.4 implements the brute force method using the brute() function, with $N = 10$.

The results are as follows.

```
a_opt = 1.894e-02
b_opt = -3.736e+01
f_min = 3.736e-01
Elapsed time = 2.090e-03
```

The results are, to the pertinent precision, equal to our own brute force method of Example 10.1 with $N = 1000$, but are now obtained much faster, namely 2.090e-3 sec vs. 6.27 sec because of the much smaller N ($= 10$). And the results are accurate. They are equal to values as found with the Least Squares method – with the selected number of digits, cf. Example 10.5.

http://techteach.no/control/python/model_adapt_temp_scipy_brute.py

Listing 10.4: model_adapt_temp_scipy_brute.py

```
"""
Model adaptation to global temperature change data using Scipy brute()
```

```
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""

# %% Import of packages

import numpy as np
from scipy import optimize
import time

# %% Definition of objective function

def fun_obj(params):
    a = params[0]
    b = params[1]
    y_pred_array = a*x_array + b
    e_array = y_obs_array - y_pred_array
    sspe = sum(e_array*e_array)
    return sspe

# %% Data

x_array = np.arange(1980, 2023)

y_obs_array = np.array([0.26,0.32,0.14,0.31,0.16,0.12,
                        0.18,0.32,0.39,0.27,0.45,0.40,
                        0.22,0.23,0.32,0.45,0.33,0.46,
                        0.61,0.38,0.39,0.54,0.63,0.62,
                        0.53,0.68,0.64,0.66,0.54,0.66,
                        0.72,0.61,0.65,0.68,0.75,0.90,
                        1.02,0.92,0.85,0.98,1.02,0.85,0.89])

# %% Initialization

N_resol_param = 10

a_lb = 0.0
a_ub = 0.03
a_step = (a_ub - a_lb)/(N_resol_param - 1)

b_lb = -50
b_ub = 0
b_step = (b_ub - b_lb)/(N_resol_param - 1)

params_ranges = (slice(a_lb, a_ub, a_step),
                 slice(b_lb, b_ub, b_step))

# %% Starting timer to calculate execution time

tic = time.time()

# %% Solving the optim problem with optimize.brute()

# finish_setting = None
```

```
finish_setting = optimize.fmin  # Nelder-Mead optim

result_optim = optimize.brute(fun_obj, params_ranges,
                              full_output=True,
                              finish=finish_setting)
params_optim = result_optim[0]
f_min = result_optim[1]

# %% Optimal parameter values

a_est = params_optim[0]
b_est = params_optim[1]

# %% Stopping timer

toc = time.time()
t_elapsed = toc-tic

# %% Presentation of result

print(f'a_est = {a_est:.3e}')
print(f'b_est = {b_est:.3e}')
print(f'f_min = {f_min:.3e}')
print('Elapsed time =', f'{t_elapsed:.3e}')
```

[End of Example 10.3]

10.3.2 Adaptation of static models using nonlinear programming (NLP)

An nonlinear programming (NLP) solver can find the optimum (minimum) of an objective function which is a nonlinear function of the optimization variables. Of course, it can find optimum also of an objective functions which is a linear function of the optimization variables, since linear functions are just special cases of nonlinear functions.

Note: NLP solvers are iterative solvers, and they may only find – or get stuck at – a local optimum – not a (better) global optimum, depending on wether you were unlucky (initial value 1) or lucky (initial value 1) with the initial (guessed) values of parameter p. This is illustrated in Figure 10.9.

Example 10.4 *Adaptation of a static model to global temperature data using using an NLP solver of Scipy*

See Example 10.1. In that example, the parameters a and b in the following model was estimated from observations with the brute force method of optimization.

$$y = ax + b \tag{10.24}$$

The data are given in Program 10.1.

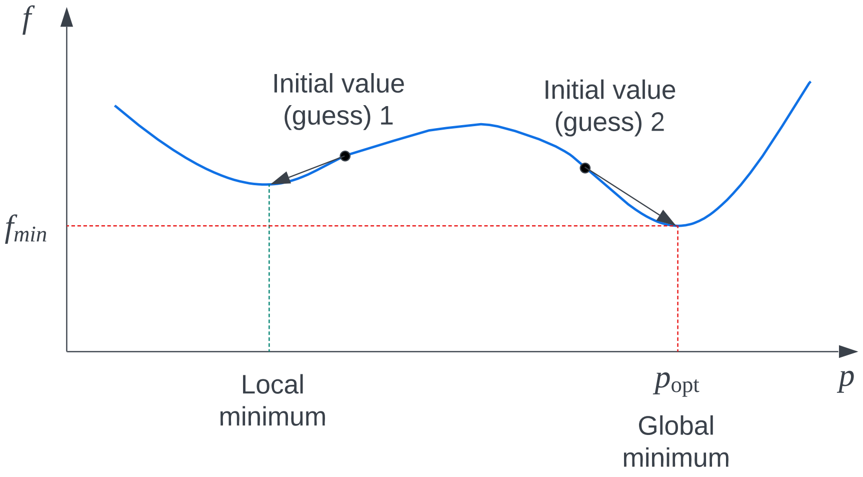


Figure 10.9: Illustration of local optima and global optimum of an objective function. p is the optimization variable (model parameter to be estimated).

Let's see how a and b can be estimated with an NLP solver, namely the scipy.optimize.minimize() function in the Scipy package of Python. Program 10.5 implements the model adaptation. There are several alternative solvers that the minimize() function can use. In program 10.5, I have selected the Nelder-Mead solver (also denoted the downhill simplex method) which finds the minimum after a kind of geometrical search for minimum based on subsequent calculations of objective function values.

As guessed or initial values for search, I selected the same values as derived in Example 10.1, namely:

$$a_{\text{guess}} = 0.015 \tag{10.25}$$

$$b_{\text{guess}} = -29.4 \tag{10.26}$$

Results:

```
a_opt = 1.894e-02
b_opt = -3.736e+01
f_min = 3.736e-01
```

These results are accurate. They are equal to values as found with the Least Squares method – with the selected number of digits, cf. Example 10.5.

http://techteach.no/control/python/model_adapt_temp_scipy_nelder_mead.py

Listing 10.5: model_adapt_temp_scipy_nelder_mead.py

```
"""
Model adaptation to global temp change data using Scipy minimize
with Nelder-Mead solver.
```

```
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""

# %% Import

import numpy as np
import scipy.optimize

# %% Defining functions

def fun_objective(params):
    a = params[0]
    b = params[1]
    y_pred_array = a*x_array + b
    e_array = y_obs_array - y_pred_array
    sspe = sum(e_array*e_array)
    return sspe


# %% Data

x_array = np.arange(1980, 2023)

y_obs_array = np.array([0.26,0.32,0.14,0.31,0.16,0.12,
                        0.18,0.32,0.39,0.27,0.45,0.40,
                        0.22,0.23,0.32,0.45,0.33,0.46,
                        0.61,0.38,0.39,0.54,0.63,0.62,
                        0.53,0.68,0.64,0.66,0.54,0.66,
                        0.72,0.61,0.65,0.68,0.75,0.90,
                        1.02,0.92,0.85,0.98,1.02,0.85,0.89])

# %% Guessed values (initial values) of optim variables

a_guess = -0.188
b_guess = 432
x_guess = np.array([a_guess, b_guess])

# %% Solving the optim problem

res = scipy.optimize.minimize(fun_objective, x_guess,
                              method = ’nelder-mead’)

# %% The results of the optimization

(a_est, b_est) = res.x
f_min = res.fun

# %% Displaying the optimal solution

print(f’a_est = {a_est:.3e}’)
print(f’b_est = {b_est:.3e}’)
print(f’f_min = {f_min:.3e}’)
```

[End of Example 10.4]

10.3.3 Adaptation of static models using ordinary least squares method

10.3.3.1 The standard regression model

Assume given the following model:

$$y = \varphi_1\theta_1 + \cdots + \varphi_n\theta_n \tag{10.27}$$

$$= \underbrace{\begin{bmatrix} \varphi_1 & \cdots & \varphi_n \end{bmatrix}}_{\phi} \underbrace{\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}}_{\theta} \tag{10.28}$$

$$= \phi\theta \tag{10.29}$$

which is called the regression model. In (10.29):

- φ_i is the regression variable (with known value).
- ϕ is the regression vector (with known value).
- y is the observed variable (with known value).
- θ is an unknown parameter vector to be estimated with the LS-method.

Note that the regression model is *linear in the parameter vector* θ. If the model is nonlinear, you can not use the LS method. Instead, you can use some optimization method for nonlinear models, e.g. the brute force method, cf. Ch. 10.3.1, or nonlinear programming, cf. Ch. 10.4.2.

Assume that we have m corresponding values of y and ϕ. Then we can write the following m equations according to the model:

$$y_1 = \varphi_{11}\theta_1 + \cdots + \varphi_{1n}\theta_n = \phi_1\theta$$

$$\vdots$$

$$y_i = \varphi_{i1}\theta_1 + \cdots + \varphi_{in}\theta_n = \phi_i\theta$$

$$\vdots$$

$$y_m = \varphi_{m1}\theta_1 + \cdots + \varphi_{mn}\theta_n = \phi_m\theta$$

These m "stacked" equations can more compactly be written as

$$\underbrace{\begin{bmatrix} \vdots \\ y_i \\ \vdots \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} \vdots \\ \varphi_i \\ \vdots \end{bmatrix}}_{\varPhi} \theta \tag{10.30}$$

or just:

$$Y = \Phi\theta \tag{10.31}$$

where Y and Φ consist of known data and θ is the unknown parameter of which we will calculate or estimate a value θ using the LS-method.[1]

10.3.3.2 The LS problem

We define the *equation-error vector* or *prediction-error vector*[2], E, as the difference between the left side and the right side of (10.31):

$$E = \begin{bmatrix} \vdots \\ e_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ y_i - \varphi_i\theta \\ \vdots \end{bmatrix} = Y - \Phi\theta \tag{10.32}$$

Figure 10.10 illustrates the equation-errors or prediction errors for the case of the model

$$y = \phi\theta \tag{10.33}$$

to be fitted to two data points.

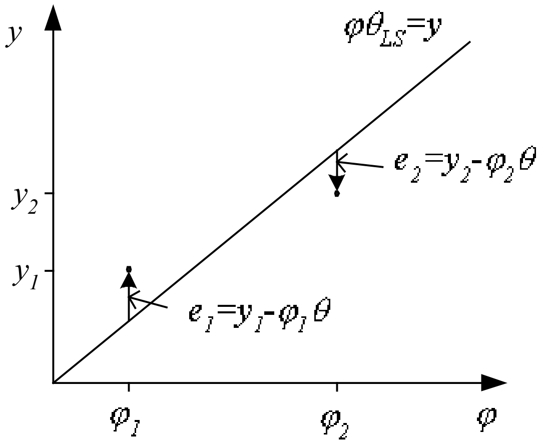


Figure 10.10: Equation-errors or prediction errors e_1 and e_2 for a simple case.

The problem is to calculate a value – an estimate – of the unknown parameter-vector θ so

[1] In matematical literature, (10.31) is more often written on the form $b = Ax$. I have used symbols which are common in the field of system identification.

[2] The name prediction-error vector is because the term $\Phi\theta$ can be regarded as a prediction of the observed (known) "output" y.

that the following quadratic criterion function, $V(\theta)$, is minimized:

$$V(\theta) = {e_1}^2 + {e_2}^2 + \cdots + {e_m}^2 \tag{10.34}$$
$$= E^T E \tag{10.35}$$
$$= (Y - \Phi\theta)^T (Y - \Phi\theta) \tag{10.36}$$
$$= \left(Y^T - \theta^T \Phi^T\right) (Y - \Phi\theta) \tag{10.37}$$
$$= Y^T Y - Y^T \Phi\theta - \theta^T \Phi^T Y + \theta^T \Phi^T \Phi\theta \tag{10.38}$$

In other words, the problem is to estimate θ so that the sum of quadratic prediction errors is minimized.

10.3.3.3 The LS solution

Since $V(\theta)$ is a quadratic function of the unknown parameters θ, the minimum value of $V(\theta)$ can be calculated by setting the derivative of V with respect to θ equal to zero. This is illustrated in Figure 10.11.

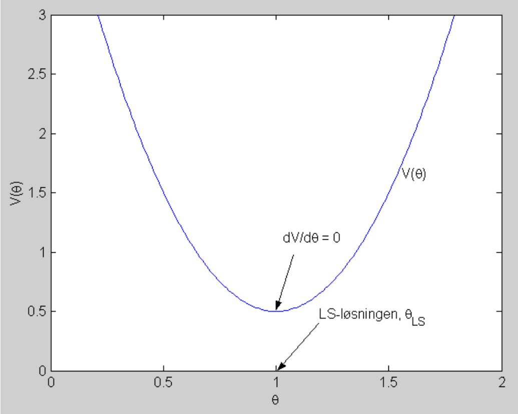


Figure 10.11: The LS solution θ_{est} corresponds to the minimum value of the quadratic function $V(\theta)$, and can be calculated by setting the derivative $V(\theta)/d\theta$ to zero.

Using the differentiation rule (45.5) on (10.38), and then setting the derivative equal to zero, gives

$$\frac{dV(\theta)}{d\theta} = 2\Phi^T \Phi\theta - 2\Phi^T Y \overset{!}{=} 0 \text{ (vector)} \tag{10.39}$$

or

$$\Phi^T \Phi\theta = \Phi^T Y \tag{10.40}$$

(10.40) is called the normal equation. The θ-solution of (10.40) can be found by pre-multiplying (10.40) with $(\Phi^T \Phi)^{-1}$. The result is

$$\theta_{\text{est}} = (\Phi^T \Phi)^{-1} \Phi^T Y \tag{10.41}$$

which is *the LS-solution* of (10.31)[3]. All right side terms in (10.31) are known. We may denote (10.41) as the "batch LS formula".

[3] $(\Phi^T \Phi)^{-1} \Phi^T$ is the so-called pseudo-inverse of Φ.

Note: To apply the LS-method, the model must be written on the regression model form (10.31), which consists of m (10.29) "stacked".

Example 10.5 *Adaptation of a static model to global temperature changes data using the LS method*

In previous examples, the parameters a and b in the following temperature change model was estimated from given data (observations):

$$y = ax + b \tag{10.42}$$

The data are given in Program 10.1.

The model (10.42) is linear in its parameters a and b. We can therefore estimate the parameters a and b using the LS method.

We start by writing the model on the standard regression form:

$$y = ax + b = \begin{bmatrix} x, & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \varphi\theta \tag{10.43}$$

The estimate is given by (10.41), which is repeated here:

$$\theta_{\text{LS}} = (\Phi^T \Phi)^{-1} \Phi^T Y \tag{10.44}$$

In (10.44):

$$Y = \begin{bmatrix} y_0 \\ \vdots \\ y_{42} \end{bmatrix} = \begin{bmatrix} 0.26 \\ \vdots \\ 0.89 \end{bmatrix}$$

$$\varPhi = \begin{bmatrix} \varphi_0 \\ \vdots \\ \varphi_{42} \end{bmatrix} = \begin{bmatrix} x_0, & 1 \\ \vdots & \vdots \\ x_{42}, & 1 \end{bmatrix} = \begin{bmatrix} 1980, & 1 \\ \vdots & \vdots \\ 2022, & 1 \end{bmatrix}$$

Program 10.6 implements the above calculations.

The results are:

```
a_est = 1.894e-02
b_est = -3.736e+01
f_min = 3.736e-01
```

which we can regard as the "true" estimates.

Comparing with the brute force method and the brute() function, one benefit of the LS method is that there is no need to define any range of the values of the parameters to be estimated. A drawback of the LS method is that the model has to be linear in the parameters, i.e. the LS method is selective regarding applicable models.

http://techteach.no/control/python/model_adapt_temp_least_squares.py

Listing 10.6: model_adapt_temp_least_squares.py

```python
"""
Model adaptation to global temperature change data using least squares method
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""

# %% Import of packages

import numpy as np

# %% Data

x_array = np.arange(1980, 2023)

y_obs_array = np.array([0.26,0.32,0.14,0.31,0.16,0.12,
                        0.18,0.32,0.39,0.27,0.45,0.40,
                        0.22,0.23,0.32,0.45,0.33,0.46,
                        0.61,0.38,0.39,0.54,0.63,0.62,
                        0.53,0.68,0.64,0.66,0.54,0.66,
                        0.72,0.61,0.65,0.68,0.75,0.90,
                        1.02,0.92,0.85,0.98,1.02,0.85,0.89])

# %% Regression model

Phi_col_0 = np.array(x_array)
Phi_col_1 = np.array(np.ones(len(x_array)))
Phi = np.array([Phi_col_0, Phi_col_1]).T
Y = np.array([y_obs_array]).T

# %% Calculating LS estimate

theta = np.linalg.inv(Phi.T @ Phi) @ Phi.T @ Y
a_est = theta[0, 0]
b_est = theta[1, 0]

# %% Calculating value of objective function

y_pred_array = a_est*x_array + b_est
pe = y_obs_array - y_pred_array # prediction error
f_min = sum(pe*pe)

# %% Presentation of result

print(f'a_est = {a_est:.3e}')
print(f'b_est = {b_est:.3e}')
print(f'f_min = {f_min:.3e}')
```

Comments to Program 10.6:

- The LS estimate () is calculated with the code theta = np.linalg.inv(Phi.T @ Phi) @

Phi.T @ y which implements the batch LS formula (10.44).

- The np.linalg.inv() function belongs to the linalg module in the numpy package.
- The @ operator performs matrix multiplication of matrices represented as 2D-arrays in Python.

[End of Example 10.5]

Of course, there is a built-in function in Python which implements least squares estimation, namely lstsq() in Numpy.

Example 10.6 *Adaptation of a static model to global temperature changes data using lstsq() in Nympy*

Program 10.7 uses lstsq() to estimate a and b in model (10.42). The results are as with our native LS code in Example 10.5.

http://techteach.no/control/python/model_adapt_temp_lstsq.py

Listing 10.7: model_adapt_temp_lstsq.py

```
"""
Model adaptation to global temp change data using lstsq() in Numpy
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""

# %% Import of packages

import numpy as np

# %% Data

x_array = np.arange(1980, 2023)

y_obs_array = np.array([0.26,0.32,0.14,0.31,0.16,0.12,
                        0.18,0.32,0.39,0.27,0.45,0.40,
                        0.22,0.23,0.32,0.45,0.33,0.46,
                        0.61,0.38,0.39,0.54,0.63,0.62,
                        0.53,0.68,0.64,0.66,0.54,0.66,
                        0.72,0.61,0.65,0.68,0.75,0.90,
                        1.02,0.92,0.85,0.98,1.02,0.85,0.89])

# %% Regression model

phi_0 = np.array(x_array)
phi_1 = np.array(np.ones(len(x_array)))
Phi = np.array([phi_0, phi_1]).T
Y = np.array([y_obs_array]).T
```

```
# %% Results of LS estimate with lstsq() function

ls_result = np.linalg.lstsq(Phi, Y, rcond=None)
theta = ls_result[0]
sspe_array = ls_result[1]

a_est = theta[0][0]
b_est = theta[1][0]
f_min = sspe_array[0]

# %% Presentation of results

print(f'a_est = {a_est:.3e}')
print(f'b_est = {b_est:.3e}')
print(f'f_min = {f_min:.3e}')
```

[End of Example 10.6]

10.3.3.4 Properties of the LS estimate

The variance of θ_{LS} is:

$$\text{Var}(\theta_{\text{est}}) \quad \equiv \quad \text{diag}\left\{ \text{Cov}\left(\theta_{\text{est}}\right) \equiv \text{E}\left[\left(\theta_{\text{est}} - \theta_0\right)\left(\theta_{\text{est}} - \theta_0\right)^T \right] = \sigma_e^2 \left(\Phi^T \Phi\right)^{-1} \right\} \tag{10.45}$$

where sub-index "e" represents prediction (or model) error. θ_0 are the "true" parameter values.${\sigma_e}^2$ is the variance of the measurement noise (or model error). What do we mean with "variance of θ_{LS}"? It is the variance of a large (ideally: infinite) number of θ_{LS} estimates where each θ_{LS} is calculated with some realization of the random model error e.

If you do not know the value of ${\sigma_e}^2$, you can *estimate* it with:

$${\sigma_e}^2 = \frac{V\left(\theta_{\text{est}}\right)}{N - n} \tag{10.46}$$

where: N is the number of observations. n is the number of parameters to be estimated. $V\left(\theta_{\text{est}}\right)$ is given by (10.38), i.e.:

$$V\left(\theta_{\text{est}}\right) = \left(Y_{\text{obs}} - \Phi\theta_{\text{est}}\right)^T \left(Y_{\text{obs}} - \Phi\theta_{\text{est}}\right) \tag{10.47}$$

You can use (10.46) together with (10.47) to calculate confidence intervals of θ_{est} if you assume that θ_{est} is normally distributed. For example, as you may know from the field of statistics, the 95 % confidence interval of a random estimate θ_{est} given the known, or estimated σ_e given by (10.46), is

$$\theta_{\text{est}} - 2\sigma_e \leqq \theta_{\text{est}} \leqq \theta_{\text{est}} + 2\sigma_e \tag{10.48}$$

The 95 % confidence interval is the interval in which there is 95% probability that the true parameter values is, assuming that the statistical properties of the given data (time series) is representative for data from any other experiment.

Note that the confidence interval does not express any certainty of the calculated estimate itself. Actually, there is no uncertainty at all related to that estimate as the estimate is just a number calculated from the observations with given formulas.

10.3.3.5 Criterion for convergence of estimate towards the true value

The prediction-error vector is

$$E = Y - \Phi\theta_{\text{est}} \tag{10.49}$$

It can be shown that the LS-estimate θ_{LS} converges towards the true value θ_0 of the parameter vector as the number m of sets of observations goes to infinity, only if E is so-called white noise, which is defined in Ch. 31.3.1. White noise means that the elements of E are random numbers, and the vector E has zero mean value. The "opposite" of white noise is coloured noise. E becomes coloured if there are systematic equation errors. Such systematic equation errors can be reduced or eliminated by choosing a more accurate model structure.

A special example of coloured noise is a constant or bias, say b, having value different from zero. For example, a model like (10.50) includes the bias, b.

$$y = ax + b \tag{10.50}$$

10.4 Adaptation of dynamic models to data

10.4.1 Adaptation of dynamic models using brute force optimization

10.4.1.1 Introduction

The brute force method of optimization can be used for adaptation of dynamic models to given data. In Python, you can program the brute force method from scratch as shown in Example 10.7, or using the built-in function scipy.optimize.brute() of the Scipy package, as shown in Example 10.8.

10.4.1.2 Adaptation of dynamic models using brute force optimization

Example 10.7 *DC motor model adaptation to simulated data with brute force optimization*

Figure 10.12 shows a DC motor with tachogenerator. The motor including a mathematical model are presented in Ch. 39.8.

The model is a time constant model:

$$S' = \left[K_u\left(u + L\right) - S\right]/t_c \tag{10.51}$$

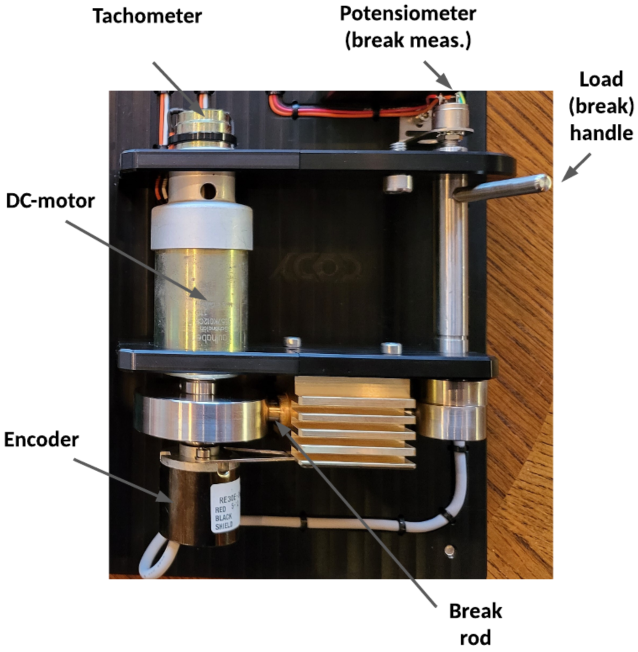


Figure 10.12: DC motor.

where: S [krpm = kilo revolutions per minute] is the er rotational speed. u [V] is the control signal to the motor. L [V] is a voltage which represents the load torque acting on the motor. K [krpm/V] is the motor gain. t_c [s] is the motor time constant.

We will estimate K and t_c from a simulated experiment. We assume for simplicity that both L and the initial state S_{init} are known and have values zero, although we may extend this example to estimate also L and S_{init}.

The objective function to be minimized, is:

$$f_{\text{obj}} = \text{SSPE} = \sum_{k=1}^{N} {e_k}^2 \tag{10.52}$$

where:

$$e_k = S_{\text{obs},k} - S_{\text{pred},k} \tag{10.53}$$

where $S_{\text{obs},k}$ is here the observed (simulated) S, and $S_{\text{pred},k}$ is the predicted S.

We use brute force optimization with the following intervals of K and t_c:

$$K \in [0.05, \ldots, 0.55] \tag{10.54}$$

$$t_c \in [0.1, \ldots, 1.1] \tag{10.55}$$

We assume that the true values of the model parameters are:

$$K_{\text{true}} = 0.15 \tag{10.56}$$

$$t_{c,\text{true}} = 0.30 \tag{10.57}$$

And, as mentioned above, we assume

$$S_{\text{init}} = 0 \tag{10.58}$$

and

$$L = 0 \tag{10.59}$$

In the simulated experiment we vary the control signal, u, as an up-down-up-signal.

The Python program 10.8 implements the parameter estimation. Firstly, the program generates a simulated "experimental" data series, and then uses this data series as the basis of the parameter estimation.

http://techteach.no/control/python/estim_brute_force_Ku_tc_sim_motor.py

Listing 10.8: estim_brute_force_Ku_tc_sim_motor.py

```python
"""
Estimation of gain and time constant in simulated DC motor
using brute force method
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""

# %% Import of packages

import matplotlib.pyplot as plt
import numpy as np


# %% Definition of functions

def fun_sim(modelparams, S_init, u_array, L, N, ts):

    (K, tc) = modelparams
    S_sim_k = S_init
    S_sim_array = np.zeros(N)

    for k in range(0, N):
        # Simulation algorithm (Euler step):
        dS_sim_dt_k = (K*(u_array[k] + L) - S_sim_k)/tc
        S_sim_kp1 = S_sim_k + ts*dS_sim_dt_k
        S_sim_array[k] = S_sim_k
        S_sim_k = S_sim_kp1  # Time shift

    return S_sim_array


def fun_calc_objfun(modelparams, S_init, S_obs_array,
                    u_array, L, N, ts):

    S_pred_array = fun_sim(modelparams, S_init, u_array, L, N, ts)
    pe_array = S_obs_array - S_pred_array
    f_obj = sum(pe_array*pe_array)

    return f_obj

# %% Time settings

t_start = 0  # [s]
t_stop = 7
ts = 0.02
N = int(((t_stop - t_start)/ts)) + 1
t_array = np.linspace(t_start, t_stop, N)

# %% Create input signal

u_array = np.zeros(N)

for k in range(N):
    t_k = k*ts
    if t_start <= t_k < 1:  u_array[k] = 0
    elif 1 <= t_k < 3:  u_array[k] = 1
    elif 3 <= t_k < 5:  u_array[k] = -1
    else: u_array[k] = 0

# %% Creating simulated observation data

K_true = 0.15
tc_true = 0.30
L = 0
modelparams = (K_true, tc_true)
S_init = 0

S_sim_array = fun_sim(modelparams, S_init, u_array, L, N, ts)
S_obs_array = S_sim_array

# %% Arrays of candidates of estimated param values

N_params = 10

K_lb = 0.05  # Lower bound
K_ub = 0.55  # Upper bound

tc_lb = 0.1
tc_ub = 1.1

K_array = np.linspace(K_lb, K_ub, N_params)
tc_array = np.linspace(tc_lb, tc_ub, N_params)

# %% For loop implementing estimation with grid optim

f_obj_min = np.inf  # Initialization of sspe

for K in K_array:
    for tc in tc_array:

        modelparams = (K, tc)  # params with candidate K and tc

        #Calculating objective function (sspe):
        f_obj = fun_calc_objfun(modelparams, S_init,
                                S_obs_array,
                                u_array, L, N, ts)

        #Improving the previous optim solution:
        if f_obj <=  f_obj_min:
            f_obj_min = f_obj
            K_est = K
            tc_est = tc

# %% Displaying the optimal solution = param estimates

print(’-------------------’)
print(f’K_true = {K_true:.4f}’)
print(f’tc_true = {tc_true:.4f}’)
print(’-------------------’)
print(f’K_est = {K_est:.4e}’)
print(f’tc_est = {tc_est:.4e}’)
print(f’f_obj_min = {f_obj_min:.7f}’)

# %% Simulation with estimated parameters in the model

S_init = 0

modelparams = (K_est, tc_est)
S_sim_adapted_model_array = fun_sim(modelparams, S_init,
                                    u_array, L, N, ts)

# %% Plotting:

plt.close("all")
plt.figure(1, figsize=(12, 9))

plt.subplot(2, 1, 1)
plt.plot(t_array, S_obs_array, ’b’, label=’S_obs’)
plt.plot(t_array, S_sim_adapted_model_array, ’r’,
         label=’S_sim’)
plt.grid()
plt.xlabel(’t [s]’)
plt.ylabel(’[krpm]’)
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(t_array, u_array, ’g’, label=’u’)
plt.grid()
plt.xlabel(’t [s]’)
plt.ylabel(’[V]’)
plt.legend()
```

```
# plt.savefig('plot_grid_K_tc_dcmotor_simdata.pdf')
plt.show()
```

Figure 10.13 shows a plot of the control signal u and the simulated speed S.

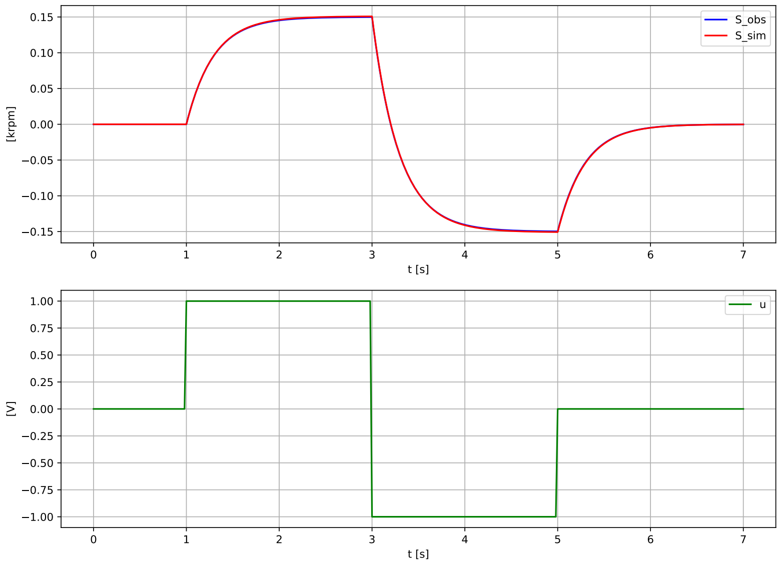


Figure 10.13: Plots of the control signal u and the simulated speed S used in the parameter estimation.

Table 10.1 shows the result of the parameter estimation with two different resolutions of the parameters K and t_c.

Table 10.1: Results of estimation of parameters K_u and t_c using grid optimization.

Parameter	$N_{\text{resolution}}$	K [krpm/V]	t_c [s]	$f_{\text{obj,min}}$
True value	–	0.150	0.300	–
Estimated	10	0.161	0.322	0.0164
Estimated	100	0.151	0.302	0.000137

Above, the true values of K and t_c *are not among* the candidate values. If we had ensured that the true values were among the candidate values, the estimated values by the brute force method would have been these true values.

[End of Example 10.7]

10.4.1.3 Model adaptation of dynamic models with Python's brute() function

In Example 10.7, the brute force method was implemented from scratch. Alternatively, we can use the scipy.optimize.brute() function of the Python Scipy package which implements the brute force method, cf. Section 10.3.1.4.

Example 10.8 *DC motor model adaptation to simulated data with the scipy.optimize.brute() function*

The parameter estimation problem is as in Example 10.7.

The intervals of K and t_c are the same as in Example 10.7, namely:

$$K \in [0.05, \ldots, 0.55] \tag{10.60}$$

$$t_c \in [0.1, \ldots, 1.1] \tag{10.61}$$

The Python program 10.9 below implements the parameter estimator. A resolution of 10 is selected for each of the parameters K_u and t_c. The brute() function is set to run a final Nelder-Mead optimization algorithm to obtain an accurate optimal solution (optimal parameter estimates). Example 10.3 explains how to set this option.

http://techteach.no/control/python/estim_bf_scipy_Ku_tc_sim_motor.py

Listing 10.9: estim_bf_scipy_Ku_tc_sim_motor.py

```
"""
Estimation of gain and time constant and load torque on sim DC motor
with the brute force optim method in Scipy
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""
# %% Import of packages:

import numpy as np
from scipy import optimize

# %% Definition of functions:

def fun_sim(modelparams, S_init, u_array, L, N, ts):

    (K, tc) = modelparams
    S_sim_k = S_init
    S_sim_array = np.zeros(N)

    for k in range(0, N):
        # Simulation algorithm (Euler step):
        dS_sim_dt_k = (K*(u_array[k] + L) - S_sim_k)/tc
        S_sim_kp1 = S_sim_k + ts*dS_sim_dt_k
        S_sim_array[k] = S_sim_k
        S_sim_k = S_sim_kp1  # Time shift
    return S_sim_array


def fun_objective(x):

    K = x[0]
    tc = x[1]
    modelparams = (K, tc)

    S_pred_array = fun_sim(modelparams, S_init, u_array, L, N, ts)

    pe_array = S_obs_array - S_pred_array

    f_obj = sum(pe_array*pe_array)
    return f_obj


# %% Time settings:

t_start = 0  # [s]
t_stop = 7
ts = 0.02
N = int(((t_stop - t_start)/ts)) + 1
t_array = np.linspace(t_start, t_stop, N)

# %% Create input signal:

u_array = np.zeros(N)

for k in range(N):
    t_k = k*ts
    if t_start <= t_k < 1:  u_array[k] = 0
    elif 1 <= t_k < 3:  u_array[k] = 1
    elif 3 <= t_k < 5:  u_array[k] = -1
    else: u_array[k] = 0

# %% Creating simulated observation data:

K_true = 0.15
tc_true = 0.3
L = 0  # [V]
modelparams = (K_true, tc_true)
S_init = 0

S_sim_array = fun_sim(modelparams, S_init, u_array, L, N, ts)
S_obs_array = S_sim_array

# %% Creating arrays of candidate parameter values:

N_params_resolution = 10

K_ub = 0.55  # Upper bound
K_lb = 0.05  # Lower bound
```

```
K_step = (K_ub - K_lb)/(N_params_resolution - 1)

tc_ub = 1.1
tc_lb = 0.1
tc_step = (tc_ub - tc_lb)/(N_params_resolution - 1)

x_ranges = (slice(K_lb, K_ub, K_step),
            slice(tc_lb, tc_ub, tc_step))

# %% Solving the optim problem with optimize.brute():

# Options for the finish argument:
finish_setting = optimize.fmin
# finish_setting = None

result_est = optimize.brute(fun_objective, x_ranges,
                              full_output=True,
                              finish=finish_setting)

params_optim = result_est[0]
f_obj_min = result_est[1]

# %% The optimal parameters values:

K_est = params_optim[0]
tc_est = params_optim[1]

# %% Displaying the results:

print('-------------------')
print(f'K_true = {K_true:.4f}')
print(f'tc_true = {tc_true:.4f}')
print('-------------------')
print(f'K_est = {K_est:.4f}')
print(f'tc_est = {tc_est:.4f}')
print(f'f_obj_min = {f_obj_min:.4f}')
```

The result of the parameter estimation is shown in Table 10.2. The brute() function was able to estimate the parameters accurately despite the realtively poor resolution of 10 for each of the parameters. The good result is due to the final optimization with the Nelder-Mead optimization algorithm (selected as an option in brute() function).

Table 10.2: Result of estimation of parameters K and T with the brute() function with a final Nelder-Mead optimization.

Parameter	$N_{\text{resolution}}$	K [krpm/V]	t_c [s]	$f_{\text{obj,min}}$
True value	–	0.150	0.300	–
Estimated	10	0.150	0.300	0.0

[End of Example 10.8]

10.4.2 Adaptation of dynamic models using nonlinear programming (NLP)

The optimization problem of parameter estimation can be solved with nonlinear programming (NLP). NLP solvers iterate towards the optimal solution (best parameter estimate), but may unfortunately get stuck at a local optimum instead of at the desired global optimum. Figure 10.9 in Section 10.3.2 illustrates this.

Example 10.9 *DC motor model adaptation to simulated data with the Nelder-Mead optimizer in Scipy*

The parameter estimation problem is as in Example 10.7: We will now use the Nelder-Mead NLP solver to estimate K_u and t_c.

Guessed (initial) values are selected as:

$$K_{\text{guess}} = 0.3 \tag{10.62}$$

$$t_{\text{c,guess}} = 0.6 \tag{10.63}$$

Python program 10.10 implements the parameter estimator.

http://techteach.no/control/python/estim_nelder_mead_scipy_Ku_tc_sim_motor.py

Listing 10.10: estim_nelder_mead_scipy_Ku_tc_sim_motor.py

```
"""
Estimation of gain and time constant in simulated DC motor
using the Nelder Mead optim solver in Scipy minimize function
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""
# %% Import:

import numpy as np
import scipy.optimize

# %% Definition of functions:

def fun_sim(modelparams, S_init, u_array, L, N, ts):

    (K, tc) = modelparams
    S_sim_k = S_init
    S_sim_array = np.zeros(N)

    for k in range(0, N):
        # Simulation algorithm (Euler step):
        dS_sim_dt_k = (K*(u_array[k] + L) - S_sim_k)/tc
        S_sim_kp1 = S_sim_k + ts*dS_sim_dt_k
        S_sim_array[k] = S_sim_k
        S_sim_k = S_sim_kp1  # Time shift

    return S_sim_array

def fun_objective(x):

    K = x[0]
    tc = x[1]
    modelparams = (K, tc)

    S_pred_array = fun_sim(modelparams, S_init, u_array, L, N, ts)

    pe_array = S_obs_array - S_pred_array
    f_obj = sum(pe_array*pe_array)

    return f_obj

# %% Time settings:

t_start = 0  # [s]
t_stop = 7
ts = 0.02
N = int(((t_stop - t_start)/ts)) + 1
t_array = np.linspace(t_start, t_stop, N)

# %% Create input signal:

u_array = np.zeros(N)

for k in range(N):
    t_k = k*ts
    if t_start <= t_k < 1:  u_array[k] = 0
    elif 1 <= t_k < 3:  u_array[k] = 1
    elif 3 <= t_k < 5:  u_array[k] = -1
    else: u_array[k] = 0

# %% Creating simulated observation data:

K_true = 0.15
tc_true = 0.30
L = 0
modelparams = (K_true, tc_true)
S_init = 0

S_sim_array = fun_sim(modelparams, S_init, u_array, L, N, ts)
S_obs_array = S_sim_array

# %% Guessed values (initial values) of optim variables:

K_guess = 0.3
tc_guess = 0.6
x_guess = np.array([K_guess, tc_guess])

# %% Solving the optim problem:
```

```
res = scipy.optimize.minimize(fun_objective, x_guess,
                              method = 'nelder-mead',
                              options = {'disp': True})

# %% The results of the optimization:
(K_est, tc_est) = res.x
f_obj_min = res.fun

# %% Displaying the optimal solution:
print('--------------------')
print(f'K_true = {K_true:.4f}')
print(f'tc_true = {tc_true:.4f}')
print('--------------------')
print(f'K_est = {K_est:.4f}')
print(f'tc_est = {tc_est:.4f}')
print(f'f_obj_min = {f_obj_min:.4f}')
```

The result of the parameter estimation is shown in Table 10.3. The parameter estimates are virtually identical to the true values.

Table 10.3: Result of estimation of parameters K and t_c with the minimize() function with Nelder-Mead optimzation solver.

Parameter	K [krpm/V]	t_c [s]	$f_{\text{obj,min}}$
True value	0.150	0.300	–
Estimated	0.150	0.300	0.0

[End of Example 10.9]

10.4.3 Adaptation of dynamic models using the least squares method

Assume that the dynamic model which you want to adapt to data, can be written on the form of (10.29), which is repeated here for convenience:

$$y = \varphi\theta \tag{10.64}$$

With m observations, the stacked model is:

$$\underbrace{\begin{bmatrix} \vdots \\ y_k \\ \vdots \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} \vdots \\ \varphi_k \\ \vdots \end{bmatrix}}_{\Phi} \theta \tag{10.65}$$

where $k \in [0,\ m-1]$ is the time index. Then, the parameters can be estimated with the ordinary least squares (LS) method. The parameter estimates are given by (10.41), which is repeated here:

$$\theta_{\text{LS}} = (\Phi^T\Phi)^{-1}\Phi^T Y \tag{10.66}$$

Example 10.10 *Estimation of parameters of a dynamic model using ordinary LS method*

The parameter estimation problem is as in Example 10.7. The motor model is repeated here for convenience:

$$S' = \left[K\left(u + L\right) - S\right]/t_c \tag{10.67}$$

We will estimate K and t_c. L is are assumed having value 0.

We define the parameter vector θ as

$$\theta = \left[\begin{array}{c} K \\ t_c \end{array}\right] \tag{10.68}$$

The model (10.67) can be written on the standard regression form (10.64) as follows, where k is the time index:

$$\underbrace{S_k}_{y_k} = \underbrace{\left[(u_k + L),\ -S_k'\right]}_{\varphi_k}\underbrace{\left[\begin{array}{c} K \\ t_c \end{array}\right]}_{\theta} \tag{10.69}$$

We may calculate S_k' as backward differentiation (but other methods may be used, too):

$$S_k' = \frac{S_k - S_{k-1}}{t_s} \tag{10.70}$$

where t_s is the time step.

The parameter vector is can now be estimated with the LS formula (10.66):

$$\theta_{\text{LS}} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y \tag{10.71}$$

where Φ and Y are "stacked" values of φ_k and y_k, respectively.

Python program 10.10 implements the parameter estimator.

http://techteach.no/control/python/estim_least_squares_Ku_tc_sim_motor.py

Listing 10.11: estim_least_squares_Ku_tc_sim_motor.py

```
"""
Estimation of gain and time constant in simulated DC motor
using ordinary LS
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 10
"""

# %% Import of packages:

import numpy as np

# %% Definition of simulation function:

def fun_sim(modelparams, S_init, u_array, L, N, ts):
    (K, tc) = modelparams
    S_sim_k = S_init
    S_sim_array = np.zeros(N)

    for k in range(0, N):
        # Simulation algorithm (Euler step):
        dS_sim_dt_k = (K*(u_array[k] + L) - S_sim_k)/tc
        S_sim_kp1 = S_sim_k + ts*dS_sim_dt_k
        S_sim_array[k] = S_sim_k
        S_sim_k = S_sim_kp1  # Time shift

    return S_sim_array

# %% Time settings:

t_start = 0  # [s]
t_stop = 7
ts = 0.02
N = int(((t_stop - t_start)/ts)) + 1
t_array = np.linspace(t_start, t_stop, N)

# %% Create input signal:

u_array = np.zeros(N)

for k in range(N):
    t_k = k*ts
    if t_start <= t_k < 1:  u_array[k] = 0
    elif 1 <= t_k < 3:  u_array[k] = 1
    elif 3 <= t_k < 5:  u_array[k] = -1
    else: u_array[k] = 0

# %% Creating simulated observation data:

K_true = 0.15
tc_true = 0.30
L = 0
modelparams = (K_true, tc_true)
S_init = 0

S_array = fun_sim(modelparams, S_init, u_array, L, N, ts)

# %% Regression model

dS_dt_array = (S_array[1:] - S_array[:-1])/ts

Phi_col_0 = np.array(u_array[:-1]) + L
Phi_col_1 = np.array(-dS_dt_array)

Phi = np.array([Phi_col_0, Phi_col_1]).T
Y = np.array([S_array[:-1]]).T

# %% Calculating LS estimates
```

```
theta = np.linalg.inv(Phi.T @ Phi) @ Phi.T @ Y
theta_0 = theta[0, 0]
theta_1 = theta[1, 0]

K_est = theta_0
tc_est = theta_1

# %% Presenting estimates

print('--------------------')
print(f'K_true = {K_true:.3e}')
print(f'tc_true = {tc_true:.3e}')
print('--------------------')
print(f'K_est = {K_est:.3e}')
print(f'tc_est = {tc_est:.3e}')
print('--------------------')
```

The results of the parameter estimation is shown in Table 10.4. The parameter estimates are identical to the true values.

Table 10.4: Result of estimation of parameters K and t_c with the (ordinary) least squares method.

Parameter	K [krpm/V]	t_c [s]
True value	0.150	0.300
Estimated	0.150	0.300

[End of Example 10.10]

Least squares method is super fast, but ...

The (ordinary) least squares (LS) method is super fast as it uses a formula, (10.66), and no iterative or search algorithm algorithm, to find the optimal solution, i.e. the parameter estimates. This is a benefit of the method. But the model (10.64) is *linear* in the parameters. If the model is nonlinear in the parameters, you can not apply the least squares method. In such cases you can use e.g. the brute force method, cf. Section 10.4.1, or nonlinear programming (NLP), cf. Section 10.4.2.

Linear models are just special cases of nonlinear models. Therefore, you can use the LS method also on models being linear in its parameters. The only drawback about that, compared to using the LS method, is that brute force method or an NLP method will take longer time to execute. However, I guess that in most cases the computational time needed to execute the brute force method or an NLP method is of little or no importance. Therefore, my general recommendation is to regard the brute force method or an NLP method – and not the LS method – as you main method for parameter estimation.

10.5 Recursive (real-time) model adaptation

The parameter estimation described in this section is a *batch* estimation since it operates on the whole data series available. An alternative term to batch estimation is *full information estimation*. If you assume that the parameters may change continuously during the time interval of interest, the parameter estimates obtained with batch estimation may not be good estimates at recent (newest) times. In such cases, you may consider *online parameter estimation* where the estimates are updated continuously based on the most recent process measurement available.

Some alternative methods for online parameter estimation are:

- Kalman Filter, which is presented in Ch. 32.
- Recursive least squares method (RLS) with forgetting factor (Saelid & Foss 1983) – not presented in this book. It can be regarded as a special case of the Kalman Filter.
- Moving horizon estimation (MHE) (Boegli 2014) – not presented in this book.

In MHE and Kalman Filter, the parameters are estimated as state variables. The original state vector is augmented with parameter states. Therefore, the Kalman Filter used for parameter estimation is denoted *augmented* Kalman Filter, and the MHE can similarly be denoted augmented MHE.

Among those two alternatives, I generally recommend the Kalman Filter since it is easier to implement and executes faster. I have experiences relatively simple cases where MHE fails to estimate the parameters while the Kalman Filter works well.

10.6 *Problems for Chapter* 10

Problem 10.1 *Characterizing models as linear and nonlinear*

Give an example of a model that is linear in the parameters, and an example of a model that is nonlinear in the parameters.

Problem 10.2 *How to check if estimation has a chance to succeed*

Assume that you have programmed some parameter estimation method. How can you check if your estimator is able to produce parameter estimates successfully?

Problem 10.3 *Bias in LS estimate*

This problem demonstrates that a LS-estimate converges towards an erroneous value – in other words the estimate is biased – if the noise or the model error has a mean value different from zero.[4]

The constant K in the model

$$y = K \tag{10.72}$$

is be estimated. Assume that the real (true) value of K is K_0. The observation y_k is

$$y_k = K_0 + e_k \tag{10.73}$$

where e_k is noise (or equation error or model error or prediction error). Calculate the LS estimate of K (you can assume that there are N observations). Assume that y_k in (10.72) is given by (10.73).

Problem 10.4 *Which model order?*

Figure 10.14 shows the criterion function V as a function of model order n in a fictitous problem about parameter estimation.

Which order should be selected?

Problem 10.5 *Model adaptation to consumer price index (CPI) in Norway*

The annual consumer price index (CPI) in Norway is available at https://www.ssb.no/en/statbank/table/03014. There you can select the period (years) for which you want to see the CPI values.

[4] In general, the estimate will be biased if the noise is *coloured* (non-white).

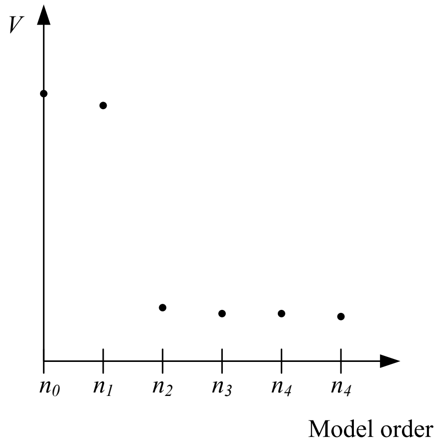


Figure 10.14: The criterion function V as a function of model order n

In the problems below, you are to estimate parameters a and b of the model

$$y = ax + b \tag{10.74}$$

to the CPI values of the years 2000–2009 (10 years) using the specified methods. In (10.74), y is CPI, and x is year. Also, both observed and predicted values of y should be plotted in the same figure.

1. Plot the data of y and x

2. Adapt the model with the Brute force method programmed in elementary Python code

3. Adapt the model with the Scipy brute function with ultimate optimization based on Nelder-Mead optimization

4. Adapt the model with NLP with Scipy minimize function with Nelder-Mead optimization

5. Adapt the model with the ordinary least squares method in elementary Python code

6. Adapt the model with the least squares method with the Numpy lstsq function

7. Is the prediction accurate?

One use of models is to predict values for future times, where we – naturally – do not have any observations. It is then interesting to see, afterwards, how good the predictions actually were. (If the predictions turn out to be bad, the mathematical model should be re-evaluated.)

Plot the predicted CPI, i.e. CPI as calculated with formula ([eq:kpi]) where a and b are as found in part task b, for the period 2010–2019 (10 years), which is a period that was not used for the model fitting.

Do you think that the predicted CPI matches well with the actual CPI in the period 2010–2019? In other words: Has it been shown that the adapted model for 2000–2009 is (was) usable for predicting future CPI in the period 2010–2019? (Only a qualitative answer is expected here.)

Problem 10.6 *Predicting the temperature change*

See Example 10.1. The model (10.10) with $a = 1.8979 \cdot 10^{-2}$ and $b = -3.7437 \cdot 10^{1}$ can be used to predict the change, y, in the Earth's temperature. Predict y in year 2040!

Problem 10.7 *Analysis of LS estimate*

In Example 10.5, Program 10.6 uses the ordinary least squares method to estimate the parameters a and b in the model

$$y = ax + b$$

to given data (observations) of temperature change, y, at given years, x. Modify Program 10.6 to calculate the 95 % confidence intervals of $a_{\text{est}}$and b_{est}.

Problem 10.8 *Minimum order at estimation of time delay*

Assume that you know in advance that a given physical process has time delay of 2 sec. You are to estimate a discrete-time transfer function to from experimental input and output (measurement) sequences. The sampling time is 0.5 sec. You are not sure about the order of the transfer function, but what is the minimum order that it should have?

Problem 10.9 *Mathematical modeling of a DC motor*

Here is a mathematical model of an electric motor (a DC motor) with load torque:

$$J\omega' = \frac{K_T}{R_a}(v_a - K_e\omega) + T_L \tag{10.75}$$

Assume that the inertia J and the torque T_L will be estimated with the LS method. The motor is excited with the armature voltage v_a. The rotational speed ω is measured with a tachogenerator. K_T, R_a and K_e have known values.

Write the model on the standard form $y = \varphi\theta$. Use the center difference method to calculate ω' (see below).

The center difference method

Using y as the name of the variable or signal, the center difference method is

$$y(t_k)' \approx \frac{y(t_{k+1}) - y(t_{k-1})}{2t_s} \tag{10.76}$$

For a given time step t_s, the center difference method gives a somewhat more accurate approximation to y than the forward difference method and the backward difference method. Actually, the center difference method is the average of those two approximations.

Problem 10.10 *Adaption of linear dynamic models*

Assume that you want to adapt a linear dynamic model, e.g. a transfer function, to experimental data. If the model is actually nonlinear, a linear model will give a good representation of the system only near an operating point. How can you process the input and output data (the sequences) before using them in system identification to improve the accuracy of the estimated linear model, assuming the process is nonlinear?

Problem 10.11 *Is is possible to estimate the parameters?*

Given the following "time constant" model:

$$t_c x' = Ku + d - x$$

where x is the measurement signal, u is the control signal, and d is the disturbance. K and t_c are model parameters. x and u have known values (at the sampling points of time).

In the problems below: "Static" means "constant". You should explain your answers, but no rigorous explanations are expected.

1. Assume that both K and d have known values. Is it possible to estimate t_c from static data?
2. Assume that t_c is known. Is it possible to estimate both K and d from static data?

Problem 10.12 *LS estimation of dynamic model*

Given the model:

$$x_{k+1} = ax_k + bu_k \tag{10.77}$$

Explain how you can estimate the assumed constant parameters a and b with the ordinary least squares method using data series (time series) of sampled values of x and u. In your answer, you are not expected to actually calculate the estimates, but the formula of the estimates should be given.

Problem 10.13 *Estimation of parameters and load torque on a simulated DC motor with Scipy's brute() function*

In Example 10.8 we estimated the motor parameters K and t_c from simulated data using Scipy's brute function.

Write a Python program, starting with the program available in Example 10.7, that estimates K and t_c and also an assumed constant load voltage ("load torque"), L. You can assume that the true but assumed unknown value is $L_{\text{true}} = -0.25$ V.

Problem 10.14 *Estimation of parameters and load torque on a real DC motor with Scipy's brute() function*

In Problem 10.13, K, t_c, and L on a simulated DC motor are estimated. In the present problem, these values are estimated from data of a real motor.

The experimental data of the real motor are available at this link:

http://techteach.no/control/python/data_dc_motor.txt

The sampling time is 0.02 s (as you can also see in the data file). The file has the following columns of data:

- Colum 0: Time in seconds.
- Colum 1: The control signal u in voltage, adjusted manually more or less randomly.
- Colum 2: The tachogenerator voltage S_{tacho}, which can be converted to a corresponding rotational speed in krpm (kilo revolutions per minute) with:

$$S_{\text{krpm}} = S_{\text{tacho}}/K_t \tag{10.78}$$

where:

$$K_t = 5.0 \text{ V/krpm} \tag{10.79}$$

You can load the data file to Python with e.g. the loadtxt() function of the Numpy package.

Based on the program given in the solution of Problem 10.13, create a program which estimates K, t_c, and L from u and S_{krpm}, and (after the estimation) simulates S_{krpm} with the estimated parameters, and plots the simulated S together with the real S_{krpm}. Does it look like the adapted model represents the real motor well?

Problem 10.15 *Model adaptation to a real air heater with the scipy.brute() function*

Section 39.5 describes a laboratory air heater. A data file from an experiment on the air heater is available in Section 39.5.5.

Write a Python program which estimates the following variables using the data file: Gain K_h. Time constant t_c. Time delay t_d. Environmental temperature T_{env}. Initial value of the outlet temperature, T_{init}. In the call of the brute() function, you can select the option with final optimization, i.e. setting the finish argument to optimize.fmin.

Simulate the air heater with the control signal of the data file, and plot both the simulated temperature and the observed (measured) temperature in the same diagram. Does the adapted model represent the real air heater accurately?

Problem 10.16 *Comparing parameter estimation methods*

Range the following parameter estimation methods (approaches) in terms of program (algorithm) execution speed:

- Nonlinear programming (NLP)
- Ordinary Least Squares method
- Gridding (or brute force) method

Also, for each of these methods, give an important applicability limitation.

10.7 *Solutions to problems for Chapter* 10

Solution to Problem 10.1

Example of a model which is linear in the parameters (a and b):

$$x' = au + bx \tag{10.80}$$

A model which is nonlinear in the parameters (c and d):

$$x' = c(du–x) \tag{10.81}$$

However, in the latter example, the model can be written as a model being linear in the parameters e and c where

$$e = cd \tag{10.82}$$

The reformulated model is:

$$x' = eu - cx \tag{10.83}$$

Solution to Problem 10.2

1. Generate simulated data using a model with the known, true parameters.
2. Estimate the parameters from the simulated data.
3. Check if the parameter estimate becomes equal to the true parameter values.

Solution to Problem 10.3

K is given by

$$\begin{aligned} y_k &= K && (10.84)\\ &= 1 \cdot K && (10.85)\\ &= \varphi\theta && (10.86) \end{aligned}$$

The LS estimate is

$$
\begin{aligned}
\theta_{\mathrm{LS}} &= K_{\mathrm{LS}} \qquad (10.87)\\
&= (\Phi^T\Phi)^{-1}\Phi^T y \qquad (10.88)\\
&= \left(\begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} K_0 + e_1 \\ K_0 + e_2 \\ \vdots \\ K_0 + e_N \end{bmatrix} \qquad (10.89)\\
&= N^{-1} \cdot \left[N \cdot K_0 + \sum_{k=1}^{N} e_k \right] \qquad (10.90)\\
&= K_0 + \frac{1}{N} \sum_{k=1}^{N} e_k \qquad (10.91)\\
&= K_0 + m_e \qquad (10.92)
\end{aligned}
$$

where m_e is the mean value of the noise. From (10.92) you can see that K_{LS} *does not converge* towards K_0 if the mean value m_e is *different from zero*.

Solution to Problem 10.4

According to the Parsimony Principle, you should select the smallest order with relatively small value of the criterion function. So, you should select order n_2.

Solution to Problem 10.5

1. Plot the data of y and x

Python program 10.10 plots the data, see Figure 10.15.

http://techteach.no/control/python/plot_cpi_data.py

Listing 10.12: plot_cpi_data.py

```
"""
Plotting CPI data
Finn Aakre Haugen, TechTeach. finn@techteach.no
2023 12 11
"""

#%% Import of packages

import numpy as np
import matplotlib.pyplot as plt

```

```
#%% Data

x_array = np.arange(2000, 2009+1)

y_obs_array = np.array([75.5, 77.7, 78.7, 80.7, 81.0,
                        82.3, 84.2, 84.8, 88.0, 89.9])

# %% Plotting

plt.close(’all’)
plt.figure(1)

plt.plot(x_array, y_obs_array,’bo-’, label=’y_obs’)
plt.legend()
plt.grid()
plt.xlabel(’x [year]’)
plt.ylabel(’[CPI index]’)

plt.savefig(’plot_cpi_2000_2009.pdf’)
plt.show()
```

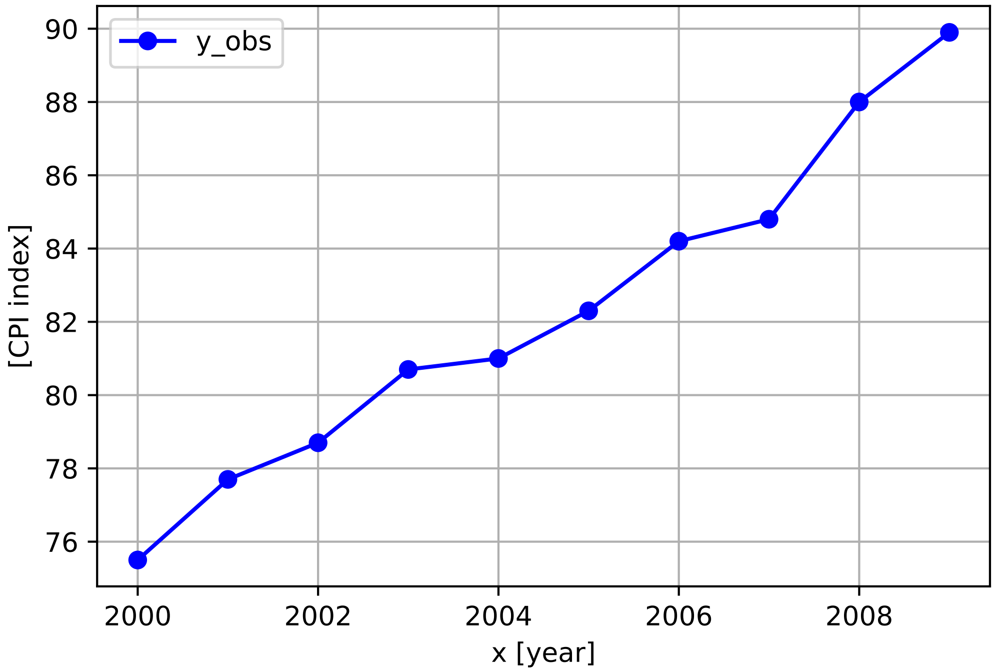


Figure 10.15: Plot of CPI data.

2. Brute force method programmed in elementary Python code

We need to determine a range of candidate values of both a and b:

- *Range of a (slope)*: From the data the slope is approximately:

$$a \approx \frac{90 - 75}{10} = 1.5 \tag{10.93}$$

- *Range of b (the constant term)*: From the model we get:

$$b = y - ax \tag{10.94}$$

  where we may use the data point ($x = 2000, y = 75$), to give

$$b \approx 75.5 - 1.5 \cdot 2000 = -2924 \tag{10.95}$$

Based on these values of a and b, we define the following ranges of candidate values of a and b:

$$a \in [1.0,\ 2.0] \tag{10.96}$$

$$b \in [-4000,\ -2000] \tag{10.97}$$

The following Python program estimates a and b, and plots the observed y and the predicted y, see Figure 10.16.

http://techteach.no/control/python/model_adapt_cpi_elementary_bf.py

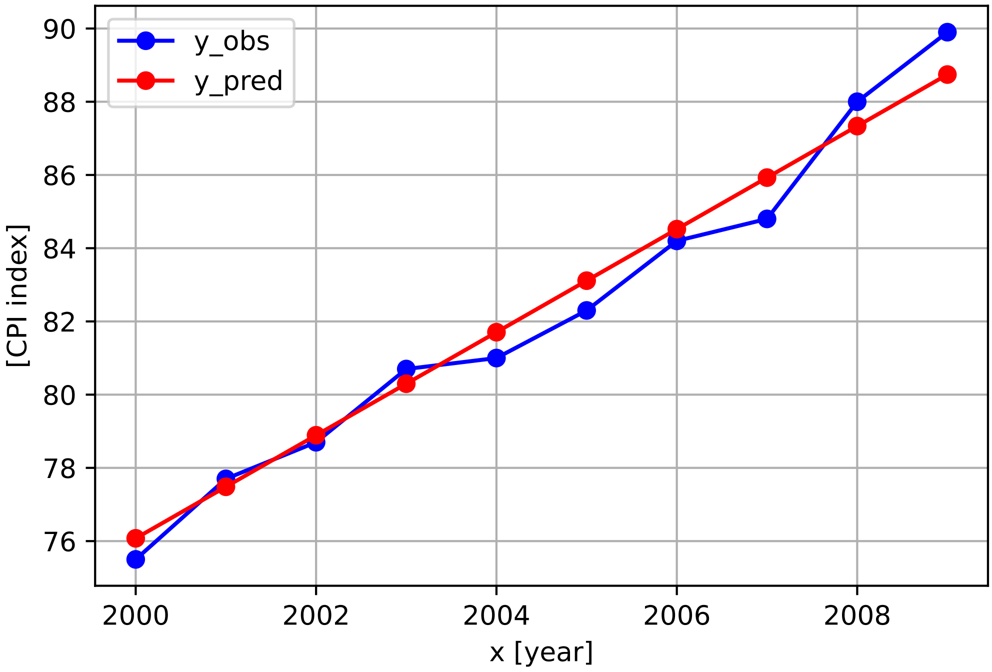


Figure 10.16: Plot of observed and predicted CPI data with elementary brute force method.

The results are:

```
a_est = 1.4074e+00
b_est = -2.7387e+03
f_min = 4.8917e+00
```

3. Scipy brute function with ultimate optimization based on Nelder-Mead optimization

The following Python program estimates a and b using a relative small value of N, namely 10. (Although the program plots observed and predicted y, I do not show the plot here.)

http://techteach.no/control/python/model_adapt_cpi_scipy_bf.py

Results:

```
a_est = 1.479e+00
b_est = -2.882e+03
f_min = 4.304e+00
```

It is interesting that $f_{\min}$with $N = 10$ is slightly smaller than with the elementary brute force method with $N = 1000$. The smaller $f_{\min}$ is due to the ultimate optimization used in the Scipy brute function.

C:\techteach.no\publications\mod_sim_con\python\model_adapt_cpi_scipy_nelder_mead.py

4. NLP with Scipy minimize function with Nelder-Mead optimization

Python program:

http://techteach.no/control/python/model_adapt_cpi_scipy_nelder_mead.py

In the program, (10.93), i.e. 1.5, is used as guessed (or initial) value of a and (10.95), i.e. -2924, as guessed value of b.

Results:

```
a_est = 1.479e+00
b_est = -2.882e+03
f_min = 4.304e+00
```

which are the same as with the Scipy brute function.

5. Ordinary least squares method in elementary Python code

The least squares method does not require any guessed value of a and b.

Python program:

http://techteach.no/control/python/model_adapt_cpi_elementary_ls.py

Results:

```
a_est = 1.479e+00
b_est = -2.882e+03
f_min = 4.304e+00
```

which are the same as with the Scipy brute function and with the NLP Scipy minimize function.

6. The least squares method with the Numpy lstsq function

Python program:

http://techteach.no/control/python/model_adapt_cpi_lstsq.py

Results:

```
a_est = 1.479e+00
b_est = -2.882e+03
f_min = 4.304e+00
```

which are the same as with the elementary implementation of the least squares method.

7. Is the prediction accurate?

The following Python predicts y and plots the observed y and the predicted y over the years 2010-2019 using the model parameters $a = 1.479$ and $b = -2882$, see Figure 10.17. It seems that the model was able to predict the CPI usefully only for the first 5 future years.

http://techteach.no/control/python/predict_cpi_2010_2019.py

Solution to Problem 10.6

$$y_{\text{pred}} = ax + b = 1.8979 \cdot 10^{-2} \cdot 2040 - 3.7437 \cdot 10^{1} = 1.28^{\circ}\text{C}$$

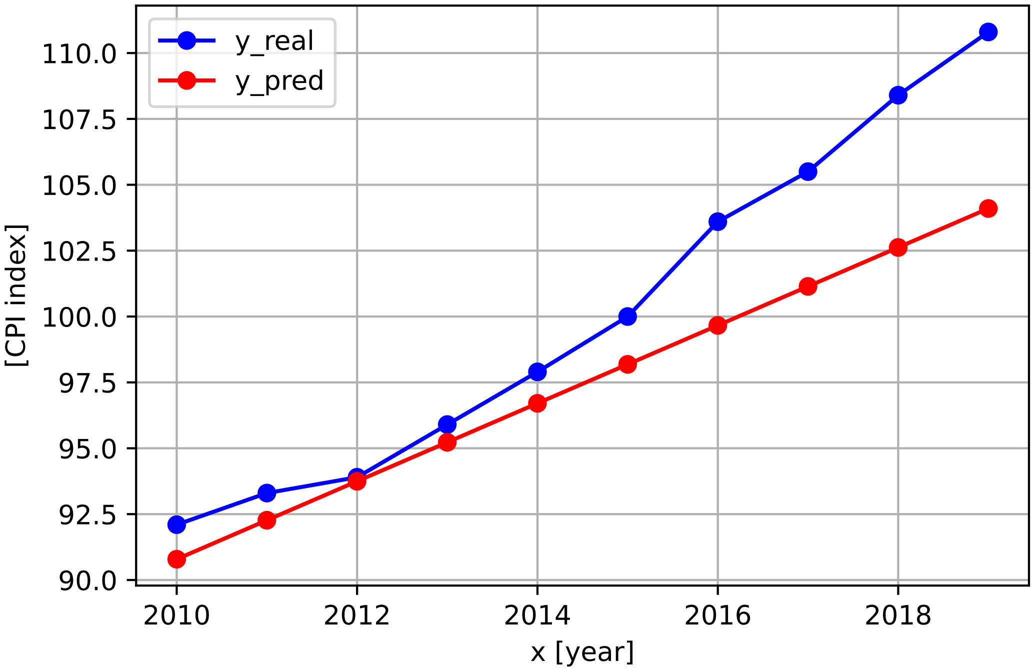


Figure 10.17: Observed y and the predicted y over the years 2010-2019.

Solution to Problem 10.7

See the following Program:

http://techteach.no/control/python/estimate_analysis_ls_estim_temp.py

The results are shown in the box below.

```
a_est = 1.894e-02
b_est = -3.736e+01
f_min = V = 3.736e-01
std_a = 1.173e-03
std_b = 2.347e+00
confident_interval_a = [0.0166 0.0189 0.0213]
confident_interval_b = [-42.054 -37.36 -32.665]
```

So, the 95 % confidence intervals are:

$$a_{\text{est}}\colon [0.166,\ 0.0213] \tag{10.98}$$

$$b_{\text{est}}\colon [-42.054,\ -32.665] \tag{10.99}$$

Solution to Problem 10.8

The order should be large enough to include the time delay. One time step of 0.5 sec corresponds to a time delay of 0.5 s, and one such time delay is represented by the factor z^{-1} in the transfer function. The model should therefore include $2/0.5 = 4$ such factors. Hence, the mimimum order of the transfer function is 4.

Solution to Problem 10.9

The model written on the standard LS form:

$$\underbrace{\frac{K_T}{R_a}\left[v_a(t_k) - K_e\omega(t_k)\right]}_{y} = \underbrace{\left[\begin{array}{cc} \omega(t_k)' & -1 \end{array}\right]}_{\varphi} \underbrace{\left[\begin{array}{c} J \\ T_L \end{array}\right]}_{\theta} \tag{10.100}$$

$\omega(t_k)'$ is calculated with the center difference method:

$$\omega(t_k)' \approx \frac{\omega(t_{k+1}) - \omega(t_{k-1})}{2t_s} \tag{10.101}$$

where t_s is the time step.

Solution to Problem 10.10

You can remove the mean values of both the input and the output signals (time series or sequences), and use the devations as new input and output. In more detail: Assume that $\{u(k)\}$ is the original input signal and $\{y(k)\}$ is the output signal, and that m_u and m_y are the respective mean values. The deviation signals are then

$$\{du(k)\} = \{u(k) - m_u\} \tag{10.102}$$

and

$$\{dy(k)\} = \{y(k) - m_y\} \tag{10.103}$$

The signals $\{du(k)\}$ and $\{dy(k)\}$ are used as input and output signals.

Solution to Problem10.11

Given the following “time constant” model:

$$t_c x' = Ku + d - x$$

where x is the measurement signal, u is the control signal, and d is the disturbance. K and t_c are model parameters. x and u have known values (at the sampling points of time).

In the problems below: “Static” means “constant”. You should explain your answers, but no rigorous explanations are expected.

1. t_c can not be estimated from static data because dx/dt is zero, effectively removing t_c from the model, and therefore, whatever static values of x and u, we can not calculate an estimate of t_c.

2. The static model becomes:

$$0 = Ku_s + d – x_s$$

giving:

$$x_s = Ku_s + d$$

Since x_s and u_s are known, the above equation is one quation with two unknowns, K and d, so there is no unique solution for K and d. Hence, we can not excpect any estimate of K and d to be equal to the true value (i.e. the estimates are not consistent).

Solution to Problem 10.12

The given model must be written on the regression model form, which is

$$y_k = \varphi_k \theta \tag{10.104}$$

where θ is the parameter vector to be estimated.

The model written on the regression model form:

$$\overbrace{x_{k+1}}^{y_k} = \overbrace{[x_k, \ u_k]}^{\varphi_k} \overbrace{\begin{bmatrix} a \\ b \end{bmatrix}}^{\theta}$$

The parameter vector estimate is calculated with the LS formula given in the formula list:

$$\theta_{LS} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y$$

where:

$$\Phi = \begin{bmatrix} \vdots \\ \varphi_k \\ \vdots \end{bmatrix}$$

$$Y = \begin{bmatrix} \vdots \\ y_k \\ \vdots \end{bmatrix}$$

For simplicity, the lower and upper indexes of φ_k and y_k are omitted in the vectors (arrays) above.

Solution to Problem 10.13

The Python program named 'prog_grid_motor_K_T_L_estim_sim.py' which is available via the link below implements the model adaptation. Firstly, the program generates a simulated data series, and then uses this data series as the basis of the parameter estimation.

http://techteach.no/control/python/estim_motor_K_tc_L_bf_sim.py

Figure 10.18 shows a plot of the control signal u and the observed (simulated) S.

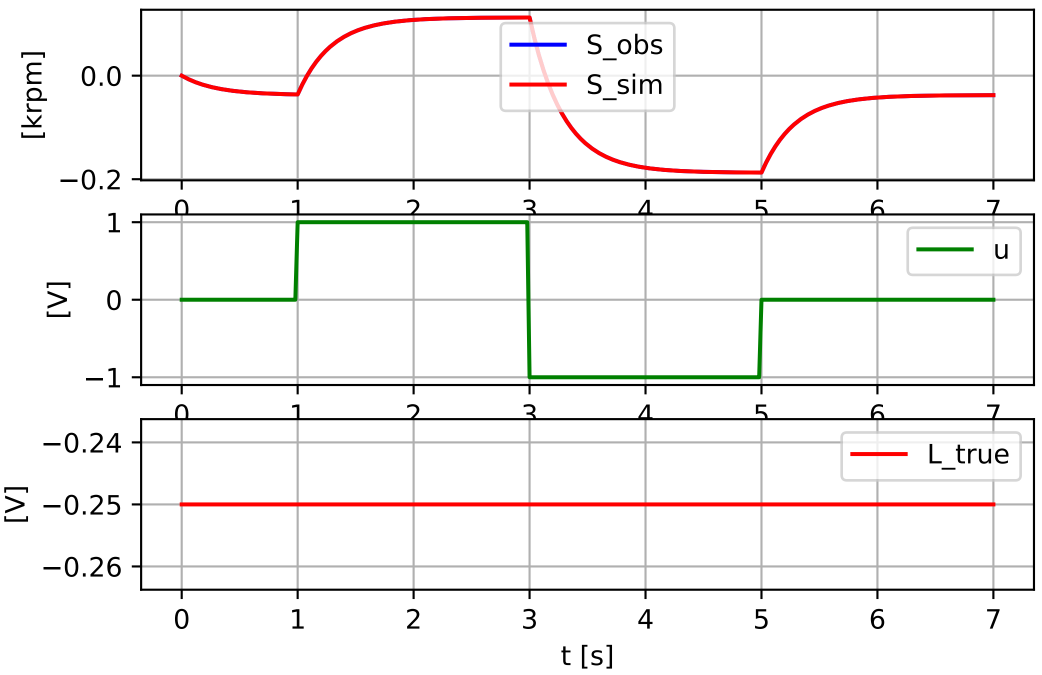


Figure 10.18: Control signal u and observed (simulated) S which is the basis of the model adaptation.

The results of the parameter estimation are:

```
K_true = 0.1500
tc_true = 0.3000
L_true = -0.2500
-------------------
K_est = 0.1500
tc_est = 0.3000
L_est = -0.2500
f_obj_min = 0.0000
```

Perfect estimation!

Solution to Problem 10.14

The following Python program implements the parameter estimation:

http://techteach.no/control/python/estim_bf_scipy_Ku_tc_L_real_motor.py

Figure 10.19 shows a plot of the control signal u and the simulated S together with the real S. It seems that the model represents the motor well.

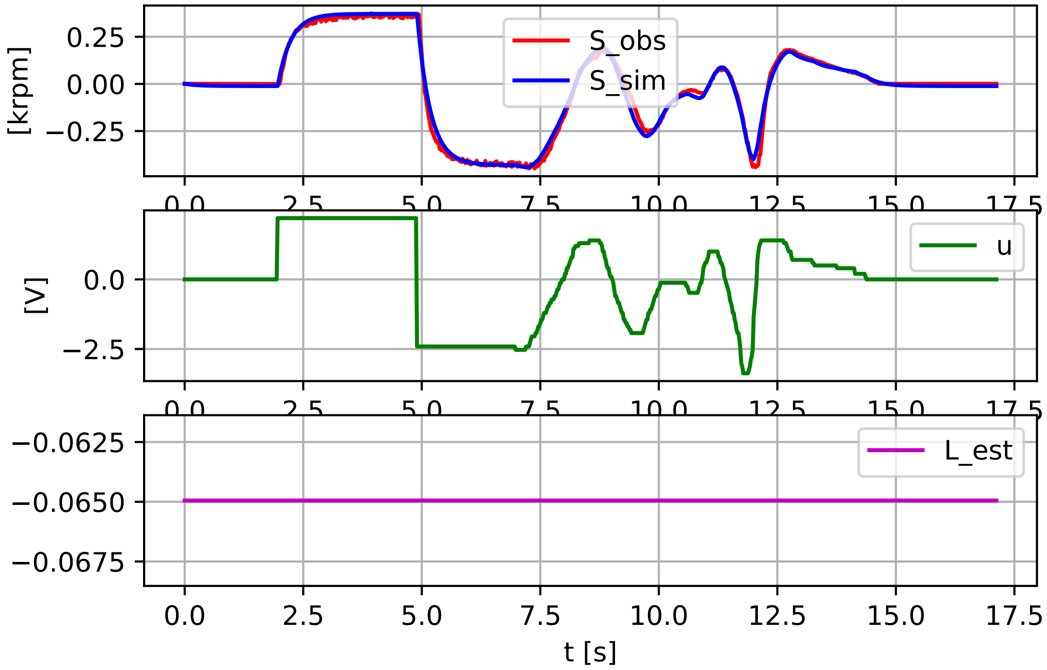


Figure 10.19: Plots of control signal u, simulated S and real S, and estimated L.

The results of the parameter estimation are:

```
K_est = 0.1741
tc_est = 0.2854
L_est = -0.0649
f_obj_min = 0.4507
```

L_est is negative, which indicates a breaking (friction) torque acting on the motor – makes sense.

Solution to Problem 10.15

The Python program named 'prog_estim_brute_airheater_real.py' which is available via the link below implements the model adaptation.

http://techteach.no/control/python/estim_bf_scipy_real_airheater.py

The result of the estimation is shown in the box below.

```
Kh_est = 3.9730
tc_est = 27.8173
td_est = 2.7852
T_env_est = 22.0161
T_init_est = 23.1379
f_obj_min = 32.5108
```

Figure 10.20 shows a plot of the control signal u and the simulated T together with the observed (real) T. It seems that the model represents the air heater well.

Solution to Problem 10.16

Range in terms of program (algorithm) execution speed:

1. Ordinary Least Squares method.
   Applicability limitation: The model must be linear in the parameters.
2. Nonlinear programming (NLP).
   Applicability limitation: There is a chance that the parameter estimates only satisfy a local minimum of the optimization criterion, and hence are not the global optimal estimates.
3. Brute force method.
   Applicability limitation: Due to limited resolution in the parameter estimate candidates, the precicely optimal parameter estimates are not found.

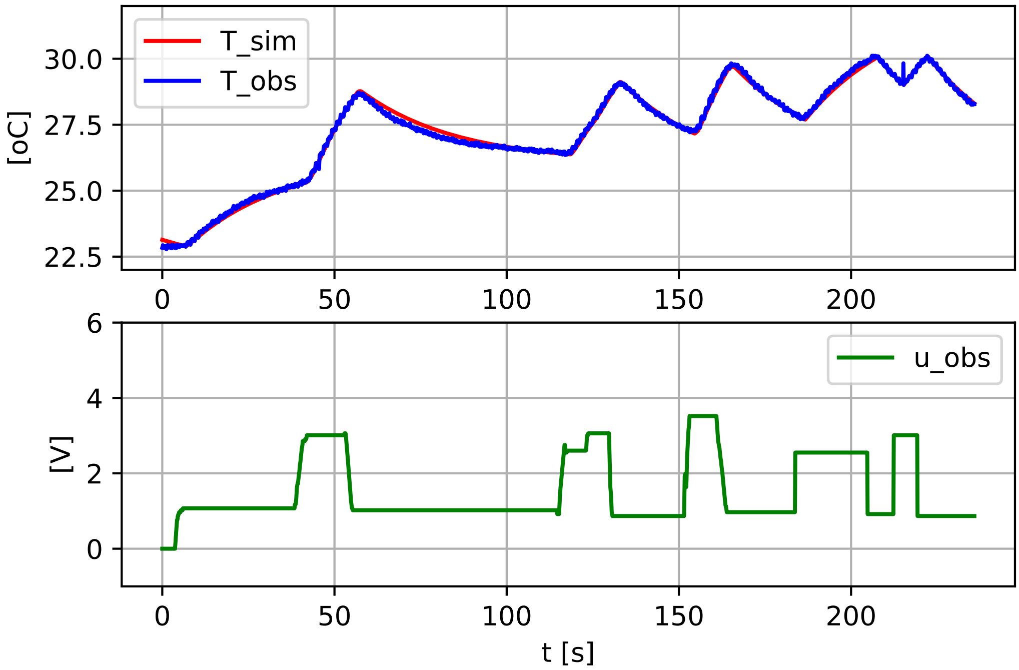


Figure 10.20: Plots of control signal u, and simulated T and observed (real) T.

Part IV

BASIC CONTROL METHODS

Chapter 11

PID control (continued)

11.1 Introduction

You met the PID controller – both in continuous-time form and in discrete-time form – in Section 1.3.4. The present chapter describes several additional aspects of the PID controller.

11.2 Transfer function of the PID controller

The continuous-time PID controller is:

$$u \;=\; u_{\mathrm{man}} + \underbrace{K_c e}_{u_p} + \underbrace{\frac{K_c}{T_i}\int_0^t e\,d\tau}_{u_i} + \underbrace{K_c T_d\, {e_f}'}_{u_d} \tag{11.1}$$

where e_f is the lowpass filtered control error. Assuming a continuous-time time constant filter, the filter model is

$${e_f}' = (e - e_f)\,/t_{fd} \tag{11.2}$$

where t_{df} [s] is the filter time constant.

In some situations, typically analysis and certain forms of simulations, we need the transfer function from control error e to control signal u of the PID controller. Let's derive this transfer function from (11.1) and (11.2).

We take the Laplace transform of (1.12) while omitting the term u_{man}:

$$U(s) = K_c E(s) + \frac{K_c}{T_i}\cdot\frac{1}{s}\cdot E(s) + K_c T_d \cdot s \cdot E_f(s) \tag{11.3}$$

Here we need to calculate $E_f(s)$. It can be found by calculating the transfer function from $E(s)$ to $E_f(s)$ from (11.2). We get

$$E_f(s) = \frac{1}{t_{fd}s + 1}E(s) \tag{11.4}$$

Inserting $E_f(s)$ from (11.4) for $E_f(s)$ in (11.3) gives

$$\begin{aligned} U(s) &= K_c E(s) + \frac{K_c}{T_i} \cdot \frac{1}{s} \cdot E(s) + K_c T_d \cdot s \cdot E_f(s) && (11.5) \\ &= K_c E(s) + \frac{K_c}{T_i} \cdot \frac{1}{s} \cdot E(s) + K_c T_d \cdot s \cdot \frac{1}{T_{fd} s + 1} E(s) && (11.6) \\ &= \left(K_c + \frac{K_c}{T_i s} + \frac{K_c T_d s}{t_{fd} s + 1} \right) E(s) && (11.7) \end{aligned}$$

Thus, the transfer function of the PID controller is

$$C(s) = \frac{U(s)}{E(s)} = K_c + \frac{K_c}{T_i s} + \frac{K_c T_d s}{T_{fd} s + 1} \tag{11.8}$$

where

$$t_{fd} = a T_d \tag{11.9}$$

where typically

$$a = 0.1 \tag{11.10}$$

11.3 Practical aspects of the PID controller

11.3.1 Reverse or direct controller action?

11.3.1.1 What is meant by reverse action and direct action?

A PID controller shall have either reverse action mode or direct action mode. It is crucial that you select correctly between these two options. If set incorrectly, *the control system becomes unstable, totally useless, and perhaps even dangerous*, since the controller will adjust the control signal in the wrong direction.

By definition:

- Reverse action mode = Positive K_c
- Direct action mode = Negative K_c

What is behind the terms reverse action and direct action? Assume for simplicity a P controller:

$$u = u_{\text{man}} + K_c (r - y) \tag{11.11}$$

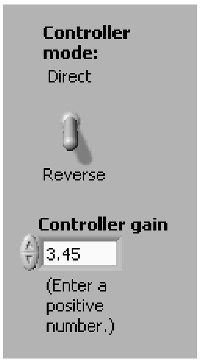
Assume there is a change, Δy, in y. The correesponsing change in the control signal is then (u_{man} and r are assumed constant):

$$\Delta u = -K_c \Delta y \tag{11.12}$$

Assume a positive Δy (i.e. the process variable increases). Then:

- If K_c is positive, Δu becomes negative, i.e. Δu and Δy has the opposite sign – hence the controller reacts "reversely".
- If K_c is negative, Δu becomes positive, i.e. Δu and Δy has the same sign – hence the controller reacts "directly".

Note that for industrial controllers it is common that the user always enters a positive value for the controller gain parameter (this is actually the absolute value of the controller gain). To set the sign of the resulting gain, the user must in addition select either reverse action or direct action using a dedicated parameter. This is illustrated in Figure 11.1.



Controller mode:
Direct
Reverse

Controller gain
3.45
(Enter a positive number.)

Figure 11.1: Setting the controller mode (reverse, direct) and the (absolute value of) the controller gain

You can practice with reverse and direct action modes with the following SimView simulator:

http://techteach.no/simview/pid_reverse_direct

11.3.1.2 How to select between reverse action and direct action modes?

Definition of process gain

Let us start by defining the term process gain, K_{process}. Figure 11.2 shows two processes to be controlled, Process 1 and Process 2. For both, the control signal is denoted u, and the process variable, or actually its measurement, is denoted y. The process blocks include the actuator and the sensor, in addition to the "process" itself (tank, motor, reactor). The figure shows the responses due to a *positive* step change, Δu, in the control signal. The sign of the change of y, denoted Δy, decides the sign of K_{process}:

- Assume $\Delta u > 0$. If $\Delta y > 0$, then $K_{\text{process}} > 0$, i.e. positive process gain.

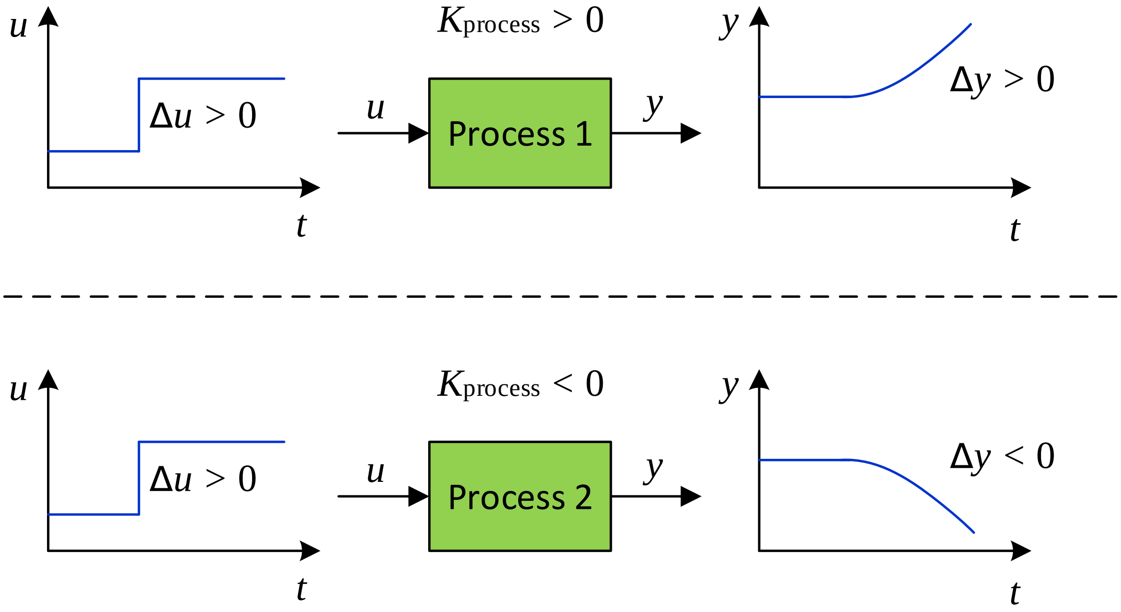


Figure 11.2: Step response test to determine the signal of the process gain, K_{process}, for two processes.

- Assume $\Delta u > 0$. If $\Delta y < 0$, then $K_{\text{process}} < 0$, i.e. negative process gain.

So, from a step response test on the process, we can conclude about the sign of K_{process}.

The step response test above does not have to be a real or simulated test. Probably it is sufficient that you imagine (simulate in your head) a step response test. Alternatively, you can conclude about the sign of K_{process} directly from a process model, e.g. a transfer function. Some examples:

- Positive K_{process}, namely 5:

$$H_p(s) = \frac{Y(s)}{U(s)} = \frac{5}{2s+1}$$

- Negative K_{process}, namely -5:

$$H_p(s) = \frac{-5}{2s+1}$$

- Positive K_{process}, in the form of the integrator gain, 10:

$$H_p(s) = \frac{10}{s}$$

- Negative K_{process}, in the form of the integrator gain, -10:

$$H_p(s) = \frac{-10}{s}$$

Rule for selecting between reverse action and direct action

To ensure that the control loop is stable, it is necessary that the signs of the controller gain, K_c, and the process gain, K_{process}, are the same. Thus, we can use the sign of K_{process} to select between reverse and direct action mode, as follows:

- For $K_{\text{process}} > 0$, select $K_c > 0$, i.e. reverse action mode.
- For $K_{\text{process}} < 0$, select $K_c < 0$, i.e. direct action mode.

Figure 11.3 illustrates the above rule.

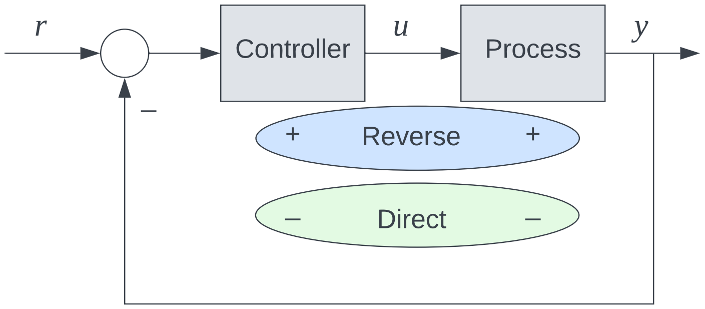


Figure 11.3: Illustration of rule for selecting between reverse and direct action of a PID controller.

Example 11.1 *Reverse or Direct action in the level controller?*

Figure 11.4 shows a level control system for a liquid tank where the control variable controls the outflow of the tank, and it is assumed that increasing control signal gives increasing outflow.

Shall the level controller, LC, have reverse or direct action mode? To answer the question, let's find the signal of K_{process}. Assume a positive step change in the control signal u, i.e. in the outflow through the pump. Does this change cause a positive or a negative change in the level y? Negative! Therefore, $K_{\text{process}} < 0$, and we conclude that shall have *direct action*.

What if the pump was in the inlet instead?[1]

[1] Then the controller shall have reverse action.

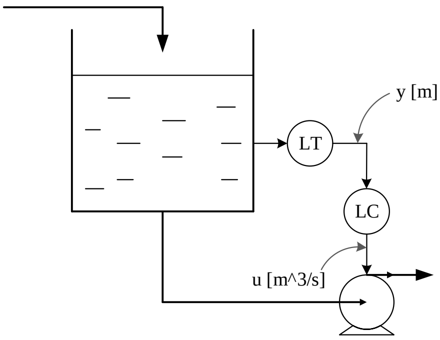


Figure 11.4: Tank with level manipulated by a pump in the outlet.

What if the pump in the outlet is replaced with a valve where the valve opening, and hence the flow, is reduced if the control signal is increased? (This could be a Fail Open valve, see Figure 2.4.)[2]

[End of Example 11.1]

11.3.2 Reducing P-kick and D-kick caused by reference changes

Abrupt changes of the reference r, for example step changes, creates an abrupt change of the control error, which in turn may cause unfortunate kicks in the control variable. The problem is related to the P term and, to an even higher degree, to the D term of the PID controller. These kicks are denoted proportional kick or P-kick and derivative kick or D-kick, respectively. Such kicks may cause mechanical actuators to move abruptly, resulting in excessive wear.

Why does these kicks come?

P-term kick

The P-term is:

$$u_p = K_p \left(w_p r - y_{\text{mf}} \right) \tag{11.13}$$

In the standard PID controller, $w_p = 1$. Assume that r changes as a step. Then there is also a step in u_p, causing a proportional kick, a P-kick, in the total control signal (in which u_p is one additive term).

D-term kick

The D-term is:

$$u_d = K_p T_d \left(w_d r - y \right)' \tag{11.14}$$

In the standard PID controller the weight w_d is 1. Assume that y_{sp} changes as a step. Since the time-derivative of a step is an impulse, the step causes an impulse in u_d – a D-kick – in the total control signal, u.

[2] Then the controller shall have reverse action.

How to solve the problems about P-kick and D-kick?

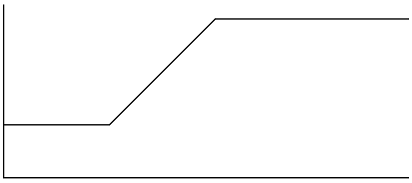


Figure 11.5: The change of the reference from one value to another may follow a ramp instead of a step to avoid kicks in the control signal.

Two solutions are:

- **Smooth reference changes:** Perhaps the best solution to this problem is to allow only smooth reference changes. Commercial controllers have functions to implement rampwise changes between two reference values. This is denoted ramping the reference, see Figure 11.5.
- **Reduce reference weights**: Another solution is to reduce the reference weights w_p and w_d. Regarding w_p, it is not so common to reduce it, but if it is, $w_p = 0.3$ is suggested Åstrøm & Hägglund (1995). Regardring w_d, it is actually quite common to set $w_d = 0$, causing the reference to be removed completely from the derivative term. In many commercial controllers $w_d = 0$ is a fixed factory setting.

Example 11.2 *Avoiding controller kicks during* reference *step change*

Figure 11.6 shows simulated responses of a PID control system with and without reduction of the reference weight in the P-term and the D-term, and with reference ramping. The reduced weights are $w_p = 0.3$ and $w_d = 0$. The PID parameters are $K_c = 3.6$, $T_i = 2.0$ s, $T_d = 0.5$ s. The simulations demonstrates that the control signal is smoother with reduced weights and with reference ramping.

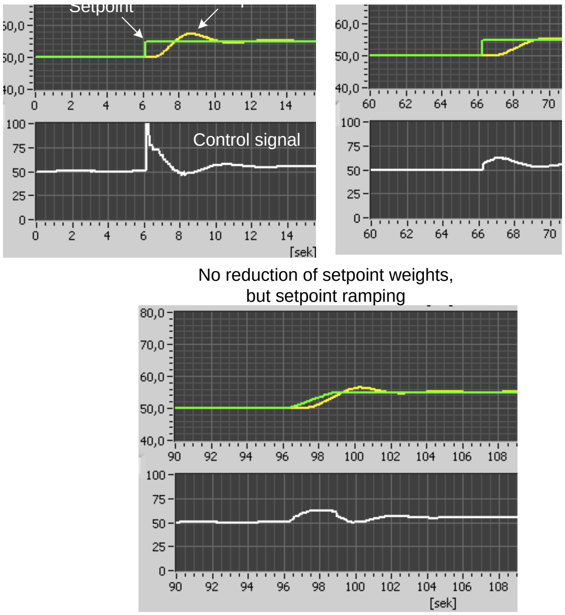
The simulations in this example are made with the following SimView simulator:

http://techteach.no/simview/control_kick

[End of Example 11.2]

11.3.3 Integrator anti wind-up

The problem



No reduction of setpoint weights, but setpoint ramping

Figure 11.6: Example 11.2: Responses in PID control system with and without reduction of reference weight and with reference ramping.

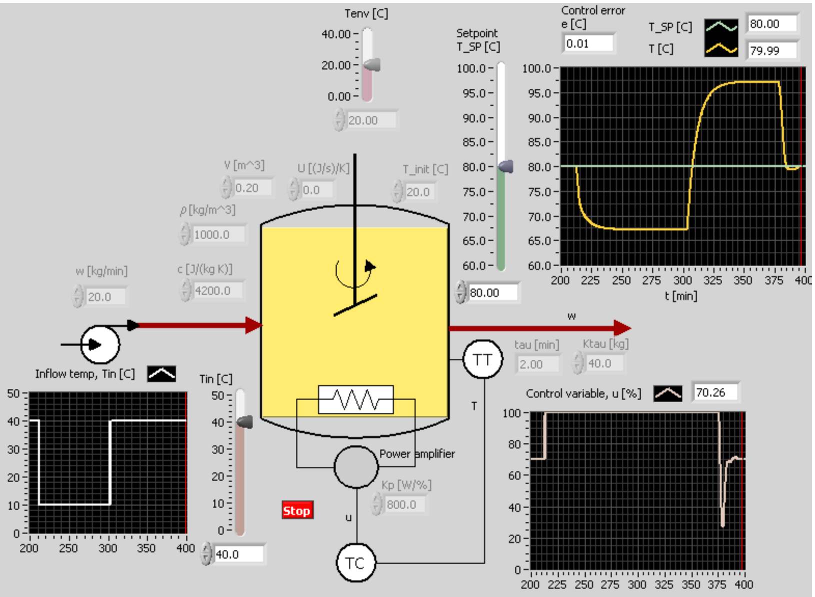
Wind-up is a problem related to the integral term of a PID controller – the I-term may get a large and continuously increasingly value – *it winds up* – if large disturbances or large references or sensor failure is causing the actuator to reach its saturation limits, i.e. a maximum limit and a minimum limit. Examples of such limits are: A power amplifier (for a heater or a motor) not deliver an infinitely amount of power; A valve can not have an infinitely large opening and can not be more closed than closed. Under normal process operation the control variable should not reach the saturation limits, but at extreme conditions, it can reach the limits.

Let's study an example.

Example 11.3 *Anti wind-up*

Figure 11.7 shows the front panel of a simulator for a temperature control system for a liquid tank with continuously mass flow. The responses shown in the figure are commented below. The disturbance is here the inlet temperature T_{in}, which is is changed as a step from 40 °C to 10 °C at approx. 210 min and back to 40 °C at approx. 300 min. The temperature setpoint r is 70 °C (constant). The controller has been tuned with the Ziegler-Nichols closed loop method. The maximum value of the control variable is 100 % and the minimum value is 0 %.

The PID controller used in the simulation shown in Figure 11.7 has no limitation on its integral term – hence it has no anti wind-up (later we will activate anti wind-up). In the simulation, T_{in} is reduced to 10 °C, causing the actuator (heating element) to go into saturation (100 %) trying to compensate for the (cold) disturbance. It can be shown that the control variable u should have a value of 122.5 % (which corresponds to more heat



Tenv [C] 20.00
V [m^3] 0.20
U [(J/s)/K] 0.0
T_init [C] 20.0
ρ [kg/m^3] 1000.0
w [kg/min] 20.0
c [J/(kg K)] 4200.0
Setpoint T_SP [C] 80.00
Control error e [C] 0.01
T_SP [C] 80.00
T [C] 79.99
tau [min] 2.00
Ktau [kg] 40.0
Control variable, u [%] 70.26
Kp [W/%] 800.0
Tin [C] 40.0

Figure 11.7: Temperature control without anti wind-up.

power than what is available) to be able to compensate for $T_{in} = 10$ °C. But because of the limitation of the heater capacity, the controller is not able to compensate fully for the disturbance, and the control error remains non-zero.

The non-zero error makes the integral term, u_i, increase, or: *wind up*. At $t = 300$ min, the disturbance was set back to the normal value of 40 °C. It was observed (but not shown in Figure 11.7), that the integral term at that time reached appox. 2200 %! Due to the very large I-term and the disturbance being back at normal ("hot") value, the total power delivered to the tanks is too large, causing the temperature to become too large. The control error has become negative, causing the I-term to decrease, which is good. But, it takes a long time until the I-term is again within a normal range, and in that time the temperature is about 17 degrees above the reference. Hence, there is a long-lasting large control error, which is a problem.

The solution

A practical PID controller must be able to cope with the possibility of integrator wind-up, that is, it must have some *integral anti wind-up* mechanism. Fortunately, you can assume that anti wind-up is implemented in commercial controllers.

The principle of an anti wind-up mechanism is simple: Since the problem is that the integral term increases continuously during actuator saturation, the solution is to halt the integration of the control error when the control signal reaches either its maximum or its minimum limit. Figure 11.8 illustrates the solution. The I term is represented with the tank, and anti wind-up is realized with a weir so that the integration of the liquid (the error) is halted at a certain maximum level.

How to realize anti wind-up in a discrete-time PID algorithm? Recall the I term of the PID

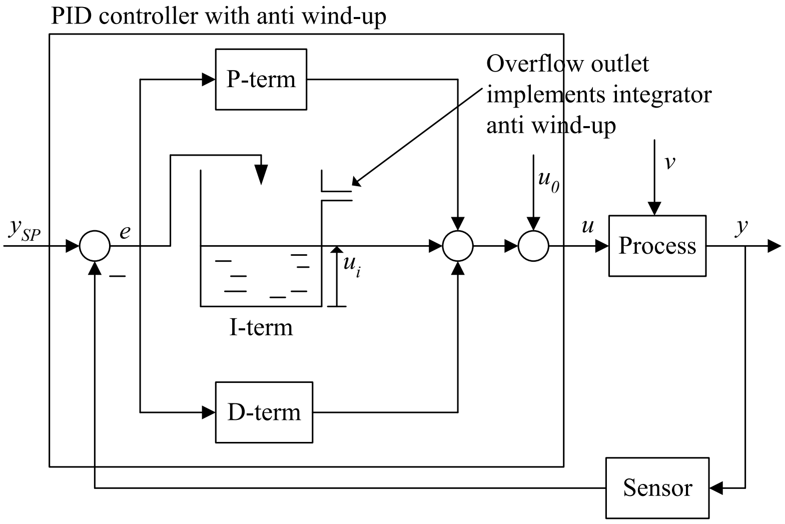


Figure 11.8: An analogy of integrator anti wind-up: A tank with weir to stop the integration at a certain level.

algorithm:

$$u_{i,k} = u_{i,k-1} + \frac{K_p T_s}{T_i} e_k \tag{11.15}$$

Anti wind-up can be realized by stopping the integration (or accumulation) of u_i when the following condition is met, namely that a (preliminary) calculation of u is greater than the maximum of u, typically 100%, or less than the minimum of u, typically 0%. In (11.15), the integration is stopped by forcing the term at the right to zero, so that the I term becomes

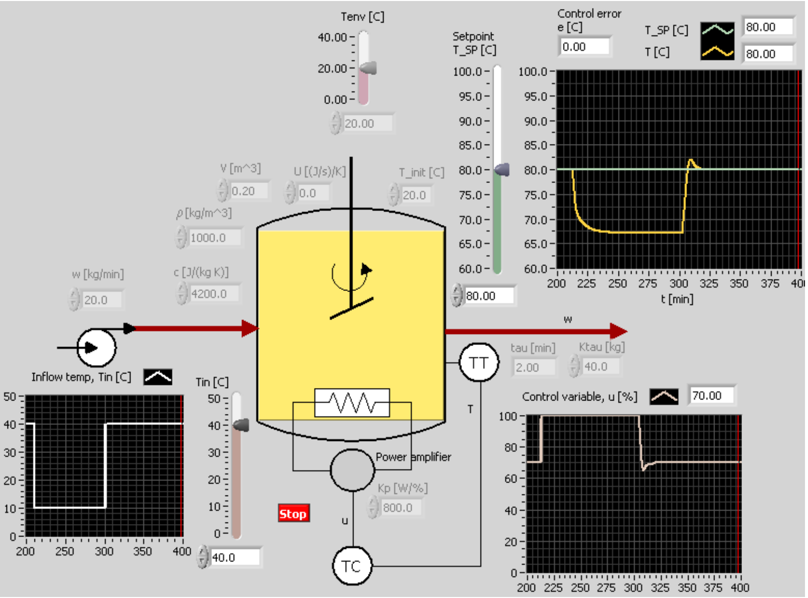
$$u_{i,k} = u_{i,k-1} + 0 \cdot e_k \tag{11.16}$$

Back to the temperature control system. Figure 11.9 shows the responses *with* integrator anti wind-up, which means that the updating of the I-term is stopped when the control signal is at its maximum (here 100 %) or minimum value (here 0 %). The simulations clearly show that it is beneficial to use integrator anti wind-up: The temperature returns much sooner to the reference after the disturbance has changed back to its normal value. This is because the I-term stayed at a reasonable value (in the simulation it was observed that the I-term had the value of a few %) despite the control error was large due to the limit of the heater.

The simulations in this example are made with the following SimView simulator:

http://techteach.no/simview/antiwindup

[End of Example 11.3]



Tenv [C] 20.00

Setpoint T_SP [C] 80.00

Control error e [C] 0.00

T_SP [C] 80.00

T [C] 80.00

V [m^3] 0.20

U [(J/s)/K] 0.0

T_init [C] 20.0

ρ [kg/m^3] 1000.0

w [kg/min] 20.0

c [J/(kg K)] 4200.0

tau [min] 2.00

Ktau [kg] 40.0

Tin [C] 40.0

Kp [W/%] 800.0

Control variable, u [%] 70.00

Figure 11.9: Temperature control with anti wind-up.

11.3.4 Bumpless transfer between manual and auto modes

It is important that the control signal does not jump (too much) when the controller is switched from automatic to manual mode, or from manual to automatic mode. In other words, the transfer between the automatic and manual modes should be *bumpless*. Bumpless transfer is implemented in commercial controllers.

We can implement bumpless transfer in a PID algorithm as follows:

- **Bumpless transfer from manual to automatic mode**: Set $u_{i,k-1} = 0$ to empty the I term just before switching between the modes.
- **Bumpless transfer from automatic to manual mode**: Set $u_{\text{man}} = u$, and halt the I term by setting $u_{i,k} = u_{i,k-1}$ during manual mode.

You can practice bumpless transfer between manual and automatic modes with the following SimView simulator:

http://techteach.no/simview/bumpless_transfer

11.4 *Problems for Chapter 11*

Problem 11.1 *Transfer function of PI controller*

What is the transfer function of a PI controller?

Problem 11.2 *Reverse or direct action?*

Figure 11.10 shows a pressure control system. Assume that increasing the control signal to the valve increases the valve opening. Will you set the controller to have reverse action or direct action?

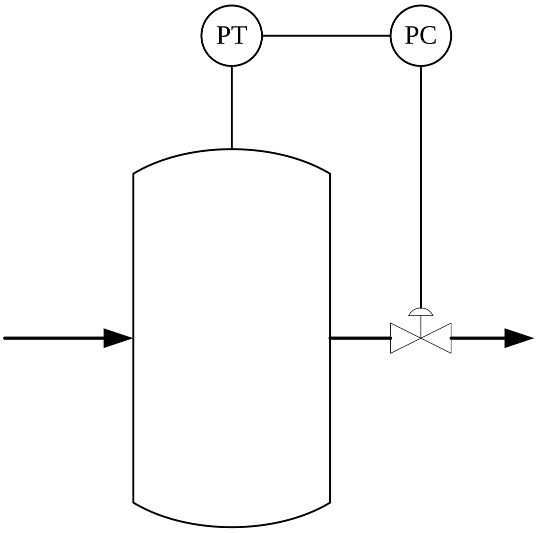


Figure 11.10: Pressure control system

Problem 11.3 *Derivative kick*

A derivative term – or D-term – with the possibility of reference weight reduction is

$$u_d = K_c T_d \left(w_d r - y\right)' \tag{11.17}$$

(It is here, for simplicity, assumed that there is no lowpass filter acting on the D-term.) Assume that there is no reduced weight of the reference, i.e.

$$w_d = 1 \tag{11.18}$$

Assume that the process measurement y is constant, and that the reference r is changed as a step at time $t = 0$. Describe (qualitatively) the corresponding response in the control signal u_d due to this step. What kind of signal is this response?

Problem 11.4 *Anti wind-up*

One appliaction where it is particularly important with anti wind-up is limiting control. Figure 11.11 shows a gas tank with one inlet and two outlets. The purpose of the normal control loop is to keep the gas pressure at the normal reference (or setpoint) SP_1, say 2 bar. The purpose of the limiting control loop is to limit the pressure to the higher reference SP_2, say 4 bar. What is the control error of controller PC_2 under normal conditions? Why is it particularly important that this controller has anti wind-up?

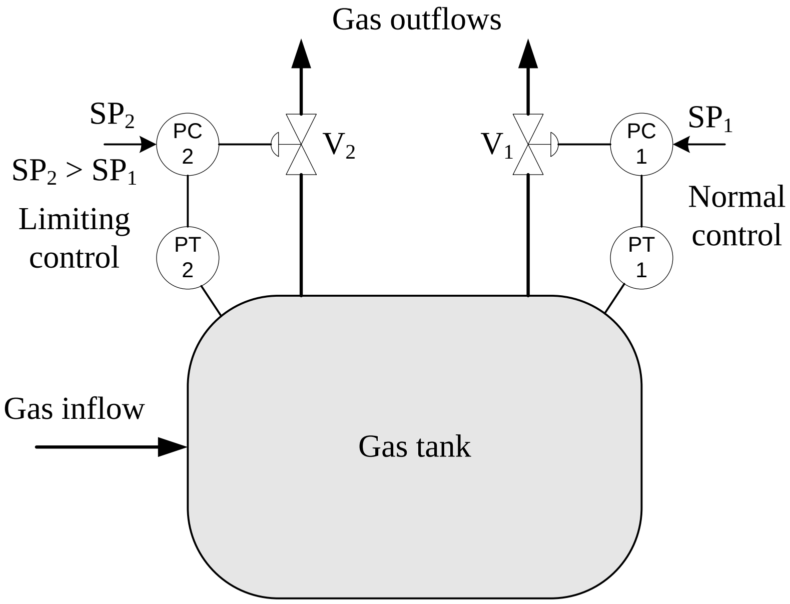


Figure 11.11: Limiting control

11.5 *Solutions to problems for Chapter 11*

Solution to Problem 11.1

By setting $T_d = 0$ in (11.19), we get the transfer function of a PI controller:

$$C(s) = \frac{u(s)}{e(s)} = K_c + \frac{K_c}{T_i s} = \frac{K_c(T_i s + 1)}{T_i s} \tag{11.19}$$

Solution to Problem 11.2

Assume that for some reason the presssure is larger than the pressure reference. The controller must react to this by ensuring that the valve opening is increase, which means that the valve control signal is increased. Therefore, the controller must have directe action.

Solution to Problem 11.3

The control signal is

$$u_d = K_c T_d \frac{d\,(\text{Setpoint step} - \text{constant})}{dt} \tag{11.20}$$

The time-derivative of a reference step is an *impulse*, which is a signal if infinite amplitude and with infinite duration, see Figure 11.12. So, the stepwise change of the reference causes an impulse-like change of the control signal. That is the *derivative kick*.

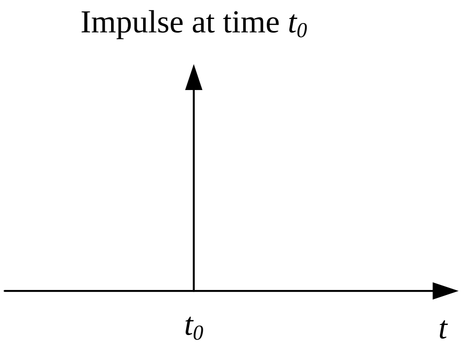


Figure 11.12: Impulse.

Solution to Problem 11.4

Under normal conditions the control error of PC_2 is

$$e_2 = \text{SP}_1 - \text{SP}_2 = 4 - 2 \text{ bar} = 2 \text{ bar} \tag{11.21}$$

This sustained non-zero control error would have caused the integral term of controller PC_2 to wind up – or, actually, to wind down to a very large negative value, making the controller virtually inactive if the pressure rises and comes close to SP_2, which makes the controller useless for limiting control.

Chapter 12

Transfer functions of feedback control systems

12.1 Introduction

Transfer functions of feedback control systems can be useful for:

- Analytical calculation of responses, as shown in the present chapter.
- Simulation, cf. Section 13.3.
- Frequency response analysis, cf. Chapter 22.

12.2 Definition of reference tracking and disturbance compensation

Figure 12.1 shows a principal block diagram of a control system.

There are two input signals to the control system, namely the reference r and the disturbance d. The value of the control error e is our primary concern (it should be small, preferably zero). Therefore we can say that e is the (main) output variable of the control system. The value of e expresses the *performance* of the control system: The less e, the higher performance. e is influenced by r and d. Let us therefore define the following two properties of control systems:

- The **reference tracking property** of the control system concerns the relation between r and e.
- The **disturbance compensation property** of the control system concerns the relation between d and e.

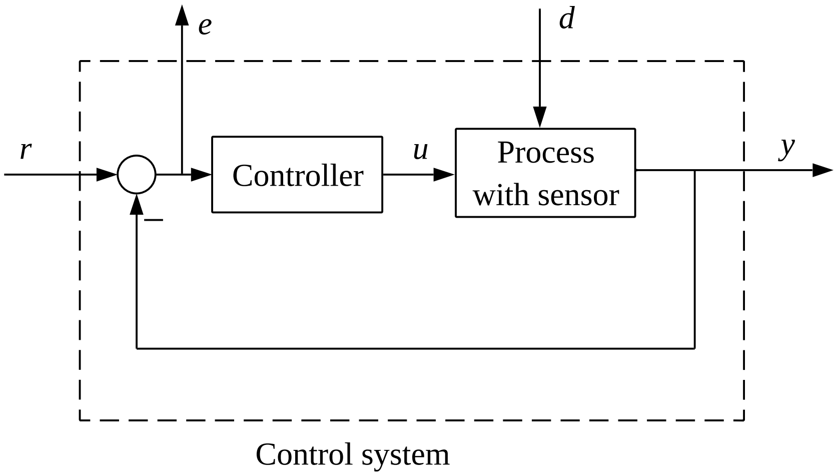


Figure 12.1: Principal block diagram of a control system.

Totally, the reference tracking and disturbance compensation properties express the performance of the control system.

12.3 Sensitivity transfer function

12.3.1 Definition of Sensitivity transfer function

We assume that the control system has a transfer function-based block diagram as shown in Figure 12.2.

We regard the reference $R(s)$ and the disturbance $D(s)$ as input variables and the control error $E(s)$ as the output variable of the system. Thus, we will derive the transfer function from $R(s)$ to $E(s)$ and the transfer function from $D(s)$ to $E(s)$.

Below, we use the *loop transfer function*. It is defined as the product of the transfer functions in the loop, see Figure 12.2:

$$H_l(s) = H_p(s)H_c(s) \tag{12.1}$$

From the block diagram in Figure 12.2 we the can write the following expressions for $E(s)$:

$$\begin{aligned} E(s) &= R(s) - Y(s) & (12.2)\\ &= R(s) - [H_d(s)D(s) + H_p(s)U(s)] & (12.3)\\ &= R(s) - [H_d(s)D(s) + H_pC(s)E(s)] & (12.4) \end{aligned}$$

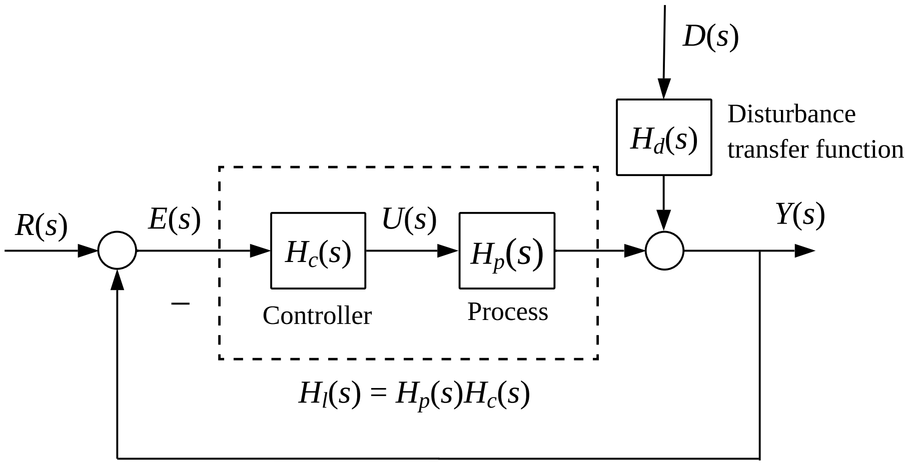


Figure 12.2: Transfer function-based block diagram of a control system.

Solving (12.4) for $E(s)$ gives:

$$E(s) = \left[\frac{1}{1+H_p(s)H_c(s)}\right] R(s) + \left[-\frac{1}{1+H_p(s)H_c(s)}H_d(s)\right] D(s) \tag{12.5}$$

or

$$E(s) = \left[\frac{1}{1+H_l(s)}\right] R(s) + \left[-\frac{1}{1+H_l(s)}H_d(s)\right] D(s) \tag{12.6}$$

The *sensitivity transfer function*: can be defined as the transfer funtion from $R(s)$ to $E(s)$:

$$H_s(s) = \frac{1}{1+H_l(s)} \tag{12.7}$$

This definition makes sense as it tells how sensitive the error is with respect to the reference.

Now, (12.6) can be written:

$$E(s) = \underbrace{H_s(s) \cdot R(s)}_{E_r(s)} + \underbrace{[-H_s(s)H_d(s)] \cdot D(s)}_{E_d(s)} \tag{12.8}$$

where:

$$E_r(s) = H_s(s) \cdot R(s) \tag{12.9}$$

and

$$E_d(s) = [-H_s(s)H_d(s)] \cdot D(s) \tag{12.10}$$

12.3.2 Calculation of response in control error

From (12.8) we can calculate the control error for any reference and any disturbance signal (assuming we know their Laplace transforms).

In the following we discuss $E_r(s)$ and $E_d(s)$.

12.3.2.1 Response in error due to reference

The response in the control error due to the *reference* is

$$E_r(s) = H_s(s)R(s) = \frac{1}{1+H_l(s)}R(s) \tag{12.11}$$

which gives a quantitative expression of the *tracking property* of the control system.

The *static* tracking property is given by static error when $r(t) = r_s$ is constant (static), regarded as a step signal of amplitude r_s starting for time $t = 0$. The Laplace transform of $r(t)$ is, cf. (44.7),

$$R(s) = \mathcal{L}\{r(t)\} = \frac{r_s}{s} \tag{12.12}$$

This error can be calculated as follows.[1]

$$\begin{aligned}
e_{r,s} &= \lim_{t\to\infty} e_r(t) && (12.13)\\
&= \lim_{s\to 0} s \cdot E_r(s) && (12.14)\\
&= \lim_{s\to 0} s \cdot H_s(s)R(s) && (12.15)\\
&= \lim_{s\to 0} s \cdot H_s(s)\frac{r_s}{s} && (12.16)\\
&= H_s(0)r_s && (12.17)
\end{aligned}$$

From (12.17) we see that *the reference tracking property of the control system are good if the sensitivity transfer function H_s has small (absolute) value – ideally zero.*

12.3.2.2 Response in error due to disturbance

The response in the control error due to the disturbance is

$$E_d(s) = -H_s(s)H_d(s)D(s) \tag{12.18}$$

which expresses the *disturbance compensation property* of the control system.

The *static* disturbance compensation property is given by *static* static error when $d(t) = d_s$ is constant (static), regarded as a step signal of amplitude d_s starting for time $t = 0$. The Laplace transform of $d(t)$ is

$$D(s) = \mathcal{L}\{d(t)\} = \frac{d_s}{s} \tag{12.19}$$

[1] The Final Value Theorem of the Laplace transform is used here.

The *static* disturbance compensation property is given by

$$e_{d_s} = \lim_{t\to\infty} e_d(t) = \lim_{s\to 0} s \cdot E_d(s) \tag{12.20}$$

$$= \lim_{s\to 0} s \cdot [-H_s(s)H_d(s)D(s)] \tag{12.21}$$

$$= \lim_{s\to 0} s \cdot \left[-H_s(s)H_d(s)\frac{d_s}{s}\right] \tag{12.22}$$

$$= -H_s(0)H_d(0)d_s \tag{12.23}$$

From (12.23) we see that the *disturbance compensation property is good if the sensitivity transfer function H_s has a small (absolute) value (close to zero).*

12.4 Tracking transfer function

12.4.1 Definition of Tracking transfer function

The *tracking transfer function* $H_t(s)$ is the transfer function from the reference $R(s)$ to the process output $Y(s)$:

$$Y(s) = H_t(s)R(s) \tag{12.24}$$

Let's find $H_t(s)$. From the block diagram in Figure 12.2:

$$Y(s) = H_p(s)U(s) = H_p(s)H_c(s)E(s) = H_p(s)H_c(s)\left[R(s) - Y(s)\right] \tag{12.25}$$

Solving for the process output $Y(s)$:

$$Y(s) = \frac{H_p(s)H_c(s)}{1 + H_p(s)H_c(s)}R(s) \tag{12.26}$$

So, the tracking transfer function is

$$H_t(s) = \frac{Y(s)}{R(s)} = \frac{H_p(s)H_c(s)}{1 + H_p(s)H_c(s)} = \frac{H_l(s)}{1 + H_l(s)} \tag{12.27}$$

$H_t(s)$ and $H_s(s)$ are actually related, as

$$H_t(s) = 1 - H_s(s) \ \text{ or } \ H_t(s) + H_s(s) = 1 \tag{12.28}$$

12.4.2 Calculation of response in control error

The *static* tracking property is given by the static tracking transfer function $T(0)$:

$$y_s = \lim_{t\to\infty} y(t) \tag{12.29}$$

$$= \lim_{s\to 0} s \cdot Y(s) \tag{12.30}$$

$$= \lim_{s\to 0} s \cdot H_t(s)R(s) \tag{12.31}$$

$$= \lim_{s\to 0} s \cdot H_t(s)\frac{r_s}{s} \tag{12.32}$$

$$= H_t(0)r_s \tag{12.33}$$

The *tracking property is good if the tracking transfer function H_t has (absolute) value equal to or close to 1* (since then y will be equal to or close to r).

12.5 *Problems for Chapter 12*

Problem 12.1 *Transfer function of a feedback control system*

Given a process with the following transfer function from control signal to measurement signal:

$$H_p(s) = \frac{K_i}{s} \tag{12.34}$$

1. Characterize this process in terms of its dynamics (just a single term is expected in your answer).
2. Calculate the gain K_c of a P controller for the process so that the control system gets a (closed-loop) time constant of t_{cc} [s].
3. What is the pole of the control system?
4. Assuming K_i is negative, does the controller have reverse action or direct action?

Problem 12.2 *Proving that the static tracking property is perfect*

Assume given a “time constant” process with transfer function

$$H_p(s) = \frac{K}{t_c s + 1} \tag{12.35}$$

The process is controlled with a PI controller having transfer function

$$H_c(s) = K_c + \frac{K_c}{T_i s} = K_c \frac{T_i s + 1}{T_i s} \tag{12.36}$$

You can assume that the controller is tuned so that the control system has ok stability.

Assume that the reference is constant (static), r_s. Prove that the static control error is zero, i.e. perfect tracking.

12.6 *Solutions to problems for Chapter 12*

Solution to Problem 12.1

1. Process dynamics: Integrator.
2. Tuning the P controller: Controller transfer function is

$$H_c(s) = K_c \tag{12.37}$$

Loop transfer function:

$$H_l(s) = H_p(s)H_c(s) = \frac{K_i}{s}K_c \tag{12.38}$$

Tracking transfer function:

$$H_t(s) = \frac{H_l(s)}{1 + H_t(s)} = \frac{K_cK_i}{s + K_cK_i} = \frac{1}{\frac{1}{K_cK_i}s + 1} = \frac{1}{t_{cc}s + 1} \tag{12.39}$$

The closed loop time constant is

$$t_{cc} = \frac{1}{K_cK_i} \tag{12.40}$$

Solving for K_c gives

$$K_c = \frac{1}{t_{cc}K_i} \tag{12.41}$$

3. The pole of control system is the root of the characteristic equation whih is:

$$s + K_cK_i = 0 \tag{12.42}$$

Hence, pole becomes

$$p = -K_cK_i = -\frac{1}{t_{cc}} \tag{12.43}$$

4. Assuming K is negative, K_c becomes negative (since t_{cc} must be positive to have a stable control system). A negative K_c means that the controller has direct action.

Solution to Problem 12.2

Let's use (12.16):

$$e_{r,s} = \lim_{s\to 0} s \cdot H_s(s)R(s) \tag{12.44}$$

where

$$H_s(s) = \frac{1}{1 + H_l(s)} = \frac{1}{1 + H_p(s)H_c(s)} = \frac{1}{1 + \frac{K}{t_cs+1} \cdot K_c\frac{T_is+1}{T_is}} = \frac{T_is(t_cs + 1)}{T_is(t_cs + 1) + KK_c(T_is + 1)} \tag{12.45}$$

and

$$R(s) = \frac{r_s}{s} \tag{12.46}$$

Now, (12.44) becomes

$$e_{r,s} = \lim_{s\to 0} s \cdot H_s(s)R(s) = \lim_{s\to 0} s \cdot \frac{T_is(t_cs + 1)}{T_is(t_cs + 1) + KK_c(T_is + 1)} \frac{r_s}{s} = 0 \tag{12.47}$$

So, it is proven that the static control error is zero, i.e. perfect tracking.

Chapter 13

Simulation of PID control systems

13.1 Introduction

This chapter demonstrates three different ways of simulating PID control systems:

- Simulation with elementary code in Python
- Simulation of transfer functions with Python Control package
- Simulation of block-diagrams with OpenModelica

The same control system is simulated in all three cases, the level control system of a wood chips tank, see Figure 13.1.

The tank is presented in Appendix 39.2. The mathematical process model presented in the appendix is repeated here for convenience:

$$\rho A h'(t) \quad = \quad u(t - t_d) - F_{\text{out}}(t) \tag{13.1}$$

h [m] is the wood chips level, u [kg/s] is the control signal which generates a flow rate of chips via the feed screw, F_{out} [kg/s] is the flow rate of the outflow of the tank, $\rho = 145$ kg/m^3 is wood chips density, $A = 13.4$ m^2 is the cross-sectional area of the tank, and $t_d = 250$ s is the transportation time or time delay on the conveyor belt.

The PI level controller has settings according to the SIMC (Skogestad 2003) PI tuning method for "integrator with time delay" processes (cf. Example 14.6):

$$K_c = 3.89 \text{ [\%/m]} \tag{13.2}$$

$$T_i = 1000 \text{ s} \tag{13.3}$$

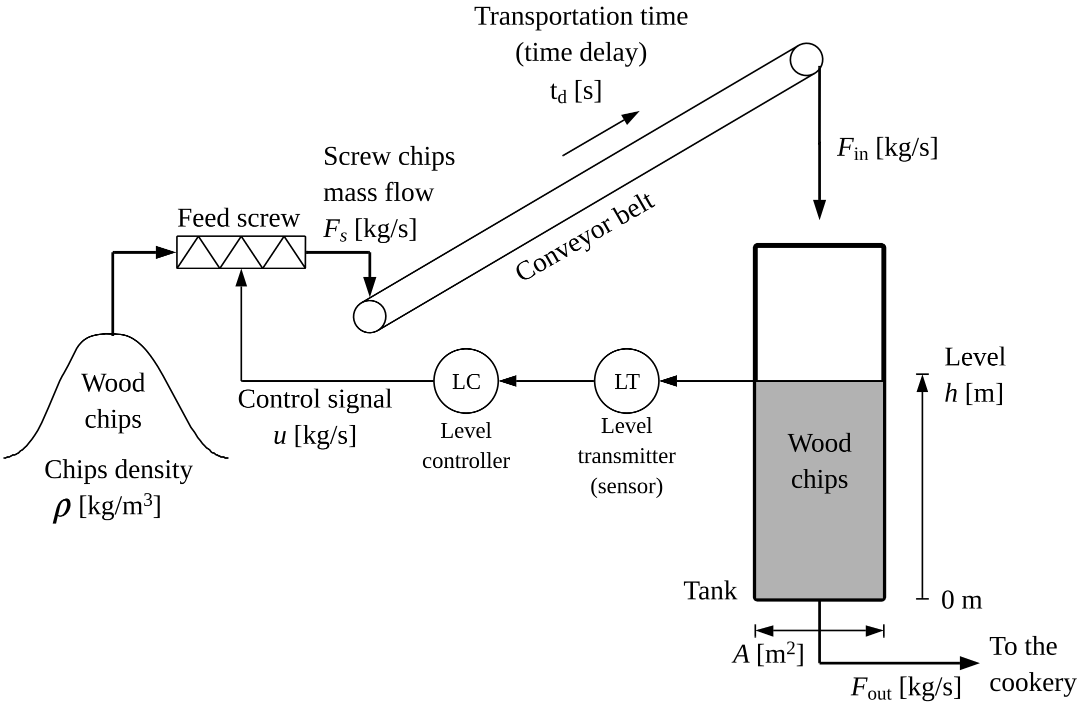


Figure 13.1: Level control system of a wood chips tank.

13.2 Simulation with elementary code in Python

The basis of the simulation algorithm is s state space mode form of the process model (13.1). We get the state space model by dividing by ρA:

$$h'(t) \quad = \quad [u(t - t_d) - F_{\text{out}}(t)] \, / (\rho A) \tag{13.4}$$

The PI controller algorithm is presented in Section 1.3.4.3.

The simulator will calculate the level response due to a step change of the level reference r at time $t = 1000$ s and a step change of the outflow F_{out} at time $t = 6000$ s.

The Python program implementing the simulator is available from the following link:

http://techteach.no/control/python/sim_pi_control_chiptank.py

Comments to the Python program:

- The PI controller is coded in the function fun_pi_con().

- The process simulator based on the Euler simulation algorithm is coded in the function fun_process_sim.
- The above two functions are invoked in the For loop of the simulator.

Figure 13.2 shows the simulated level response.

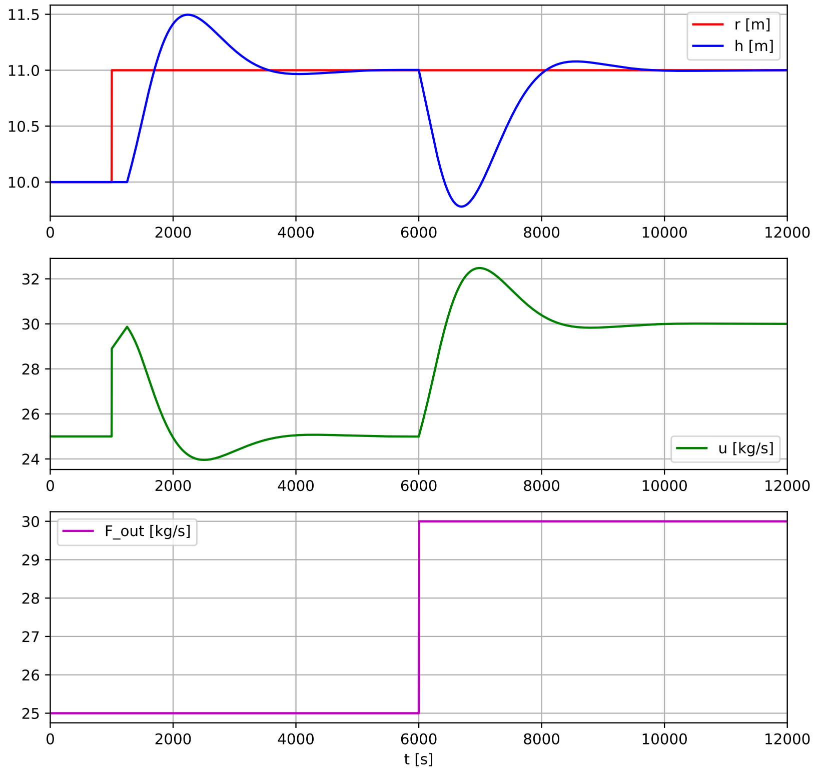


Figure 13.2: Simulation of level control system of wood chips tank with elementary Python code.

13.3 Simulation of transfer function models with Python Control package

We start by drawing a block diagram of the control system using transfer functions blocks, see Figure 13.3.

The transfer function $H_c(s)$ of the PI controller is developed in Section 11.2, but is repeated

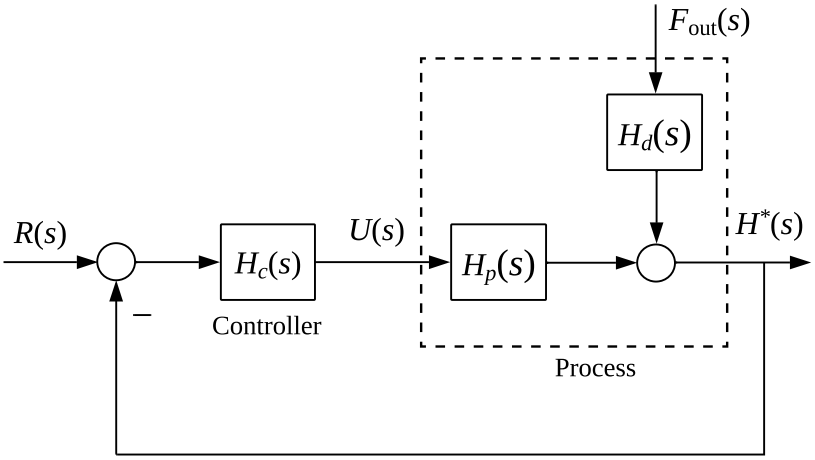


Figure 13.3: Block diagram of the temperature control system with transfer functions.

here for convenience:

$$H_c(s) = K_c + \frac{K_c}{T_i s} \tag{13.5}$$

We derive the process transfer functions $H_p(s)$ and $H_d(s)$ from the model (13.1): Taking the Laplace transform, the model becomes

$$\rho A s H^*(s) \quad = \quad e^{-t_d s}U(s) - F^*_{\text{out}}(s) \tag{13.6}$$

where I have used the star symbol in $H^*(s)$ to make it clear that it is not a transfer function[1] but the Laplace transformed $h(t)$, i.e. $H^*(s) = \mathcal{L}\{h(t)\}$. From (13.6), we get

$$H^*(s) = \underbrace{\frac{1}{\rho A s} e^{-t_d s}}_{H_p(s)} U(s) + \underbrace{\left(-\frac{1}{\rho A s}\right)}_{H_d(s)} F^*_{\text{out}}(s) \tag{13.7}$$

In (13.7), I will assume the disturbance is zero and therefore neglect $H_d(s)$, since in applications where we use transfer function models, it is quite common to do such a simplification of the model.

The Python program implementing the simulator using the Python Control package is available from the following link:

http://techteach.no/control/python/sim_pi_level_control_woodchip_tank_pycon.py

Figure 13.4 shows the simulated level response due to a step change of the level reference.

[1] I have used H as the general symbol for a transfer function in the book.

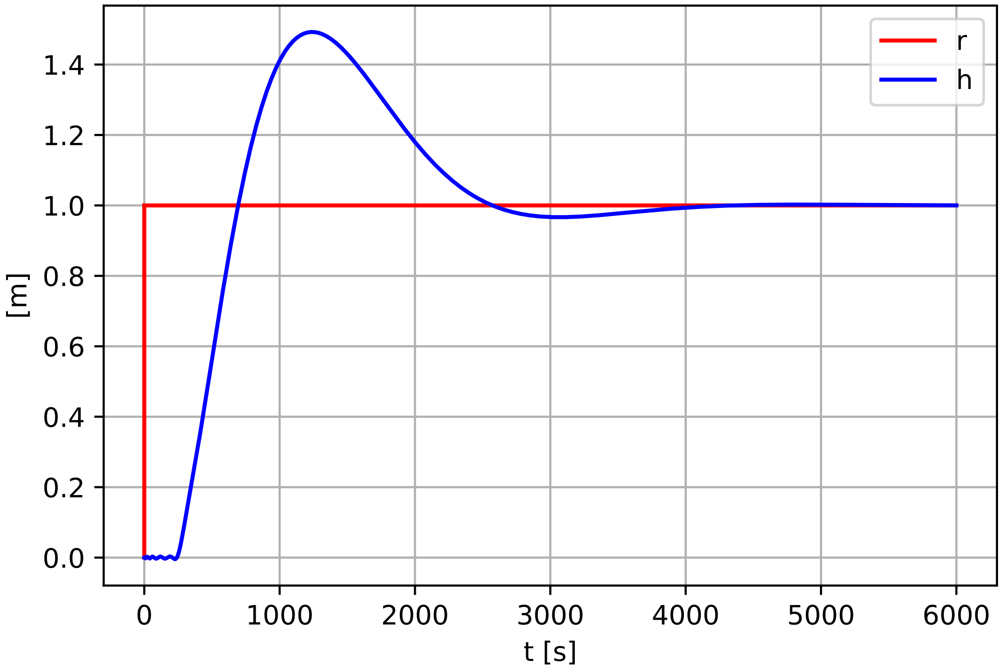


Figure 13.4: Simulation of level control system of wood chips tank with Python Control package.

Comments to the simulation:

- The small oscillations in the beginning of the response are due to the Padé approximation of the time delay.
- The control.forced_response function of the Python Control package has been used to run the simulation.

13.4 Simulation of block diagram model with OpenModelica

Firstly, we draw a detailed mathematical block diagram of the control system, as a preparation for implementation in OpenModelica, see Figure 13.5.

Secondly, we implement and run a simulator of the control system in OpenModelica. We can use a Constant-block to set the manual control signal $u_{\rm man}$ to 25 kg/s, and a LimPID block (PID with limitations) to set the limits of the controller output signal to minimum -25 and maximum 25 kg/s, which, when combined with the $u_{\rm man}$ block, implements limitation of the control signal to the range 0-50 kg/s.

Figure 13.6 shows the block diagram model of the level control system as built in OpenModelica.

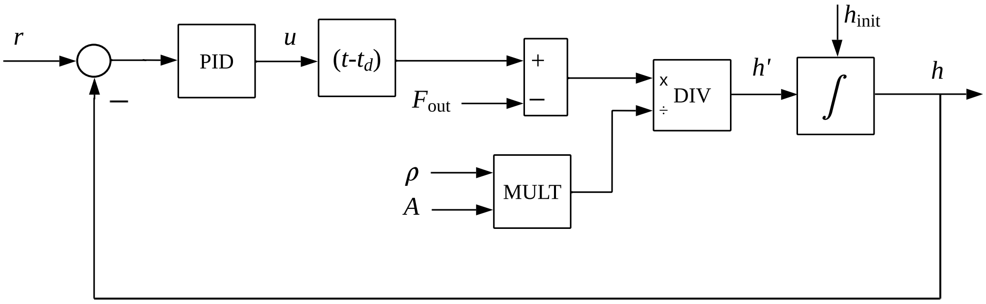


Figure 13.5: Block diagram of the level control system.

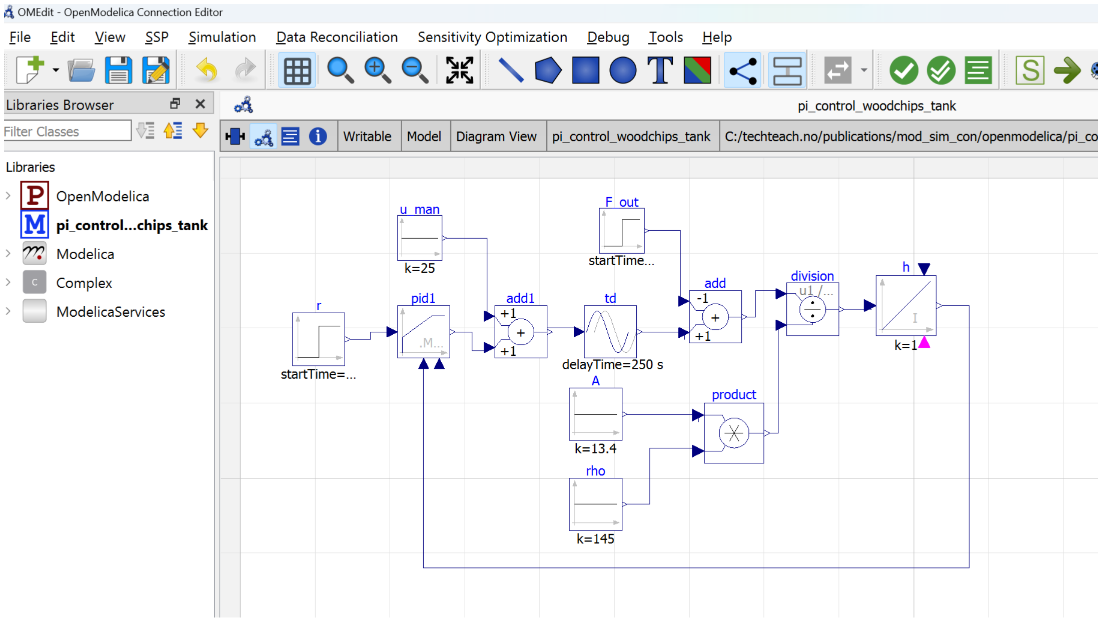


Figure 13.6: Block diagram model of the level control system as built in OpenModelica.

The OpenModelica simulator is available from the following link:

http://techteach.no/control/openmodelica/pi_control_woodchips_tank.mo

Figure 13.7 shows the simulated level response due to a step change of the level reference and a step change of the outflow.

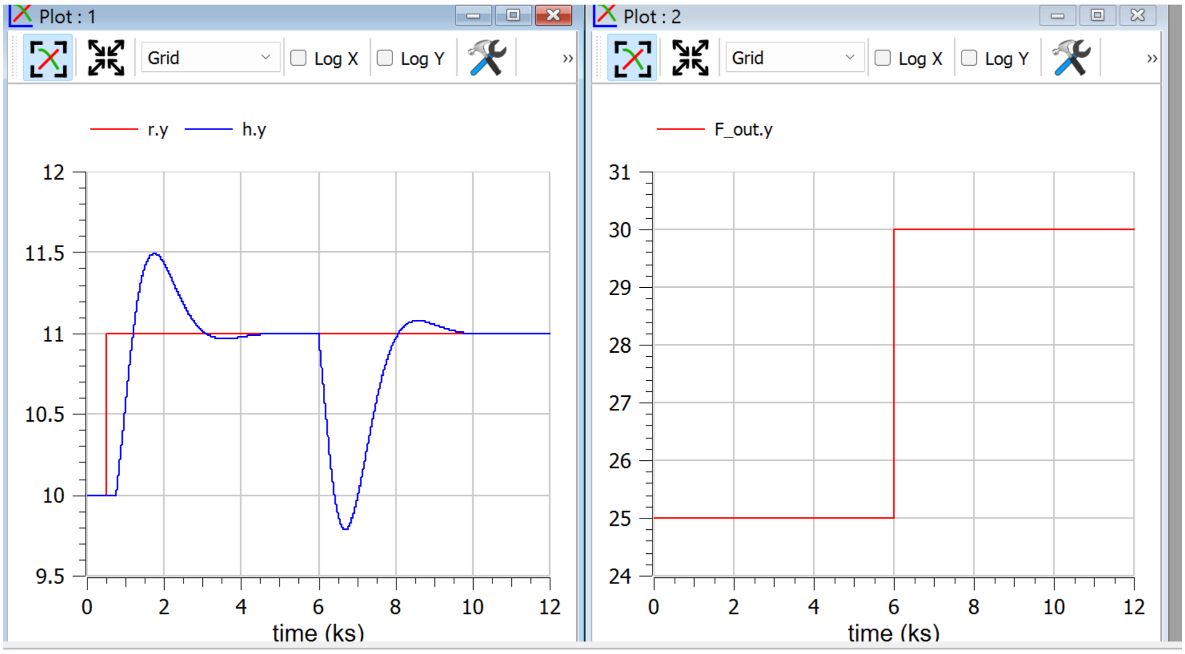


Figure 13.7: Simulation of level control system of wood chips tank with OpenModelica.

13.5 *Problems for Chapter 13*

Information for Problems 13.1, 13.2, and 13.3.

Each of the problems is about the temperature control system of the kettle which is presented in Appendix 39.6. Figure 13.8 shows a process & instrumentation diagram (P&I D) of the control system.

A process model based on energy balance of the water in the kettle is presented in Appendix 39.6, but repeated here for convenience:

$$CT'(t) = [P(t - t_d) + G\,(T_a - T(t))] \tag{13.8}$$

T [°C] is water temperature. T_a [°C] is ambient (room) temperature. P [W] is electric power supplied to the heater. $C = 2101$ J/K is heat capacity of the water. $G = 2.34$ W/K is thermal conductivity of the plastic jacket. t [s] is time. $t_d = 20$ s is time delay (so, the electric power is time delayed before it affects the water temperature).

In each of the problems, you can use a PI controller with the following PI settings:

$$K_c = 26.3\ [\%/\text{m}] \tag{13.9}$$

$$T_i = 160\ \text{s} \tag{13.10}$$

These settings are based on the SIMC (Skogestad 2003) PI settings for "time constant with time delay", cf. Section 14.8.4.2. The gain and the time constant of the process are presented in the solution to Problem 13.2. In the SIMC PI settings, I have set the closed

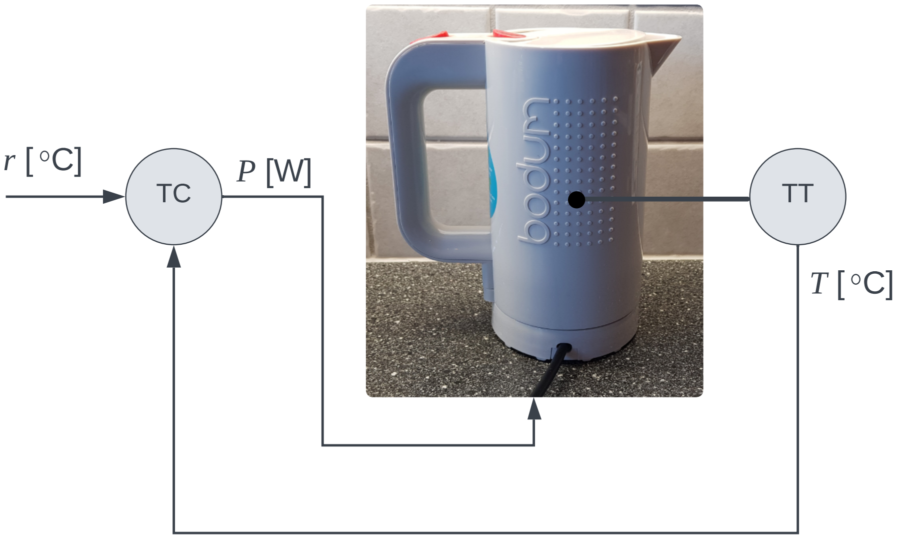


Figure 13.8: Temperature control system of the kettle.

loop time constant to $t_{cc} = 60$ sec (found by trial-and-error in simulations, to avoid too aggressive control signal).

In each of the problems, you should use the following information:

- The initial temperature is 20 °C.
- The ambient (room) temperature is 20 °C (constant).
- Temperature reference is changed as a step from 20 to 30 degrees at point of time which you can select yourself.
- The reference and the temperature should be plotted in the same figure. Also the control signal (supplied power) should be plotted, in a separate figure, but you may drop this plot in Problem 13.2.
- The simulation should start at $t = 0$, and stop at $t = 1000$ s.
- Time step can be set to $t_s = 1$ s.

Problem 13.1 *Simulator of temperature control system of kettle using elementary Python code*

Implement and run a simulator of the control system using elementary Python code. You may start from scratch, or base your solution on the program in Section 13.2.

Problem 13.2 *Simulator of temperature control of kettle using transfer function models in Python Control Package*

Implement a simulator of the control system using transfer functions in Python Control package. It is ok if you implement only the process transfer function $H_p(s)$ from control signal $P^*(s)$ (supplied power) to temperature $T^*(s)$ in the simulator, similar to what I did in Section 13.3. You can use control.forced_response function to run the simulation. You may start from scratch, or base your solution on the program in Section 13.3.

Hints for solving the problem:

Firstly, draw (by hand is ok) a block diagram of the control system using transfer function blocks. You can neglect the impact of the ambient temperature in this problem.

Secondly, write the PI controller transfer function $H_c(s)$ and process transfer functions $H_p(s)$ and $H_d(s)$. However, you can drop implementing $H_p(s)$, cf. my comment above.

Finally, program and run the simulator.

Problem 13.3 *Simulator of a PI control system using OpenModelica*

Firstly, draw (by hand is ok) a detailed mathematical block diagram of the control system, as a preparation for implementation in OpenModelica. It is ok if you represent the PI controller with just one block. Hint: A block diagram of just the kettle (no controller) was developed in Example 5.1.

Secondly, implement and run a simulator of the control system in OpenModelica. You should use the LimPID block (PID with limitations) to set the limits of the controller output signal to minimum 0 and maximum 700 W.

13.6 *Solutions to problems for Chapter 13*

Solution to Problem 13.1

The basis of the simulation algorithm is a state space mode form of the process model (13.8). We get the state space model by dividing by C, to get

$$T'(t) = \{[P(t-t_d) + G(T_a - T(t))]\}/C \tag{13.11}$$

The PI controller algorithm is presented in Section 1.3.4.3.

The Python program implementing the simulator is available from the following link:

http://techteach.no/control/python/sim_pi_control_kettle_elem_code.py

Comments to the Python program:

- The PI controller is coded in the function fun_pi_con().
- The process simulator based on the Euler simulation algorithm is coded in the function fun_process_sim.
- The above two functions are invoked in the For loop of the simulator.

Figure 13.9 shows the simulated temperature response due to a step change of the reference r.

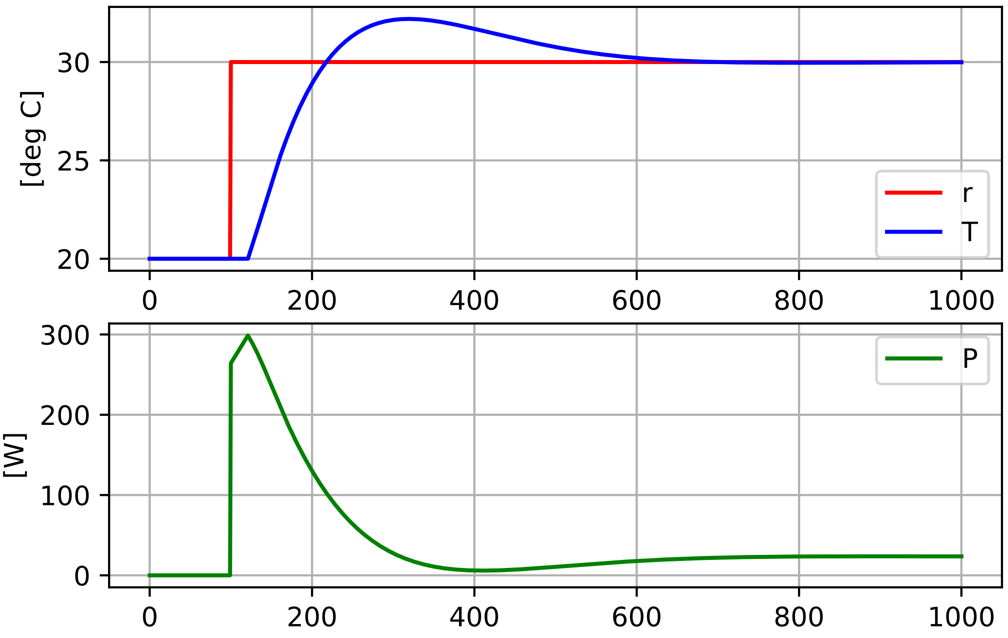


Figure 13.9: Simulation of the temperature control system with elementary Python code.

Solution to Problem 13.2

Figure 13.10 shows a block diagram with transfer functions of the temperature control system.

The PI controller transfer function is developed in Section 11.2, but is repeated here for convenience:

$$H_c(s) = K_c + \frac{K_c}{T_i s} \tag{13.12}$$

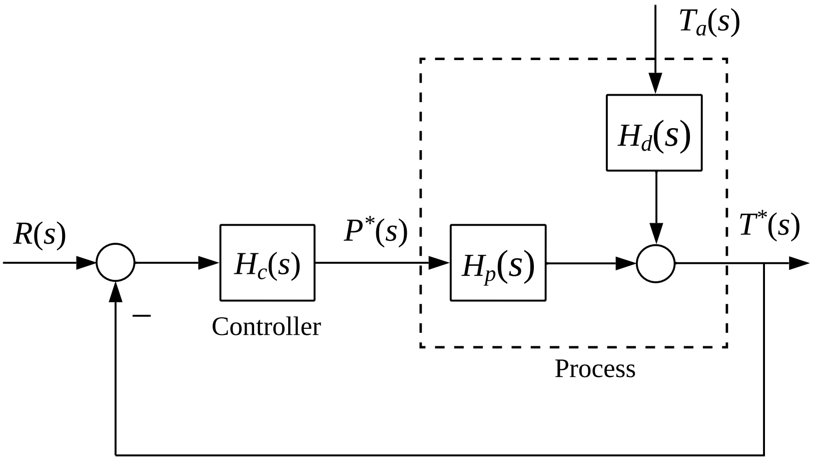


Figure 13.10: Block diagram of the temperature control system with transfer functions.

We derive the process transfer functions $H_p(s) = T^*(s)/P^*(s)$ and $H_d(s) = T^*(s)/T_a(s)$ from the model (13.8): Taking the Laplace transform, the model becomes

$$CsT^*(s) \quad = \quad e^{-t_d s}P^*(s) + G\left[T_a^*(s) - T^*(s)\right] \tag{13.13}$$

From (13.13) we get

$$T^*(s) = \underbrace{\frac{K}{t_c s + 1} e^{-t_d s}}_{H_p(s)} P^*(s) + \underbrace{\frac{K_d}{t_c s + 1}}_{H_d(s)} T_a^*(s) \tag{13.14}$$

where

$$K = \frac{1}{G} = \frac{1}{2.34 \text{ W/K}} = 0.427 \text{ K/W} \tag{13.15}$$

$$K_d = 1 \text{ K/K} \tag{13.16}$$

$$t_c = \frac{C}{G} = \frac{2101 \text{ J/K}}{2.34 \text{ W/K}} = 898 \text{ s} \tag{13.17}$$

In (13.14), I will assume the disturbance T_a is zero and therefore neglect $H_d(s)$, since in applications where we use transfer function models, is quite common to do such a simplification of the model.

The Python program using the Python Control package implementing the simulator is available from the following link:

http://techteach.no/control/python/sim_pi_control_kettle_pycon.py

Figure 13.11 shows simulated T due to a step change of r.

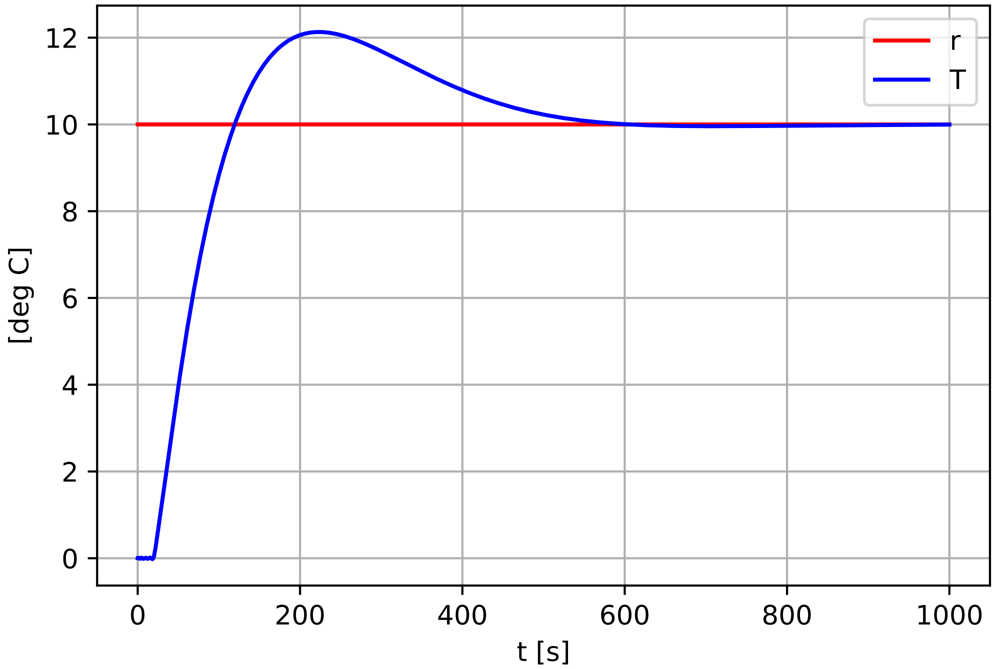


Figure 13.11: Simulation of temperature control system of kettle with Python Control package.

Solution to Problem 13.3

Figure 13.12 shows a block diagram of the temperature control system of the kettle.

Figure 13.13 shows the block diagram model of the temperature control system as built in OpenModelica.

Comment to the OpenModelica model:

- I have used the LimPID block (PID with limitations) because the controller output signal should be limited between 0 and 700 W. (It is not possible to set the output limit on the other PID block denoted just PID.)

The OpenModelica simulator shown in Figure 13.13 is available from the following link:

http://techteach.no/control/openmodelica/temp_control_kettle.mo

Figure 13.14 shows the simulated responses.

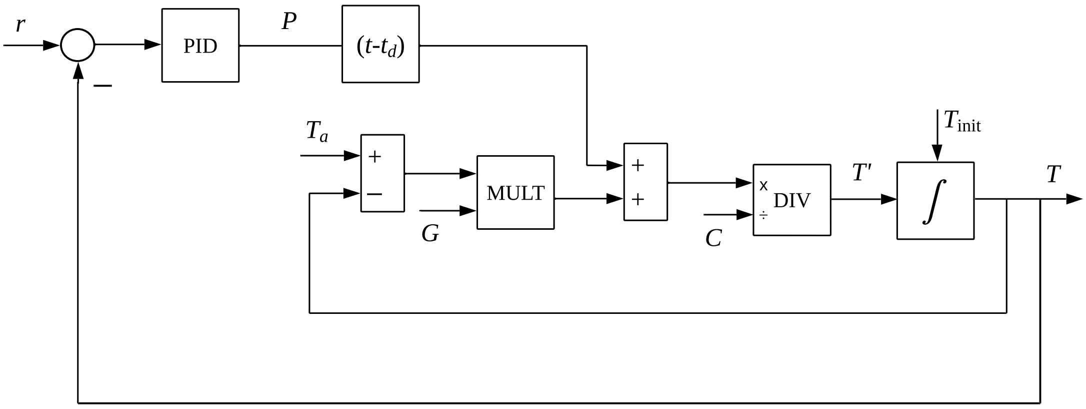


Figure 13.12: Block diagram of the temperature control system.

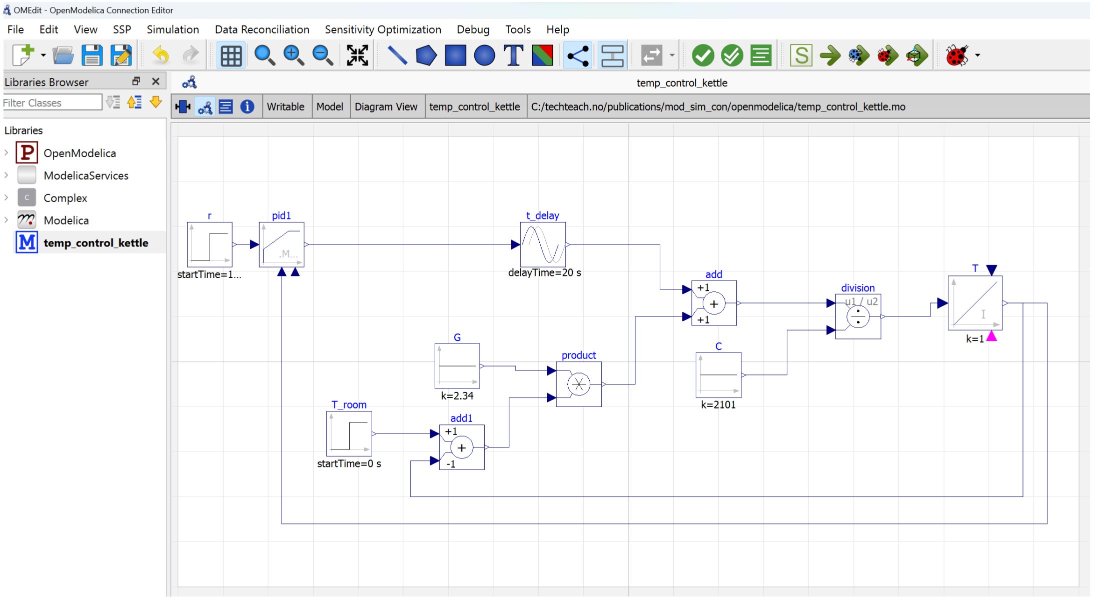


Figure 13.13: Block diagram model of the temperature control system as built in OpenModelica.

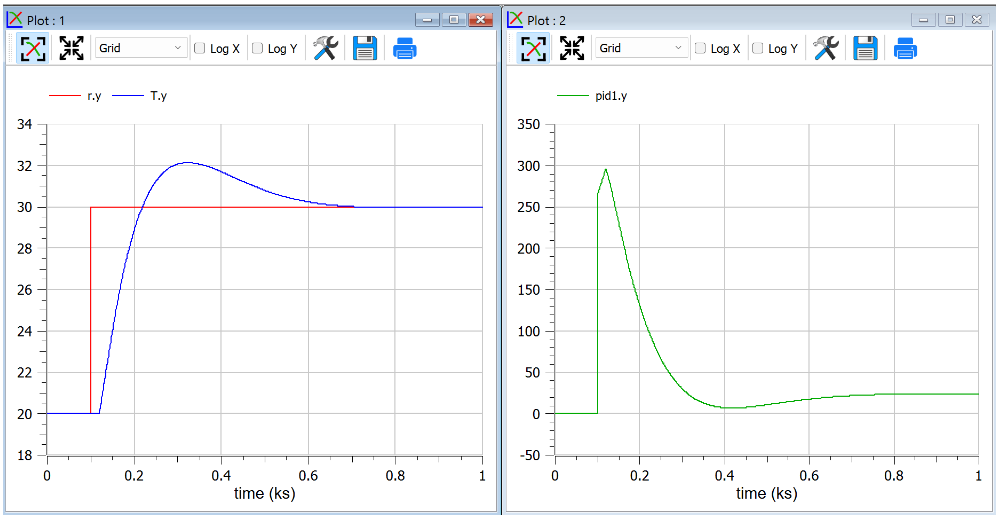


Figure 13.14: Simulation of the temperature control system with OpenModelica.

Chapter 14

Tuning of PID controllers

14.1 Introduction

PID controller tuning is to find proper values of the controller parameters K_c, T_i and T_d so that the specifications of the control system are met. Generally, the specifications comprise:

- *Stability*, which is an absolute requirement to a control system. A control system must be stable – otherwise it is useless, and it may even be dangerous since variables in the control system be oscillate or diverge to large, uncontrolled values.
- *Speed*. In most cases, we want the control system to act as fast as possible. However, some control systems should act slowly, as in averaging level control, cf. Ch. 16.4. In averaging control, the level control of a buffer tank or magazine is made slow by purpose. The slow level control will attenuate variations in the inflow to the magazine, so the outflow, which is manipulated by the level controller, varies less than the inflow – therefore the term averaging control.

Let's look closer at these specifications.

Stability

Stability can be determined from a mathematical model of the control system (as pole placement, cf. Ch. 20, or stability margins, cf. Ch. 15 and 20), or from the way the control system behaves in the "time domain" after it has been excited with an appropriate test signal, typically a step in the reference. Let's take a closer look on control system stability in the time domain.

Ziegler and Nichols Ziegler & Nichols (1942), who developed a PID controller tuning method which is later named after them, cf. Section 14.2, defined "ok stability" as "one quarter decay ratio", see Figure 14.1. The figure shows the response due to a step change in the reference. The ratio between amplitudes A_2 and A_1 is one quarter:

$$\frac{A_2}{A_1} = \frac{1}{4} \tag{14.1}$$

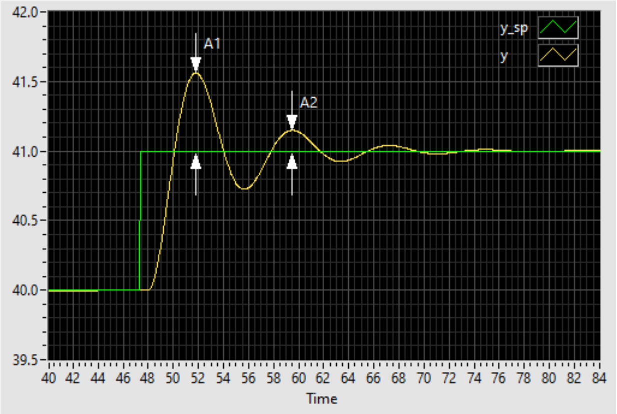


Figure 14.1: The definition of “one quarter decay ratio”

The one quarter decay ratio is the stability you can expect for a control system where you have used one of the Ziegler-Nichols tuning methods, however, there is no guarantee that the ratio is actually one quarter. Most control practitioners are not so happy with such a stability – it should be better (smaller ratio; i.e. better damping). This chapter presents several methods that typically gives better stability.

Speed

Typically, there is a compromise between stability and fastness. This is illustrated in Figure 14.2. It shows the response in the process output variable due to a step change of the reference. If you want very good stability in the control system, the system will not be so fast. If you want very fast control, the system will have poor stability.

Since stability is the most important specification, the controller tuning aims at giving satisfactory stability, and the obtained fastness must be accepted.

The controller tuning methods

This chapter presents several controller tuning methods. Most of them are experimental methods. But that does not mean that these methods can be applied only to a physical system on which you perform experiments; You may use the methods on a *simulator* of the control system!

Some times I am asked which of the controller tuning methods is the best, or which do I recommend trying first. Based on my work is quite a few practical projects, I think SIMC

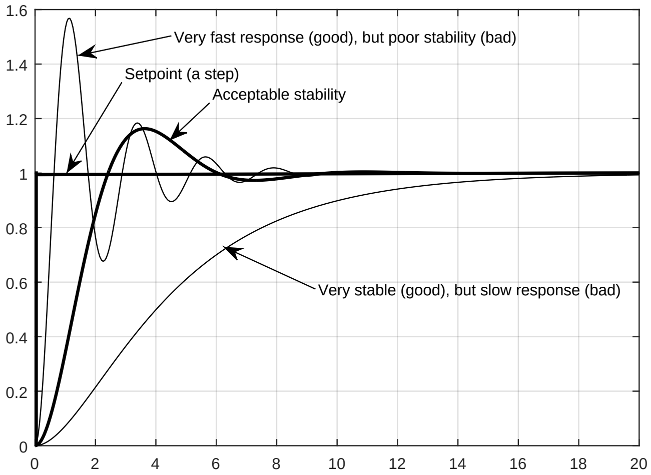


Figure 14.2: The typical compromise between good stability as fastness of a control systems

method, cf. Section 14.8, is the best method overall. It can be used both for fast control and for slow control, and the controller can be retuned directly from the tuning formulas if the process dynamics, for example the process gain or the process time delay, varies. And as an experimental method, it requires in principle only one relatively short experiment, while other methods typically requires several experiments.

14.2 Ziegler-Nichols closed loop method

Introduction

Ziegler and Nichols published in 1942 a paper Ziegler & Nichols (1942) where they described two methods for tuning the parameters of P-, PI- and PID controllers. These two methods go under various names, e.g. the Ziegler-Nichols closed loop method (which is described in this section) and the Ziegler-Nichols open loop method, cf. Section 14.9. One may say that the Ziegler-Nichols closed loop method remains as the most known PID tuning method, despite decades of research and development. The Ziegler-Nichols closed loop method is also often used as a reference method when new methods are tested.

There is no guarantee that a given control system tuned with one of the Ziegler-Nichols methods actually gets a stability as “one quarter decay ratio”, but the stability is probably not very different.

The method

The Ziegler and Nichols closed loop method is based on experiments executed on an established control loop (a real system or a simulated system), see Figure 14.3.

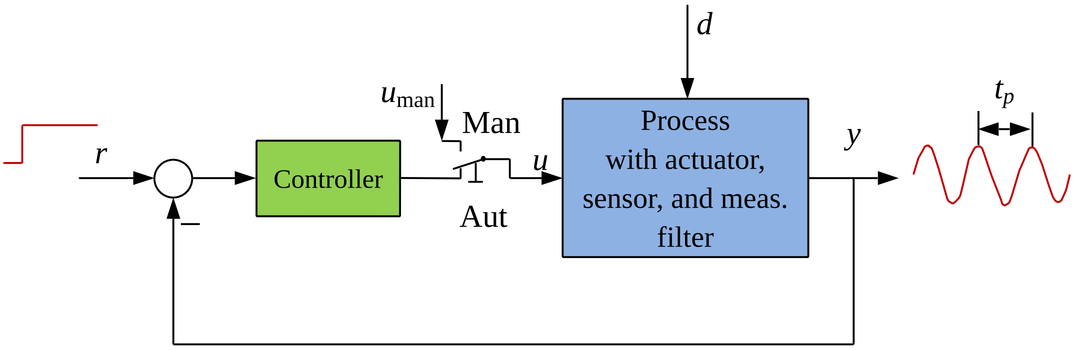


Figure 14.3: The Ziegler-Nichols closed loop method is applied on the closed loop system with the controller in automatic mode.

The tuning procedure is as follows:

Ziegler and Nichols closed loop controller tuning method

1. Bring the process to (or as close to as possible) the specified *operating point* of the control system to ensure that the controller during the tuning is "feeling" representative process dynamic[1] and to minimize the chance that variables during the tuning reach limits. You can bring the process to the operating point by manually adjusting the control variable, with the controller in manual mode, until the process variable is approximately equal to the reference.

2. Turn the PID controller into a *P controller* with gain $K_c = 0$ (set $T_i = \infty$ and $T_d = 0$). Close the control loop by setting the controller in automatic mode.

3. Increase K_c until there are *sustained oscillations* in the signals in the control system, e.g. in the process measurement, after an excitation of the system. (The sustained oscillations corresponds to the system being on the stability limit.) This K_c value is denoted the *ultimate (or critical) gain*, K_{cu}.

   The excitation can be a step in the reference. This step must be small, for example 5% of the maximum reference range, so that the process is not driven too far away from the operating point where the dynamic properties of the process may be different. On the other hand, the step must not be too small, or it may be difficult to observe the oscillations due to the inevitable measurement noise.

   It is important that K_{cu} is found without the actuator being driven into any saturation limit (maximum or minimum value) during the oscillations. If such limits are reached, you will find that there will be sustained oscillations for any (large) value of K_c, e.g. 1000000, and the resulting K_c-value (as calculated from the Ziegler-Nichols formulas, cf. Table 14.1) is useless (the control system will probably be unstable). One way to say this is that K_{cu} must be the smallest K_c value that drives the control loop into sustained oscillations.

4. Measure the *ultimate (or critical) period* t_p of the sustained oscillations.

5. *Calculate the controller parameter values* according to Table 14.1, and use these parameter values in the controller.

6. If the stability of the control loop is poor, try to improve the stability by decreasing K_c by a factor of say 2. If that does not help enough, you may in addition try to increase T_i by a factor of say 2.

Table 14.1: The Ziegler-Nichols controller settings.

	K_c	T_i	T_d
P controller	$0.5K_{cu}$	∞	0
PI controller	$0.45K_{cu}$	$\frac{t_p}{1.2}$	0
PID controller	$0.6K_{cu}$	$\frac{t_p}{2}$	$\frac{t_p}{8} = \frac{T_i}{4}$

Example 14.1 *PI controller tuning with the Ziegler-Nichols closed loop method*

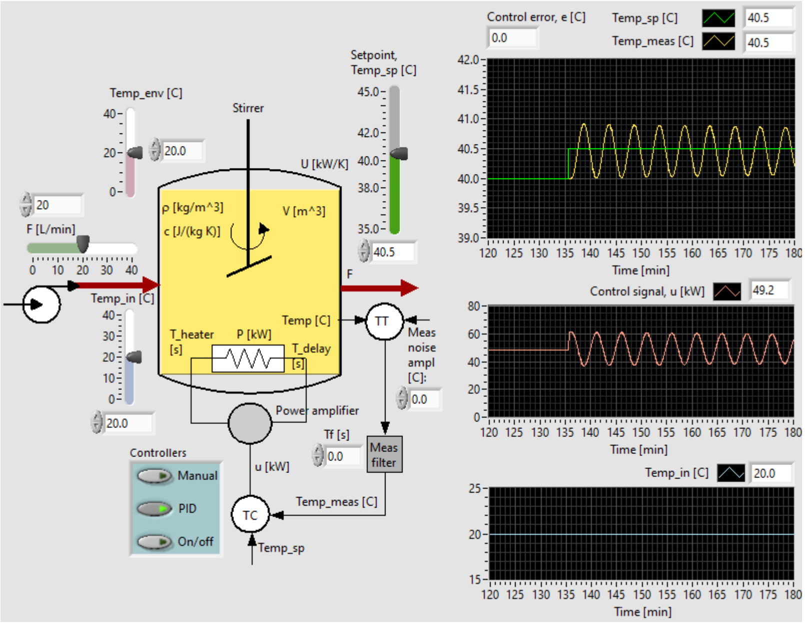


Figure 14.4: The Ziegler-Nichols tuning experiment

Figure 14.4 shows a temperature control system with simulated responses of the Ziegler-Nichols tuning experiment. The ultimate gain is

$$K_{cu} = 26.5 \tag{14.2}$$

The ultimate periode is read off as

$$t_p = 5 \text{ min} = 300 \text{ s} \tag{14.3}$$

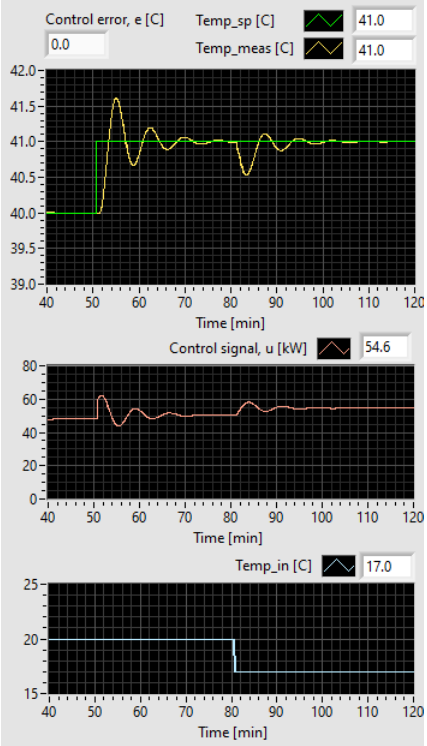
This gives the following Ziegler-Nichols PI settings:

$$K_c = 0.45K_{cu} = 0.45 \cdot 26.5 = 11.9 \tag{14.4}$$

and

$$T_i = \frac{t_p}{1.2} = \frac{300 \text{ s}}{1.2} = 250 \text{ s} \tag{14.5}$$

Figure 14.5 shows the simulated responses with the above PI settings. The excitations are step changes in reference and disturbance (inlet temperature). The stability can be characterized with an amplitude damping ration of approximately 0.30 which is somewhat larger (worse) than 0.25 which is the ratio that Ziegler and Nichols prescribed.


Control error, e [C] 0.0

Temp_sp [C] 41.0

Temp_meas [C] 41.0

Control signal, u [kW] 54.6

Temp_in [C] 17.0

Figure 14.5: Simulated temperature control system with PI settings according to the Ziegler-Nichols method.

[End of Example 14.1]

Some comments to the Ziegler-Nichols closed loop method

1. *You do not know in advance the amplitude of the sustained oscillations.* The amplitude depends partly of the initial value of the process measurement. By using the Åstrøm-Hägglund's tuning method described in Section 14.5 instead of the Ziegler-Nichols closed loop method, you have full control over the amplitude, which is beneficial, of course.

2. *For sluggish processes it may be time consuming* to find the ultimate gain in physical experiments. The Åstrøm-Hägglund's method reduces this problem since the oscillations come automatically.

3. If the operating point varies and if the process dynamic properties depends on the operating point, you should consider using some kind of *adaptive control or gain scheduling*, where the PID parameter are adjusted as functions of the operating point.

   If the controller parameters shall have fixed value, they should be tuned in the worst case as stability is regarded. This ensures proper stability if the operation point varies. The worst operating point is the operation point where the process gain has its greatest value and/or the time delay has its greatest value.

4. *The responses in the control system may become unsatisfactory* with the Ziegler-Nichols method. 1/4 decay ratio may be too much, that is, the damping in the loop is too small. A simple re-tuning in this case is to reduce the K_c somewhat, for example by 20%.

In the beginning

The Ziegler and Nichols methods have definitely proven to be useful, but they actually met some resistance in the beginning. In Blickley (1990) Ziegler reports from a meeting in the American Society of Mechanical Engineers (ASME): "*The questions at the end were pretty bitter because they (the 'old-timers') could not stomach this ultimate sensitivity*[2]*. The questions got worse and worse, and I was answering them. Finally a little guy in the back of the room got up. He was from Goodyear. Since he was on the committee he had received an advance copy of the paper. He stuttered some, and stammered out for all to hear: 'We had one process in our plant, a very bad one, and so I tried this method and it just worked perfectly.' That broke up the meeting.*"

14.3 Relaxed Ziegler-Nichols PI settings

It may happen that the stability of the control system becomes somewhat poor with the Ziegler-Nichols method. In Haugen & Lie (2013), alternative formulas for tuning PI controllers based on the Ziegler-Nichols oscillations are derived. The method will provide better stability than tuning with the original Ziegler-Nichols method. We can denote the resulting settings the Relaxed Ziegler-Nichols settings.

The PI settings are derived from a combination of the Ziegler-Nichols method and the SIMC (Skogestad 2003) method. The PI setting formulas are:

$$K_c = \frac{2}{\pi\left(1+k_r\right)}K_{cu} \tag{14.6}$$

$$T_i = \frac{k_r+1}{2}t_p \tag{14.7}$$

[2] *which implies that the control system is on the stability limit and* oscillates

where k_r is a tuning parameter. With $k_r = 1$, the PI settings are equal to the SIMC PI settings for a process assumed to be "integrator with time delay", cf. Section 14.8. Since the purpose of the Relaxed Ziegler-Nichols settings is relaxed tuning, I suggest even more relaxation than with $k_r = 1$, namely

$$k_r = 1.5 \tag{14.8}$$

Using $k_r = 1.5$ in (14.6) and (14.7) gives $K_c = 0.25K_{cu}$ and $T_i = 1.25t_p$, which are the Relaxed Ziegler-Nichols PI settings. They are presented in Table 14.2 together with the original Ziegler-Nichols PI settings.

Table 14.2: Relaxed Ziegler-Nichols PI settings and the original Ziegler-Nichols settings

PI settings	K_c	T_i
Relaxed Ziegler-Nichols	$0.25K_{cu}$	$1.25t_p$
Original Ziegler-Nichols	$0.45K_{cu}$	$\frac{t_p}{1.2}$

To summarize:

Relaxed Ziegler-Nichols PI settings

1. Accomplish the Ziegler-Nichols experiment to find the ultimate gain, K_{cu}, and ultimate period, t_p.
2. Tune the PI controller using

$$K_c = 0.25K_{cu} \tag{14.9}$$

$$T_i = 1.25t_p \tag{14.10}$$

Example 14.2 *Relaxed Ziegler-Nichols PI tuning*

In Example 14.1 we found the following parameters in the Ziegler-Nichols experiment:

$$K_{cu} = 26.5$$

$$t_p = 5 \text{ min} = 300 \text{ s}$$

This gives the following Relaxed Ziegler-Nichols PI settings:

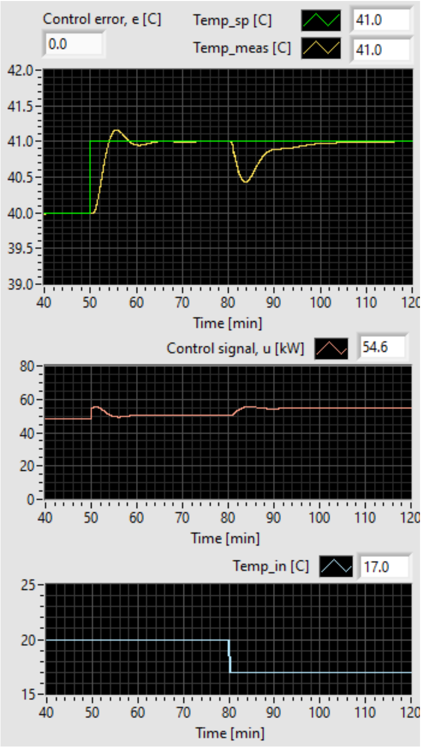
$$K_c = 0.25K_{cu} = 0.25 \cdot 26.5 = 6.63$$

and

$$T_i = 1.25t_p = 1.25 \cdot 300 = 375 \text{ s}$$

Figure 14.6 shows simulations with these PI settings. The stability is clearly improved comparing with the stability with the Ziegler-Nichols method, cf. Figure 14.5.

[End of Example 14.2]



Control error, e [C] 0.0

Temp_sp [C] 41.0

Temp_meas [C] 41.0

Control signal, u [kW] 54.6

Temp_in [C] 17.0

Figure 14.6: Simulation of the temperature control system with PI settings according to the Relaxed Ziegler-Nichols method.

14.4 Quasi Ziegler-Nichols tuning

Here is another method – called the Quasi Ziegler-Nichols tuning method – to cope with poor stability of a control loop – assuming PI control.[3] The poor stability may be the result of a Ziegler-Nichols tuning applied to a process which has relatively small time delay, e.g. less than 10 % of the process time constant. However, the method may be used also on other control systems showing poor stability.

The method is as follows:

Quasi Ziegler-Nichols PI tuning

1. It is assumed that the original PI settings are K_{c0} and T_{i0}, found with any PI tuning method, and that the control system shows poor stability with these settings. Assume that you can observe poorly damped oscillations with period t_{p0}.

[3] This method has not a solid scientific background. It stems from an idea I got while working on tuning a PI controller for biogas production on a pilot plant around year 2012. The method worked out very well on both a simulator of the plant and the real plant, and it turns out to work well also on other systems.

2. Improved PI setting may be obtained by applying Ziegler-Nichols PI tuning *pretending* that these oscillations are true Ziegler-Nichols oscillations:

$$K_c = 0.45 K_{c0} \tag{14.11}$$

$$T_i = \frac{t_{p0}}{1.2} \tag{14.12}$$

Note that the original value of T_{i0} is not used in these formulas (although T_{i0} is mentioned in item 1).

Example 14.3 *Quasi Ziegler-Nichols method*

In Example 14.1 we found the following PI settings for the temperature control system with the Ziegler-Nichols method:

$$K_c = 11.9 \tag{14.13}$$

$$T_i = 250\,\text{s} \tag{14.14}$$

Figure 14.5 shows simulations with these settings. Assume that we are not happy with the stability. From Figure 14.5 we read off

$$t_{p0} = 8\,\text{min} = 480\,\text{s}$$

The Quazi Ziegler-Nichols method gives:

$$K_c = 0.45 K_{c0} = 0.45 \cdot 11.9 = 5.36 \tag{14.15}$$

$$T_i = \frac{t_{p0}}{1.2} = \frac{480\,\text{s}}{1.2} = 400 \tag{14.16}$$

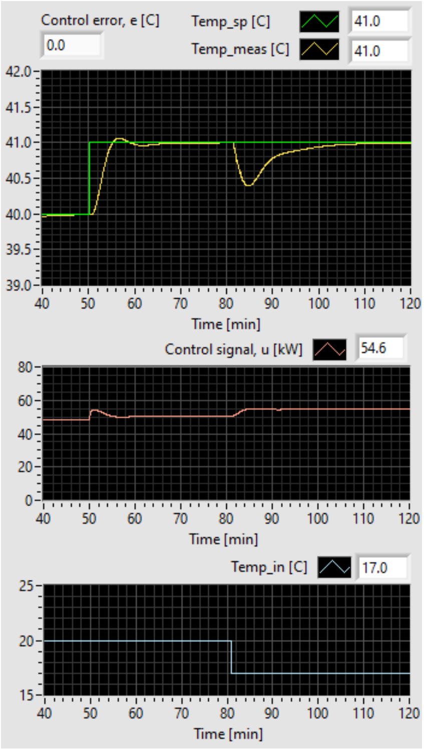
Figure 14.7 shows simulations with these PI settings. The stability is clearly improved comparing with the stability with the Ziegler-Nichols method, cf. Figure 14.5.

[End of Example 14.3]

14.5 Åstrøm-Hägglund Relay tuning method

If the process has slow dynamics, the trial-and-error procedure to find the critical gain $K_{c,u}$ in the Ziegler-Nichols method may take a long time. Åstrøm and Hägglund Åstrøm & Hägglund (1995) found a clever solution that may reduce the time considerably as the trial-and-error procedure is avoided. The PID controller to be tuned is replaced (in the tuning session) by an On-off controller! As we saw in Ch. 1.3.5, an On-off gives sustained oscillations in the control loop, and the sustained oscillations come automatically. Figure 14.8 illustrates this solution. An On-off controller behaves like a relay. Hence, the method is often referred to as the Åstrøm and Hägglund Relay tuning method, or simply the Relay tuning method

Now, the important questions are:



Control error, e [C] 0.0

Temp_sp [C] 41.0

Temp_meas [C] 41.0

Control signal, u [kW] 54.6

Temp_in [C] 17.0

Figure 14.7: Simulation of the temperature control system with PI settings according to the Quasi Ziegler-Nichols method.

1. Is the period of the oscillation, t_p, the same with the On-off controller as with a P controller with ultimate gain for the given control loop?
2. Can we estimate the ultimate gain, K_{cu}, from oscillations with the On-off controller?

Answer to the first question:

It turns out that the period is very similar. Hence, the oscillations with the On-off controlles can be used as t_p in the Ziegler-Nichols formulas.

Answer to the second question:

In general, the gain of a system can be estimated from the ration

$$K = \frac{A_{\text{out}}}{A_{\text{in}}} \tag{14.17}$$

where A_{in} is the amplitude of an assumed sine function at the input, and A_{out} is the amplitude of the corresponding sine function at the output. In Figure 14.8, the amplitude at the input of the On-off controller is the same the amplitude of the process output. Hence,

$$A_{\text{in}} = A_y \tag{14.18}$$

which can estimated from the observed response (with the On-off controller).

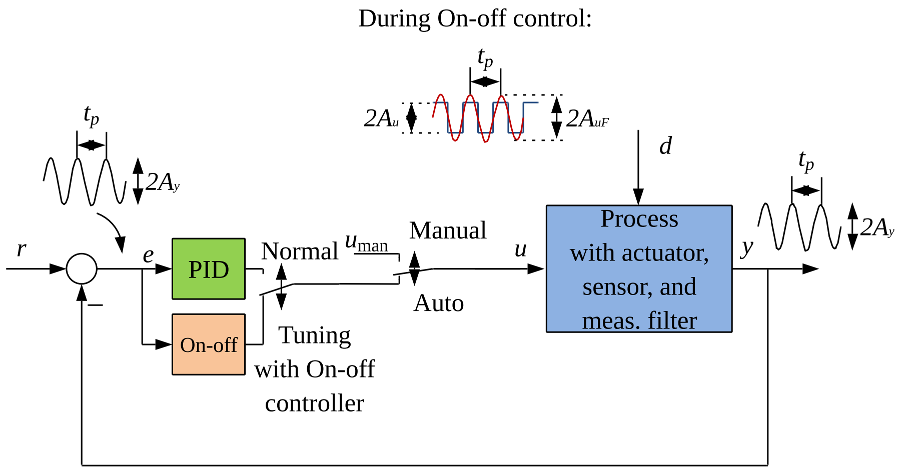


Figure 14.8: With an On-off in the place of the PID controller oscillations come automatically.

What is A_{out}? With On-off control, the control signal, u, is a square wave (created automatically as a result of the On-off controller function). Åstrøm and Hägglund approximated this square wave with the basic sine function of the Fourier series approximation to the square wave. According to the Fourier theory, the amplitude of the sine function is

$$A_{uF} = \frac{4A_u}{\pi} \tag{14.19}$$

where A_u is the amplitude of the square wave, cf. Figure 14.8. Often the range of the control signal is [0%, 100%], giving $A_u = 50\%$. Now, A_{out} in (14.17) is

$$A_{\text{out}} = A_{uF} = \frac{4A_u}{\pi}$$

As a result of the above, the ultimate gain to be used in the Ziegler-Nichols tuning method can be estimated as

$$K_{cu} = \frac{A_{\text{out}}}{A_y} = \frac{A_{uF}}{A_y} = \frac{\frac{4A_u}{\pi}}{A_y} = \frac{4A_u}{\pi A_y} = 1.27\frac{A_u}{A_y} \tag{14.20}$$

To summarize:

The Relay method for PI controller tuning

1. Replace the PI controller with an On-off controller, which automatically causes the control system to oscillate.

2. Read off the period, t_p, of the oscillations.

3. Read off the amplitude, A_u, of the square-shaped oscillations of the control signal.

4. Read off the amplitude, A_y, of the typically sinusoidal oscillations of the process output variable.

5. Calculate the ultimate controller gain:

$$K_{cu} = 1.27\frac{A_u}{A_y} \tag{14.21}$$

6. Calculate the PI controller settings (assuming Ziegler-Nichols formulas):

$$K_c = 0.45K_{cu} \tag{14.22}$$

$$T_i = \frac{t_p}{1.2} \tag{14.23}$$

7. Replace the on/off regulator with a PI regulator (for normal operation).

Example 14.4 *PI controller tuning with the Relay tuning method*

See Example 14.1. We will now tune the temperature controller TC as a PI controller using On-off tuning. Figure 14.9 shows the responses with On-off controller with

$$A_u = 40 \text{ kW} \tag{14.24}$$

From the simulation, we read off

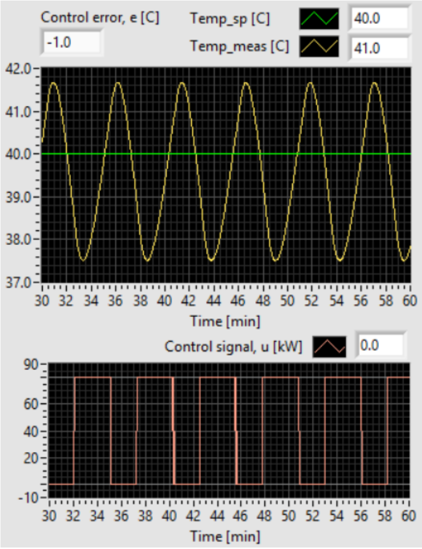


Figure 14.9: Simulated responses with On-off controller.

$$A_y = \frac{41.7\,^\circ\mathrm{C} - 37.5\,^\circ\mathrm{C}}{2} = 2.1\,^\circ\mathrm{C} \tag{14.25}$$

giving

$$K_{cu} = \frac{4A_u}{\pi A_y} = \frac{4 \cdot 40\ \mathrm{kW}}{\pi \cdot 2.1\,^\circ\mathrm{C}} = 24.3\,\mathrm{kW}/^\circ\mathrm{C}$$

and

$$t_p = 5.2\,\mathrm{min} = 312\,\mathrm{s}$$

which are quite similar to the values found with the Ziegler-Nichols method in Example 14.1, namely $K_{cu} = 26.5$ kW/°C and $t_p = 300$ s, respectively.

The PI settings may now be calculated using the Ziegler-Nichols formulas, see Table 14.1, alternatively, the Relaxed Ziegler-Nichols formulas, see Table 14.114.2). Assuming the Ziegler-Nichols formulas, we get

$$K_c = 0.45K_{c,u} = 0.45 \cdot 24.3\,\mathrm{kW}/^\circ\mathrm{C} = 10.9\,\mathrm{kW}/^\circ\mathrm{C} \tag{14.26}$$

$$T_i = \frac{312\ \mathrm{s}}{1.2} = 260\ \mathrm{s} \tag{14.27}$$

[End of Example 14.4]

14.6 Auto-tuning

Auto-tuning is automatic tuning of controller parameters in a well-defined experiment. It is common that commercial controllers offers auto-tuning. The operator starts the auto-tuning via some button or menu choice on the controller. The controller then executes automatically a pre-planned experiment either on the uncontrolled process (i.e. the open loop system), or on the control loop (i.e. the closed loop system) – depending on the auto-tuning method implemented. After this experiment, the controller is automatically set into normal operation with the tuned parameters.

One example of an auto-tuner is the so-called relay tuner which uses an automatic implementation of the On-off PID tuning method presented in Section 14.5. Such an auto-tuner is found in e.g. PID controllers by ABB and by Fuji. When the auto-tuning phase is started, an On-off (or relay) controller is used as the controller in the control loop. The On-off controller sustained oscillations in control loop come automatically. From the amplitude and the period of these oscillations, proper PID controller parameters are calculated by an algorithm in the controller (the algorithm does not necessarily use the Ziegler-Nichols formulas). Just a couple of periods are needed for the autotuner to have enough information to accomplish the tuning. The autotuner activates the tuned PID controller automatically after the tuning has finished.

14.7 Good Gain method

The Good Gain method aims at obtaining acceptable stability as explained above. It is a simple method which has proven to give good results on laboratory processes and on simulators. The method is based on experiments on a real or simulated control system, see Figure 14.10.

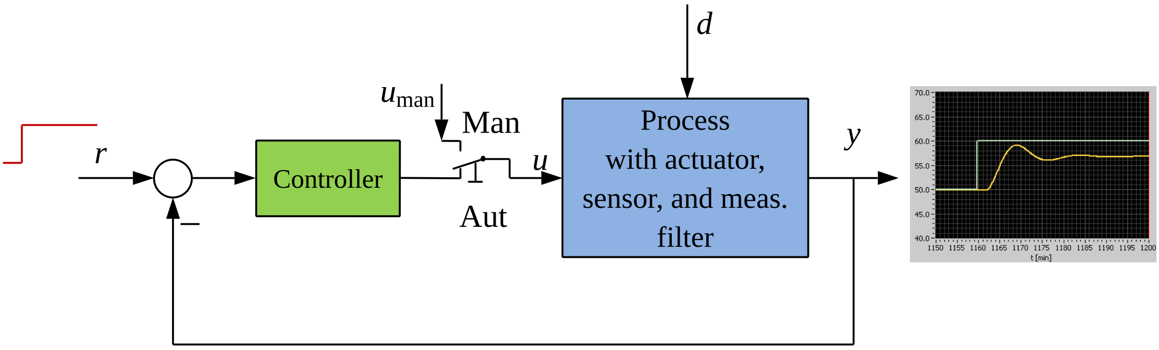


Figure 14.10: The Good Gain method for PID tuning is applied to the established control system

The procedure described below assumes a PI controller, which is the most commonly used controller function. However, a comment about how to include the D-term, so that the controller becomes a PID controller, is also given.

The Good Gain PI controller tuning method

1. Bring the process to or close to the normal or specified operation point by adjusting the nominal control signal u_0 (with the controller in manual mode).

2. Ensure that the controller is a P controller with $K_c = 0$ (set $T_i = \infty$ and $T_d = 0$). Increase K_c until the control loop gets good (satisfactory) stability as seen in the response in the measurement signal after e.g. a step in the reference or in the disturbance (exciting with a step in the disturbance may be impossible on a real system, but it is possible in a simulator). If you do not want to start with $K_c = 0$, you can try $K_c = 1$ (which is a good initial guess in many cases) and then increase or decrease the K_c value until you observe some overshoot and a barely observable undershoot (or vice versa if you apply a reference step change the opposite way, i.e. a negative step change), see Figure 14.11. This kind of response is assumed to represent good stability of the control system. This gain value is denoted $K_{c,\mathrm{GG}}$.

   It is important that *the control signal is not driven to any saturation limit* (maximum or minimum value) during the experiment. If such limits are reached the K_c value may not be a good one – probably too large to provide good stability when the control system is in normal operation. So, you should apply a relatively small step change of the reference (e.g. 5% of the reference range), but not so small that the response drowns in noise.

3. Set the integral time T_i equal to

$$T_i = 1.5t_{\text{ou}} \tag{14.28}$$

where t_{ou} is the time between the overshoot and the undershoot of the step response (a step in the reference) with the P controller, see Figure 14.11.[4] Note that for most systems (those which does not containt a pure integrator) there will be offset from reference because the controller during the tuning is just a P controller.

4. Because of the introduction of the I-term, the loop with the PI controller in action will probably have somewhat reduced stability than with the P controller only. To compensate for this, the K_c can be reduced somewhat, e.g. to 80% of the original value. Hence,

$$K_c = 0.8K_{c,\text{GG}} \tag{14.29}$$

5. If you want to include the D-term, so that the controller becomes a PID controller[5], you can try setting T_d as follows:

$$T_d = \frac{T_i}{4} \tag{14.30}$$

which is the T_d–T_i relation that was used by Ziegler and Nichols (Ziegler & Nichols 1942).

6. You should check the stability of the control system with the above controller settings by applying a step change of the reference. If the stability is poor, try reducing the controller gain somewhat, possibly in combination with increasing the integral time.

Example 14.5 *PI tuning with the Good Gain method*

Figure 14.12 shows the Good Gain tuning experiment on the simulated temperature system shown in Figure 14.4.

From the response we read off:

$$K_{c,\text{GG}} = 8.0 \tag{14.31}$$

and

$$t_{\text{ou}} = 4 \text{ min} = 240 \text{ s} \tag{14.32}$$

which give the following Good Gain PI settings

$$K_c = 0.8K_{c,\text{GG}} = 0.8 \cdot 8.0 = 6.4 \tag{14.33}$$

$$T_i = 1.5t_{\text{ou}} = 1.5 \cdot 240 \text{ s} = 360 \text{ s} \tag{14.34}$$

Figure 14.13 shows simulations with these PI settings. The stability is clearly improved comparing with the stability with the Ziegler-Nichols method, cf. Figure 14.5.

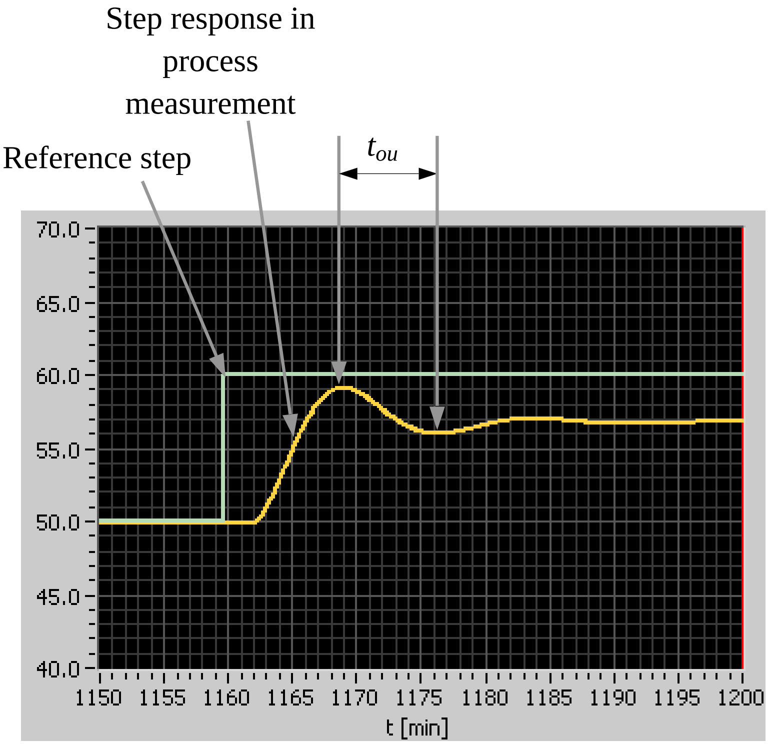


Figure 14.11: The Good Gain method: Reading off the time between the overshoot and the undershoot of the step response with P controller.

[End of Example 14.5]

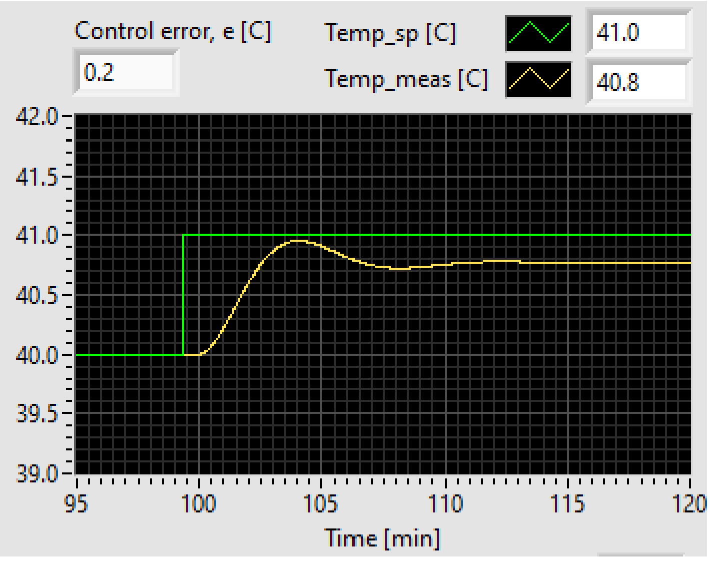
14.8 SIMC controller tuning method

14.8.1 Background of the SIMC method

The SIMC method (Skogestad 2003) is a model-based tuning method where the controller parameters are expressed as functions of the process model parameters. It is assumed that the control system has a block diagram as shown in Figure 14.14.

Comments to this block diagram:

- The process transfer function $H_p(s)$ is *a combined transfer function of the process, the sensor, and the measurement lowpass filter*. For simplicity we may denote this transfer function the "process transfer function", although it is a combined transfer function.
- The process transfer function may stem from a single step-response experiment with



Control error, e [C] 0.2

Temp_sp [C] 41.0

Temp_meas [C] 40.8

Figure 14.12: The Good Gain tuning experiment on the simulated temperature system shown in Figure 14.4

the process, as explained in subsequent sections.

- The block diagram shows a disturbance acting on the process. Information about this disturbance is not used in the tuning, but if you are going to test the tuning on a simulator to see how the control system compensates for a process disturbance, you should add a disturbance at the point indicated in the block diagram, which is at the process input. It turns out that in most processes the dominating disturbance influences the process dynamically at the "same" point as the control variable. Such a disturbance is called an *input disturbance*. Here are a few examples:
  - Liquid tank: The control variable controls the inflow. The outflow is a disturbance.
  - Motor: The control variable controls the motor torque. The load torque is a disturbance.
  - Thermal process: The control variable controls the power supply via a heating element. The power loss via heat transfer through the walls and heat outflow through the outlet are disturbances.

The design principle of the SIMC method is as follows. The control system *tracking transfer function* $H_t(s)$, which is the transfer function from the reference to the (filtered) process measurement, is *specified* as a first order transfer function with time delay:

$$H_t(s) = \frac{Y(s)}{U(s)} = \frac{1}{t_{cc}s + 1}e^{-t_d s} \tag{14.35}$$

where t_{cc} is the time **c**onstant of the **c**ontrol system which *the user must specify*, and t_d is the process time delay which is *given* by the process model (the method can however be used for processes without time delay, too). Figure 14.15 shows as an illustration the response in y after a step in the reference r with the tracking transfer function (14.35).

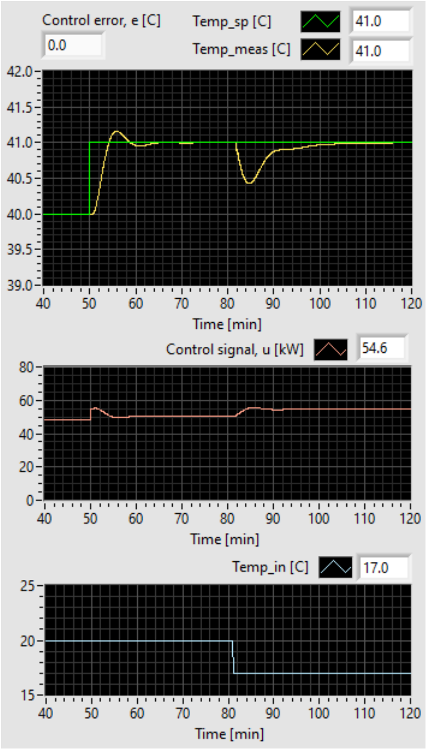


Figure 14.13: Simulation of the temperature control system with PI settings according to the Good Gain method.

From the block diagram shown in Figure 14.14 the tracking transfer function is, cf. the Feedback rule in Figure 8.3,

$$H_t(s) = \frac{H_p(s)H_c(s)}{1 + H_p(s)H_c(s)} \tag{14.36}$$

Here, the only unknown is the controller transfer function, $H_c(s)$. Solving for $H_c(s)$, gives

$$H_c(s) = \frac{H_t(s)}{[1 - H_t(s)]\, H_p(s)}$$

By setting (14.36) equal to (14.35) and making some simplifying approximations to the time delay term, the controller, $H_c(s)$, becomes a PID controller or a PI controller for the assumed process transfer function, $H_p(s)$.

14.8.2 PI controller tuning for "integrator with time delay" processes

14.8.2.1 Mathematical model and dynamics

The process model of an "integrator with time delay" is

$$y'(t) = K_i u(t - t_d) \tag{14.37}$$

The corresponding process transfer function is

$$\frac{Y(s)}{U(s)} = H_p(s) = \frac{K_i}{s} e^{-t_d s} \tag{14.38}$$

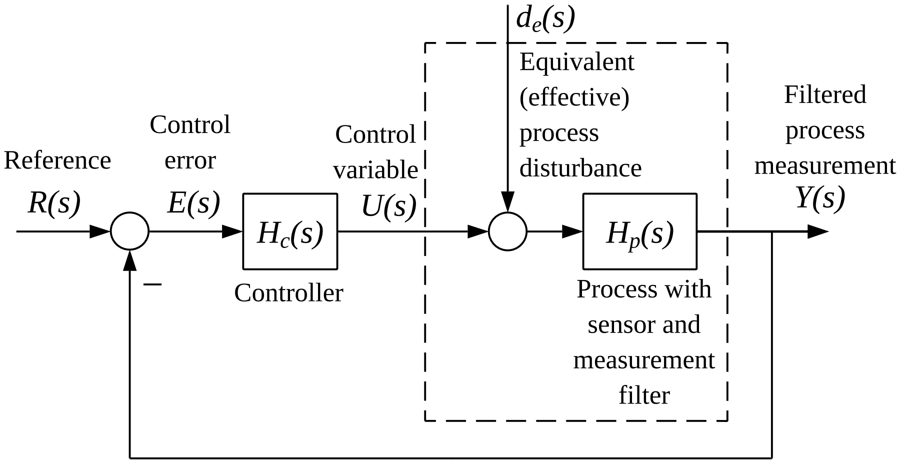


Figure 14.14: Block diagram of the control system in the SIMC controller tuning method (Skogestad 2003).

Figure 14.16 illustrates the dynamics of "integrator with time delay" processes with a step response, which is in the form of a delayed ramp. Note that the step is in *the control signal* (the step is applied with the controller – if present – in manual mode).

14.8.2.2 Controller settings

The SIMC controller settings are:

$$K_c = \frac{1}{K_i\left(t_{cc} + t_d\right)} \tag{14.39}$$

$$T_i = 2\left(t_{cc} + t_d\right) \tag{14.40}$$

$$T_d = 0 \tag{14.41}$$

Thus, the controller is a PI controller.

In the original SIMC formulas, the factor 2 in (14.40) is 4. This gives good reference tracking. But the disturbance compensation may become unneccessarily sluggish. To obtain faster disturbance compensation, factor 2 can be used Haugen & Lie (2013).[6] The drawback of using 2 instead of 4 is that there will be somewhat more overshoot in the reference step response, and that the stability of the control loop will be somewhat reduced. Also, the robustness against changes of process parameters (e.g. increase of process gain and increase of process time delay) will be somewhat reduced.

Skogestad's suggestion for selecting t_{cc}

[6] So, we may denote the above settings as modified SIMC settings, but for simplicity I say just SIMC settings.

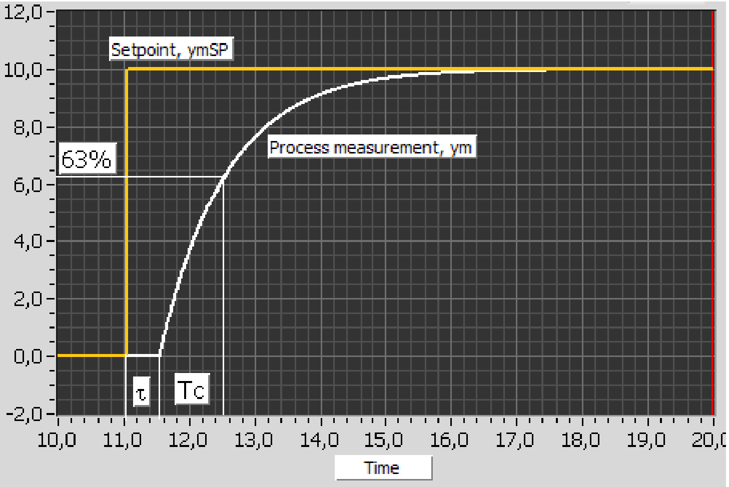


Figure 14.15: Response due to a reference step with the specified tracking transfer function (14.35) in the SIMC controller tuning method

If you do not know how to specify t_{cc} in (14.39) and (14.40), you may use Skogestad's advice in his SIMC method:

$$t_{cc} = t_d \tag{14.42}$$

Consequently, (14.39) and (14.40) give the following PI settings:

PI settings for "integrator with time delay" processes

$$K_c = \frac{1}{2K_i t_d} \tag{14.43}$$

$$T_i = 4t_d \tag{14.44}$$

Example 14.6 *SIMC tuning of the level PI controller of a wood chips tank*

A level control system of a wood chips tank with conveyor belt is presented in Ch. 13.1. The P&I D is shown in Figure 13.1.

A mathematical model of the tank with conveyor belt based on mass balance of the wood chips is described in Ch. 39.2. I repeat the model here:

$$\rho A h'(t) = u(t - t_d) - F_{\text{out}}(t) \tag{14.45}$$

where h [m] is wood chips level, ρ [kg/m^3] is wood chips density, A [m^2] is cross-sectional area, t_d [s] is transportation time or time delay of the conveyor belt.

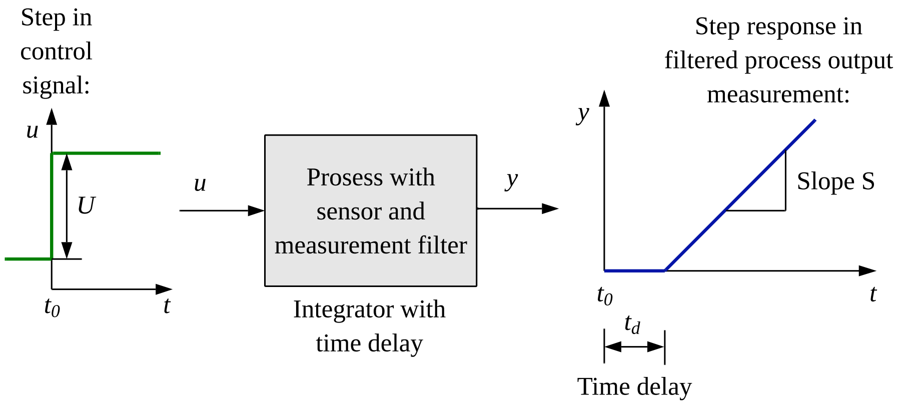


Figure 14.16: Step response of an "integrator with time delay" process.

I will now show how to apply (14.43)-(14.44) to tune the level PI controller.

The PI settings are independent of the outflow rate F_{out}. The model relating control signal u and process output y is then

$$\rho A h'(t) = u(t - t_d) \tag{14.46}$$

which can be written as

$$h'(t) = \frac{1}{\rho A} u(t - t_d) = K_i u(t - t_d) \tag{14.47}$$

which is on the form (14.37) with

$$K_i = \frac{1}{\rho A} \tag{14.48}$$

Thus, the model is "integrator with time delay" with the integrator gain given by (14.48).

Although not necessary for the purpose of controller tuning, we may also derive the process transfer function, $H_p(s)$, from $U(s) = \mathcal{L}\{u(t)\}$ to $H(s) = \mathcal{L}\{h(t)\}$ from (14.45). It is

$$H_p(s) = \frac{H(s)}{U(s)} = \frac{1}{\rho A s} e^{-t_d s} = \frac{K_i}{s} e^{-t_d s} \tag{14.49}$$

which is on the form (14.38).

The PI settings (14.43)-(14.44) become

$$K_c = \frac{1}{2K_i t_d} = \frac{1}{2 \cdot \frac{1}{\rho A} \cdot t_d} = \frac{1}{2 \cdot \frac{1}{145\ [\mathrm{kg/m^3}] \cdot 13.4\ [\mathrm{m^2}]} \cdot 250\ [\mathrm{s}]} = 3.89\ \frac{(\mathrm{kg/s})}{\mathrm{m}} \tag{14.50}$$

$$T_i = 4t_d = 4 \cdot 250\,\mathrm{s} = 1000\,\mathrm{s} \tag{14.51}$$

A Python program implementing a simulator in elementary Python code is available from the following link:

http://techteach.no/control/python/sim_skogestad_pi_tuning_chipstank.py

Figure 14.17 shows the simulated level response due to a step change of the level reference r and a step change of the outflow rate F_{out}. The responses indicate that the stability of the control loop is satisfactory.

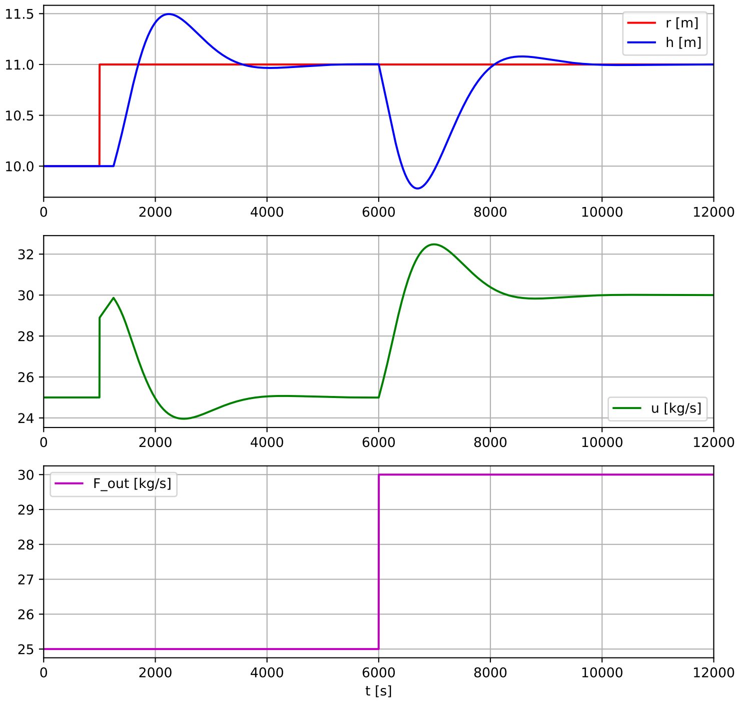


Figure 14.17: Simulation of level control system of wood chips tank.

[End of Example 14.6]

14.8.2.3 PI controller tuning for "pretended integrator with time delay" processes

The PI settings (14.43) – (14.44) may be used even if the process dynamics is not actually "integrator with time delay". In such cases, we have to pretend that the process dynamics is "integrator with time delay". I have experienced in several practical applications that this is a useful simple approach to PI controller tuning.

Figure 14.18 shows the actual step response (blue curve) of a process and the step response (red curve) of a pretended "integrator with time delay" process. From the pretended

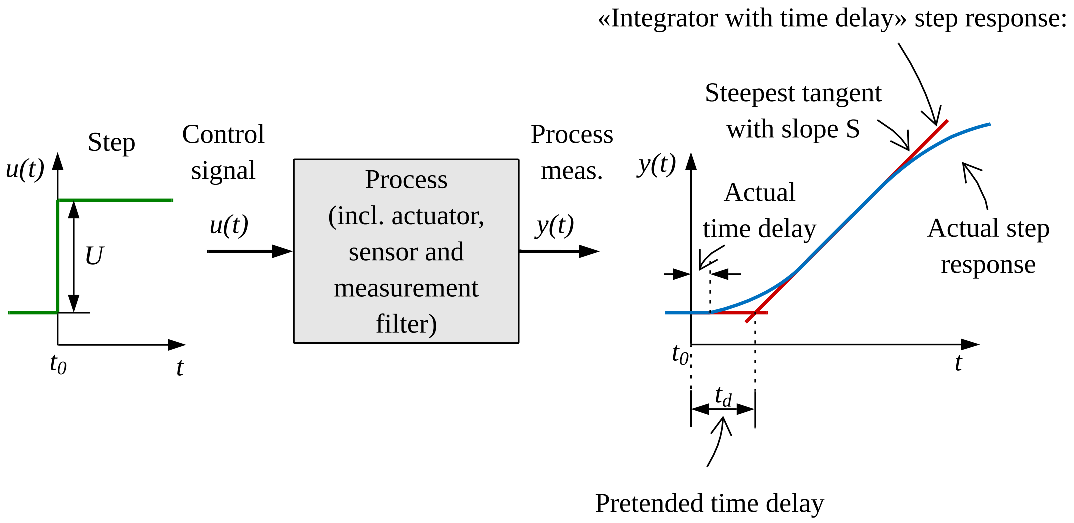


Figure 14.18: Actual step response of a process and the step response of a pretended "integrator with time delay" process.

response, we read off the pretended time delay, t_d, and the slope, S, of the steepest tangent:

- t_d is the time measured from the point of time that the step change was made to the point of time where the tangent crosses the base line of the response.
- S is the slope of the steepest tangent. For example, if the process response is measured in unit of %, and the time unit is second, then S has the unit of %/s.

For an integrator, the slope of the step response is

$$S = K_i U \tag{14.52}$$

where K_i is the integrator gain and U is the amplitude of the step applied to the control signal. That the slope is $K_i U$, can be seen directly from (14.37) by setting u equal to U and neglecting the time delay. From (14.52) we can calculate K_i as

$$K_i = \frac{S}{U} \tag{14.53}$$

Hence, K_i can be interpreted as the normalized slope of the step response of the integrator.

Now that we have found the values of t_d and K_i, we can tune the PI controller with (14.43) – (14.44), which are repeated here:

PI settings for "pretended integrator with time delay" processes

$$K_c = \frac{1}{2K_i t_d} \tag{14.54}$$

$$T_i = 4t_d \tag{14.55}$$

I have found the PI settings (14.54)-(14.55) very useful in various practical projects, so I dare to denote them the "Golden PI settings".

If uncertain: Overestimate!

If you are uncertain when reading off the values of S and t_d from the step response, cf. Figure 14.18, you should *overestimate* their values, i.e. give them (slightly) too large values rather than (slightly) too small because it is *safe regarding stability* of the control system. This is because relatively large values of K_i and t_d in (14.54)-(14.55) gives smaller values of K_c and T_i, which imrpoves the stability of the control loop.

You may wonder

The blue curve in Figure 14.18 indicates "time constant with time delay" process dynamics, or actually some higher order dynamics. The SIMC method actually includes PI tuning formulas also for such process dynamics. However, it is more challenging, and definitely more time-consuming, to adapt such models than to adapt the "integrator with time delay" process model to the step response. As an example, I once did a PI controller tuning of a biogas flow rate controller on a real biogas reactor. Such reactors have very slow dynamics, with a time constant of days. If I should have waited for the biogas response to have settled to adapt a model, I would wait for days. Instead, I pretended that the reactor was a "integrator with time delay" process, and read off the time delay and the slope after just a few hours (when I saw that the response in the biogas measurement was about to flatten). So, I saved a lot of time.

Relating to PI for "time constant with time delay" processes

In Section 14.8.4 the SIMC PI tuning formulas for "time constant with time delay" processes are presented. From those formulas, we can find that the PI settings for "time constant with time delay" processes are identical with the PI settings for "integrator with time delay" processes if

$$t_d \leq \frac{t_{cp}}{4} \tag{14.56}$$

Example 14.7 *SIMC PI tuning of a temperature controller*

Let us tune a PI controller for the heated liquid tank shown in Figur 14.4. Figure 14.19

shows the step response in the temperature, T, due to a step change in the control signal, u, to the heater.

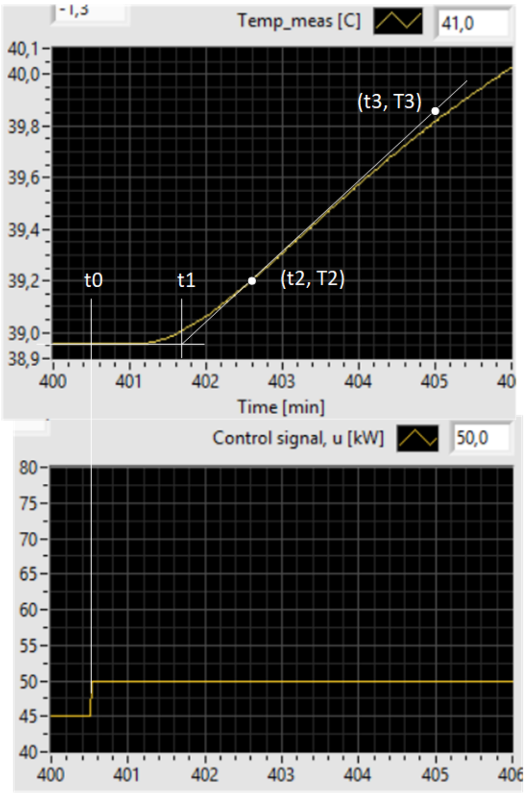


Figure 14.19: Step response in the temperature, T, due to a step change in the control signal, u, to the heater.

The step amplitude is

$$U = 50\,\text{kW} - 45\,\text{kW} = 5\,\text{kW}$$

I have drawn the steepest tangent to the temperature curve. From Figure 14.19 we can read off the following values (approximately):

$$t_0 = 400.5\,\text{min}$$

$$t_1 = 401.7\,\text{min}$$

$$t_2 = 402.6\,\text{min}$$

$$t_3 = 405.0\,\text{min}$$

$$T_2 = 39.20\,^\circ\text{C}$$

$$T_3 = 39.85\,^\circ\text{C}$$

This gives:

$$S = \frac{T_3 - T_2}{t_3 - t_2}$$

$$K_i = \frac{S}{U}$$

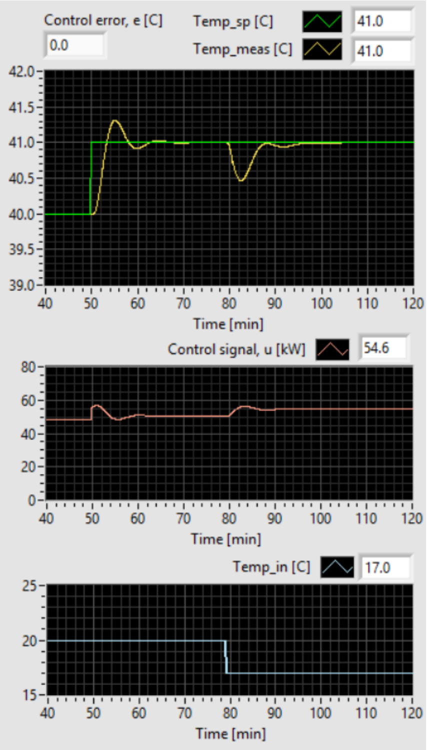
$$t_d = t_1 - t_0$$

The PI settings become:

$$K_c = \frac{1}{2K_i t_d} = 7.69\,\text{kW}/^\circ\text{C} \tag{14.57}$$

$$T_i = 4t_d = 4.8\,\text{min} = 288\,\text{s} \tag{14.58}$$

Figure 14.20 shows the response in the temperature with these PI settings. The reference is changed as a step from 40 to 41 °C at $t = 50$ min, and the inlet temperature, T_{in}, which is a disturbance, is changed from 20 to 17 °C at $t = 80$ min. The stability of the control system seems to be ok.



Control error, e [C] 0.0

Temp_sp [C] 41.0

Temp_meas [C] 41.0

Control signal, u [kW] 54.6

Temp_in [C] 17.0

Figure 14.20: PI control with SIMC PI settings: Temperature response due to a reference step.

[End of Example 14.7]

14.8.3 PI controller tuning for "pure integrator"

In Section 14.8.2, you saw how to tune PI controller for an "integrator with time delay" process. The controller settings are (14.43)-(14.44) based on the hand-rule $t_{cc} = t_d$. Now assume that the process is a "pure integrator", i.e. it has no time delay. Then you can not just set the time delay t_d to zero in (14.43)-(14.44) since that would give you $K_c = \infty$ and

$T_i = 0$, which does not work, of course. Instead, we must recall the original SIMC controller settings for "integrator with time delay" processes:

$$K_c = \frac{1}{K_i\left(t_{cc} + t_d\right)} \tag{14.59}$$

$$T_i = 2\left(t_{cc} + t_d\right) \tag{14.60}$$

With $t_d = 0$, the above settings become

$$K_c = \frac{1}{K_i t_{cc}} \tag{14.61}$$

$$T_i = 2t_{cc} \tag{14.62}$$

Here, we must specify t_{cc} to some reasonable value larger than zero. If you do not know what could be a reasonable value, you can simulate the control system for various values of t_{cc}. If the control signal (controller output signal) is changing too quickly, or often reaches the maximum and minimum values for reasonable changes of the reference or the disturbance, the controller is too aggressive, and you can try increasing t_{cc}.

Let's look at an example of an integrator process, namely a water tank. Although water is assumed, the material can of course be something else, for example youghurt to be pumped from a storage tank at the end of a youghurt production line and into a cups to be shipped to shops.

Example 14.8 *SIMC tuning of a PI level controller*

Figure 14.21 shows a water tank with varying inflow. The level controller manipulates the outflow via a pump. We assume there is zero or negligible time delay between the pump flow control signal and the level (process output measurement). The symbols $\mathit{\Delta} F_{\text{out}}$ and $\mathit{\Delta} h_{\max}$ in the figure are refered to later in this example.

The figure shows the front panel of the following SimView simulator:

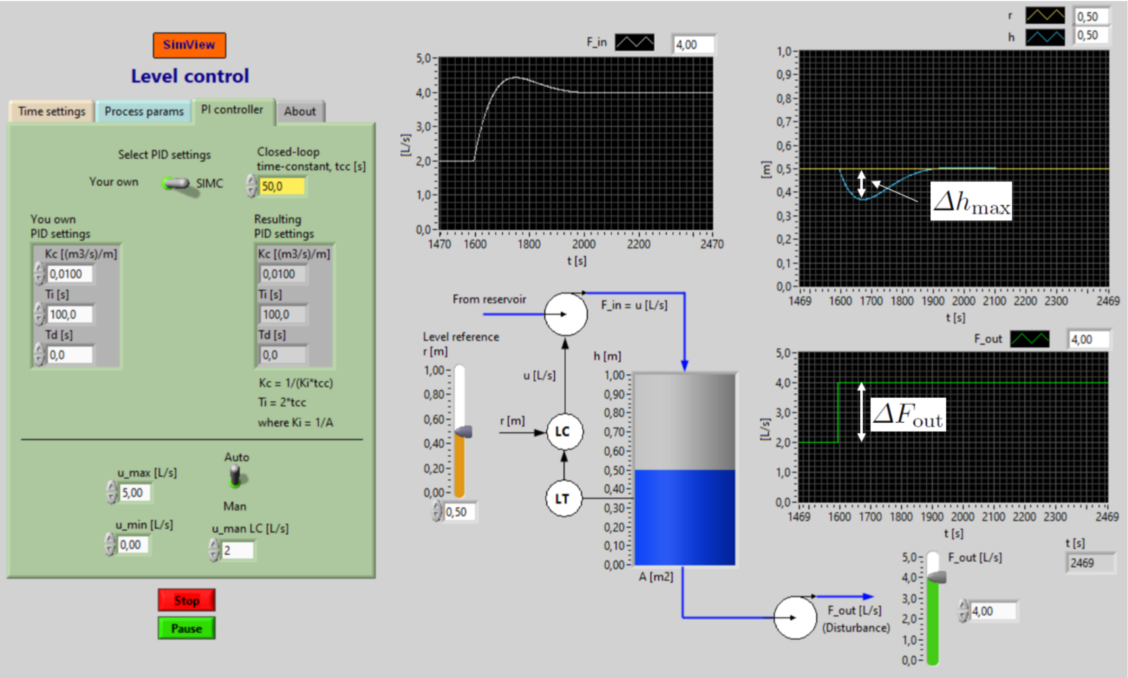
http://techteach.no/simview/level_control/

In (14.61)-(14.62), we need to know the integrator gain, K_i, and wee need to specify the closed loop time constant, t_{cc}. Both topics are addressed below.

What is K_i?

K_i is the process integrator gain. It can be found from a process model. Mass balance of the water is

$$\rho A h' = \rho F_{\text{in}} - \rho F_{\text{out}} \tag{14.63}$$



SimView

Level control

Closed-loop time-constant, tcc [s]: 50,0

You own PID settings: Kc [(m3/s)/m] 0,0100; Ti [s] 100,0; Td [s] 0,0

Resulting PID settings: Kc [(m3/s)/m] 0,0100; Ti [s] 100,0; Td [s] 0,0

Kc = 1/(Ki*tcc)

Ti = 2*tcc

where Ki = 1/A

u_max [L/s]: 5,00

u_min [L/s]: 0,00

u_man LC [L/s]: 2

F_in: 4,00

r: 0,50

h: 0,50

F_out: 4,00

Level reference r [m]: 0,50

F_out [L/s]: 4,00

t [s]: 2469

Figure 14.21: Liquid tank with a level controller.

where ρ [kg/m^3] is water density, A [m^2] is cross-sectional area of the tank, F_{in} [m^3/s] is inflow, and F_{out} [m^3/s] is outflow. In the following, ρ has been cancelled out. We assume that the level controller output signal, u, is in unit [m^3/s] and that the actual value of F_{in} is equal to u. We substitute u with F_{in} in (14.63). Now, (14.63) can be written as

$$h' = (u - F_{\text{out}})/A \tag{14.64}$$

which is the process model. It represents integrator dynamics, cf. Section 9.2. Let's find the integrator gain K_i, which we need in (14.61), from a step response test (alternatiely, we can find it from the process transfer function). Assume for simplicity that all variables in (14.64) initially are zero, and that we apply a step change with amplitude (or height) U in u. Then (14.64) gives

$$h' = S = \frac{1}{A}U \tag{14.65}$$

where S is the slope (rate of change) of the ramp-formed level response. K_i is the normalized slope, i.e.

$$K_i = \frac{S}{U} = \frac{1}{A} \tag{14.66}$$

which we will use in (14.61).

How to specify t_{cc}?

t_{cc} is the closed loop time constant. Therefore, it seems natural to specify t_{cc} as the time constant, i.e. the 63% rise time, in the level due to a step change of the level reference. However, in level control systems the level reference is typically constant, so specifying the 63% rise time of the reference step response is not particularly meaningful for such control systems. What makes more sense is to specify t_{cc} to obtain an acceptable level response due to a worst-case outflow variation, which we here will assume is a stepwise change. I will now just present a result from the later chapter about averaging level control[7] (Chapter 16.4) where I have derived the following formula for specifying t_{cc}:

$$t_{cc} = \frac{A\Delta h_{\text{max}}}{\Delta F_{\text{out}}} \tag{14.67}$$

In (14.67), Δh_{max} is the maximum steady state change in level due to an assumed maximum outflow step change of amplitude ΔF_{out} [m^3/s]. Note: (14.67) is conservative, which here means that the actual Δh_{max} will be somewhat smaller for a given ΔF_{out}, so there is a design margin. Figure 14.21 shows ΔF_{out} and Δh_{max}.

Summing up the PI settings

1. From the allowed Δh_{max} due to a change of the outflow as a step of amplitude ΔF_{out}, calculate t_{cc} as

$$t_{cc} = \frac{A\Delta h_{\text{max}}}{\Delta F_{\text{out}}} \tag{14.68}$$

2. Then, t_{cc} is used to tune the PI controller as follows:

$$K_c = \frac{A}{t_{cc}} \tag{14.69}$$

$$T_i = 2t_{cc} \tag{14.70}$$

Example numerical values

For the SimView simulator:

$$A = 0.5\,\text{m}^2 \text{ (given)}$$
$$\Delta F_{\text{out}} = 2.0\,\text{L/s} = 0.002\ \text{m}^3/\text{s (specified)}$$
$$\Delta h_{\text{max}} = 0.2\,\text{m(specified)}$$

Then, from (14.68) we get

$$t_{cc} = \frac{A\Delta h_{\text{max}}}{\Delta F_{\text{out}}} = \frac{0.5\,\text{m}^2 \cdot 0.2\,\text{m}}{0.002\,\text{m}^3/\text{s}} = 50\,\text{s} \tag{14.71}$$

Finally, (14.69) and (14.70) gives

$$K_c = -\frac{A}{t_{cc}} = -\frac{0.5\,\text{m}^2}{50\,\text{s}} = -0.01\ (\text{m}^3/\text{s})/\text{m} \tag{14.72}$$

$$T_i = 2t_{cc} = 2 \cdot 50\,\text{s} = 100\,\text{s} \tag{14.73}$$

Figure 14.21 shows a simulation with step change in the outflow of $\Delta F_{\text{out}} = 2.0\,\text{L/s}$. We see that the level, as expected, is reduced somewhat less than 0.2 due to the outflow step change. Thus, the specification of the control system is satisfied!

[End of Example 14.8]

[7] In averaging level control, the pump (actuator) is at the outlet insted of at the inlet as in the present example.

14.8.4 PI controller tuning for “time constant with time delay” processes

14.8.4.1 Mathematical model and dynamics

The process model of a “time constant with time delay” process is

$$t_{cp} y'(t) = K u(t - t_d) - y(t) \tag{14.74}$$

where t_{cp} is the time constant of the process, K is the gain, and t_d is the time delay.

The process transfer function corresponding to (14.74) is

$$\frac{Y(s)}{U(s)} = H_p(s) = \frac{K}{t_{cp}s + 1} e^{-t_d s} \tag{14.75}$$

Figure 14.22 illustrates the dynamics of “time constant with time delay” processes with a step response, which is in the form of a delayed exponential curve.

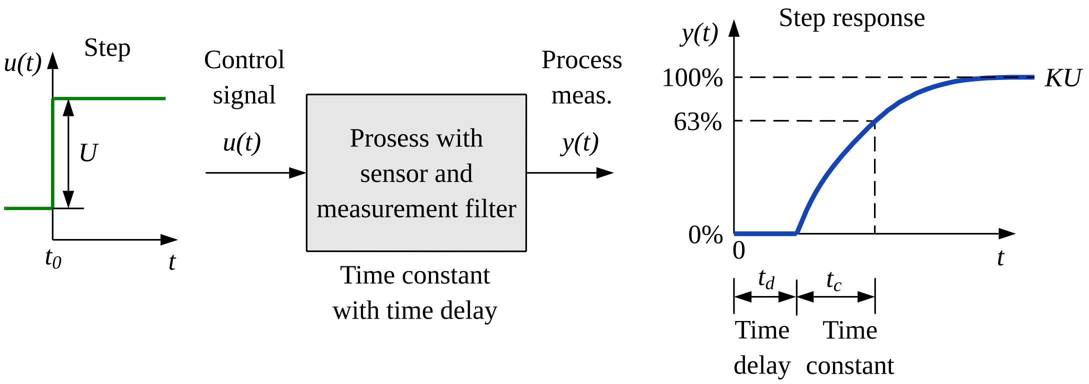


Figure 14.22: How the transfer function parameters K, t_{cp}, and t_d appear in the step response of a “time constant with time delay process”.

14.8.4.2 Controller settings

The SIMC controller settings are

$$K_c = \frac{t_{cp}}{K\,(t_{cc} + t_d)} \tag{14.76}$$

$$T_i = \text{minimum}\left[t_{cp},\, 2\,(t_{cc} + t_d)\right] \tag{14.77}$$

$$T_d = 0 \tag{14.78}$$

Thus, the controller is a PI controller.

In the original SIMC formulas, the factor 2 in (14.77) is actually 4. I suggest using 2 instead of 4 since this gives faster disturbance compensation.

Skogestad's suggestion for selecting t_{cc}

If you do not know how to specify t_{cc} in (14.76) and (14.77), you may use Skogestad's advice in his SIMC method:

$$t_{cc} = t_d \tag{14.79}$$

With this choice, the PI settings are:

SIMC PI settings for time constant with time delay processes

$$K_c = \frac{t_{cp}}{2Kt_d} \tag{14.80}$$

$$T_i = \min\left(T,\, 4t_d\right) \tag{14.81}$$

Note that if $t_d \le t_{cp}/4$, i.e. for a relatively small time delay, the PI settings are identical to the settings for "integrator with time delay" processes, cf. Section 14.8.2.

Example 14.9 *Control of "time constant with time delay" process*

Let us try the SIMC method for tuning a PI controller for the process transfer function

$$H_p(s) = \frac{K}{t_{cp}s+1}e^{-t_d s} \tag{14.82}$$

where

$$K = 1;\ t_{cp} = 1 \text{ s};\ t_d = 0.5 \text{ s} \tag{14.83}$$

We use (14.42):

$$t_{cc} = t_d = 0.5 \text{ s} \tag{14.84}$$

The controller parameters are as follows:

$$K_c = \frac{t_{cp}}{K\left(t_{cc}+t_d\right)} = \frac{1}{1\cdot(0.5+0.5)} = 1 \tag{14.85}$$

$$T_i = \min\left[t_{cp},\, c\left(t_{cc}+t_d\right)\right] \tag{14.86}$$

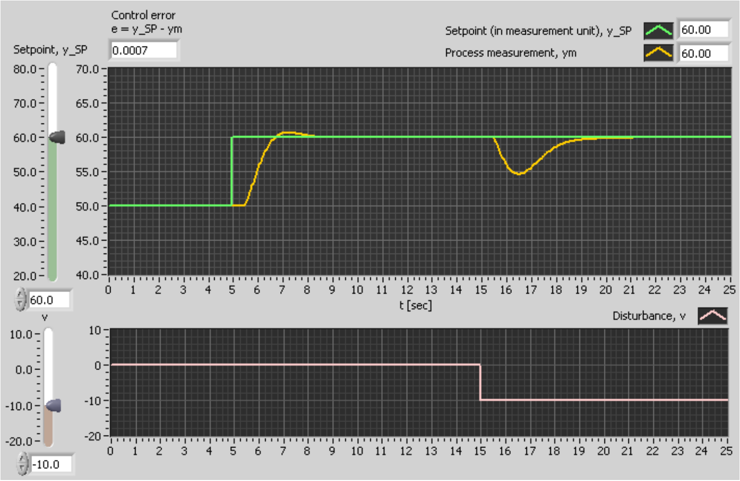
$$= \min\left[1,\, 2\left(0.5+0.5\right)\right] \tag{14.87}$$

$$= \min\left[1,\, 2\right] \tag{14.88}$$

$$= 1 \text{ s} \tag{14.89}$$

$$T_d = 0 \tag{14.90}$$

Figure 14.23 shows control system responses with the above PID settings. At time 5 sec the reference is changed as a step, and at time 15 sec the disturbance is changed as a step. The responses, and in particular the stability of the control systems, seem ok.



Control error e = y_SP - ym: 0.0007

Setpoint (in measurement unit), y_SP: 60.00

Process measurement, ym: 60.00

Setpoint, y_SP: 60.0

v: -10.0

Figure 14.23: Example 14.9: Simulated responses in the control system with SIMC controller tuning.

[End of Example 14.9]

14.8.5 PID controller tuning for "double integrator" processes

14.8.5.1 Mathematical model

Typically, there is no or just a negligible time delay in processes which have models like double integrators (motion systems). Although the SIMC method provides PID settings for double integrators with time delay processes, I here neglect any time delay.

The differential equation model of a double integrator is:

$$y'' = K_{ii}u \tag{14.91}$$

The corresponding process transfer function is

$$\frac{Y(s)}{U(s)} = H_p(s) = \frac{K_{ii}}{s^2} \tag{14.92}$$

Example 14.10 *Model of a double integrator*

An example of a double integrator is a body (ship, robot, satellite) actuated by a force, but not by any speed-dependent damping forces (i.e. no so-called viscous damping). One example is a ship. Chapter 39.9 presents a model of a ship that also includes hydrodynamic forces and wind forces. Now, assume that the ship speed is zero relative to water and wind. (If the water and wind speeds are zero, this means that the ship is at rest.) Then, there are no damping forces acting on the ship. The model of the ship is given by Newton's 2nd law:

$$my'' = u \tag{14.93}$$

or:

$$y'' = \frac{1}{m}u = K_{ii}u \tag{14.94}$$

where: m [kg] is ship mass, y [m] is position, u [N] is propeller force.

(14.94) is a double integrator.

The still state will be the most critical operating point in position control because there is no natural damping on the movement by the environment. If you are going to tune a position controller for the ship in a dynamic positioning system (DP system), you should tune the controller in this operating point since the stability will be imporved in any other operating point (where the ship moves).

[End of Example 14.10]

14.8.5.2 Controller settings

It can be shown using theoretical stability analysis that the controller for a double integrator must have derivative action, otherwise the control loop is guaranteed to be unstable. Thus, controller should be selected as a PID controller. The control loop will be unstable with a PI controller.

The SIMC method provides the following PID settings of a so-called serial PID controller:

$$K_{c,\text{serial}} = \frac{1}{4K_{ii}\,{t_{cc}}^2} \tag{14.95}$$

$$T_{i,\text{serial}} = 4t_{cc} \tag{14.96}$$

$$T_{d,\text{serial}} = 4t_{cc} \tag{14.97}$$

A serial PID controller comprises a PI controller in series with a PD controller, which means that the transfer function of the PID controller is equal to the product of the transfer function of the PI term and the transfer function of the PD term:

$$H_{c,\text{PID, serial}}(s) = H_{c,\text{PI}}(s) \cdot H_{c,\text{PD}}(s) \tag{14.98}$$

where

$$H_{c,\text{PI}}(s) = K_c\frac{T_is+1}{T_is} \tag{14.99}$$

and

$$H_{c,\text{PD}}(s) = T_ds+1 \tag{14.100}$$

SIMC PID settings for a parallel PID controller

In this book, the parallel PID controller is assumed. In the parallel PID controller, the P, I, and D terms are added. In terms of transfer functions:

$$H_{c,\text{PID, parallel}}(s) = H_{c,\text{P}}(s) + H_{c,\text{I}}(s) + H_{c,\text{D}}(s) \tag{14.101}$$

where:

$$H_{c,\text{P}}(s) = K_c \tag{14.102}$$

$$H_{c,\text{I}}(s) = \frac{K_c}{T_i s} \tag{14.103}$$

$$H_{c,\text{D}}(s) = K_c T_d s \tag{14.104}$$

From the above two PID transfer function models, we can derive the following transformation formulas:

$$K_{c,\text{parallel}} = K_{c,\text{serial}} \left(1 + \frac{T_{d,\text{serial}}}{T_{i,\text{serial}}} \right) \tag{14.105}$$

$$T_{i,\text{parallel}} = T_{i,\text{serial}} \left(1 + \frac{T_{d,\text{serial}}}{T_{i,\text{serial}}} \right) \tag{14.106}$$

$$T_{d,\text{parallel}} = T_{d,\text{serial}} \frac{1}{1 + \frac{T_{d,\text{serial}}}{T_{i,\text{serial}}}} \tag{14.107}$$

So, to find the SIMC settings for a parallel PID controller, we do as follows:

1. Calulate the settings for a serial PID controller, (14.95)-(14.97).
2. Transform those serial settings to settings for a parallel PID controller using the transformation formulas (14.105)-(14.107).

Following the two-step procedure above, we get the following (improved) PID setting for a parallel PID controller:

Parallel PID controller settings for a double integrator

$$K_c = \frac{1}{2K_{ii}\, {t_{cc}}^2} \tag{14.108}$$

$$T_i = 8t_{cc} \tag{14.109}$$

$$T_d = 2t_{cc} \tag{14.110}$$

It turns out that the actual time constant of the control system with the parallel PID controller becomes about 2 times the specified time constant, t_c. To compensate for this inaccuracy, I have used $t_{cc}/2$ instead of t_{cc} in (14.95)-(14.97).

Example 14.11 *PID tuning of a dynamic positioning system*

Ch. 39.9 presents a mathematical model of a ship. Assume

$$m = 71164 \text{ tonn} \tag{14.111}$$

Assume the following specification:

$$t_{cc} = 100 \text{ s} \tag{14.112}$$

The PID settings become:

$$K_c = 3558 \text{ N/m} \tag{14.113}$$

$$T_i = 800 \text{ s} \tag{14.114}$$

$$T_d = 200 \text{ s} \tag{14.115}$$

A simulator in elementary Python code is available on:

http://techteach.no/control/python/sim_pid_tuning_dynpos.py

In the simulator, a time constant filter with time constant $T_f = 0.1T_d$ is included in the D-term of the controller.

Figure 14.24 shows the simulated step response in position of the control system. The stability seems ok. The observed time constant (63% rise time) is not approx. 80 s, which is not very different quite from the specified value of 100 s.

[End of Example 14.11]

14.9 Ziegler-Nichols open loop method

In Ziegler & Nichols (1942), Ziegler and Nichols presented their closed loop method. In that article, they presented a second PID tuning method which is denoted the Ziegler-Nichols open loop method, or the Ziegler-Nichols process reaction curve method. In this method, the PID settings are calculated from the integrator gain, K_i, and the time delay, t_d, as in the SIMC method assuming "integrator with time delay" process dynamics, cf. Section 14.8.2.3. The Ziegler og Nichols PI settings for a PI controller is (they derived also P settings and PID settings) are as follows.

PI settings in Ziegler-Nichols open loop method

$$K_c = \frac{0.9}{K_i t_d} \tag{14.116}$$

$$T_i = 3.3t_d \tag{14.117}$$

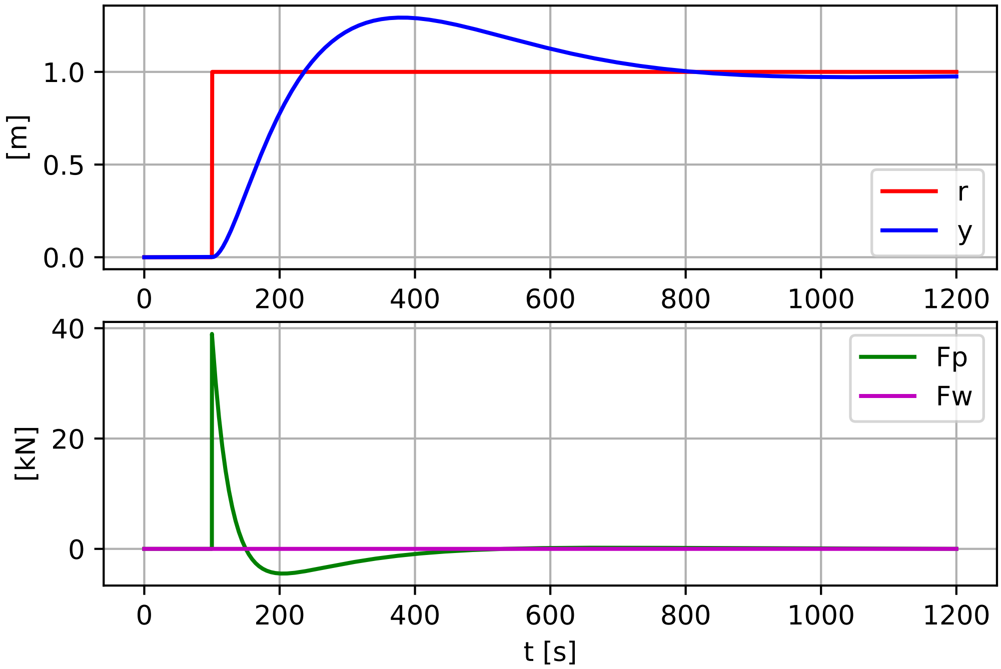


Figure 14.24: Step response in the position in a simulated dynamic positioning system.

Comparison with the SIMC PI formulas, (14.43) – (14.44): In the Ziegler and Nichols PI settings, the gain is $0.9/0.5 = 1.8$ times larger, and T_i is $4/3.3 = 1.2$ times smaller than in the (modified) SIMC PI settings. In general, both increasing K_c and reducing T_i reduce the stability of a control loop. As a consequence, Ziegler-Nichols tuning certainly gives reduced stability comparing with SIMC tuning. Therefore, I recommend using SIMC tuning instead of Ziegler-Nichols tuning.

14.10 PID tuning when process dynamics varies

14.10.1 Introduction

A well tuned PID controller has parameters which are adapted to the dynamic properties to the process, so that the control system becomes fast and stable. If the process dynamic properties varies without re-tuning the controller, the control system

- gets *reduced stability*, or
- becomes *more sluggish*.

Problems with variable process dynamics can be solved in the following alternative ways:

- **The controller is tuned in the most critical operation point**, so that when the process operates in a different operation point, the stability of the control system is just better — at least the stability is not reduced. However, if the stability is too good the tracking quickness is reduced, giving more sluggish control.
- **The controller parameters are varied in the "opposite" direction of the variations of the process dynamics**, so that the performance of the control system is maintained, independent of the operation point. Some ways to vary the controller parameters are:
  - *Model-based PID parameter adjustment*, cf. Section 14.10.2.
  - *PID controller with gain scheduling,* cf. Section 14.10.3.
  - *Model-based adaptive controller,* cf. Section 14.10.4.

Commercial control equipment is available with options for gain scheduling and/or adaptive control.

14.10.2 PID parameter adjustment based on the SIMC PID tuning method

Assume that you have tuned a PID or a PI controller for some process that has a model as assumed in the SIMC PI(D) controller tuning method, cf. Section 14.8. Assume that some of the parameters of the process model changes. How should the controller parameters be adjusted? The answer is directly given by the SIMC tuning formulas because those formulas contains the process parameters!

Example 14.12 *Adjustment of PI controller parameters for "integrator with time delay" process*

Assume that the process transfer function is

$$H_p(s) = \frac{K_i}{s}e^{-t_d s} \tag{14.118}$$

The SIMC PI settings for this process are (14.43) – (14.44), which I repeat here:

$$K_c = \frac{1}{2K_i t_d} \tag{14.119}$$

$$T_i = 4t_d \tag{14.120}$$

As an example, assume that the process gain K_i is increased to, say, twice its original value. How should the PI parameters be adjusted to maintain good behaviour of the control system? From (14.119) we see that K_c should be halved, and from (14.120) we see that T_i should remain unchanged.

As another example, assume that the process time delay t_d is increased to, say, twice its original value. From (14.119) we see that K_c should be halved, and from (14.120) we see that T_i should get a doubled value. One concrete example of such a process change is the wood chips tank. If the speed of the conveyor belt is halved, the time delay (transport delay) is doubled. And now you know how to quickly adjust the PI controller parameters if such a change of the conveyor belt speed should occur.[8]

[End of Example 14.12]

You may use the SIMC tuning formulas as the basis for adjusting the PID parameters even if you used some other method than the SIMC method for the initial tuning. In Ch. 15.1 the SIMC tuning formulas are used to retune the PI controller of the level control of the wood chips tank when parameter changes of the process (tank with conveyor belt) occur.

14.10.3 Gain scheduling of PID parameters

Figure 14.25 shows the structure of a control system of a process which may have varying dynamic properties, for example a varying gain.

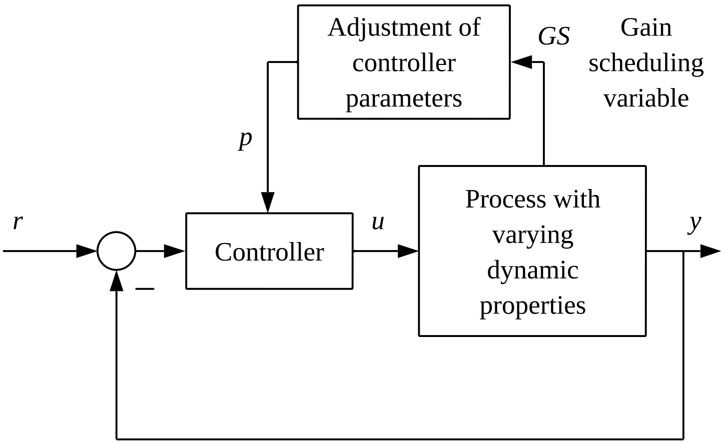

Adjustment of controller parameters
GS
Gain scheduling variable
p
r
Controller
u
Process with varying dynamic properties
y
–

Figure 14.25: Control system of a process having varying dynamic properties. The GS variable expresses or represents the dynamic properties of the process.

The *Gain scheduling variable GS* is some measured process variable which at every instant of time expresses or represents the dynamic properties of the process. As you will see in Example 14.13, GS may be the mass flow through a liquid tank.

Assume that proper values of the PID parameters K_c, T_i and T_d are found (using e.g. the Good Gain method) for a set of values of the GS variable. These PID parameter values can be stored in a parameter table – the gain schedule – as shown in Table 14.3. From this table proper PID parameters are given as functions of the gain scheduling variable, GS.

[8] What may happen if you do not adjust the controller parameters? The control system may get poor stability, or it may even become unstable.

Table 14.3: Gain schedule or parameter table of PID controller parameters

GS	K_c	T_i	T_d
GS_1	K_{c_1}	T_{i_1}	T_{d_1}
GS_2	K_{c_2}	T_{i_2}	T_{d_2}
GS_3	K_{c_3}	T_{i_3}	T_{d_3}

There are several ways to express the PID parameters as functions of the GS variable:

- **Piecewise constant**: An interval is defined around each GS value in the parameter table. The controller parameters are kept constant as long as the GS value is within the interval. This is a simple solution, but is seems nonetheless to be the most common solution in commercial controllers.

  When the GS variable changes from one interval to another, the controller parameters are changed abruptly, see Figure 14.26 which illustrates this for K_c, but the situation is the same for T_i and T_d. In Figure 14.26 it is assumed that GS values toward the left are critical with respect to the stability of the control system. In other words: It is assumed that it is safe to keep K_c constant and equal to the K_c value in the left part of the interval.

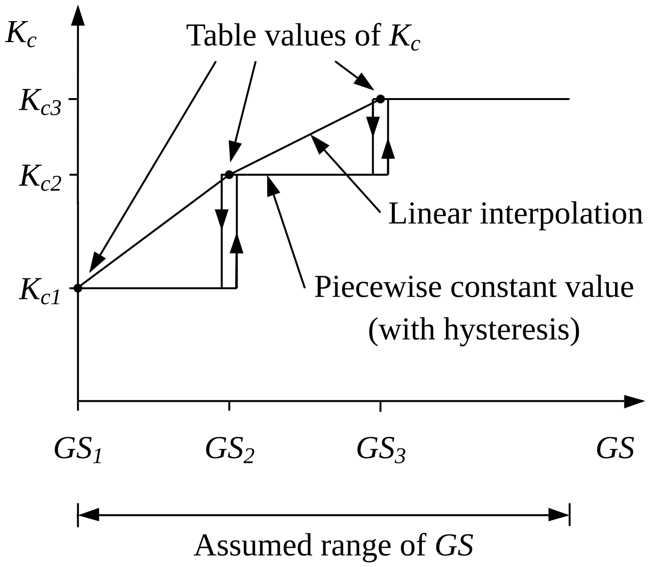


Figure 14.26: Two different ways to interpolate in a PID parameter table: Using piecewise constant values and linear interpolation.

Using this solution there will be a disturbance in the form of a step in the control variable when the GS variable shifts from one interval to a another, but this disturbance is probably of negligible practical importance for the process output variable. Noise in the GS variable may cause frequent changes of the PID parameters. This can be prevented by using a hysteresis, as shown in Figure 14.26.

- **Piecewise linear**, which means that a linear function is found relating the controller parameter (output variable) and the GS variable (input variable) between to adjacent sets of data in the table. The linear function is on the form

$$K_c = a \cdot GS + b \tag{14.121}$$

where a and b are found from the two corresponding data sets:

$$K_{c_1} = a \cdot GS_1 + b \tag{14.122}$$

$$K_{c_2} = a \cdot GS_2 + b \tag{14.123}$$

(Similar equations applies to the T_i parameter and the T_d parameter.) (14.122) and (14.123) constitute a set of two equations with two unknown variables, a and b (the solution is left to you).

- **Other interpolations** may be used, too, for example a polynomial function fitted exactly to the data or fitted using the least squares method.

Example 14.13 *PID temperature control with gain scheduling during variable mass flow*

Figure 14.28 shows the front panel of a simulator for a temperature control system for a liquid tank with variable mass flow, w, through the tank. The control variable u controls the power to heating element. The temperature T is measured by a sensor which is placed some distance away from the heating element. There is a time delay from the control variable to measurement due to imperfect blending in the tank.

The simulations in this example are made with the following SimView simulator:

http://techteach.no/simview/gain_scheduling

The process dynamics

We will initially, both in simulations and from analytical expressions, that the dynamic properties of the process *varies with the mass flow w*. The response in the temperature T after a step change in the control signal (which is proportional to the supplied power) is simulated for a large mass flow and a small mass flow. (Feedback temperature control is not active, thus open loop responses are shown.) The responses are shown in Figure 14.27.

The simulations show that the following happens when the mass flow w is reduced (from 24 to 12 kg/min): *The gain process K is larger.* It can be shown that in addition, the time constant T_t is larger, and the time delay t_d is larger. (These terms assumes that system is a first order system with time delay. The simulator is based on such a model. The model is described below.)

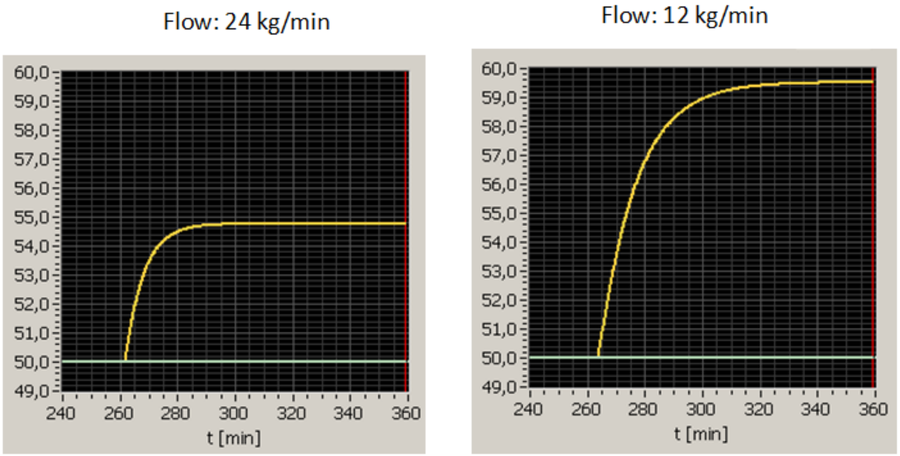


Figure 14.27: Responses in temperature T after a step in u of amplitude 10% at large mass flow and small mass flow.

Let us see if the way the process dynamics seems to depend on the mass flow w as seen from the simulations, can be confirmed from a mathematical process model.[9] Assuming perfect stirring in the tank to have homogeneous conditions in the tank, we can set up the following energy balance for the liquid in the tank:

$$c\rho V T_1'(t) = K_u u(t) + cw\left[T_{\mathrm{in}}(t) - T_t(t)\right] \tag{14.124}$$

where T_1 [K] is the liquid temperature in the tank, T_{in} [K] is the inlet temperature, c [J/(kg K)] is the specific heat capacity, V [m^3] is the liquid volume, ρ [kg/m^3] is the density, w [kg/s] is the mass flow (same out as in), K_c [W/%] is the gain of the power amplifier, u [%] is the control variable, $c\rho V T_1$ is (the temperature dependent) energy in the tank. It is assumed that the tank is isolated, that is, there is no heat transfer through the walls to the environment. To make the model more realistic, we can include a time delay t_d [s] to represent inhomogeneous conditions in the tank. Let us for simplicity assume that the time delay is inversely proportional to the mass flow:

$$t_d = \frac{K_d}{w} \tag{14.125}$$

Thus, the temperature T at the sensor is

$$T(t) = T_1\left(t - \frac{K_d}{w}\right) = T_1\left(t - t_d\right) \tag{14.126}$$

where t_d is the time delay and K_d is a constant. It can be shown that the process transfer function from control signal u to process variable T is

$$H_p(s) = \frac{T(s)}{U(s)} = \frac{K}{T_t s + 1} e^{-t_d s} \tag{14.127}$$

where

$$\text{Gain } K = \frac{K_u}{cw} \tag{14.128}$$

[9] Well, it would be strange if not. After all, we will be analyzing the same model as used in the simulator.

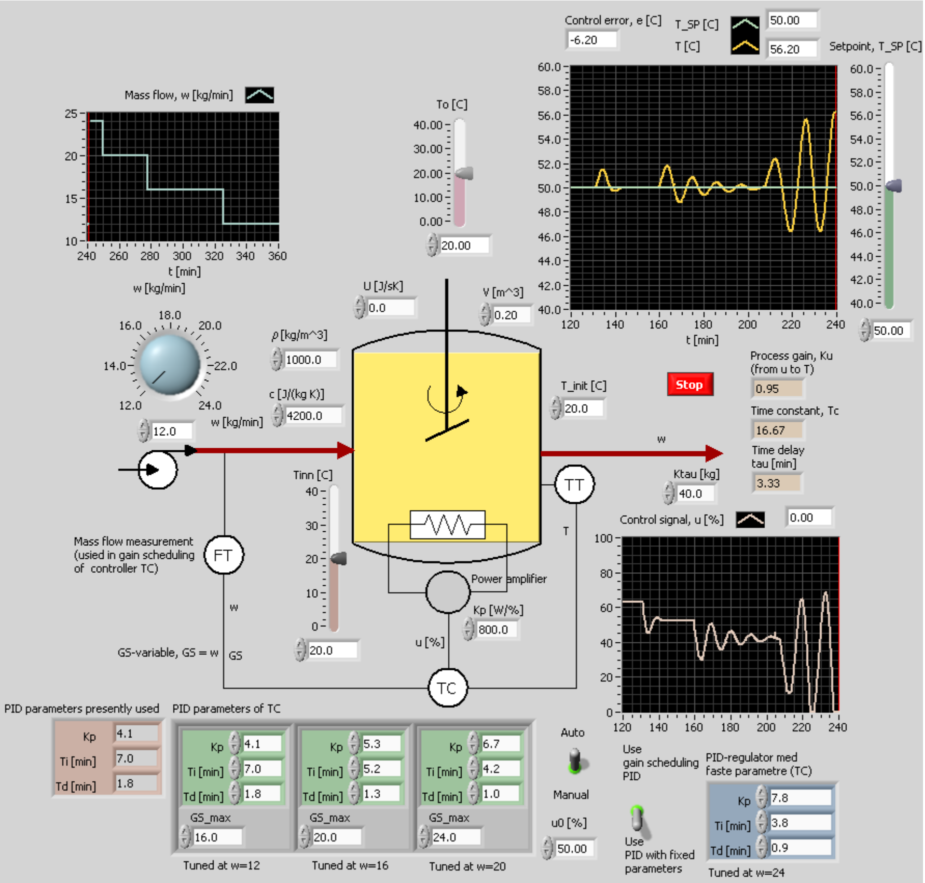
$$\text{Time-constant } t_c = \frac{\rho V}{w} \tag{14.129}$$

$$\text{Time delay } t_d = \frac{K_d}{w} \tag{14.130}$$

This confirms the observations in the simulations shown in Figure 14.27: Reduced mass flow w implies *larger process gain*, and larger time constant and larger time delay.

Heat exchangers and blending tanks in a process line where the production rate or mass flow varies, have similar dynamic properties as the tank in this example.

Control without gain scheduling (with fixed parameters)



Control error, e [C] -6.20

T_SP [C] 50.00

T [C] 56.20

Setpoint, T_SP [C] 50.00

To [C] 20.00

U [J/sK] 0.0

V [m^3] 0.20

w [kg/min] 12.0

ρ [kg/m^3] 1000.0

c [J/(kg K)] 4200.0

T_init [C] 20.0

Process gain, Ku (from u to T) 0.95

Time constant, Tc 16.67

Time delay tau [min] 3.33

Ktau [kg] 40.0

Control signal, u [%] 0.00

Tinn [C] 20.0

Kp [W/%] 800.0

u0 [%] 50.00

PID parameters presently used: Kp 4.1, Ti [min] 7.0, Td [min] 1.8

PID parameters of TC:

	Tuned at w=12	Tuned at w=16	Tuned at w=20
Kp	4.1	5.3	6.7
Ti [min]	7.0	5.2	4.2
Td [min]	1.8	1.3	1.0
GS_max	16.0	20.0	24.0

PID-regulator med faste parametre (TC): Kp 7.8, Ti [min] 3.8, Td [min] 0.9, Tuned at w=24

Figure 14.28: Example 14.13: Simulation of temperature control system with PID controller with fixed parameters tuned at maximum mass flow, which is $w = 24$kg/min.

Let us look at temperature control of the tank. The mass flow w varies. In which operating point should the controller be tuned if we want to be sure that the stability of the control system is not reduced when w varies? In general the stability of a control loop is reduced if the gain increases and/or if the time delay of the loop increases. (14.128) and (14.130) show how the gain and time delay depends on the mass flow w. According to (14.128) and (14.130) the PID controller should be tuned at minimal w. If we do the opposite, that is, tune the controller at the maximum w, the control system may actually become unstable if w decreases.

Let us see if a simulation confirms the above analysis. Figure 14.28 shows a temperature control system. The PID controller is in the example tuned at the maximum w value, which here is assumed 24 kg/min.[10] The PID parameters are

$$K_c = 7.8;\ T_i = 3.8 \text{ min};\ T_d = 0.9 \text{ min} \tag{14.131}$$

Figure 14.28 shows what happens at a stepwise reduction of w: The stability becomes worse, and the control system becomes *unstable* at the minimal w value, which is 12kg/min.

Instead of using the PID parameters tuned at maximum w value, we can tune the PID controller at minimum w value, which is 12 kg/min. The parameters are then

$$K_c = 4.1;\ T_i = 7.0 \text{ min};\ T_d = 1.8 \text{ min} \tag{14.132}$$

The control system will now be stable for all w values, but the system behaves sluggish at large w values. (Responses for this case is however not shown here.)

Control with gain scheduling

Let us see if gain scheduling maintains the stability for varying mass flow w. The PID parameters will be adjusted as a function of a measurement of w since the process dynamics varies with w. Thus, w is the gain scheduling variable, GS:

$$GS = w \tag{14.133}$$

A gain schedule consisting of three PID parameter value sets will be used. The PID controller are tuned at the following GS or w values: 12, 16 and 20 kg/min. These three PID parameter sets are shown down to the left in Figure 14.28. The PID parameters are held piecewise constant in the GS intervals. In each interval, the PID parameters are held fixed for an increasing $GS = w$ value, cf. Figure 14.26.[11] Figure 14.29 shows the response in the temperature for decreasing values of w.

The simulation shows that the *stability of the control system is maintained even if w decreases.*

Finally, assume that you have decided not to use gain scheduling, but instead a PID controller with fixed parameter settings. What is the most critical operating point, at which the controller should be tuned? Is it at maximum flow or at minimum flow?[12]

[End of Example 14.13]

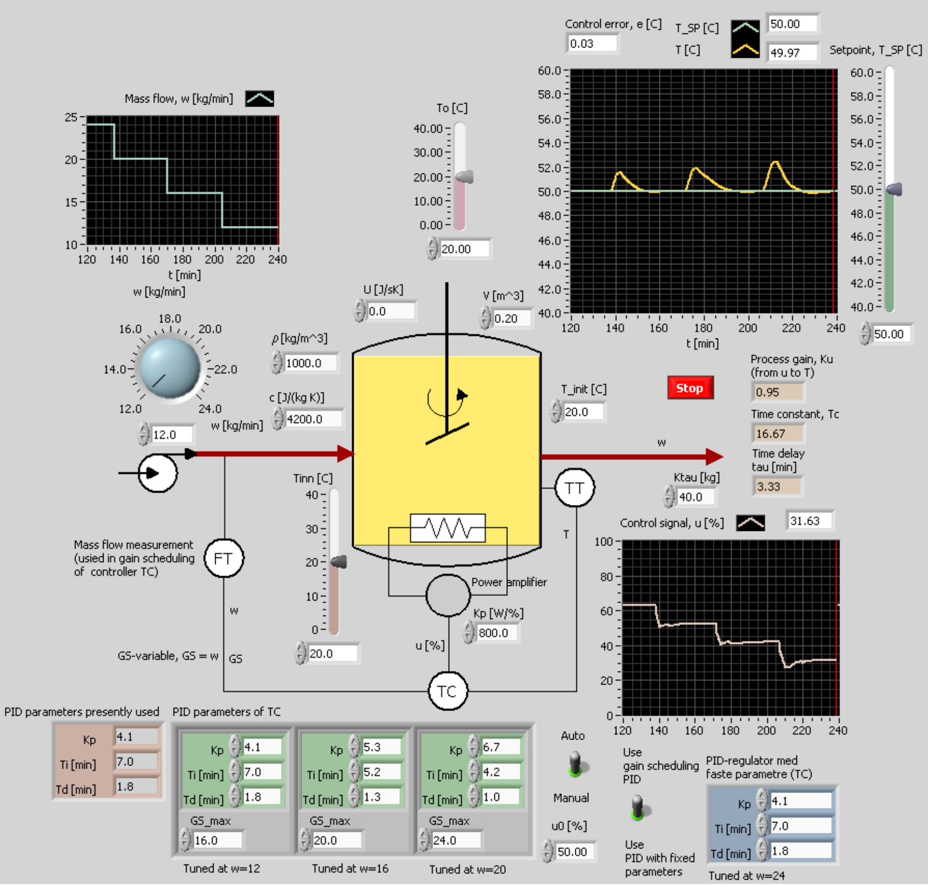
14.10.4 Adaptive controller

In an adaptive control system, see Figure 14.30,

[10] Actually, the controller was tuned with the Ziegler-Nichols Ultimate Gain method. This method is however not described in this book. The Good Gain method could have been used instead.

[11] The simulator uses the inbuilt gain schedule in LabVIEW's PID Control Toolkit.

[12] The answer is minimum flow, because at minimum flow the process gain is at maximum, and also the time delay (transport delay) is at maximum.



Control error, e [C] 0.03
T_SP [C] 50.00
T [C] 49.97
Setpoint, T_SP [C] 50.00
To [C] 20.00
U [J/sK] 0.0
V [m^3] 0.20
w [kg/min] 12.0
ρ [kg/m^3] 1000.0
c [J/(kg K)] 4200.0
T_init [C] 20.0
Process gain, Ku (from u to T) 0.95
Time constant, Tc 16.67
Time delay tau [min] 3.33
Ktau [kg] 40.0
Control signal, u [%] 31.63
Tinn [C] 20.0
Kp [W/%] 800.0
u0 [%] 50.00

	PID parameters presently used	Tuned at w=12	Tuned at w=16	Tuned at w=20	Tuned at w=24
Kp	4.1	4.1	5.3	6.7	4.1
Ti [min]	7.0	7.0	5.2	4.2	7.0
Td [min]	1.8	1.8	1.3	1.0	1.8
GS_max		16.0	20.0	24.0	

Figure 14.29: Example 14.13: Simulation of temperature control system with a gain schedule based PID controller

a mathematical model of the process to be controlled is continuously estimated from samples of the control signal (u) and the process measurement (y). The model is typically a transfer function model. Typically, the structure of the model is fixed. The model parameters are estimated continuously using e.g. the least squares method. From the estimated process model the parameters of a PID controller (or of some other control function) are continuously calculated so that the control system achieves specified performance in form of for example stability margins, poles, bandwidth, or minimum variance of the process output variable (Åstrøm & Wittenmark 1994). Adaptive controllers are commercially available, for example the ECA60 controller (ABB).

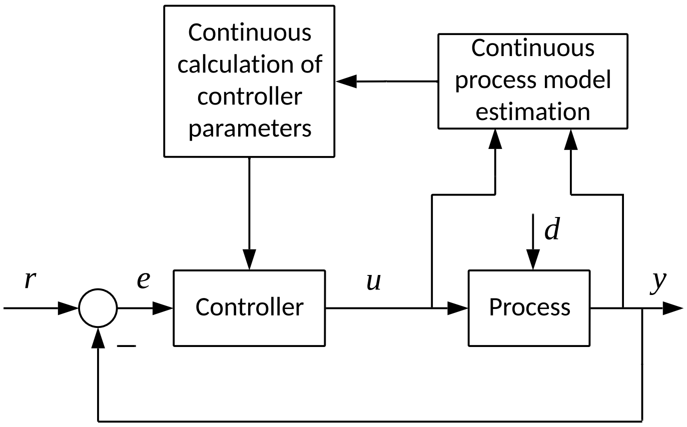

Continuous calculation of controller parameters
Continuous process model estimation
d
r
e
Controller
u
Process
y
−

Figure 14.30: Adaptive control system.

14.11 *Problems for Chapter 14*

Problem 14.1 *Ziegler-Nichols PI tuning*

Assume that in a Ziegler-Nichols experiment for tuning a PI controller, the control system oscillates with sustained oscillations of period 12 seconds when the controller gain is 10.

1. Find the PI controller settings with the original Ziegler-Nichols method.
2. Find the PI controller settings with the Relaxed Ziegler-Nichols settings.

Problem 14.2 *Relay tuning method*

Tune a PI controller for a process with the Relay tuning method using Ziegler-Nichols PI settings given the following data from an experiment with On-off control of the process.

- Period of oscillations: $t_p = 24$ s.
- Amplitude of square wave oscillations of the control signal: $A_u = 50$ %.
- Amplitude of sinusoidal oscillations of the process measurement: $A_y = 5$ %.

Problem 14.3 *Good gain PI tuning*

Figure 14.31 shows the response in the temperature of a simulated temperature control system with P controller with the following “good gain” value:

$$K_c = 4.0 \tag{14.134}$$

Tune a PI controller for this process using the Good Gain method.

What can you do with the controller tuning if turns out that the stability of the control system is too bad with this value of K_c?

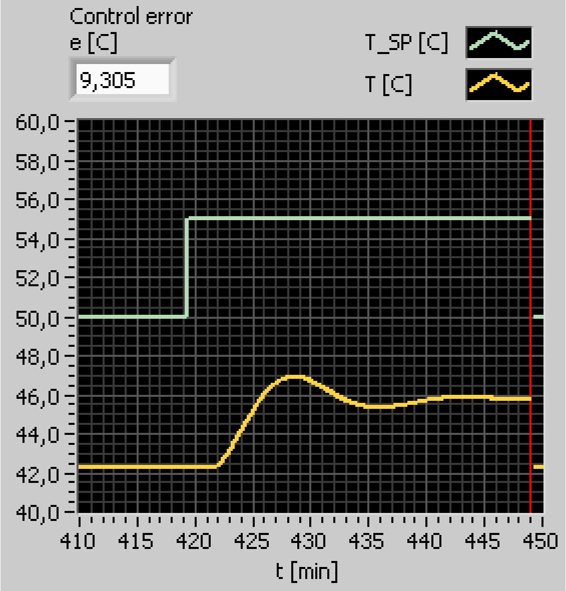


Figure 14.31: Temperature response

Problem 14.4 *Tuning of controller for "time constant with time delay" process*

Given a process where the relation between the control signal and the process measurement can be well represented as "time constant with time delay" with gain $K = 0.5$, time constant $t_{cp} = 5$ s, and time delay $t_d = 1$ s.

1. Tune a PI controller for this process using the SIMC method assuming "time constant with time delay" process dynamics.
2. Tune a PI controller for this process using the SIMC method assuming "integrator with time delay" process dynamics. Compare with the result of Problem 1.

Problem 14.5 *SIMC PI tuning from process step response*

Figure 14.32 shows the response in the process measurement y_m due to a step of amplitude $U = 2$ in the control signal u at time $t = 0$ s. Calculate settings for a PI controller for this process.

Problem 14.6 *Controller tuning when process dynamics varies*

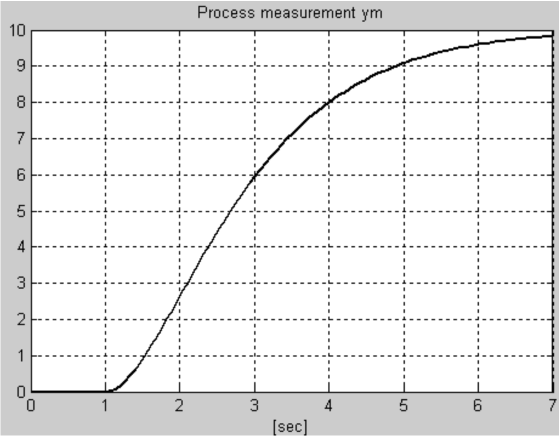


Figure 14.32: Process measurement step response for SIMC tuning.

Assume that the process to be controlled has varying process dynamics, which may cause stability problems or sluggish control. Both of the solutions A and B below are possible. Which is the best one with respect to control performance, and which is the simplest one?

A: The controller is tuned at the most critical operating point, and the controller parameters are then kept constant.

B: The controller parameters are adjusted continually so that they fit to the dynamic properties of the process at any operating point.

Problem 14.7 *In which operating point to tune the controller?*

Figure 14.33 shows a chemical reactor and a PID parameter table which is the basis of a PID controller with gain scheduling. Assume that gain scheduling is not to be used, but fixed PID settings instead. Should the controller be tuned at high temperature or at low temperature, given that it is crucial that the stability of the control system is satisfactory at any temperature?

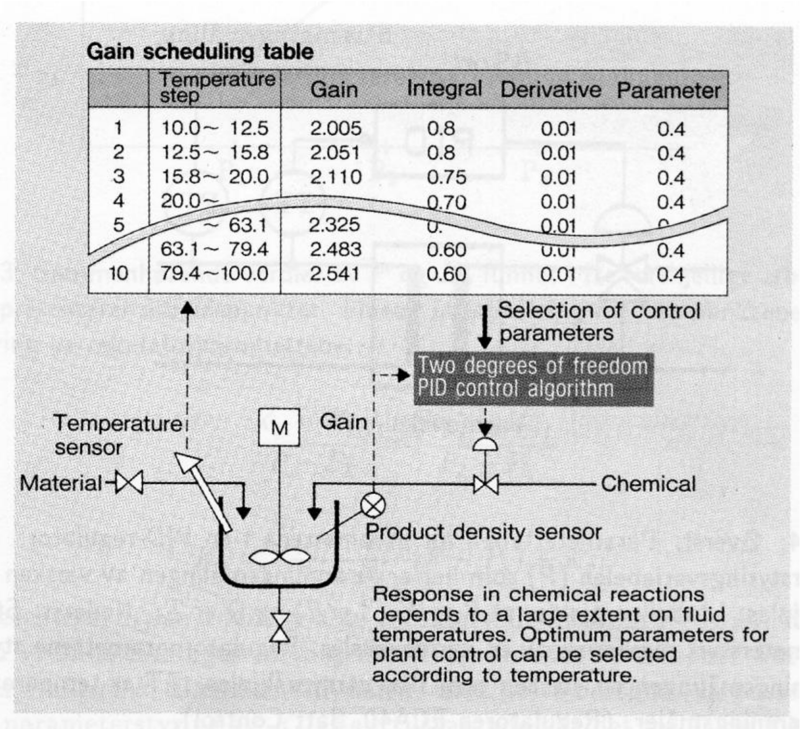
Problem 14.8 *Interpolation in a gain scheduling table*

Table 14.4 shows parts of a gain scheduling based PID controller.

Table 14.4: PID Gain Schedule

GS	K_c	T_i	T_d
$\vdots$	$\vdots$	$\vdots$	$\vdots$
20 %	0.4	5.2	1.3
30 %	0.5	4.5	1.6
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Find K_c as a function of the gain scheduling variable GS between the operating points shown in the table. The function should be based on linear interpolation.



Gain scheduling table

	Temperature step	Gain	Integral	Derivative	Parameter
1	10.0~ 12.5	2.005	0.8	0.01	0.4
2	12.5~ 15.8	2.051	0.8	0.01	0.4
3	15.8~ 20.0	2.110	0.75	0.01	0.4
4	20.0~		0.70	0.01	0.4
5	~ 63.1	2.325	0.	0.01	0.
	63.1~ 79.4	2.483	0.60	0.01	0.4
10	79.4~100.0	2.541	0.60	0.01	0.4

Selection of control parameters
Two degrees of freedom PID control algorithm
Temperature sensor
M
Gain
Material
Chemical
Product density sensor

Response in chemical reactions depends to a large extent on fluid temperatures. Optimum parameters for plant control can be selected according to temperature.

Figure 14.33: Chemical reactor and PID parameter table

14.12 *Solutions to problems for Chapter 14*

Solution to Problem 14.1

1. Original Ziegler-Nichols settings:

$$K_c = 0.45 \cdot 10 = 4.5 \tag{14.135}$$

$$T_i = 12/1.2 = 10 \text{ s} \tag{14.136}$$

2. Relaxed Ziegler-Nichols settings:

$$K_c = 0.25 \cdot 10 = 2.5 \tag{14.137}$$

$$T_i = 1.25 \cdot 12 = 15 \text{ s} \tag{14.138}$$

Solution to Problem 14.2

Calculation of ultimate gain:

$$K_{cu} = 1.27 \cdot \frac{A_u}{A_y} = 1.27 \cdot \frac{50\ \%}{5\ \%} = 12.7\ \frac{\%}{\%} \tag{14.139}$$

Ziegler-Nichols PI tuning:

$$K_c = 0.45 \cdot 12.7\ \frac{\%}{\%} = 5.72\ \frac{\%}{\%}$$

$$T_i = \frac{t_p}{1.2} = \frac{24}{1.2} = 20 \text{ s}$$

Solution to Problem 14.3

We set the controller gain to

$$K_c = 0.8 \cdot 4.0 = 3.2 \tag{14.140}$$

In Figure 14.34 we read off

$$t_{\rm ou} = 8.0 \text{ min} \tag{14.141}$$

which gives the integral time

$$T_i = 1.5 t_{\rm ou} = 12.0 \text{ min} \tag{14.142}$$

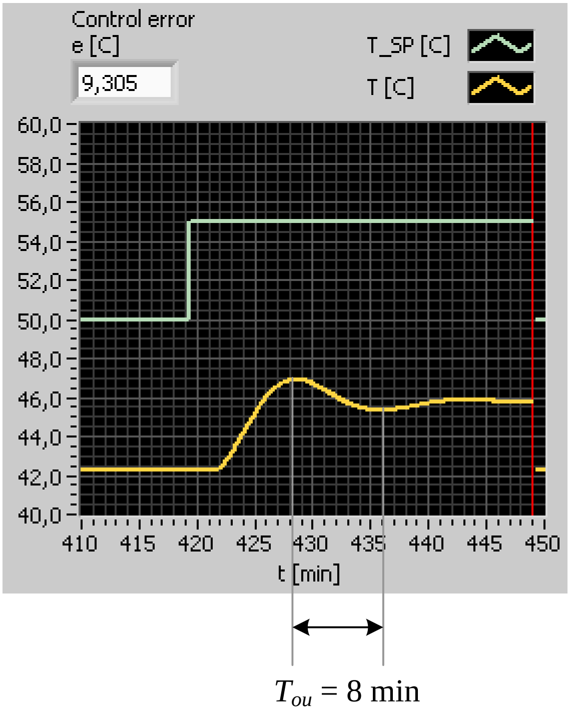


Figure 14.34: Problem 14.3: Good Gain response.

If it turns out that the stability of the control system is too bad with this value of K_c, you can try reducing the gain and/or increasing the integral time.

Solution to Problem 14.4

1. With the SIMC PI tuning formulas for “time constant with time delay”, (14.80)-(14.81), and using Skogestad’s hand-rule in his SIMC method:

$$t_{cc} = t_d = 1 \text{ s} \tag{14.143}$$

the PI settings become:

$$K_c = \frac{t_{cp}}{2Kt_d} = \frac{5}{2 \cdot 0.5 \cdot 1} = 5 \tag{14.144}$$

$$T_i = \min\left[T, \ 4t_d\right] = \min\left[5, \ 4 \cdot 1\right] = \min\left[5, \ 4\right] = 4 \text{ s} \tag{14.145}$$

Comment: Above, $\min[x, y]$ means the minimum of the two numbers x and y.

2. The SIMC PI tuning formulas for "integrator with time delay", (14.43)-(14.44), with the SIMC hand-rule

$$t_{cc} = t_d \tag{14.146}$$

are:

$$K_c = \frac{1}{2K_i t_d} \tag{14.147}$$

$$T_i = 4t_d \tag{14.148}$$

Here, K_i is the normalized steepest slope of the process step response. For a "time constant" system, the steepest slope is equal to the initial slope, which appears just after the point of time of the input step. The initial slope is given by (9.37), which is repeated here:

$$S = \frac{KU}{t_{cp}} \tag{14.149}$$

From (14.149) we get

$$K_i = \frac{S}{U} = \frac{KU}{t_{cp}}/U = \frac{K}{t_{cp}} \tag{14.150}$$

With this K_i, (14.147) and (14.148) become:

$$K_c = \frac{1}{2K_i t_d} = \frac{t_{cp}}{2Kt_d} = \frac{5}{2 \cdot 0.5 \cdot 1} = 5 \tag{14.151}$$

$$T_i = 4t_d = 4 \text{ s} \tag{14.152}$$

which are the same PI settings as in Problem 1 above. This is in accordance with a fact presented in Ch. 14.8.4.2: If $t_d > t_{cp}/4$, the PI settings for "time constant with time delay" processes and for "integrator with time delay" processes are identical.

Solution to Problem 14.5

Figure 14.35 shows the steepest tangent drawn on the process step response. From the two points labelled P2 and P3, the slope of the tangent can be calculated as:

$$S = \frac{(10.0 - 0.0)}{(4.0 - 1.3)} = 3.7 \tag{14.153}$$

The normalized slope is:

$$K_i = \frac{S}{U} = \frac{3.7}{2} = 1.85 \tag{14.154}$$

From the two points labelled P1 and P2, the (approximate) time delay is:

$$t_d = 1.3 - 0.0 = 1.3 \text{ s} \tag{14.155}$$

The PI settings become:

$$K_c = \frac{1}{2K_i t_d} = \frac{1}{2 \cdot 1.85 \cdot 1.3} = 0.21 \tag{14.156}$$

$$T_i = 4t_d = 4 \cdot 1.3 = 5.2 \text{ s} \tag{14.157}$$

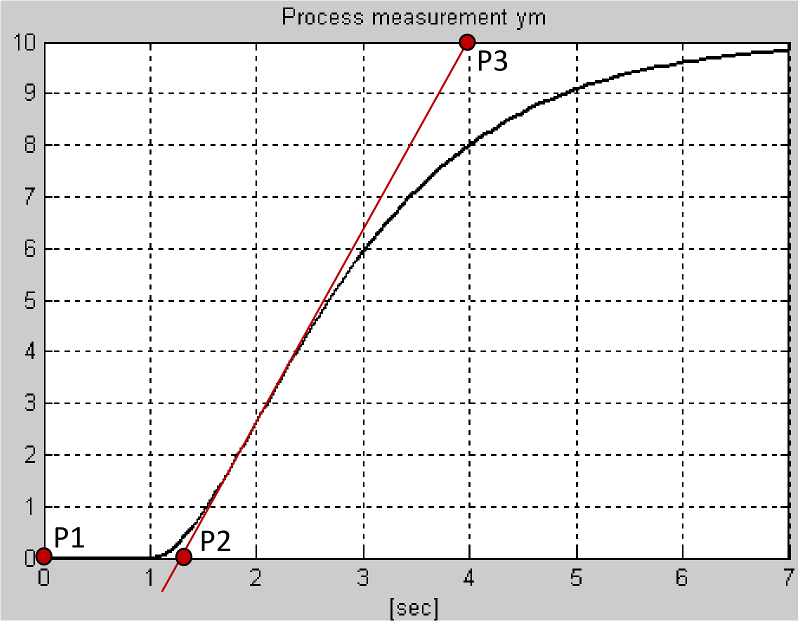


Figure 14.35: Problem 14.5: Steepest tangent drawn on the process step response.

Solution to Problem 14.6

For best control performance: B. Simplest: A.

Solution to Problem 14.7

From the table shown in Figure 14.33 we see that Gain = K_c has (should have) less value and Integral = T_i has (should have) larger value the lower the temperature. This indicates that minimum temperature is “worst case”. Therefore, a PID controller with fixed settings should be tuned at $\underline{\underline{\text{miminum temperature}}}$.

Solution to Problem 14.8

$$K_c = \frac{0.5 - 0.4}{30\% - 20\%}(GS - 20\%) + 0.4 = 0.01\%^{-1} \cdot GS + 0.2 \tag{14.158}$$

Chapter 15

Experimental analysis of control loop stability

15.1 Heuristic stability analysis

It is important to be aware that there may be stability problems in a control loop. It is a basic requirement to a control loop that it is stable. Simply stated this means that the response in any signal in control loop converges towards a finite value after a limited change (with respect to the amplitude) of the reference or the disturbance or any other input signal to the loop.

There is always a possibility that a feedback control system which is originally stable, *may become unstable* due to parameter changes in the loop. Instability implies that signals in the loop starts to increase in amplitude until some saturation limit is reached (for example, a valve have a limited opening).

As a start of the discussion of how a feedback control system can become unstable, let us repeat the block diagram of a feedback control system from the beginning of this book, see Figure 15.1.

Instability in a feedback system can be explained in two ways:

- **Too large gain in the loop**: The loop gain is the product of the gains in each of the subsystems (controller, process, sensor) in the loop. If the signal in the loop is amplified too much through the loop, it “comes back” amplified. Then, this signal is again amplified through the loop, and eventually the amplitude just increases. In other words, the control loop is unstable.
- **Too large time delay in the loop**. The effect or response due to the controller action is fed back to the controller with too much time delay so that the information in the response is out-of-date and does not reflect the state of the process good enough. Based on this poor information, the controller may generate too much or too little control action. This repeats through the loop, causing larger and larger

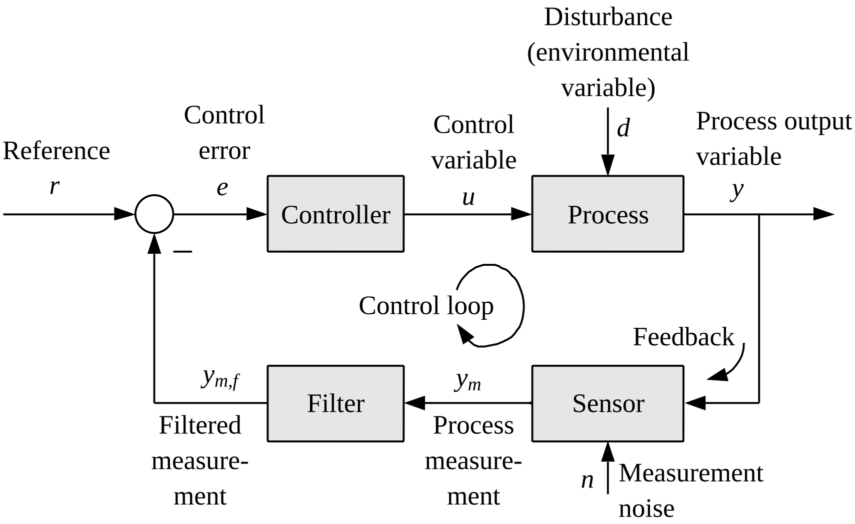


Figure 15.1: Block diagram of a feedback control system.

deviation from the reference. In other words, the control loop is unstable.

- **Too large time constant in the loop**. The effect of a too large time constant is similar to that of the a too large time delay: The response due to a change in the controller output come back to the controller with a large lag, which resembles a time delay. Furthermore, the signal is smoothed due to the filtering effect of a time constant. Both these effects cause the controller to get a relatively "false" information about the process, and it may generate a too large control signal which gives an over-compensation, eventually causing oscillations to occur. Note: The impact of a large time constant on the loop stability as described above, assumes that the controller has integral action, as in a PI or PID controller.

Above, I wrote that there will be stability problems at "too large gain", "too large time delay", and "too large time constant" in the loop. But note that you can obtain good loop stability with a successful controller tuning. For example, whatever the value of the time delay of the process, you can tune the controller so that the resulting loop stability is ok. What will be dangerous for the loop stability, is if the time delay *increases* after you have done the controller tuning!

Example 15.1 *Instability in the wood chips tank level control system*

We will here study the level control system of the wood chip tank described in Example 1.1. The simulations in the present example are made with the following SimView simulator:

http://techteach.no/simview/level_control_woodchips_tank

The controller is a PI controller with SIMC settings as found in Example 14.6:

$$K_c = 3.89 \text{ (kg/s)/m} \tag{15.1}$$

$$T_i = 1000 \text{ s} \tag{15.2}$$

Figure 15.2 shows the simulated level response due to a step change of the level reference. The simulations show that the stability of the control system is ok with the above PI settings. In the following we will impose several parameter changes in the loop which will cause stability problems.

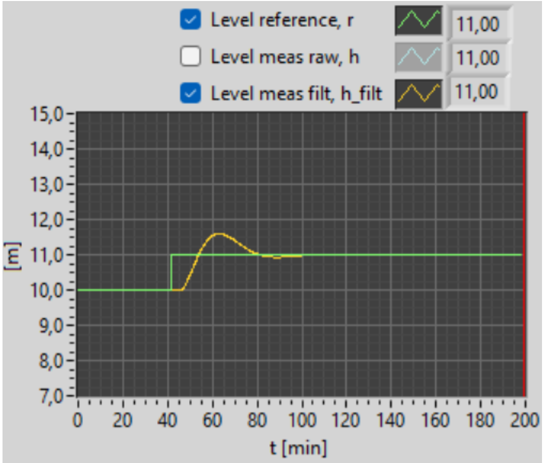


Figure 15.2: Level response due to a step change of the level reference with good PI settings, $K_c = 3.89$ and $T_i = 1000$ s. The stability of the control system is ok.

Increase of loop gain

Increase of controller gain

As the first example of increased loop gain, let's assume we were unlucky to apply an increase in the controller gain, K_c, so that it increases from from the good value of 3.89 to 10.0. Figure 15.3 shows the level response due to a step change of the level reference at $t = 80$ min. The control system has become unstable!

Can we get the stability back by retuning the controller? Of course. We just set K_c back to its good value, 3.89, and the step response will again be as in Figure 15.2.

Reduction of tank area

As the second example of increased loop gain, let's assume that the company has replaced the original tank where the cross-sectional area is $A = 13.4$ m^2, with a more narrow tank, with halved area: $A = 13.4/2 = 6.7$ m^2. This change implies that the level becomes twice as

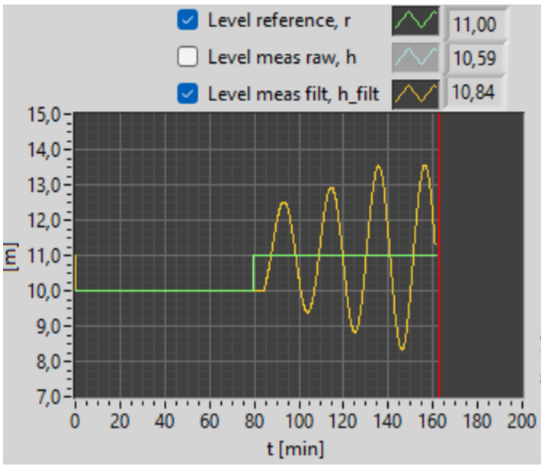


Figure 15.3: The control system is unstable due to a too large loop gain due to an increase of the controller gain, K_c.

sensitive to inflow changes, and therefore, the process gain increases by a factor of 2. This can also be seen also from the process model: The model, which is based on mass balance of the wood chips in the tank, is presented in Appendix 39.2. It is repeated here:

$$\rho A h'(t) \quad = \quad u(t-t_d) - F_{\text{out}}(t) \tag{15.3}$$

From this differential equation we can derive the following process transfer function from u to h:

$$H_p(s) = \frac{H(s)}{U(s)} = \frac{1}{\rho A s}e^{-t_d s} = \frac{K_i}{s}e^{-t_d s} \tag{15.4}$$

which is an “integrator with time delay” transfer function with integral gain

$$K_i = \frac{1}{\rho A} \tag{15.5}$$

Since K_i is inversely proportional with A, K_i. Therefore, the process gain is doubled if A is halved.

Figure 15.4 shows the level response due to a step change of the level reference at $t = 40$ min for this new tank. The control system now has poor stability, although it is still stable.

Can we get back good loop stability? Not by setting back the original tank, but by retuning the PI controller? To this end, recall the SIMC PI settings, (14.43) – (14.44), repeated here:

$$K_c = \frac{1}{2K_i t_d}; \; T_i = 4t_d \tag{15.6}$$

If A is halved, K_i is doubled. (15.6) shows how to adjust the PI settings to retain ok stability if any of the parameters K_i or t_d is changed, assuming the control system has ok stability before the change. From (15.6): If K_i is doubled, K_c should be reduced by a factor of 2, i.e., from 3.89 to 1.95, while T_i should not be kept unchanged since the time de lay, or transport time, t_d, does not depend on A. A simulation with the adjusted PI settings confirms that the control system now has good stability, but I do not show the responses here.

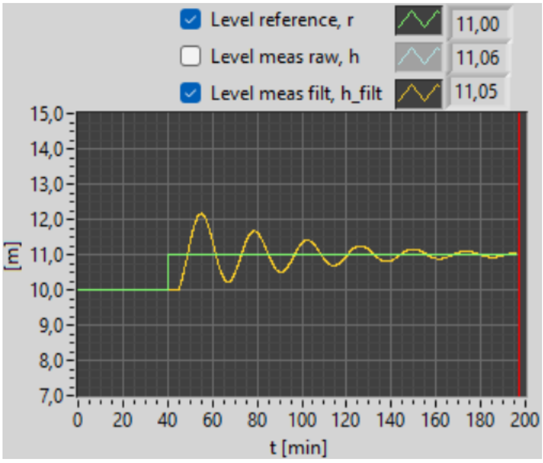


Figure 15.4: The control system has poor stability due to the increase of the loop gain caused by the decrease of the tank cross-sectional area, A.

Increase of loop time delay

Figure 15.5 shows the reference step response in the level when the time delay of the conveyor belt has been increased from the nominal value of 250 s (= 4.17 min) to 500 s (= 8.33 min), i.e. by a factor of 2. The controller has the original settings of $K_c = 3.89$ and T_i =1000 s. This increase of the process time delay gives, of course, the same increase in the control loop time delay. From the simulation in Figure 15.5 we see that the control system has become unstable, which is due to the increase of the time delay.

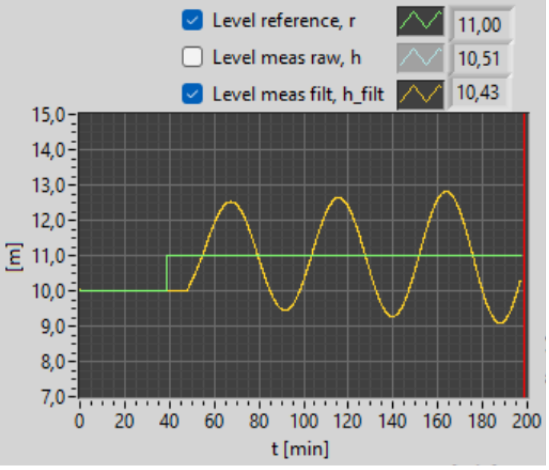


Figure 15.5: Increased time delay: The level control system has become unstable when the time delay (transportation time) of the conveyor belt is increased from the nominal value of 250 s to 500 s.

Can we get the stability back by retuning the controller? From the SIMC PI settings (15.6) we see that if the time delay, t_d, is increased by a factor of 2 as in our example, K_c should be reduced by a factor of 2, from 3.89 to 1.95. And T_i should be increased by a factor of to, from 1000 s to 2000 s. A simulation with such adjusted PI settings indeed confirms that the control system has good stability, but I do not show the responses here.

Increase of loop time constant

In this level control system, the only time constant subsystem is the measurement filter. We will study the effect on control system stability of increasing the filter time constant, t_f. Let's increase it from 20 s to 520 s, i.e. an increase of 500 s. (This particular increase simplifies calculations in this example.) The controller has the orginal settings of $K_c = 3.89$ and T_i =1000 s. Figure 15.6 shows the level response due to a step change of the reference with this t_f = 520 s. The yellow curve in Figure 15.6 is the level measurement signal after the filter, i.e. the filtered measurement, while the blue curve is the level measurement signal before the filter, i.e. the raw measurement. (We see that the filtered measurement has a big lag compared to the unfiltered measurement. The lag is due to the sluggish filter.) The increase in time constant has made the control loop unstable!

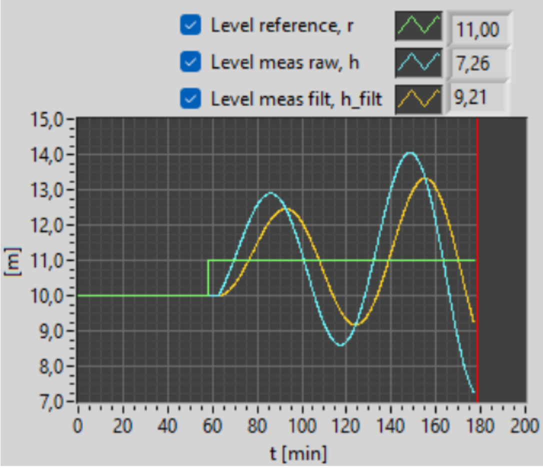


Figure 15.6: Large process time constant: The level control system has become close to unstable when the measurement filter time constant t_f is increased from the nominal value of 20 s to 520 s.

Can we get the stability back by retuning the controller? Recall the SIMC PI settings, (15.6). Now we have a challenge, since there is no time constant in (15.6). However, we can try *approximating* the time constant with a time delay. We have increased the time constant, t_f, by 500 s. So, we pretend this is an increase of the time delay by 500 s. Originally, the time delay was 250 s. Thus, we regard the increase of t_f by 500 s as an increase in t_d from 250 to 750 s, which is an increase by factor 3. According to (15.6), a 3 times larger t_d implies a reduction of K_c by factor 3 and an increase of T_i by factor 3. Therefore, the new PI settings are: $K_c = 3.89/3 = 1.30$ and $T_i = 3 \cdot 1000 = 3000$ s. A simulation with such adjusted PI settings confirms that the control system has good stability, but I do not show the responses here.

[End of Example 15.1]

Note the trick

Note the particular trick used in case of increased time constant in Example 15.1: Pretending that a time constant is a time delay! Such an approximation can be useful when retuning controllers after an increase in the filter time constant in the control loop. This approximation is safe (or robust or conservative) because an increase in the time delay has a larger (worse) impact on dynamics and stability of a control loop than an increase in the time constant has.

15.2 Experimental gain margin (GM) and phase margin (PM)

In Section 15.1 we saw that a feedback control system gets worse stability

- if the loop gain in the loop becomes larger, and/or
- if the time delay in the loop becomes larger.

A stability margin indicates how much change in the parameters in a (feedback) control system that can be tolerated before the control system comes to the stability limit, i.e. becomes marginally stable. When the control system is marginally stable, the system variables shows undamped oscillations.

In the feedback control theory, there are two common stability margins:

- **Gain margin GM**, which expresses how large increase of the loop gain that feedback loop can tolerate before it becomes marginal stable.
- **Phase margin PM**, which (indirectly) expresses how large increase of the loop time delay that the feedback loop can tolerate before it becomes marginally stable.

GM and PM can be calculated from a linear mathematical model of the control system, cf. Ch. 22.3. GM and PM can also be found experimentally on a real or simulated control system (the simulator model may be nonlinear), as explained in the following.

Figure 15.7 shows a feedback control system where two special blocks for the purpose of stability analysis have been included:

- One block with a multiplicative gain ΔK
- One block with an additional time delay increase Δt_d.

You can find GM and PM as explained below.

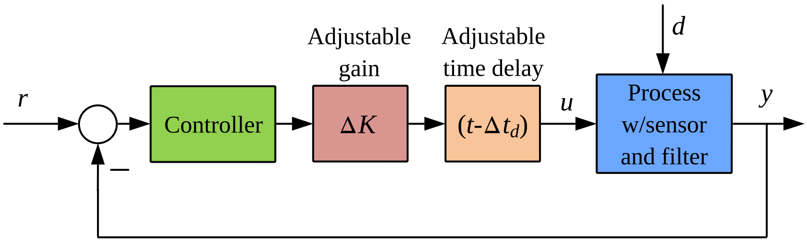


Figure 15.7: A feedback control system with blocks of multiplicative gain increase $\varDelta K$ and additional time delay increase $\varDelta t_d$.

- **GM**: Let Δt_d be 0, i.e. no time delay increase. Start with $\Delta K = 1$, i.e. no gain increase to start with. By trial-and-error, find the value of ΔK, here denoted ΔK_u (u for ultimate), that makes the control system become marginally stable (oscillatory). This gives
$$\text{GM} = \Delta K_u \tag{15.7}$$
- **PM**: Let ΔK be 1, i.e. no gain increase. Start with $\Delta t_d = 0$, i.e. no time delay increase to start with. By trial-and-error, find the value of Δt_d, here denoted $\Delta t_{d,u}$ (subindex u for ultimate), that makes the control system become marginally stable (oscillatory). It can be shown that
$$\text{PM} = \frac{\Delta t_{d,u}}{t_p} \cdot 360^\circ \tag{15.8}$$
where t_p [s] is the period of the oscillations. (15.8) is derived at the end of this section, after Example 15.2. Comment: PM measured in degrees originates from traditional frequency response theory of feedback systems.

Acceptable intervals for GM and PM are given in Seborg et al. (2004) as15.2

$$1.7 \leqslant \text{GM} \leqslant 4.0 \tag{15.9}$$

$$30^\circ \leqslant \text{PM} \leqslant 45^\circ \tag{15.10}$$

Since too poor stability *must* be avoided, while too good stability *may* be accepted, the lower limits of GM and PM are critical, while the upper limits for GM and PM are recommended, but not critical. In other words: GM should not be less than 1.7 and PM not less than 30°, while there is no disaster if GM is greater than 4.0 or if PM is greater than 45°.

Practical problems of including the extra blocks (or functions)?

Figure 15.7 contains a gain block and a time delay block where we adjust ΔK and Δt_d to find marginal stability. If, for some reason, it is difficult to include two such blocks, you can use the following alternative method to find ΔK and Δt_d (that procedure is demonstrated in Example 15.2):

- Finding ΔK_u: We adjust the *controller gain* until the feedback loop becomes marginally stable. Then, ΔK_u is equal to the multiplicative increase of the controller gain.
- Finding $\Delta t_{d,u}$: It is here assumed here that there is a time delay somewhere in the feedback loop, e.g. in the process, that can be adjusted. Then, $\Delta t_{d,u}$ is equal to the additive increase of the time delay that makes the feedback loop become marginally stable. Note: If there is no time delay in the loop at all and you can not include a time delay in the loop, either, there will not be any time delay to adjust. In such a case you must simply drop finding PM.[1]

Example 15.2 *Stability margins of the level control system of the wood chips tank*

We will again study the level control system of the wood chip tank described in Example 1.1 The controller is a PI controller with SIMC settings as found in Example 14.6:

$$K_c = 3.89 \tag{15.11}$$

$$T_i = 1000\,\text{s} \tag{15.12}$$

We will find the GM and PM of the control system from simulations with the following SimView simulator:

http://techteach.no/simview/level_control_woodchips_tank

To find GM, we adjust the regulator gain, K_c, in a simulation. To find PM, we adjust the conveyor time delay (transportation time), t_d, in a simulation. Initially, $t_d = 250$ s.

- **GM**: The control system becomes marginally stable with $K_c = 9.05$. Figure 15.8 shows the simulation with this value of K_c. The gain margin becomes

$$\text{GM} = \Delta K_u = \frac{9.05}{3.89} = 2.33$$

  which is an acceptable value since it is within the limits in (15.9).

- **PM**: Before the time delay is adjusted in a simulation, we reset K_c to 3.89. The control system becomes marginally stable with $t_d = 485$ s. Figure 15.9 shows the simulation with this value of t_d. The period of the oscillations is measured as $t_{p,u} = 47$ min $= 2820$ s. The phase margin becomes

$$\begin{aligned} \text{PM} &= \frac{\Delta t_{d,u}}{t_p} \cdot 360^\circ && (15.13)\\ &= \frac{(485\,\text{s} - 250\,\text{s})}{2820\,\text{s}} \cdot 360^\circ && (15.14)\\ &= 30.0^\circ && (15.15) \end{aligned}$$

  which is just acceptable as it is equal to the lower limit of the range cf. (15.10).

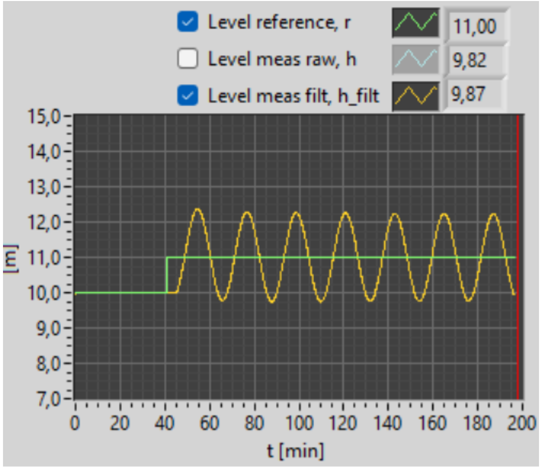


Figure 15.8: The control system is marginally stable with $K_c = 9.05$.

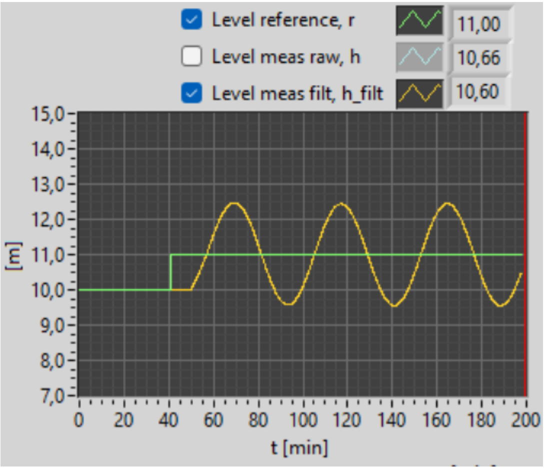


Figure 15.9: The control system is marginally stable with $t_d = 485$ s.

[End of Example 15.2]

Derivation of (15.8):

I will now derive (15.8) from the theory of stability analysis of linear feedback systems based on frequency response (this theory is presented in Ch. 22). See Figure 15.7. Assume that the feedback system is stable with $\Delta t_d = 0$ (and $\Delta K = 1$), and that Δt_d is then increased to the value $\Delta t_{d,u}$ that makes the system become marginally stable. In general, when the time delay in the loop is increased, the amplitude gain characteristic is unchanged, and in particular the gain crossover frequency, ω_c [rad/s], is unchanged, while the phase characteristic is reduced by $\omega \Delta t_d$ [rad] where ω [rad/s] is frequency. In frequence response theory, the phase margin, PM, is defined as the phase reduction (given as a positive value) which causes the phase of the loop to become $-180°$ at the gain crossover frequency, ω_c.

[1] With a simulator, we should always be able to include a time delay, but it can be difficult to do it in a real feedback control system.

Since this phase reduction stems from the time delay increase only, the phase margin is

$$\text{PM} = \omega_c \Delta t_{d,u} \text{ [rad]} \tag{15.16}$$

or

$$\text{PM} = \omega_c \Delta t_{d,u} \cdot \frac{180}{\pi} \text{ [deg]} \tag{15.17}$$

When the system is marginally stable, its response in the time domain is oscillatory, and the frequency of the oscillations is equal to ω_c (because the purely imaginary poles, $\pm j\omega_c$, are among the poles of the system). ω_c is related to the period, t_p [s], of the oscillations as follows:

$$\omega_c = \frac{2\pi}{t_p} \tag{15.18}$$

Finally, combining (15.18) and (15.17) gives (15.8), which is what I wanted to derive.

15.3 *Problems for Chapter 15*

Problem 15.1 *PB and stability*

How will the stability of the control loop change if the proportional band PB is reduced?

Problem 15.2 *Control system stability at process changes*

Figure 15.10 shows a temperature control system with two different positions of the temperature sensor. Assume that the temperature controller is tuned so that the stability of the control system is satisfactory with the sensor in position 1.

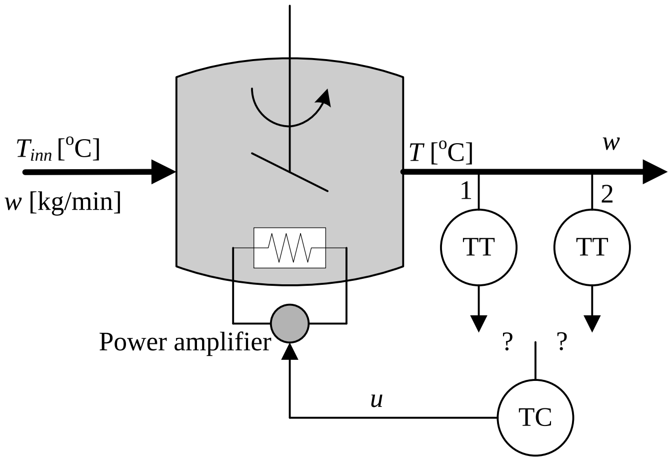


Figure 15.10: Temperature control system

1. What wil happen to the stability if the sensor is moved to position 2?
2. Assume that the sensor is in position 1. How will the stability change if the liquid flow decreases?
3. How will the stability change if the sensor gain is increased (the gain is the ratio of the sensitivity of the measurement signal in volts or amperes to the temperature)?
4. How will the stability change if the heater is substituted by a heater delivering more power per unit of the control signal?

Problem 15.3 *Explain the unstable response!*

A temperature control system is described in Problem 1.10. Figure 15.11 shows the temperature response after a relatively large reduction of the fan opening. Explain the unstable response!

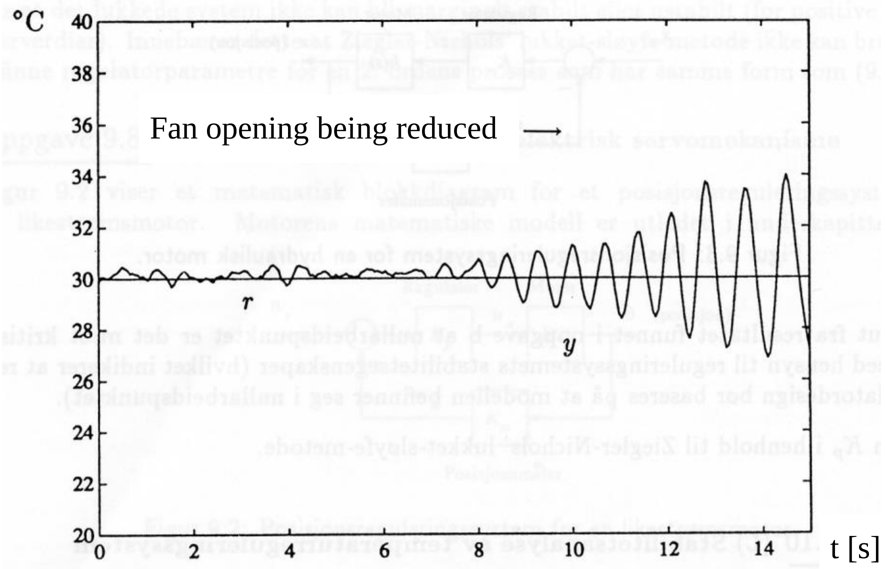


Figure 15.11: Unstable temperature control system

Problem 15.4 *Stability margins*

Given a control system that is initially stable when $\Delta K = 1$ and $\Delta t_d = 0$. The controller gain is $K_c = 5.0$, and it has been found that GM $= 2.5$.

1. Which value of K_c will make the control system become marginally stable?
2. In an experiment with the control system with $K_c = 5.0$, it has been found that a time delay increase of $\Delta t_d = 10$ s causes the control system to oscillate with a period of 100 s. What is the phase margin PM? Is this an ok value?

15.4 *Solutions to problems for Chapter 15*

Solution to Problem 15.1

Reduced PB value means increased controller gain, and therefore increased loop gain. This implies that the *stability is reduced* if the PB value is decreased.

Solution to Problem 15.2

1. The transport time from tank to sensor increases, thus increasing the time delay in the control loop, causing reduced stability. If the time delay gets too large, the control system becomes unstable.
2. With reduces flow, the time delay is increased, causing *reduced stability*.
3. The *stability is reduced* since the loop gain is increased.
4. The *stability is reduced* since the loop gain is increased.

Solution to Problem 15.3

The response is oscillatory because the control system is unstable. The instability is due to: (1) The reduced air flow which causes *increased transport delay (time delay) in the process* from heating element to temperature sensor. (2) *Increased process gain* because the reduced air flow makes the temperature more sensitive to the supplied heat (adjusted by the control signal), and increased process gain implies increased control loop gain.

Solution to Problem 15.4

1.
$$K_c = 5 \cdot 2.5 = 12.5$$

2.
$$\text{PM} = \frac{\triangle t_{d,u}}{t_p} \cdot 360^\circ = \frac{10\text{ s}}{100\text{ s}} \cdot 360^\circ = 36^\circ$$

which is an ok value, cf. (15.10).

Chapter 16

Control structures based on the PID control loop

This chapter describes a number of control methods and control structures based on the PID controller that are quite common in industry.

Also, sequential control is described. Sequential control is used to implement batch process control and automated mechanical operations. In a sequential control system PID control loops typically play a minor role, because the overall control task is not about continuous control. For example, PID control may be used to make the temperature of a batch reactor track a ramped temperature profile during the heating step of the control sequence.

16.1 Cascade control

16.1.1 The principle of cascade control

From earlier chapters we know that a control loop compensates for disturbances. If the controller has integral action the steady state control error is zero despite a change in the disturbance. What more can we wish? We may want the compensation to happen faster, so that the control error goes to zero faster. This can be achieved by *cascade control*, see Figure 16.1.

In a cascade control system there is one or more control loops inside the *primary loop*, and the controllers are in cascade.

There is usually one, but there may be two and even three internal loops inside the primary loop. The (first) loop inside the primary loop is called the *secondary loop*, and the controller in this loop is called the *secondary controller* (or slave controller). The outer loop is called the *primary loop*, and the controller in this loop is called the *primary controller* (or master-controller). The control signal calculated by the primary controller is used as the reference of the secondary controller.

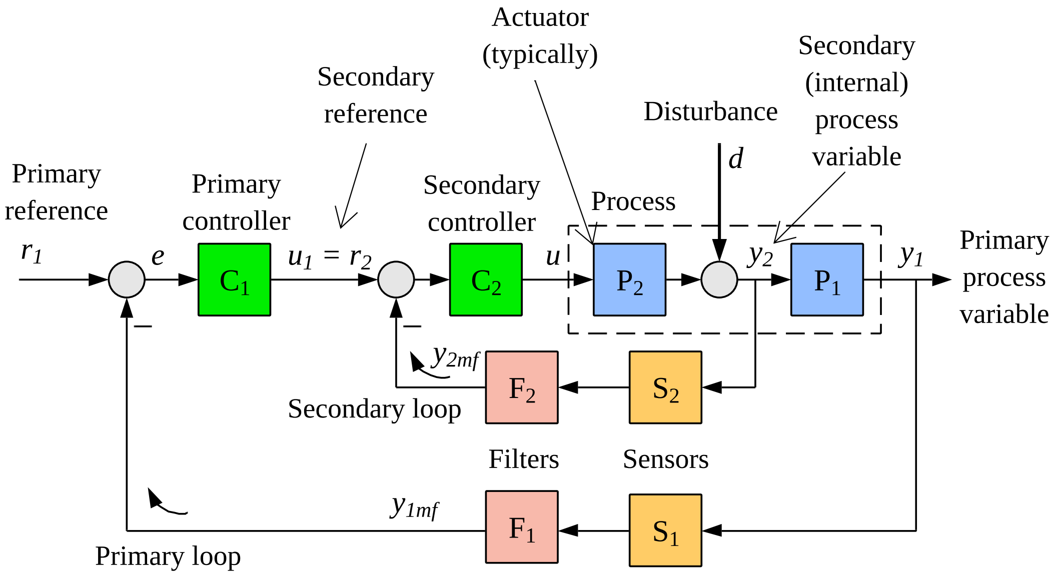


Figure 16.1: Cascade control system.

16.1.2 Benefits of cascade control

To repeat: One important reason for using cascade control is that it *gives faster (better) disturbance compensation.* In most applications the purpose of the secondary loop is to compensate quickly for the disturbance so that the response of the disturbance in the primary output variable is small. For this to happen the secondary loop must register the disturbance. This is done with the sensor S_2 in Figure 16.1.

One additional benefit of cascade control is that *the internal process variable (y_2) becomes more directly controllable, making it is easier to tune the primary controller to obtain satisfactorily good overall stability and fastness of the whole control system.* In many applications process part 2 (P_2 in Figure 16.1) is the actuator (valve, pump, motor), or is closely related to the actuator. The secondary control loop can be regarded as a *new actuator* which can be more directly controlled or manipulated by the primary controller. See Figure 16.2. The internal process variable y_2 is controlled or manipulated directly by the primary controller: The output of the primary controller, u_1, demands a value of y_2 which is then a reference r_2 of y_2, and the secondary control loop makes y_2 track r_2.

Examples:

- The internal process variable (y_2) is valve flow. The secondary loop is then a flow control loop.
- The internal process variable (y_2) is pressure after (downstream) the valve. The secondary loop is then a pressure control loop.

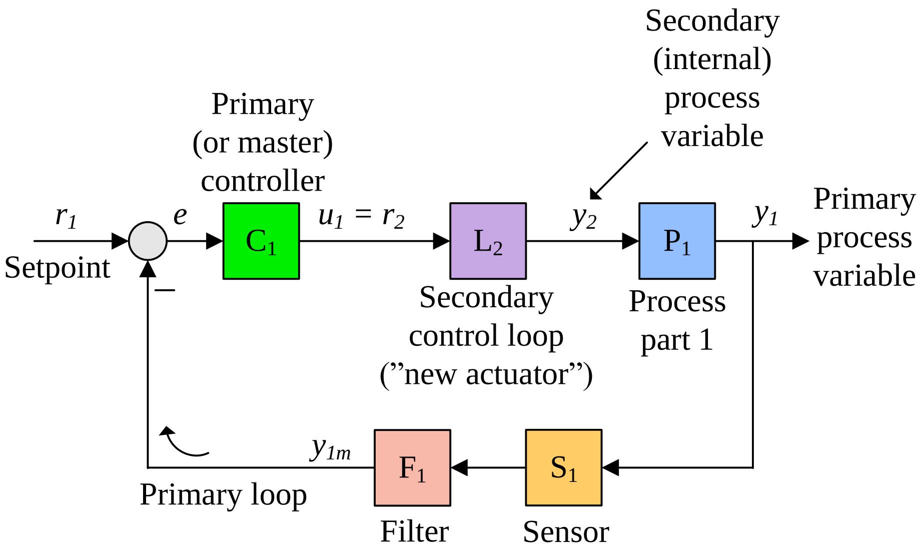


Figure 16.2: The secondary control loop can be regarded as a *new actuator.*

- The internal process variable (y_2) is valve position (valve opening). The secondary loop is then a position control loop.

The generally improved control with cascade control can be explained by the increased information about the process – there is at least one more measurement. It is a general principle that the more information you have about the process to be controlled, the better it can be controlled. Note however, that there is still only one control variable to the process, but it is based on two or more measurements.

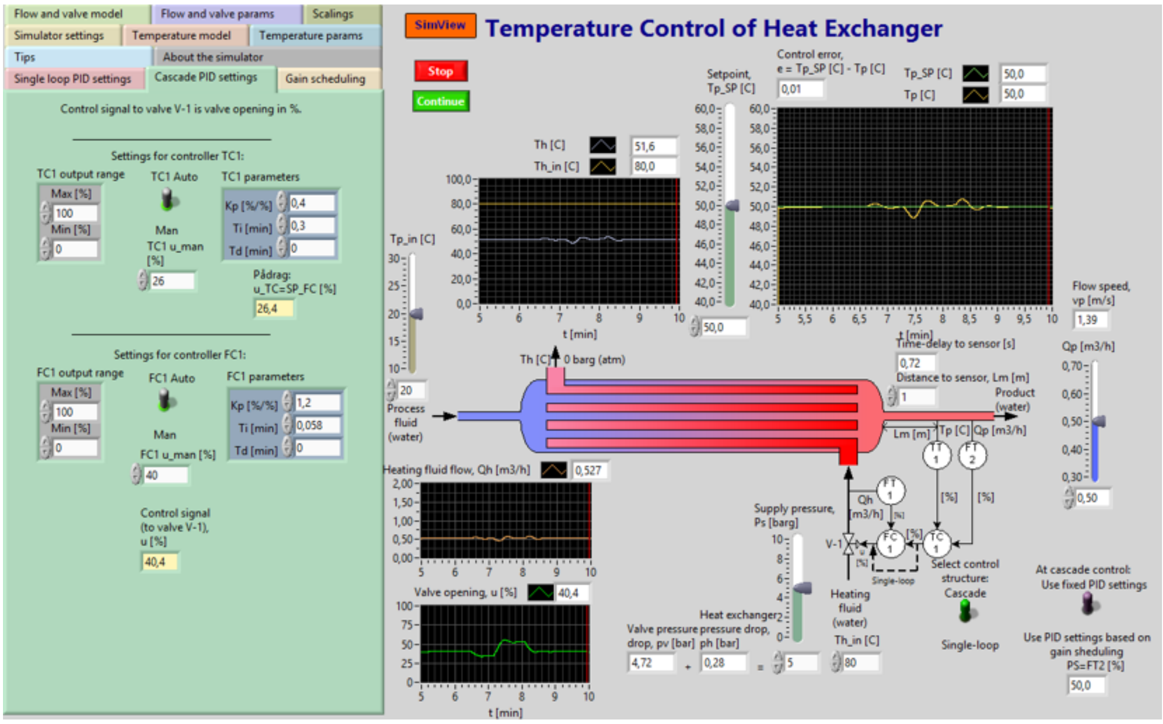
As explained above cascade control can give substantial disturbance compensation improvement. Cascade control can also give improved tracking of a varying reference, but only if the secondary loop has faster dynamics than the process part P$_2$ itself, cf. Figure 16.1, so that the primary controller “sees” a faster process. But, if there is a time delay in P$_2$, the secondary loop will not be faster than P$_2$, and then faster reference tracking can not be expected.

Are there any drawbacks of cascade control? Since cascade control requires at least two sensors a cascade control system is somewhat more expensive than a single loop control system. Industrial automation systems are typically prepared for cascade control, so no extra control hardware or software is required.

Example 16.1 *Cascade control of a heat exchanger*

Figure 16.3 shows a temperature control system of a simulated heat exchanger. The simulator is in the SimView library, and is available on http://techteach.no/simview/cascade_control_temp_heat_ex/

The control variable controls the opening of the hot water valve. The primary loop controls the product temperature. The secondary loop controls the heat flow to compensate for flow variations (disturbances) which stem from variations in the pressure drop over the valve; these pressure variations may in turn stem from variations in the pressure supply. The valve with flow control system can be regarded as a new valve with an approximate proportional relation between the control variable and the heat flow.


SimView Temperature Control of Heat Exchanger

Control signal to valve V-1 is valve opening in %.

Settings for controller TC1:
TC1 output range: Max [%] 100, Min [%] 0
TC1 Auto / Man
TC1 u_man [%] 26
TC1 parameters: Kp [%/%] 0,4; Ti [min] 0,3; Td [min] 0
Pådrag: u_TC=SP_FC [%] 26,4

Settings for controller FC1:
FC1 output range: Max [%] 100, Min [%] 0
FC1 Auto / Man
FC1 u_man [%] 40
FC1 parameters: Kp [%/%] 1,2; Ti [min] 0,058; Td [min] 0
Control signal (to valve V-1), u [%] 40,4

Th [C] 51,6
Th_in [C] 80,0
Setpoint, Tp_SP [C] 50,0
Control error, e = Tp_SP [C] - Tp [C] 0,01
Tp_SP [C] 50,0
Tp [C] 50,0
Flow speed, vp [m/s] 1,39
Time-delay to sensor [s] 0,72
Distance to sensor, Lm [m] 1
Qp [m3/h] 0,50
Tp_in [C] 20
Heating fluid flow, Qh [m3/h] 0,527
Valve opening, u [%] 40,4
Valve pressure drop, pv [bar] 4,72 + Heat exchanger pressure drop, ph [bar] 0,28 = Supply pressure, Ps [barg] 5
Th_in [C] 80
Select control structure: Cascade / Single-loop
At cascade control: Use fixed PID settings / Use PID settings based on gain sheduling
PS=FT2 [%] 50,0

Figure 16.3: Example 16.1: Cascade control of the product temperature of a heat exchanger.

Let's look at simulated temperature responses with single loop control and with cascade control. In the simulations, the various controllers are PI controllers tuned for fast and stable control.

Single loop control

Figure 16.4 shows the temperature in the product temperature (at TT1) when standard single loop control is used, i.e. the valve is manipulated directly by controller TC1. The supply pressure (disturbance) P_s is changed manually as $5 \to 8 \to 3 \to 5$ barg (approximate values).

Cascade control

Figure 16.5 shows the temperature (at TT1) when cascade control is used; The valve is

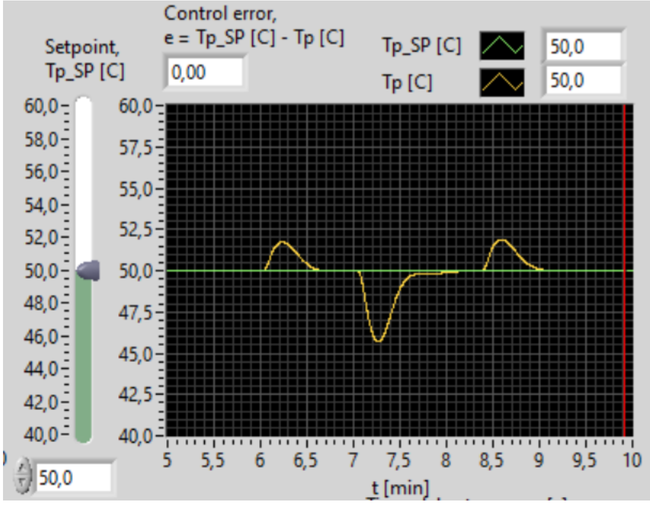


Figure 16.4: Temperature response (at TT1) with single loop control. P_s is changed manually as $5 \rightarrow 8 \rightarrow 3 \rightarrow 5$ barg.

manipulated directly by flow controller FC1. The flow reference (setpoint) to FC1 is the control signal generated by the temperature controller TC1.

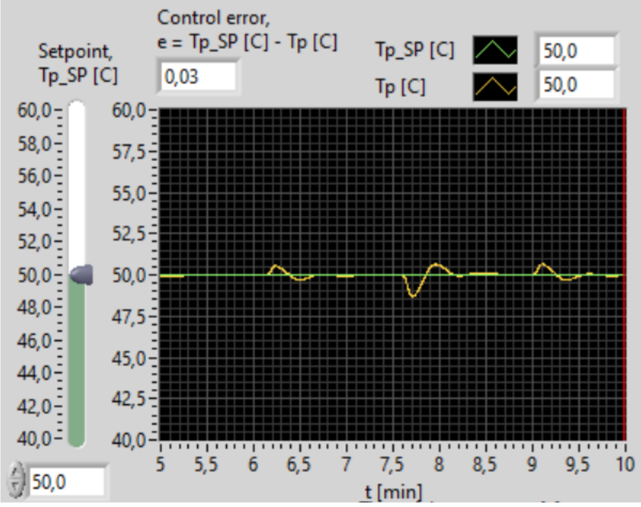


Figure 16.5: Temperature response (in TT1) with cascade control. P_s is changed manually as $5 \rightarrow 8 \rightarrow 3 \rightarrow 5$ barg.

Compare the responses shown in Figure 16.5 with those in Figure 16.4. Which control structure is the best (in this test)? The answer should be obvious.

[End of Example 16.1]

So far, we have seen examples only of two cascade control loops, but there may be additional loops. Below is an example of *three* cascade control loops.

Example 16.2 *Level control system with three cascade control loops*

Figure 16.6 shows a level control system of a tank using three cascade control loops. The loops are:

- Primary loop: Level control of the tank.
- Secondary loop: Flow rate control of the pump.
- Tertiary loop: Rotational speed control of the pump motor.

This cascade control system is found at the inlet part of some water resource recovery facilities. The tank is an inlet buffer magazine of combined water and sewage to be treated by the facility (plant). The level controller should be tuned for averaging level control, cf. Section 16.4.

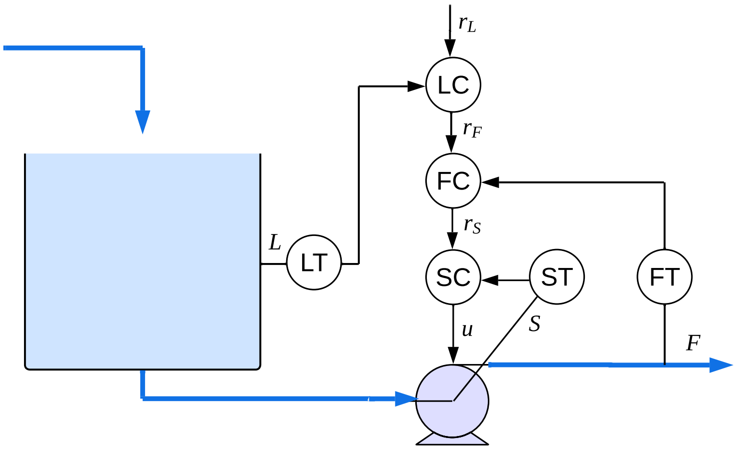


Figure 16.6: Level control system with three cascade control loops. (S = Speed.)

[End of Example 16.2]

16.1.3 Selection of control functions

The primary controller is typically a PID controller or a PI controller. The secondary controller is typically a P controller or a PI controller. The derivative action is usually not needed to speed up the secondary loop since process part 2 typically has much faster dynamics than process part 1, so the secondary loop will probably be fast enough. And in general the sensitivity of the derivative term to measurement noise is a drawback.

If the secondary controller is a PI controller, the proportional term of the controller should not have reduced reference weight, cf. Section 11.3.2. Why?[1]

[1] Because attenuating or removing the time-varying reference (which is equal to the control signal produced

16.1.4 Controller tuning

How do you *tune* the controllers of a cascade control? You can follow this procedure:

- First the secondary controller is tuned, with the primary controller in manual mode.
- Then the primary controller is tuned, the secondary controller in automatic mode.

So, you start with the inner loop, and then move outwards.

The tuning of the controllers (finding good PID settings) can be made using any of the tuning methods described in Section 14.

16.1.5 Cascade control and state feedback

Cascade control is quite similar to *state feedback control*, which is a control principle where the controller generates the control signal as a function of measurements (or estimates) of *all the state variables* of the process to be controlled, see Figure 16.7.

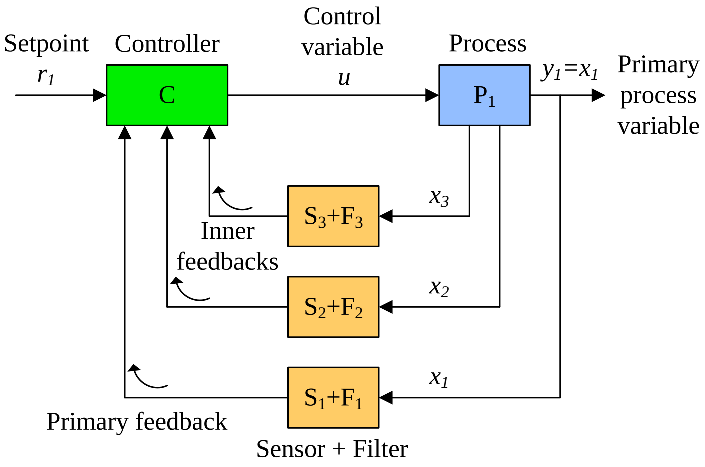


Figure 16.7: State feedback control.

It can be shown that with state feedback, you can design a control system with specified performance regarding stability and fastness for virtually any stable or unstable process. Hence, state feedback is potentially a very powerful control principle. To design a state feedback controller you need the mathematical model of the process. There are several control methods – linear and nonlinear – that are based on state feedback control, and a mathematical process model. The following methods are presented later in this book:

- Linear quadratic control (LQG), Ch. 34.

by the primary controller) of the secondary loop will reduce the ability of the secondary loop to track these reference changes, causing slower tracking by the total control system.

- Model predictive control (MPC), Ch. 35.
- Inverse dynamic control, Ch. 36.

16.2 Ratio control

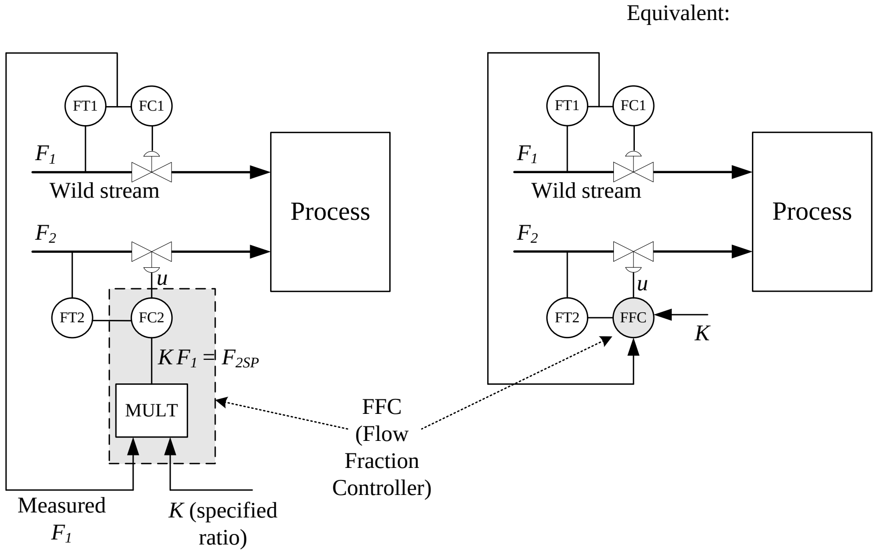


Figure 16.8: Ratio control

The purpose of *ratio control* is to control a mass flow, say F_2, so that the ratio between this flow and another flow, say F_1, is

$$F_2 = KF_1 \tag{16.1}$$

where K is a specified ratio which may have been calculated as an optimal ratio from a process model. One example is the calculation of the ratio between oil inflow and air inflow to a burner to obtain optimal operating condition for the burner. Another example is the nitric acid factory where ammonia and air must be fed to the reactor in a given ratio.

Figure 16.8 shows the most common structure of ratio control. As shown with the left diagram in the figure, the reference of the flow F_2 is calculated as K times the measured value of F_1, which is denoted the “wild stream” (this name is used, even if this stream is controlled, as shown in Figure 16.8). The diagram to the right in the figure shows a compact but equivalent representation of ratio control with the symbol FFC (Flow Fraction Controller).

16.3 Split-range control

In *split-range control* one controller controls two actuators in different ranges of the control signal span, which here is assumed to be 0 – 100%. See Figure 16.9.

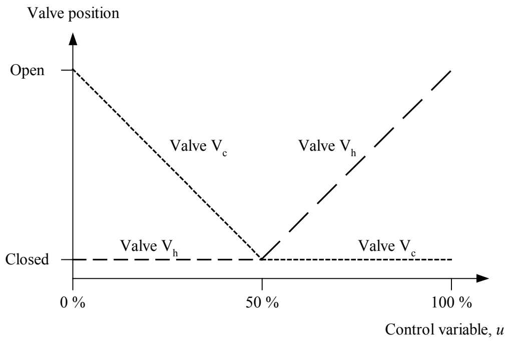


Figure 16.9: Split-range control of two valves

Figure 16.10 shows an example of split-range temperature control of a thermal process. Two valves are controlled – one for cooling and one for heating, as in a reactor.

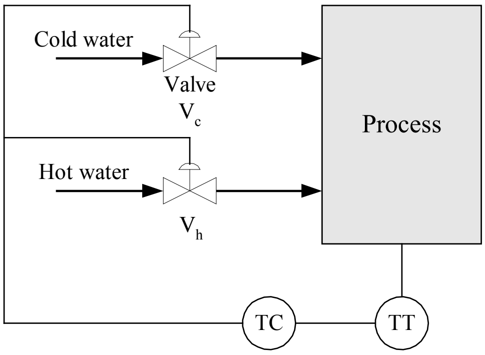


Figure 16.10: Split-range temperature control using two control valves

The temperature controller controls the cold water valve for control signals in the range 0 – 50 %, and it controls the hot water valve for control signals in the range 50 – 100 %, cf. Figure 16.9.

In Figure 16.9 it is indicated that one of the valves are open while the other is active. However in certain applications one valve can still be open while the other is active, see Figure 16.11.

One application is pressure control of a process: When the pressure drop compensation is small (as when the process load is small), valve V$_1$ is *active* and valve V$_2$ is *closed.* And when the pressure drop compensation is large (as when the process load is large), valve V$_1$ is *open* and valve V$_2$ *is still active.*

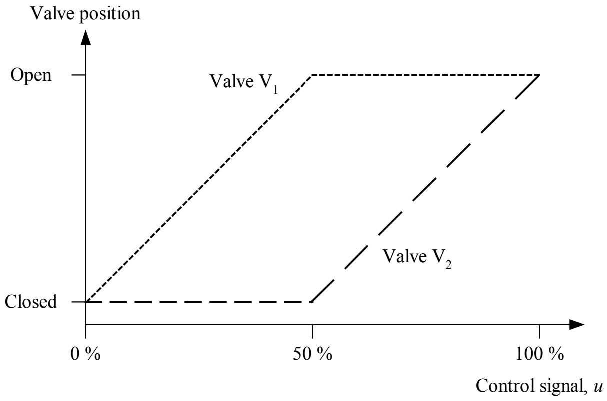


Figure 16.11: In split-range control one valve can be active while another valve is open simultaneuously.

16.4 Averaging level control

16.4.1 What is averaging level control?

Averaging level control is an important part of several process systems. Figure 16.12 shows a general buffer tank with a level control system aiming at averaging (or equalizing, or attenuating) inflow variations so that the outflow becomes smoother than the inflow. Figure 16.13 shows a block diagram of the level control system.

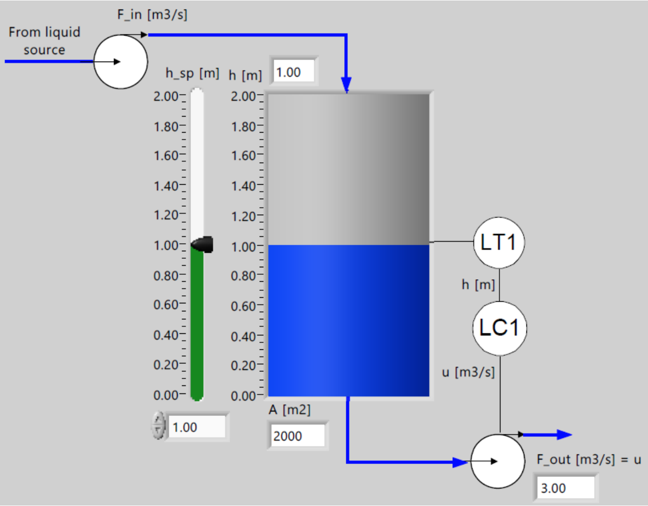


Figure 16.12: Averaging level control of a buffer tank.

The tank resembles the inlet magazine to the water resource and recovery facility (WWRF) presented in the subsequent Example 16.3 and the oil/water separator presented in presented in the subsequent Example 16.4. Based on the difference between the level reference and the level measurement provided by the sensor LT1, the level controller, LC1, manipulates the outlow through the pump to compensate for inflow variations.

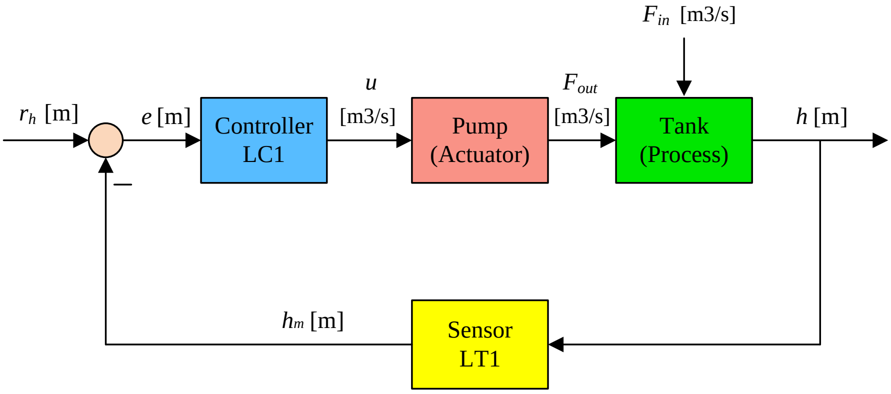


Figure 16.13: Block diagram of the temperature control system of the liquid tank.

A SimView simulator of the averaging level control system is available on:

http://techteach.no/simview/level_control_equalization_tank

Figure 16.14 shows a simulation with the level controller LC1 in automatic mode. (SimView simulator: http://techteach.no/simview/level_control_equalization_tank.) The inflow is sinusoidal, with amplitude of 1 m^3/s and period of 1200 s (refering to the WRRF, this

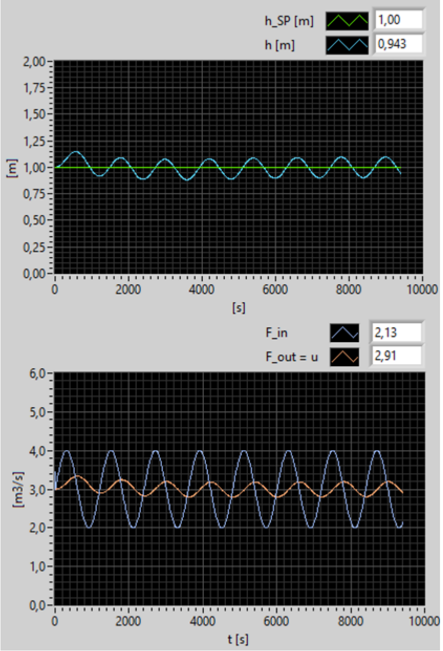


Figure 16.14: Simulation with the PI level controller in automatic mode. The controller is tuned with the SIMC method. F_{in} has a sinusoidal variation.

inflow variation may be washwater from the treatment processes subsequent to the inlet magazine in the WRRF). The bottom plot of Figure 16.14 shows that the level controlled buffer tank (magazine) attenuates the flow variations. The level is allowed to vary around the level reference. These variations are necessary to obtain the buffering or averaging of the varying inflow. It is actually a special challenge to tune the level controller in averaging level control systems, since the level control must be sufficiently compliant, not "stiff" is we typically want control system to be. Tuning for averaging control is described in Ch. 16.4.2.

Example 16.3 *Equalization or buffer magazine at the inlet of a WRRF*

Water resource recovery facilities (WWRF) are crucial facilities in modern societies. They recover resources from what has traditionally been regarded as "waste", namely sewage and other forms of organic matter.

Figure 16.15 shows the equalization or buffer magazine at the inlet of a typical WRRF.

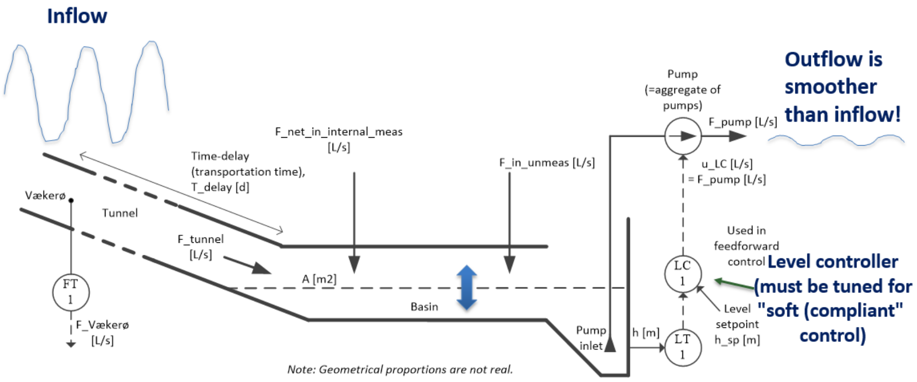

Inflow
Pump (=aggregate of pumps)
Outflow is smoother than inflow!
F_pump [L/s]
F_net_in_internal_meas [L/s]
Time-delay (transportation time), T_delay [d]
F_in_unmeas [L/s]
u_LC [L/s] = F_pump [L/s]
Vækerø
Tunnel
F_tunnel [L/s]
Used in feedforward control
Level controller (must be tuned for "soft (compliant" control)
LC 1
A [m2]
Basin
FT 1
F_Vækerø [L/s]
Pump inlet
h [m]
LT 1
Level setpoint h_sp [m]
Note: Geometrical proportions are not real.

Figure 16.15: Level control of an equalization magazine in a WRRF:

In the equalization magazine, the level should be compliant to flow variations so that variations in the inflow are attenuated through the magazine, making the outflow considerably smoother than the inflow. Smoother outflow is advantageous for the subsequent processes, e.g. for the biological treatment processes of the WRRF. The level controller must tuned for compliant (or soft, or sluggish) level control so that the volume of the tank can absorb the inlet variations.

[End of Example 16.3]

Example 16.4 *Control of a water and oil separation system*

In the oil & gas production where separators are used to separate oil and gas and water from a three-phase oil/water/gas flow coming from reservoirs.[2] Separators are level-controlled to maintain the mass balance in the separator. It it important that *slugs*,

[2] The residence in the separator causes the separation to take place due to the different densities of oil, water and gas.

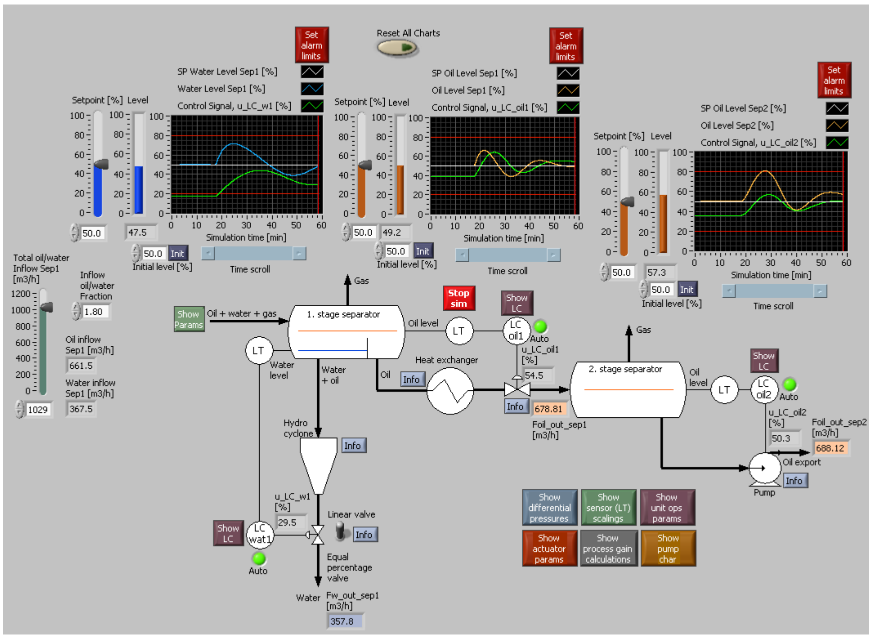


Figure 16.16: Control structure for a water, oil and gas separator system

which are sudden changes in the inflow to the separators, are attenuated through the separator, otherwise the slug will propagate through the subsequent production line. This attenuation will not take place if the level control works too fast, because in a fast level control system, the outflow is nearly the same as the input! So, slugs requires sluggish (slow) level control. On the other hand, the slugs must not cause too large level variations inside the separator, otherwise level alarms will be triggered.

A SimView simulator of level control systems of the separators on the Vigdis oil and gas plant in Norway is available on:

http://techteach.no/simview/separators

Figure 16.16 shows the front panel of the simulator. The main flow rate is given by the inflow from the reservoir. So, in this system there is no flow control loop. Instead, the flow is given by the flow coming from the reservoir. The level controllers must maintain the mass balances of the separators. (Mass of gas is maintained using pressure control, but the pressure control system is not shown in the figure.) In the first stage separator there is a level control systems for the oil and another level control systems for the water. In the second stage separator there is a level control system for the oil (it is assumed that all the water has been removed in the first stage separator).

[End of Example 16.4]

16.4.2 Tuning of the PI controller for averaging level control

Review Figure 16.12. How to tune the level controller (LC1 in the figure)? The Ziegler-Nichols controller tuning methods – both their ultimate gain method (or closed loop method), and the process reaction method (or open loop method) – are designed for giving as fast control as possible. These methods are therefore *useless* for tuning of the level controller of a buffer tank since we want sluggish control of such tanks.

However, the SIMC method is suitable here since the speed of the controller can be tuned with the closed loop time constant, t_{cc}, as tuning parameter. SIMC tuning of the averaging level controller is explained below.

Dynamically, buffer tanks are integrators (or accumulators) since the process output variable, which is the liquid level, is proportional to the integral of (or accumulation of) the control signal, which is here assumed to be the inflow pump control signal. Let us recall the SIMC formulas for integrating processes with zero time delay (in liquid buffer tanks, we can assume a negligible or zero time delay between the pump flow control signal and the level):

$$K_c = \frac{1}{K_i t_{cc}} \tag{16.2}$$

$$T_i = 2t_{cc} \tag{16.3}$$

Here:

- K_i is the process integrator gain, or the normalized step slope of the process step response (step in the control signal). As explained below,

  $$K_i = -\frac{1}{A} \tag{16.4}$$

  where A [m^2] is the surface area of the liquid in the tank.

- t_{cc}[s] is the time constant of the level control system, i.e. of the closed loop system from level reference, r, to level measurement, h. A way of tuning t_{cc} is suggested below.

Using (16.4), the PI controller settings of the level controller of a buffer tank is:

$$K_c = \frac{1}{K_i t_{cc}} = -\frac{A}{t_{cc}} \tag{16.5}$$

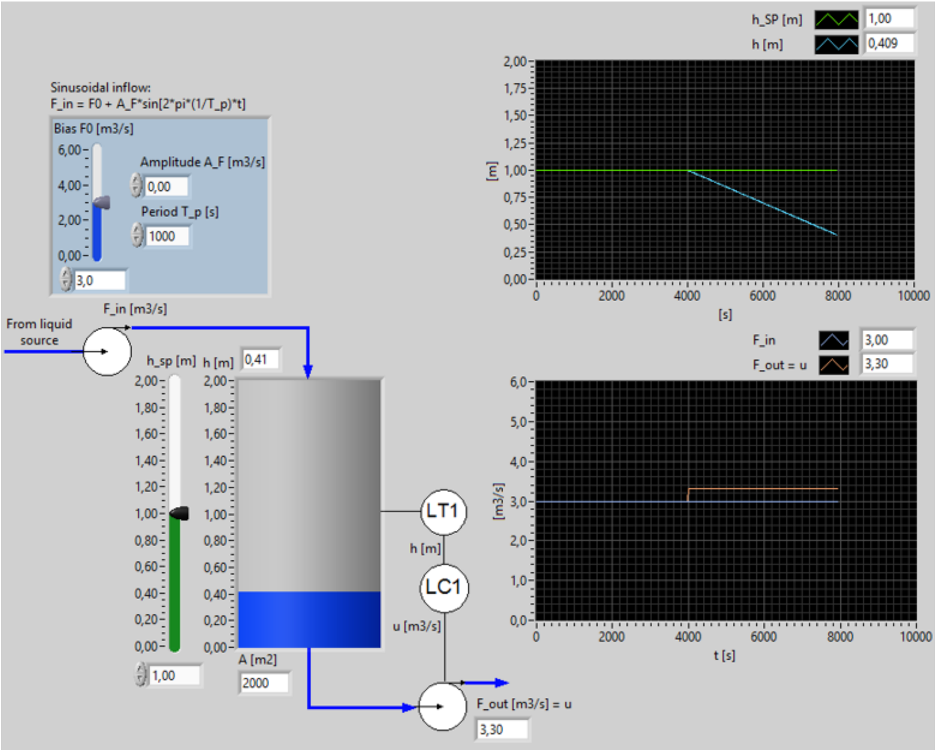
$$T_i = 2t_{cc} \tag{16.6}$$

Note: The negative sign in K_c means that the controller has *direct action*.

Derivation of (16.4), $K_i = -1/A$

Let us study an example in the form of a simulator of a level controlled buffer tank. Figure 16.17 shows the front panel of the simulator. The controller is in manual mode. The upper plot in Figure 16.17 shows the ramp-shaped response in the level, h [m], due to a step in the pump control signal, u. (The values of the model parameters may represent a water resource and recovery facility.[3])

[3] Historically denoted wastewater treatment plant, but new terminology has been introduced.



Sinusoidal inflow:
F_in = F0 + A_F*sin[2*pi*(1/T_p)*t]

Bias F0 [m3/s] 3,0

Amplitude A_F [m3/s] 0,00

Period T_p [s] 1000

h_SP [m] 1,00

h [m] 0,409

F_in 3,00

F_out = u 3,30

h_sp [m] 1,00

h [m] 0,41

A [m2] 2000

F_out [m3/s] = u 3,30

Figure 16.17: The upper plot shows the ramp-shaped response in the level due to a step in the pump control signal shown in the bottom plot. (Controller LC is in manual mode.)

Assume that the pump control signal is changed as a step of amplitude U. Figure 16.17 shows such an experiment (applied there with $U = 0.3\ \mathrm{m}^3/\mathrm{s}$). The ramp-shaped level response has slope S [m/s] equal to

$$S = -\frac{U}{A} \tag{16.7}$$

and the normalized slope becomes

$$K_i = \frac{S}{U} = \frac{-U/A}{U} = -\frac{1}{A} \tag{16.8}$$

How to specify a reasonable value of t_{cc}?

How to specify a reasonable value of t_{cc} [s] needed in the PI settings of K_c and T_i in (16.5) and (16.6), respectively? An interpretation of this time constant is how fast, in the form of 63% rise time, the level approaches to the level reference when the reference is changed as a step. This is illustrated in Figure 16.18 which shows an ideal, principal time constant response of a control system (numbers will, of course, be unique to each pertinent control system).

However, in level control systemts of buffer tanks the reference is typically constant, so specifying the 63% rise time of the reference step response is not particularly meaningful for such control systems. Instead, let's make a more meaningful specification of t_{cc}, related to the response of an inflow change. We start by assuming that the level controller is a P

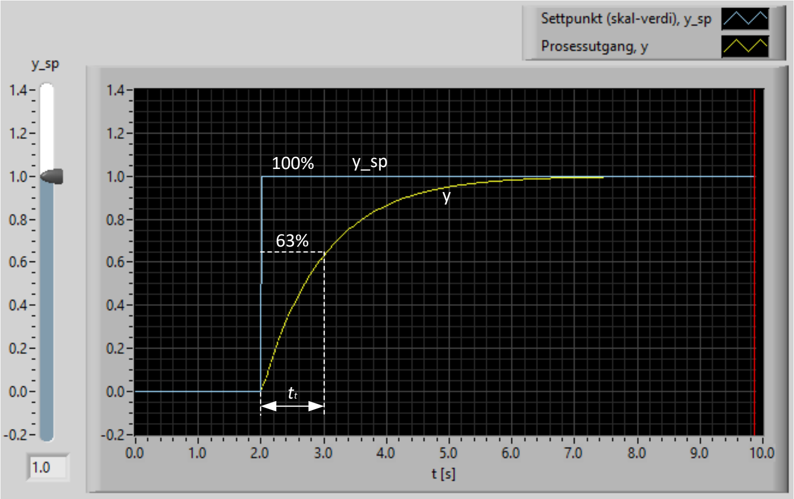


Figure 16.18: Ideal time constant response of a control system.

(proportional) controller. It can be shown, from a mathematical model of the level control system, that

$$\Delta h_{\max} = \frac{t_{cc}}{A}\Delta F_{\text{in}} \tag{16.9}$$

Here, $\Delta h_{\max}$ is the maximum steady state change in level due to an assumed maximum inflow step change of amplitude ΔF_{in} [m^3/s], see the upper plot of Figure 16.19, which shows a simulated response with P-controller (yellow curve) due to a inflow step change at t = 2000 s (shown in the bottom plot of the figure). The response with a PI-controller is also shown (blue curve).

With a PI controller that has the same t_{cc} as with the P controller, the following applies, as illustrated in Figure 16.19:

$$\Delta h_{\max} \leqslant \frac{t_{cc}}{A}\Delta F_{\text{in}} \tag{16.10}$$

Solving this inequality for t_{cc} gives

$$t_{cc} \geqslant \frac{A\Delta h_{\max}}{\Delta F_{\text{in}}} \tag{16.11}$$

We may use this upper limit of t_{cc} in (16.11) to specify t_{cc}:

$$t_{cc} = \frac{A\Delta h_{\max}}{\Delta F_{\text{in}}} \tag{16.12}$$

Summing up the PI tuning formulas

1. From the allowed $\Delta h_{\max}$ due to an inflow step change of amplitude ΔF_{in}, calculate t_{cc} as

$$t_{cc} = \frac{A\Delta h_{\max}}{\Delta F_{\text{in}}} \tag{16.13}$$

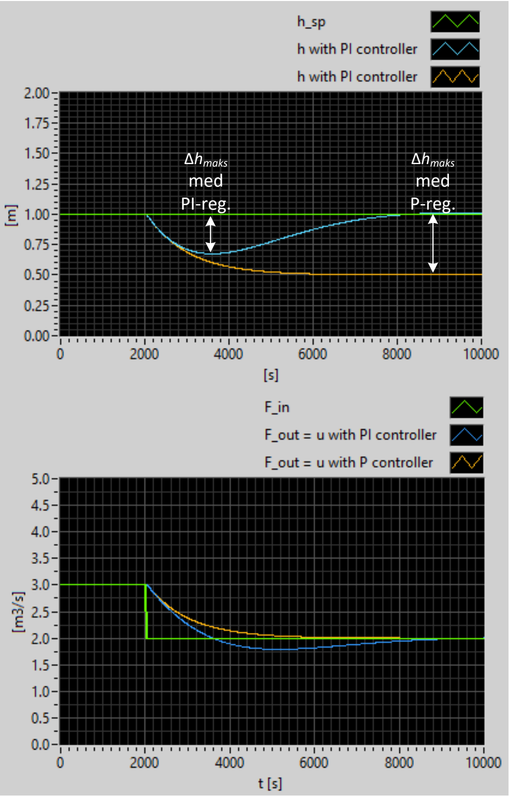


Figure 16.19: Simulations with P controller and with PI controller due to a step change in F_{in}.

2. Then, t_{cc} is used to tune the PI controller as follows:

$$K_c = -\frac{A}{t_{cc}} \tag{16.14}$$

$$T_i = 2t_{cc} \tag{16.15}$$

Note: The negative sign in K_c means that the controller has direct action.

Example 16.5 *Tuning of the PI controller for averaging level control*

The simulations in this example are made with the following SimView simulator:

http://techteach.no/simview/level_control_buffer_tank/

Assumptions:

$$A = 2000\ \text{m}^2$$

$$\Delta F_{\text{in}} = 1.0\ \text{m}^3/\text{s}$$

$$\varDelta h_{\max} = 0.5\ \mathrm{m}$$

From (16.13):

$$t_{cc} = \frac{A\varDelta h_{\max}}{\varDelta F_{\text{in}}} = \frac{2000\ \mathrm{m}^2 \cdot 0.5\ \mathrm{m}}{1.0\ \mathrm{m}^3/\mathrm{s}} = 1000\ \mathrm{s} \tag{16.16}$$

Then, from (16.14) and (16.15):

$$K_c = -\frac{A}{t_{cc}} = -\frac{2000\ \mathrm{m}^2}{1000\ \mathrm{s}} = -2\ \mathrm{m}^2/\mathrm{s} = -2\ (\mathrm{m}^3/\mathrm{s})/\mathrm{m} \tag{16.17}$$

$$T_i = 2t_{cc} = 2 \cdot 1000\ \mathrm{s} = 2000\ \mathrm{s} \tag{16.18}$$

Simulation with step change in inflow

Figure 16.19 shows a simulation with step change in the inflow of $\varDelta F_{\text{in}} = 1.0\ \mathrm{m}^3/\mathrm{s}$. The blue curves pertains to PI control (the orange is P control). We see that the level, as expected, is reduced somewhat less 0.5 due to the inflow step change. Thus, the specification of the control system is satisfied.

Simulation with sinusoidal change in inflow

Figure 16.14 shows another simulation with the controller in automatic mode. The inflow is now sinusoidal, with amplitude of 1 m^3/s and period of 1200 s (this intake variation may be washwater from the treatment processes subsequent to the inlet magazine in a wasterwater treatment plant). The bottom plot of Figure 16.14 shows that the level controlled buffer tank (magazine) attenuates the flow variations. Even greater attenuation can be achieved by increasing the control system time constant T_c, but then with the disadvantage that the level may deviate too far from the reference in the case of a large inflow change.

[End of Example 16.5]

16.5 Plantwide control

In the process industry products are created after materials being processed in a number of vessels in series, which are typically unit processes as evaporators, blending or heated tanks, buffer tanks, reactors, distillation columns, separators, absorbers, etc. Typical basic control requirements of such a production line are as follows:

- **The product flow must be controlled (to follow a reference).** This is typically done by implementing flow control of one key component or feed.
- **The product quality must be controlled** – more or less directly. Ideally, the product quality is analyzed or measured in real time, and this measurement is used to adjust feeds, supplied heat etc. so that the product quality is controlled to follow its reference using feedback. In practice, however, such real time or online measurements are typically not implemenented since they rely on time consuming laboratory analysis. Instead, the product quality is controlled indirectly by controlling the flow of the feeds needed to produce the final product, typically using ratio control.

- **The mass balance of each process vessel (tank, reactor, etc.) must be maintained** – otherwise it may run full or empty. The mass balance is maintained with level control systems for liquids, and with pressure control systems for gases. In vessels containing gas there may still not be any pressure control. It depends on the process if it necessary to control the process.

  It is crucial to select the correct reverse or direct action for each of the controllers (otherwise the control loop becomes unstable), cf. Section 11.3.1. The general rules are as follows[4]:

  - Level/gas controllers that are *upstream* relative to the flow control loop (i.e. are placed before the loop) implementing product flow control of the plant (see above) shall be set into *reverse action mode.*
  - Level/gas controllers that are *downstream* relative to the flow control loop (i.e. are placed after the loop) implementing product flow control of the plant (see above) shall be set into *direct action mode.*

- **The temperature in some of the process lines or vessels must be controlled.**

Figure 16.20 shows a basic, principal example where the above control requirements are implemented.

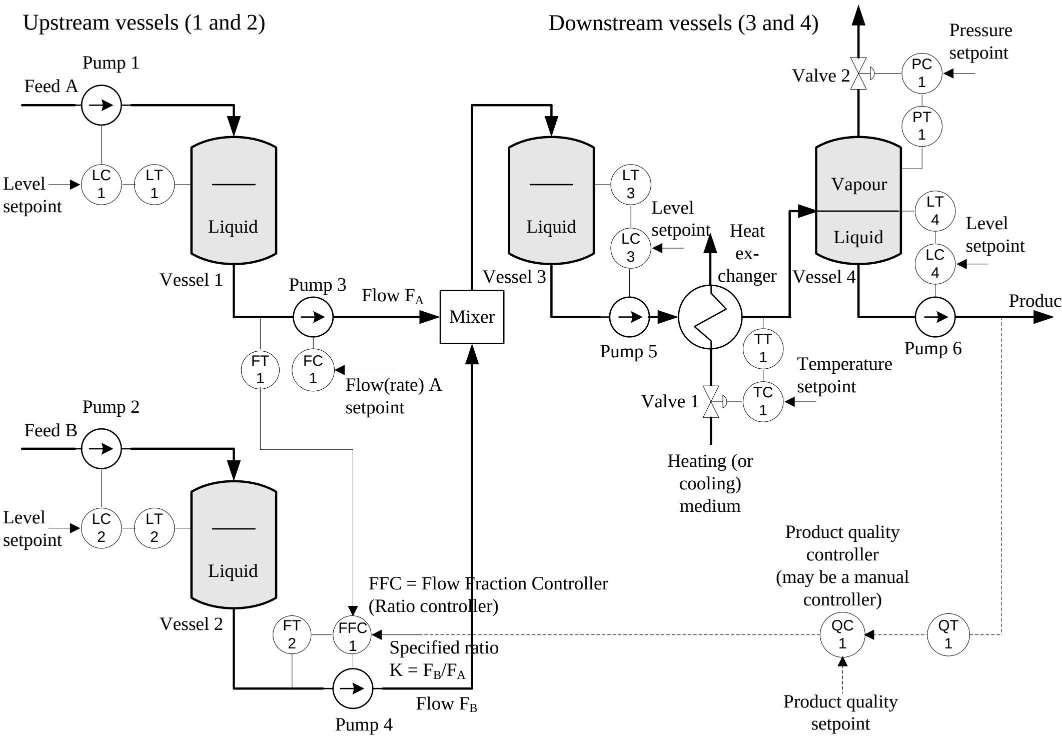


Figure 16.20: Process plant with control loops

Comments to the control system shown in Figure 16.20:

[4] It is assumed that increased control signal to the actuator increases flow through the actuator.

- Two flows, F_A and F_B, are mixed. In general, the mix of two feeds can be fed to a subsequent reactor. It is assumed that F_A contains the key component of the product, and that F_A defines the production rate. Ratio control is used to control F_B so that the ratio between the flows is as specified:

$$K = \frac{F_B}{F_A} \tag{16.19}$$

- The mass balance of upstream vessels 1 and 2 (upstream relative to the point of production flow control in the process line) is controlled by manipulating *inflows* while the mass balance of the downstreams vessels 3 and 4 tank are controlled by manipulating the *outflows*.
- In vessel 4 both the liquid level and the vapour pressure are controlled.
- The mass balances of liquids are maintained using level control. The mass balance of vapour is maintained using pressure control.
- The product quality may be controlled by adjusting the ratio K between the flows into the mixer. Ideally, this quality controller is operating in automatic mode based on online (continuous) quality measurement of the product. However, in many cases (as in the nitric acid production plant described in Example 16.6) the product quality is measured off-line using laboratory analysis which gives quality measures only a few times per day. In these cases, the quality controller is actually in the form of an operator adjusting the ratio K.
- The temperature of the inflow to vessel 4 is controlled.
- It is crucial to select correctly between Reverse action mode and Direct action mode in the controllers, cf. Section 11.3.1). Assume that for each of the pumps and the valves that increased control signal increase pump speeds and increase valve openings. Furthermore, assume that heating is needed by the heat exchanger, the control valve manipulates the amount of heating medium. Then, the controller mode of each of the controllers are as follows:
  - LC1: Reverse
  - LC2: Reverse
  - LC3: Direct
  - LC4: Direct
  - PC1: Reverse
  - TC1: Reverse
  - All flow controllers: Reverse

  Is this correct?[5]

A comprehensive overview over process control, including plantwide control principles, can be found in (Skogestad 2023).

A specific, industrial example is given in the following.

[5] No! PC1: Direct.

Example 16.6 *Control structures of a nitric acid plant*

The example is about production of nitric acid, which is used in e.g. the production of fertilizers.[6] The description of the process is based on information given by Yara company in Porsgrunn, Norway. However, I have simplified the process somewhat because too many details may obscure the basic principles. (One simplification is that the bleacher, which is used to clean the raw, brown-coloured nitric acid which is the output from the asbsorber, is not included. Another simplification is that the condenser between compressor K2 and absorber A1 is not included.) The control structure is shown in the figure is based on information given by Yara.

Figure 16.21 shows a Process & Instrumentation Diagram (P & ID) of the (simplified) nitric acid plant.

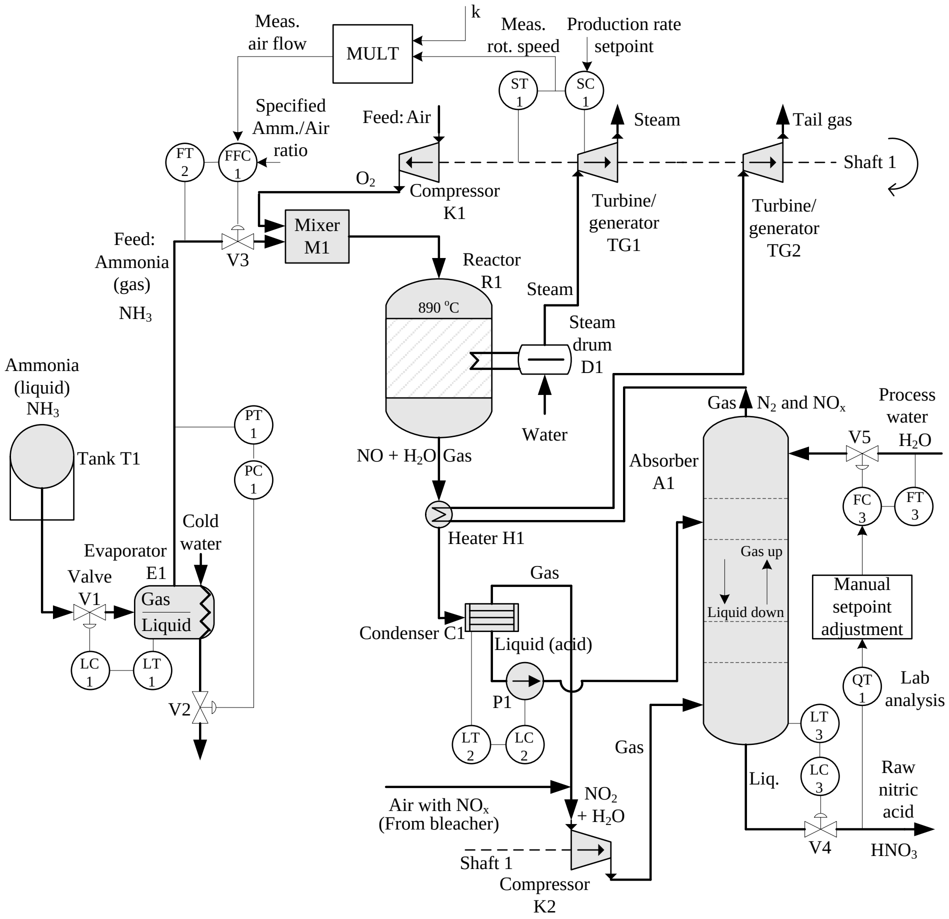


Figure 16.21: Example 16.6: Process and Instrumentation Diagram (P & ID) of simplified nitric acid plant.

Description of the nitric acid plant

[6] There are many Nitric Acid production plants around the world. One of them is at the Yara industrial plant area in Porsgrunn, Norway.

(Control structures are described below.)

The feeds (input flows) to the production are

- Ammonia
- Air
- Process water

The product (output flow) is

- Nitric acid

Below are descriptions of the components and flows related to these components:

- **Evaporator E1**: Ammonia liquid (NH_3) is evaporated using water (cold!) as heat supply.
- **Compressor K1**: Primary air is compressed.
- **Mixer M1**: Ammonia gas and air are mixed.
- **Reactor R1** (exothermal): Ammonia and oxygene (in the primary air) reacts with a platinum-rhodium catalyst to produce nitrogene monixide (NO) and water (H_2O) which are passed to condenser C1.
- **Condenser C1**: The water and some of the nitrogene dioxide (gases) are condensed into acid liquid.
- **Condenser C1**: The gaseous output is mixed with secondary air from the bleacher (the bleacher is omitted in this simplified plant), causing oxydation. The product of the oxydation is nitrogene dioxide (NO_2) and water (H_2O).
- **Compressor K2**: Nitrogene dioxide (NO_2) and water (H_2O) is compressed in compressor K2.
- **Condenser C1**: The acid liquid outflow is pumped by pump P1 to a proper tray in the absorber.
- **Absorber A1**: Process water is added at the top. This water and the acid liquid (from condenser C1) flow downwards to meet gaseous nitrogene dioxide (from condenser C2) with a chemical reaction taking place. The product of this reaction is raw nitric acid (HNO_3) liquid collected at the bottom of the absorber, to be passed to bleacher (the bleacher is omitted here), and nitrous gases exiting at the top.
- **Heater H1**: The nitrous gases are heated by the hot product out of reactor R1, before being passed to, and driving, turbine/generator TG2 which produces electric power that is exported.

- **Steam drum D1**: The reaction taking place in reactor R1 is exothermal. Some of the produced heat is used to heat water, and the resulting high-pressure steam drives turbine/generator TG1 which produces electric power that is exported.
- **Shaft 1**: The compressors and the turbines/generators have one common shaft. The turbines transfers chemical energy to motional (kinetic) energy. The generators produce electric power that is exported. The rotational speed of the common shaft determines the production rate of the plant.

Description of the control structure of the plant

- **Production rate** is controlled with rotational speed control of the common turbine/generator shaft. The mass flow of the air, which is a feed to the process, is proportional to the speed (indicated with the MULT block with a gain k in the upper part of the diagram). The stoichiometric balances are maintained with a ratio of 10 % between the mass flow of ammonia to mass flow of air. Therefore, the ammonia gas flow (feed) is controlled with ratio control based on the rotational speed.

  Also the *process water flow* into the absorber is a feed to the process. This flow is controlled with flow control with the flow reference adjusted manually according to a laboratory analysis of the nitric acid outflow (from the absorber) accomplished twice a day. The QT (Quality Transmitter) represents this analysis.
- **Mass balances** of the various vessels are maintained as follows:
  - *Evaporator E1*: Ammonia liquid level is controlled using inlet valve V1 as actuator. Since this level control loop is upstream relative to the production flow control loop, the level controller LC1 shall be set in reverse action mode.
  - *Evaporator E1*: Ammonia gas pressure is controlled using water valve V2 as actuator.
  - *Reactor* R1: The contents of the reactor is gaseous only. There is however no control of gas pressure.
  - *Condenser* C1: Acid liquid level is controlled using pump P1 as actuator. Since this level control loop is downstream relative to the production flow control loop, the level controller LC2 shall be set into direct action mode.
  - *Condenser* C1: Gas pressure is not controlled.
  - *Absorber* A1: Raw nitric acid liquid level is controlled using valve V4 as actuator. Since this level control loop is downstream relative to the production flow control loop, the level controller LC3 shall be set into direct action mode.
  - *Absorber* A1: Nitrous gas pressure at the “top” is not controlled.
- **Temperature control**: None.

[End of Example 16.6]

16.6 *Problems for Chapter 16*

Problem 16.1 *Cascade control for level control of wood chips tank*

Figure 16.22 shows a level control system of a wood chips tank. The primary loop implements level control. The secondary loop implements flow control to compensate quickly for disturbance flow d which is an outflow of large chips which are removed from the chip flow with a special mechanical filter. Insert the correct instrumentation letters for the question marks.

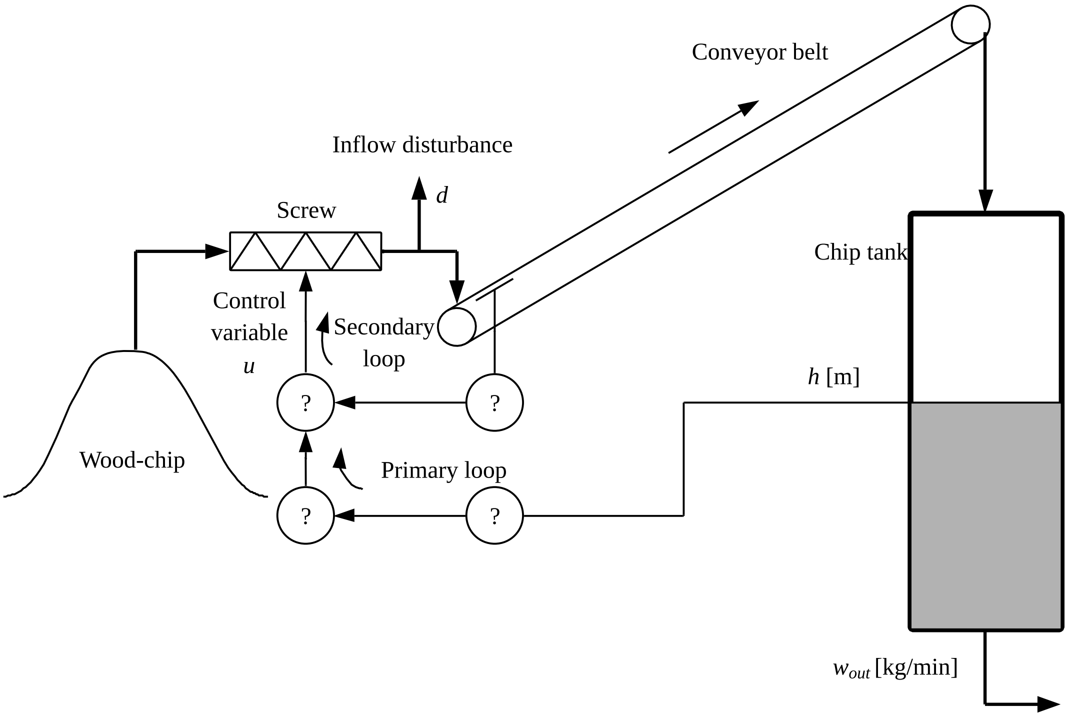


Figure 16.22: Level control system of a wood chips tank.

Problem 16.2 *Purpose of the ammonia flow control loop*

In the neutralization section of a fertilizer production plant, intermediate mother liquor flows into and out of a tank. In the tank the pH value of the liquid is controlled by adjusting the inflow of ammonia gas to the tank. The ammonia flow is flow controlled using a control valve.

Draw an instrumentation diagram of this process section. (You can use Q (for quality) as symbol for pH.) What can be the purpose of the ammonia flow control loop?

Problem 16.3 *Three control loops*

Figure 16.23 shows a control valve being used to manipulate the flow of a heating medium (liquid) into a heat exchanger where the temperature is to be controlled. The output of the temperature controller is flow command signal (flow reference) to the valve, and the output of the flow controller is a valve stem position command (position reference) to the stem moving mechanism. The stem position control system of valves is denoted *positioner*.

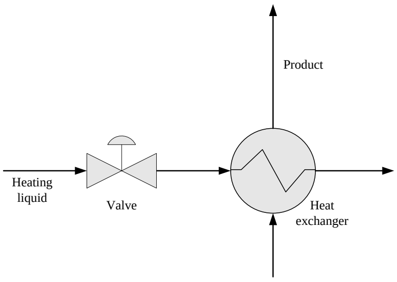


Figure 16.23: Heat exchanger.

Draw an instrumentation diagram of the total control system. You can use symbol G for position of the valve stem. (Hint: There are three control loops.)

Problem 16.4 *Which controllers?*

Figure 16.24 shows a ship. The position of the ship is controlled. Assume that it is benefical for the positional control system that the rotational speed of the propeller is controlled.

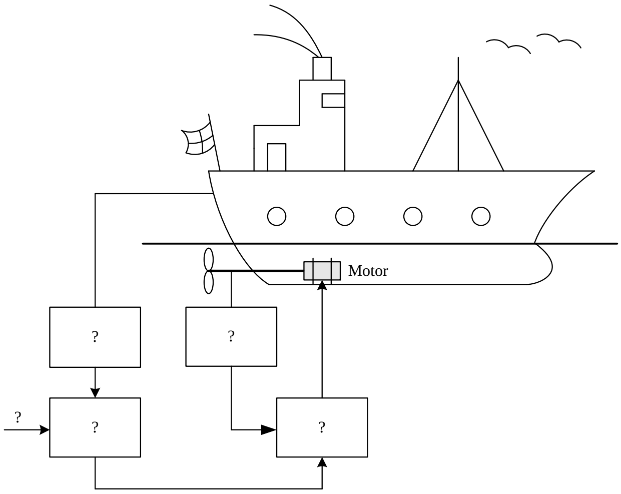


Figure 16.24: Ship

Based on the given information, substitute the question marks with proper functions (text). What are the purposes of the control loops?

Problem 16.5 *Control structure for level and flow control*

Figure 16.25 shows a tank with two inlet flows. The liquid level of the tank is to be controlled by manipulating (controlling) flow A. It is assumed that flow A is much larger than flow B. The ratio between flow B and A is specified as

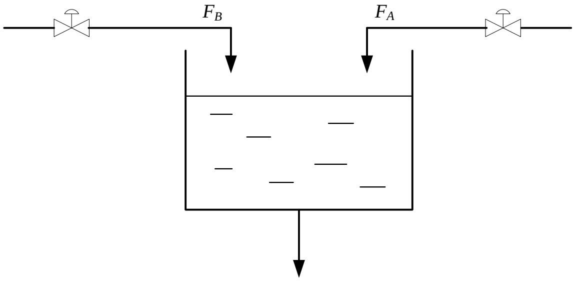


Figure 16.25: Tank

$$\frac{F_B}{F_A} = k \tag{16.20}$$

where k is a given ratio. Assume that it is necessary to have local flow control loops around each valve.

Draw a Process & Instrumentation diagram of a control system for this process.

Problem 16.6 *pH control*

Figure 16.26 shows a liquid tank where the pH value of the liquid is to be controlled with split-range control where acid flow and base flow are adjusted. Both the acid flow and the base flow are controlled with (local) flow control loops. Draw an instrumentation diagram of the tank with control system.

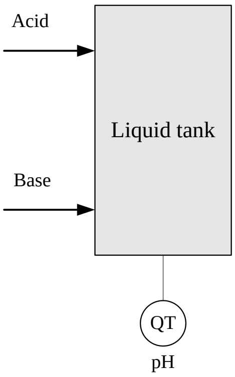


Figure 16.26: Liquid tank

Problem 16.7 *PI tuning for averaging level control*

Given a level control system with PI controller for averaging level control for a water tank with cross-sectional area $A = 50$ m^2. The largest sudden change that the inflow to the tank can have is 100 L/s. It is specified that this largest change must give a maximum level change of 20 cm. Tune the PI controller.

Problem 16.8 *Control structure of process line*

Figure 16.27 shows an uncontrolled process line consisting of a serial connection of liquid tanks. Draw a Piping & Instrumentation Diagram of the control structure of the process line based on the following specifications: The production rate (-flow) is controlled by controlling F3 to its reference. The mass balance of the liquid in each tank should be maintained. Pumps are used as actuators.

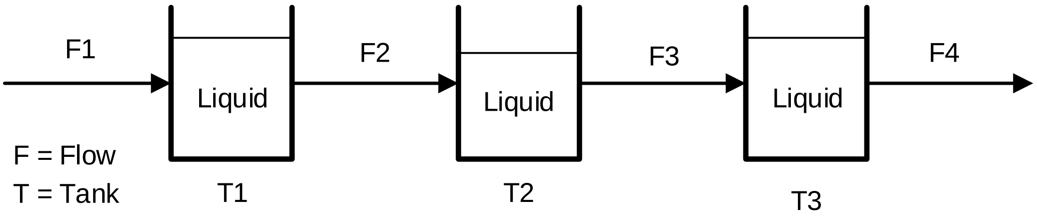


Figure 16.27: Process line.

Problem 16.9 *Control structure of process line*

Figure 16.28 shows an uncontrolled process line consisting of a serial connection of liquid tanks. Draw a Piping & Instrumentation Diagram of the control structure of the process line based on the following specifications: The production rate (-flow) is controlled by controlling F1 to its reference. The mass balance of the liquid in each tank should be maintained using pumps as actuators. The gas pressure is controlled to a reference using a control valve as actuator.

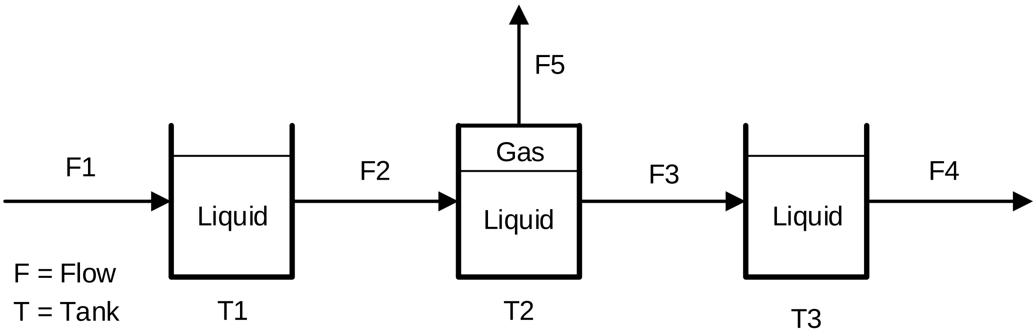


Figure 16.28: Process line.

Problem 16.10 *Control structure of level control system*

Figure 16.29 shows a wood chips tank, which is in the beginning of the pulp & paper production line. Spruce, pine and eucalyptus are used as feeds into the tank, via a conveyor belt. The percentages of each of these flows are indicated in Figure 16.29. There is a flow disturbance before the belt which is due to sieving the chip flow to remove large parts of chip.

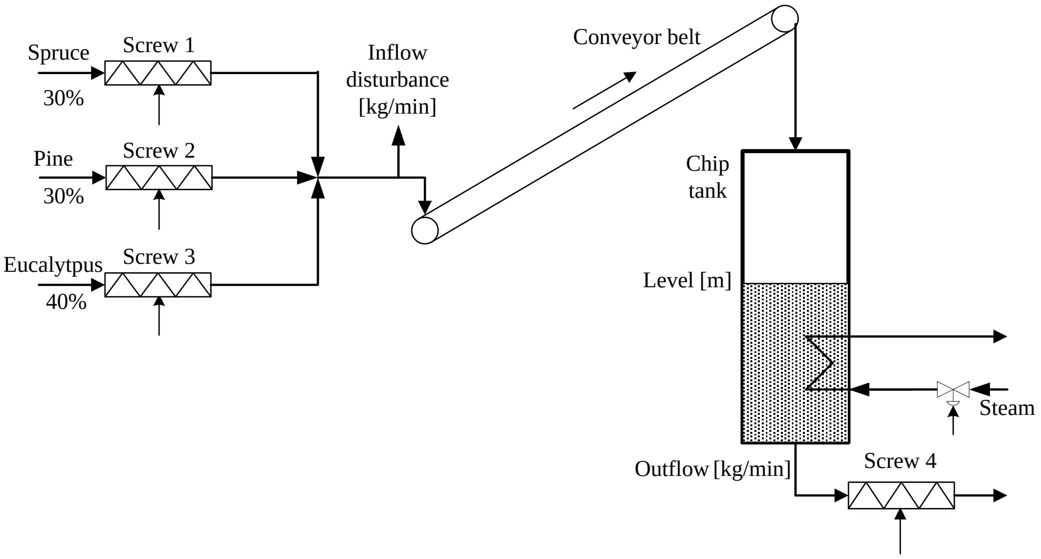


Figure 16.29: Wood chips tank

Draw a P&I (Process & Instrumentation) diagram of a control system for this part of the production line according to the following specifications:

- The production rate is controlled to a reference with flow control of Screw 4.
- The level of chip in the tank is controlled to a reference by manipulating the total inflow to the conveyor belt.
- The total chip flow into the belt is splitted into percentage flows shown in Figure 16.29. The splitting can be represented with a block with total control signal (100%) as input and three flow value outputs (30%, 30% and 40% respectively). The flows out of the three inflow screws are flow controlled.
- A flow control loop is used to compensate for the flow disturbance due to the sieveing. This flow loop is based on the measurement of the flow with a flow sensor at the beginning of the belt.
- The temperature of the chip in the tank is controlled to a reference using the steam valve.

Problem 16.11 *Control structure of destillation column*

Figure 16.30 shows an incomplete P&I (Process & Instrumentation) diagram of a controlled distillation column.

If you need it, here is some basic information about distillations columns: A distillation column contains a number of trays from where liquid can pour downwards (to the next tray) and vapour can rise upwards (to the next tray). The purpose of the distillation column is to separate the “light” component and the “heavy” component by exploiting their different boiling points of temperature. Heat is supplied to the boiler at the bottom of the column. Vapour leaving the column is condensed in the condenser. The liquid leaving the condenser is accumulated or stored in the accumulator. Part of the liquid leaving the accumulator is directed back to the column, and the rest – the distillation product – is directed to e.g. a storage tank. Ideally, the concentration of “heavy” component in the top product is zero, and the concentration of the “light” component in the bottoms product is zero. In principle this can be achieved by one quality control loop for the top product and one quality control loop for the bottoms product, but due to the dynamic properties of distrillation columns such “two-point” control is difficult to realize. Therefore, there is typically either quality control of the top product or quality control of the bottoms product.

Make the diagram shown in Figure 16.30 complete by entering letter codes in the instrumentation symbols according to these specifications: The quality of the distillate product is controlled, and there is mass balance control of various parts of the column. (The heating medium supplied to the boiler is manually controlled, so it is not adjusted by an automatic controller.)

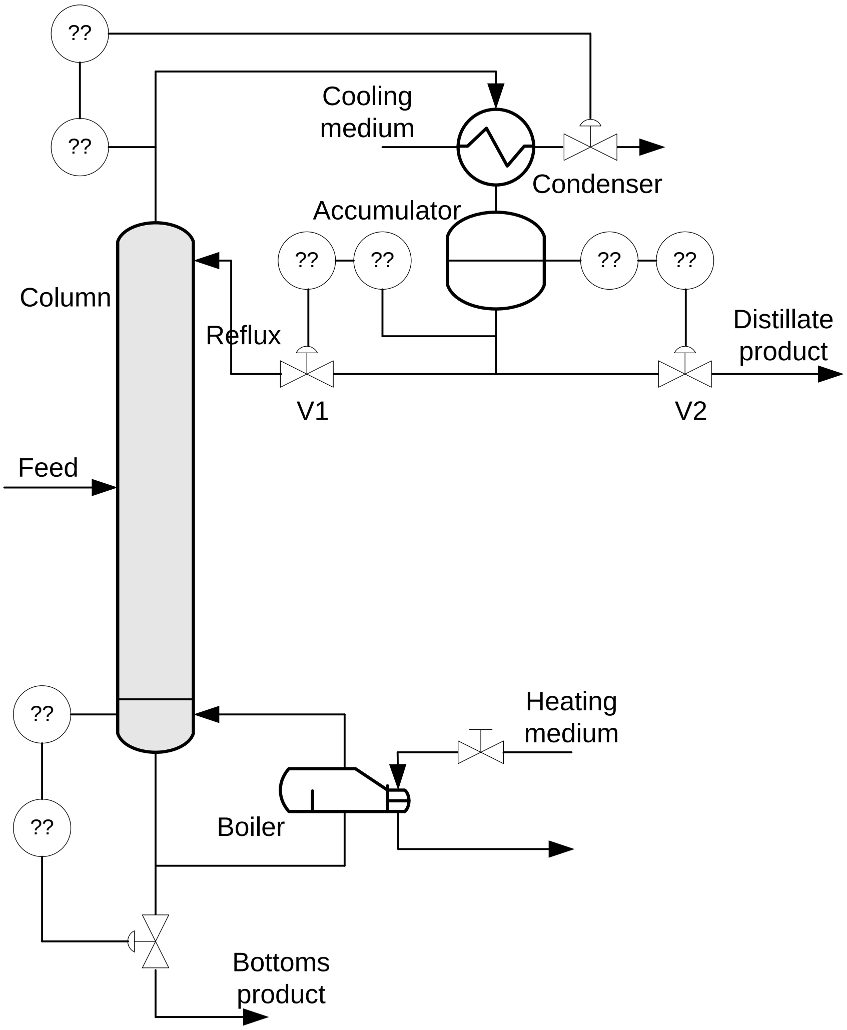


Figure 16.30: Distillation column

16.7 *Solutions to problems for Chapter 16*

Solution to Problem 16.1

Figure 16.31 shows the solution. (In the real plant, the mass flow rate is measured with a flow sensor which is in the form of a level measurement of the vertical position of the belt between two specific rollers.)

Solution to Problem 16.2

See Figure 16.32.

The purpose of the ammonia flow control loop can be to obtain an ammonia flow that tracks the flow value (flow reference) that the pH controller demands. The flow control will

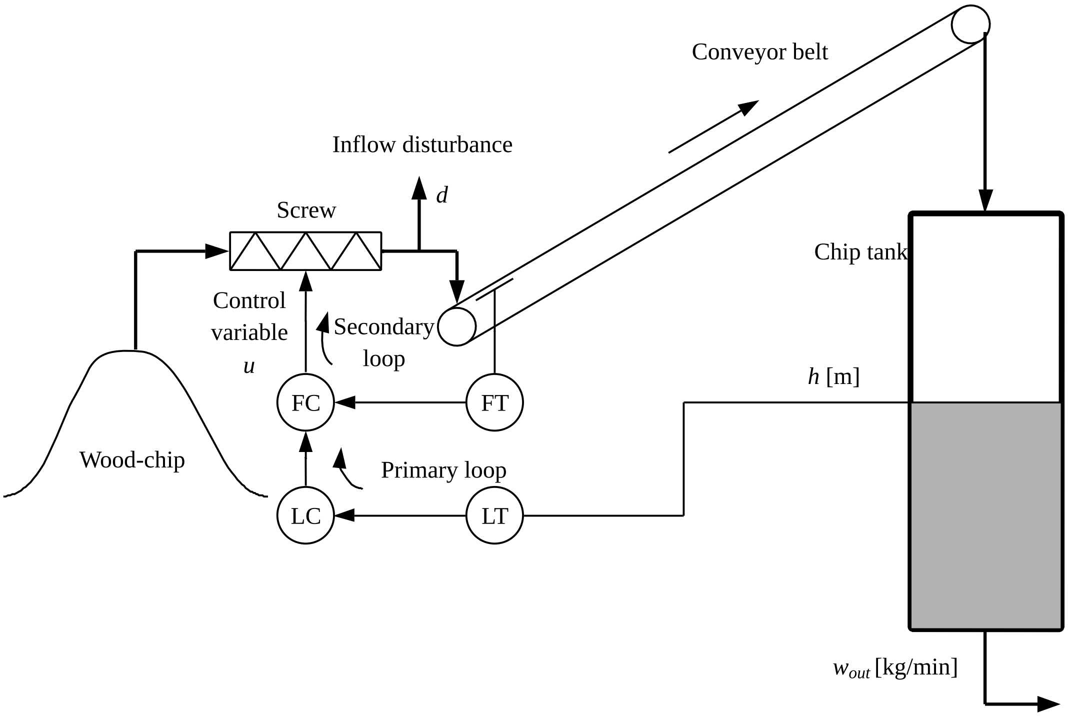


Figure 16.31: Cascade control system of wood chipss tank.

compensate for flow variations caused by e.g. pressure variations in the ammonia gas supply.

Solution to Problem 16.3

Figure 16.33 shows the instrumentation diagram.

Solution to Problem 16.4

Figure 16.34 shows the ship with control system.

The purpose of the position control loop is to make the ship track the position reference.

The purpose of the propeller speed control loop is to make the propeller speed track the speed command or reference generated by the positional controller. The speed control loop will compensate for disturbances acting on the propeller so that the propeller speed is more smooth. In addition, this speed control loop may make the tuning of the position controller easier because the position controller will control the speed more directly.

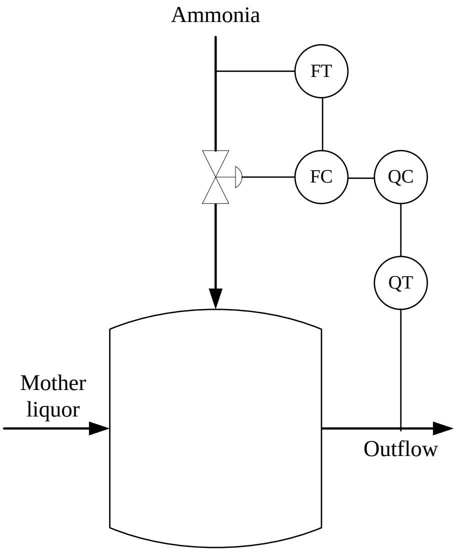


Figure 16.32: pH control system

Solution to Problem 16.5

See Figure 16.35.

Solution to Problem 16.6

Figure 16.36 shows the pH control system.

Solution to Problem 16.7

Given:

$$A = 50 \text{ m}^2 \tag{16.21}$$

$$\Delta F_{\text{in}} = 100 \text{ L/s} = 0.1 \text{ m}^3\text{/s} \tag{16.22}$$

$$\Delta h = 20 \text{ cm} = 0.2 \text{ m} \tag{16.23}$$

This gives:

$$T_c = A\frac{\Delta h}{\Delta F_{\text{in}}} = 50 \cdot \frac{0.2}{0.1} = 100 \text{ s} \tag{16.24}$$

$$K_c = -\frac{A}{T_c} = -\frac{\Delta F_{\text{in}}}{\Delta h} = -\frac{0.1}{0.2} = -0.5 \text{ (m}^3\text{/s)/m} \tag{16.25}$$

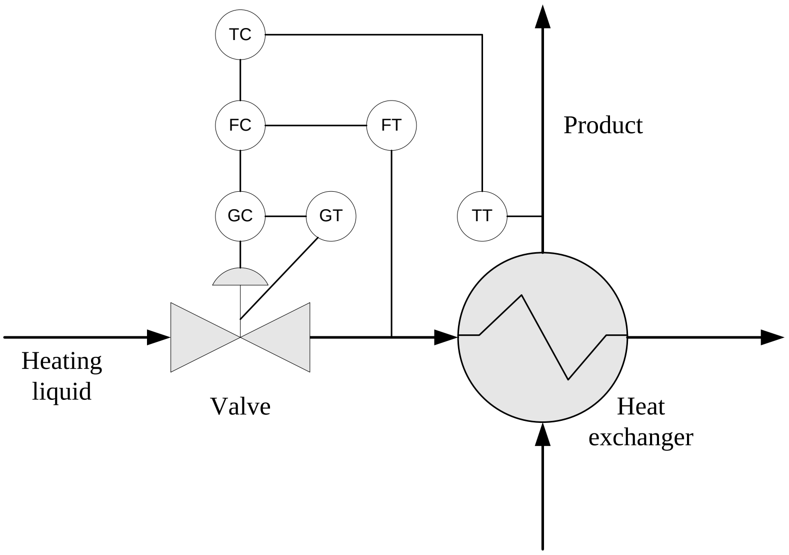


Figure 16.33: Cascade control of heat exchanger

$$T_i = 2T_c = 200 \text{ s} \tag{16.26}$$

Solution to Problem 16.8

See Figure 16.37.

Solution to Problem 16.9

See Figure 16.38.

Solution to Problem 16.10

See Figure 16.39.

Solution to Problem 16.11

See Figure 16.40.

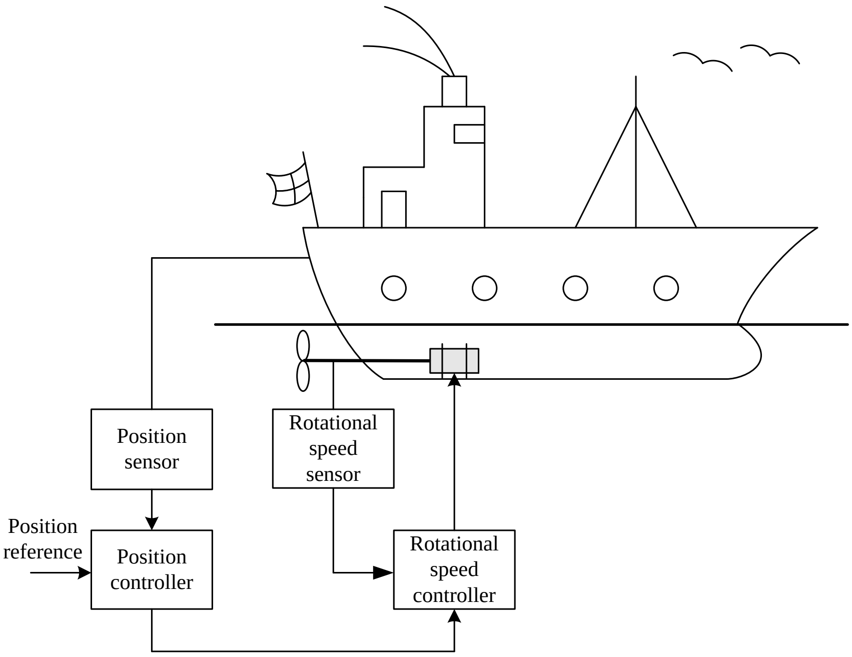


Figure 16.34: Cascade control of ship

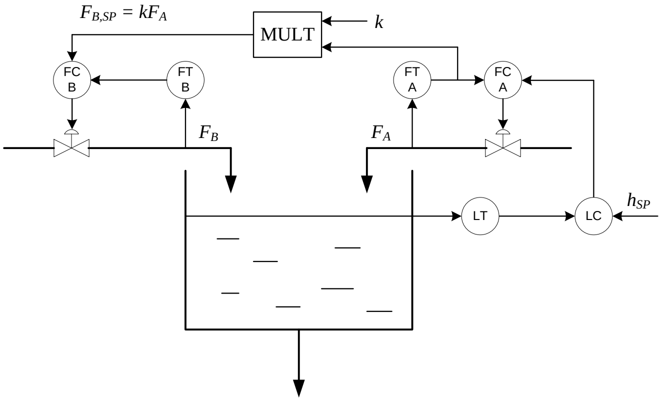


Figure 16.35: Ratio control and cascade control

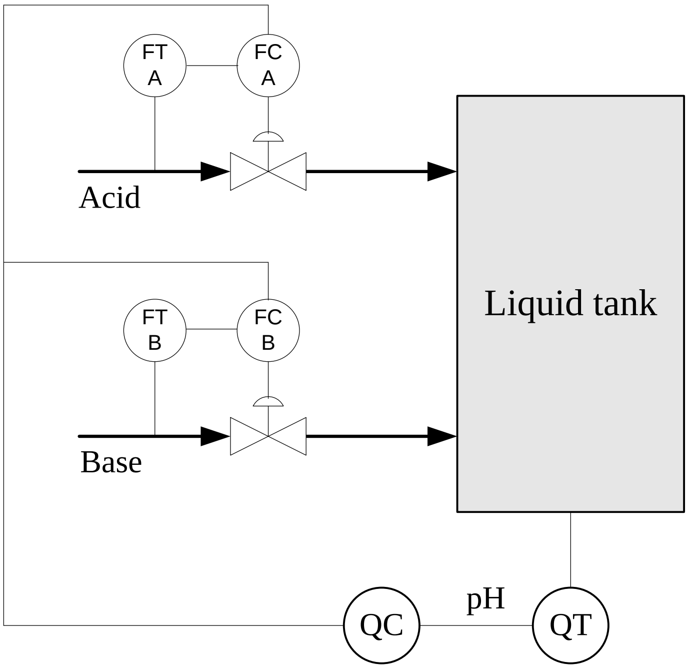


Figure 16.36: Split-range control system

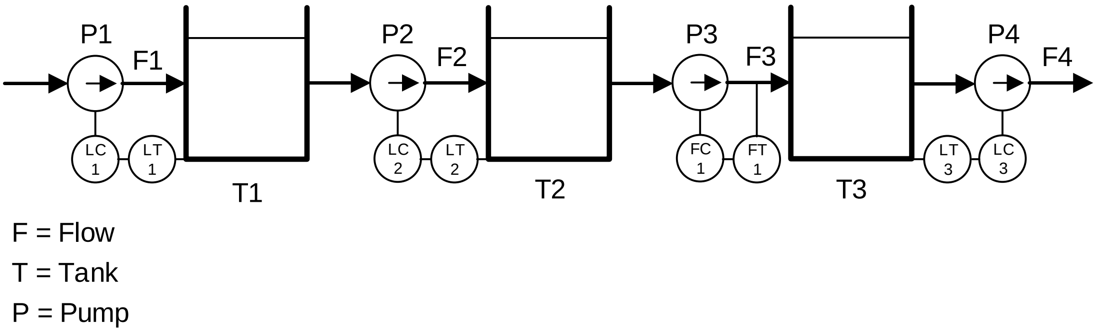


Figure 16.37: Process line with control system.

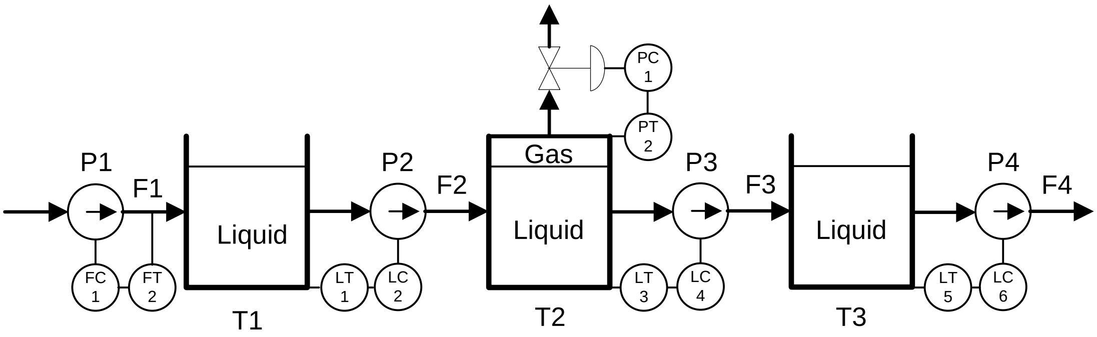


Figure 16.38: Process line with control system.

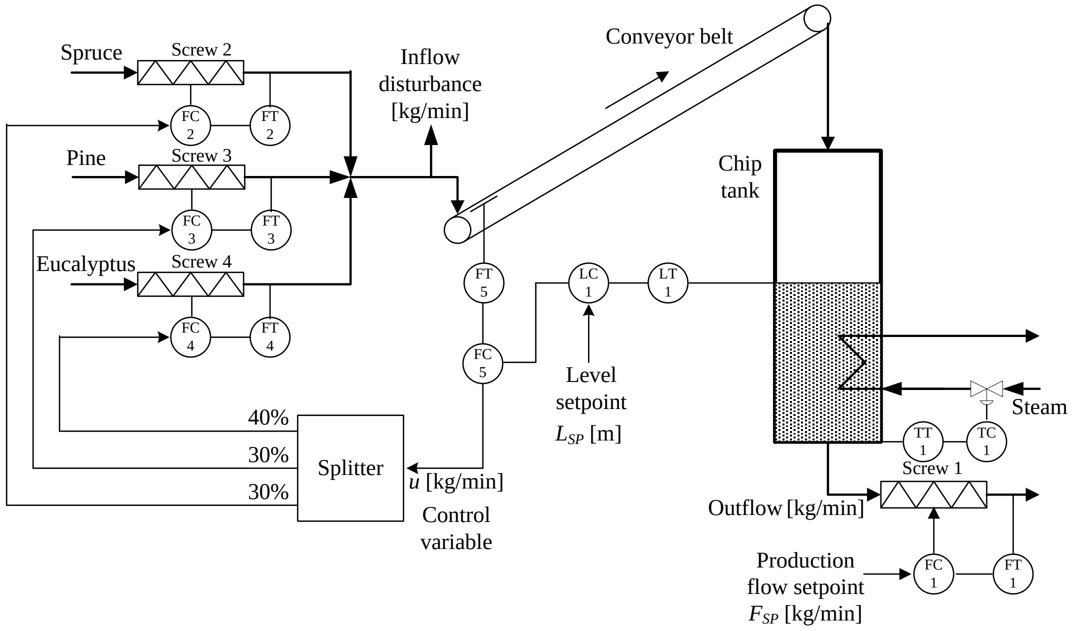


Figure 16.39: Level control system of wood chips tank

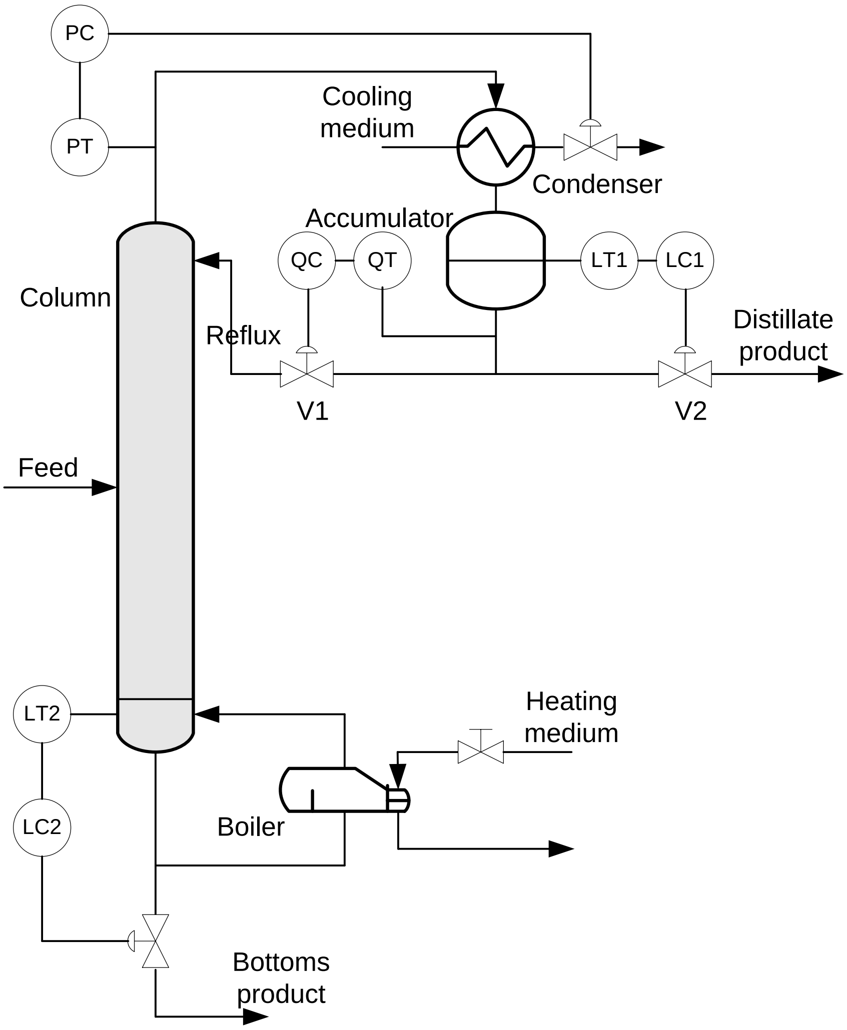


Figure 16.40: Control system of distillation column

Chapter 17

Feedforward control

17.1 Introduction

We know from Chapters 1.3 and 11 that feedback control – or error-driven control – can bring the process output variable to or close to the reference. Feedback control is in most cases a sufficiently good control method. But improvements can be made, if required. A problem with feedback is that there is no adjustment of the control variable before the control error is different from zero, since the control variable is adjusted as a function of the control error. This problem does not exist in *feedforward control*.

Feedforward control generates a contribution to the control signal using knowledge about the following three components:

- The reference, which is always known.
- The process disturbance(s), assumed measured so that its value is known.
- A mathematical process model (linear or non-linear) which conveys process knowledge in mathematical terms.

Using the above information, the feedforward controller generated (ideally) a perfect control signal making the process output variable, y, equal to its reference, r.

However, a perfect feedforward is not possible to realize since there are always some process disturbances which are not measured, and because the process model is never perfect. In other words, perfect feedforward can not be realized fully because of model errors, causing the control error to becomes different from zero. But by including feedback control, this error may be reduced. In practice, feedforward control is used together with feedback control.

17.2 The structure of feedforward control together with feedback control

Figure 17.1 shows the structure of a control system with both feedforward and feedback control.

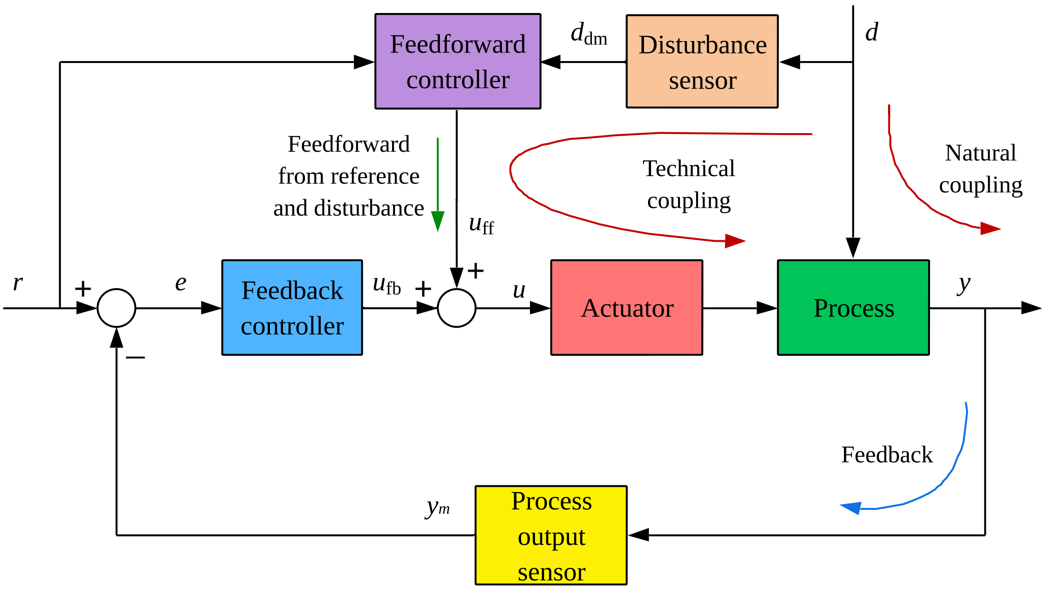


Figure 17.1: Control system with both feedforward and feedback control.

In Figure 17.1, the feedforward from disturbance can be interpreted as a constructed, technical coupling from the disturbance to the process output variable which is supposed to counteract, or cancel out, the natural coupling so that the net effect that the disturbance has on the process output y is zero, which is our aim.

The feedforward control signal is added to the feedback control signal to make the total control signal:

$$u = u_{\mathrm{fb}} + u_{\mathrm{ff}} \tag{17.1}$$

The feedforward controller, which generated u_{ff} in Figure 17.1, can be developed in several ways:

- From a differential equations process model, cf. Section 17.3.
- From a transfer functions process model. This is not described in this book, but the procedure is as for differential equation models, assuming you have found the differential equation(s) that corresponds to the given transfer functions.

- From experimental data, cf. Section 17.5 This method is model-free, and should be regarded as an approximative method, which still may give substantial improvement of control system performance.

Note that using feedforward together with feedback does not influence the stability of the feedback loop because the feedforward does not introduce new dynamics in the loop.

17.3 Designing feedforward controllers from differential equation models

The feedforward controller can be designed with the following two-step procedure.

How to design a feedforward controller

1. Substitute the process output variable, y, by its desired value, the reference, r.
2. Solve the process model for the control signal, u. Denote this u by u_{ff}, the feedforward control signal.

One practical issue: Typically, the feedforward controller comprise the time-derivative of r. The time-derivative is very sensitive to abrupt changes of r. Two solutions to this problem are:

- r is lowpass filtered before the differentiation. If the reference appears in the feedforward controller with its first order time derivative, a time constant filter will probably be ok, see Figure 17.2 where a time constant filter is assumed. If the reference appears in the feedforward controller with its second order time derivative, a second order filter should be used. A simple second order filter may be constructed with two time constant filters in series.
- Only smooth changes of r are allowed, e.g. a sigmoid or sinusoidal change between reference values.

Example 17.1 *Feedforward level control of a liquid tank*

Figure 17.3 shows a liquid tank with inflow and outflow. The level h is to be controlled using feedback with PID controller in combination with feedforward control. (The responses are explained later in this example.)

From a mass balance of the liquid in the tank we get the following process model:

$$\rho A h' = F_{\text{in}} - F_{\text{out}} \tag{17.2}$$

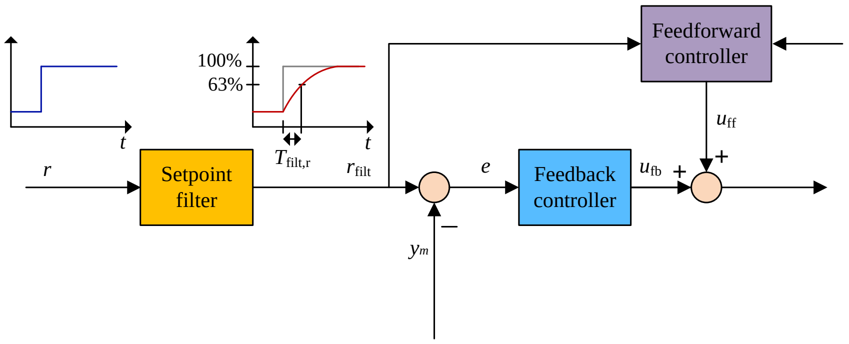


Figure 17.2: A lowpass filter generates a smooth reference to be used by the feedforward controller (and the feedback controller).

where h [m] is liquid level, F_{in} [kg/s] is mass inflow, F_{out} [kg/s] is mass outflow, A [m^2] is cross sectional area of the tank, ρ [kg/m^3] is the liquid density. F_{in} is assumed to be equal in value to the applied control signal u. By using $F_{\text{in}} = u$ the model becomes:

$$\rho A h' = u - F_{\text{out}} \tag{17.3}$$

Now, let us derive the feedforward control function from the process model (17.3), follwing the two-step procedure presented above:

Firstly, we substitute the level h by its reference r:

$$\rho A r' = u - F_{\text{out}} \tag{17.4}$$

Secondly, we solve (17.4) for the control variable u to get the feedforward control variable u_{ff}:

$$u_{\text{ff}} = \underbrace{\rho A r'}_{u_{\text{ff,r}}} + \underbrace{F_{\text{out}}}_{u_{\text{ff,d}}} \tag{17.5}$$

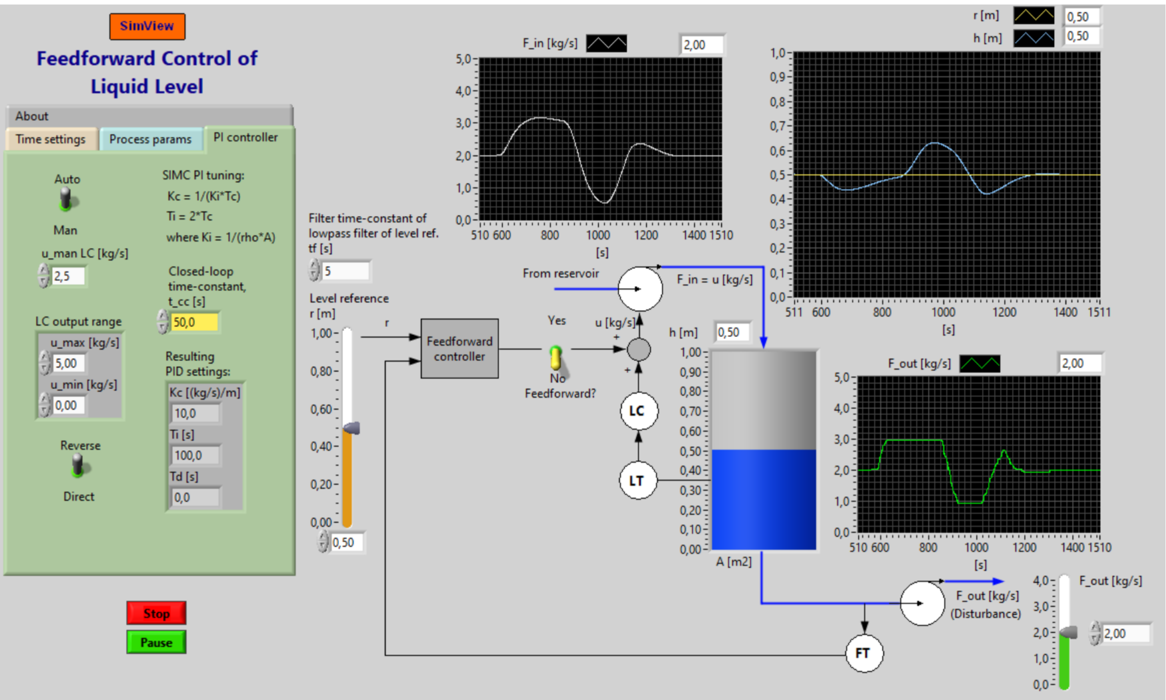
which is the ideal feedforward controller. $u_{\text{ff,r}} = \rho A r'$ represents feedforward from reference, and $u_{\text{ff,d}} = F_{\text{out}}$ represents feedforward from disturbance. We see that calculation of feedforward control signal u_{ff} requires measurement or knowledge of the following four quantities: ρ, A, F_{out}, and r'. (Figure 17.3 indicates that flow F_{out} is measured.)

Does the feedforward controller (17.5) make sense?

- The term
$$u_{\text{ff,d}} = F_{\text{out}} \tag{17.6}$$
tells that the inflow should be equal to the outflow at any instant of time to cancel the impact of the outflow in the level. Makes sense?[1]

- The term
$$u_{\text{ff,r}} = \rho A r' \tag{17.7}$$

[1] Yes.



SimView

Feedforward Control of Liquid Level

About | Time settings | Process params | PI controller

Auto / Man

u_man LC [kg/s]: 2,5

LC output range
u_max [kg/s]: 5,00
u_min [kg/s]: 0,00

Reverse / Direct

SIMC PI tuning:
Kc = 1/(Ki*Tc)
Ti = 2*Tc
where Ki = 1/(rho*A)

Closed-loop time-constant, t_cc [s]: 50,0

Resulting PID settings:
Kc [(kg/s)/m]: 10,0
Ti [s]: 100,0
Td [s]: 0,0

Filter time-constant of lowpass filter of level ref. tf [s]: 5

Level reference r [m]: 0,50

Feedforward? Yes / No

F_in [kg/s]: 2,00

r [m]: 0,50
h [m]: 0,50

h [m]: 0,50

F_out [kg/s]: 2,00

F_out [kg/s] (Disturbance): 2,00

Stop

Pause

Figure 17.3: Example 17.1: Liquid tank where the level h is controlled with feedback and feedforward control. In the simulation shown in this figure, feedforward control is *not* in use.

> tells that if the reference is changing, the inflow should be increased, or decreased – depending on the sign of r', equal to the specified rate of change of the mass in the tank, which is $\rho A r'$.

In (17.5), r should be lowpass filtered before it is time-differentiated. A first order lowpass filter in the form of a time constant filter may be used:

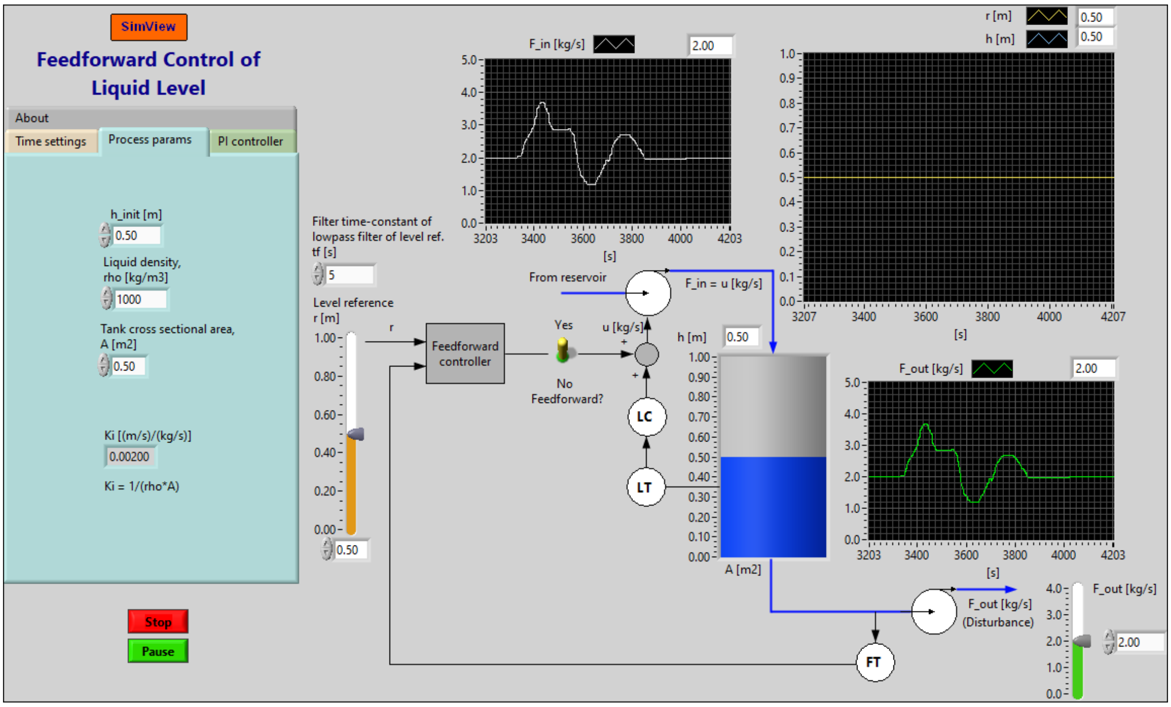
$$R_{\text{filt}}(s) = \frac{1}{t_{\text{fr}}s + 1}R(s) \tag{17.8}$$

where t_{fr} is the filter time constant which may be tuned by trial-and-error, and $R(s)$ is the Laplace transform of $r(t)$.

In the following, two cases are simulated: Level control *without* and *with feedforward control.* In both cases there is feedback control with PI controller with parameters $K_c = 10$ and $T_i = 100$ s. In the first part of the simulation which is the first 1000 seconds, the level reference r is varied while the outflow (disturbance) is constant, while in the last part of the simulation which is the last 1000 seconds, the level reference is kept constant while the outflow F_{out} is varied.

- **Without feedforward control** (only feedback control): Figure 17.3 shows the responses in the level h as the reference is varied and as the disturbance (outflow) is varies. The level deviates clearly from the reference in both situations.
- **With feedforward control** (in addition to feedback control): Figure 17.4 shows the responses in the level with almost the same variations of the reference and the

disturbance as when only feedback control is used. We see that the level now deviates very little from the reference. The control performance is substantially improved.


h_init [m] 0.50

Liquid density, rho [kg/m3] 1000

Tank cross sectional area, A [m2] 0.50

Ki [(m/s)/(kg/s)] 0.00200

Ki = 1/(rho*A)

Filter time-constant of lowpass filter of level ref. tf [s] 5

Level reference r [m] 0.50

F_in [kg/s] 2.00

r [m] 0.50

h [m] 0.50

h [m] 0.50

F_out [kg/s] 2.00

F_out [kg/s] 2.00

Figure 17.4: Example 17.1: Liquid tank where the level h is controlled with feedback control. Here, feedforward control is applied.

Note: Without feedforward control the control signal range of the PID controller is [0, 5] (in unit of kg/s). With feedforward the output signal range of the PID controller was set to [−5, +5] so that the contribution, u_{PID}, from the PID controller can be negative. If u_{PID} can not become negative, the total control signal, which is

$$u = u_{\text{PID}} + u_{\text{ff}} \tag{17.9}$$

may not get small enough value to give proper control when the outflow is small. (This was confirmed by a simulation, but the responses are not shown here.)

The simulations in this example are made with the following SimView simulator:

https://techteach.no/simview/feedforward_level_control

[End of Example 17.1]

17.4 Designing feedforward controllers from transfer function models

You can design a feedforward controller for process represented with a transfer function model using the same procedure as for differential equation models, cf. Section 17.3. Let's see what it looks like with a general transfer function model. Figure 17.5 shows a transfer function model of a process. The model is

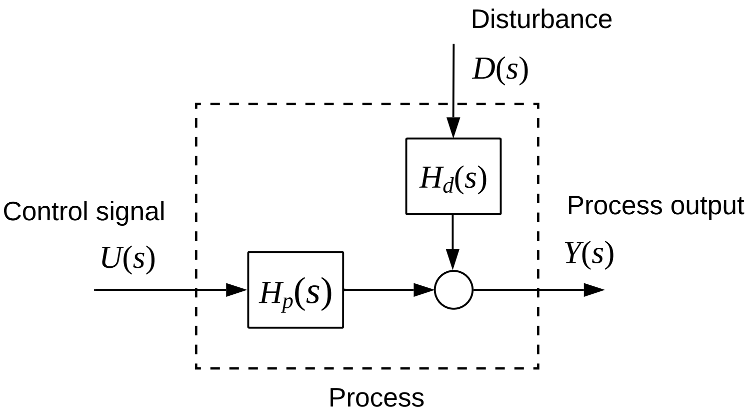


Figure 17.5: Process model i terms of transfer functions.

$$Y(s) = H_p(s)U(s) + H_d(s)D(s) \tag{17.10}$$

Firstly, we substitute the process output variable, $Y(s)$, by its desired reference, $R(s)$, to get

$$R(s) = H_p(s)U(s) + H_d(s)D(s) \tag{17.11}$$

Secondly, we solve (17.11) for the control signal, $U(s)$, and denote this $U(s)$ by $U_{\text{ff}}(s)$.

$$U_{\text{ff}}(s) = \overbrace{\underbrace{\left[\frac{1}{H_p(s)}\right]}_{H_{\text{ffr}}(s)} R(s)}^{U_{\text{ffr}}(s)} + \overbrace{\underbrace{\left[-\frac{H_d(s)}{H_p(s)}\right]}_{H_{\text{ffd}}(s)} D(s)}^{U_{\text{ffd}}(s)} \tag{17.12}$$

Eq. (17.12) is the feedforward controller!

In (17.12):

- Feedforward from reference:

$$\frac{U_{\text{ffr}}(s)}{R(s)} = H_{\text{ffr}}(s) = \frac{1}{H_p(s)} \tag{17.13}$$

- Feedforward from disturbance:

$$\frac{U_{\text{ffd}}(s)}{D(s)} = H_{\text{ffd}}(s) = -\frac{H_d(s)}{H_p(s)} \tag{17.14}$$

As explained in Section 17.1, feedforward control will typically come in *addition* to feedback control, see Figure 17.1. Based on this principle, the structure of the control system based on transfer function models is as shown in Figure 17.6.

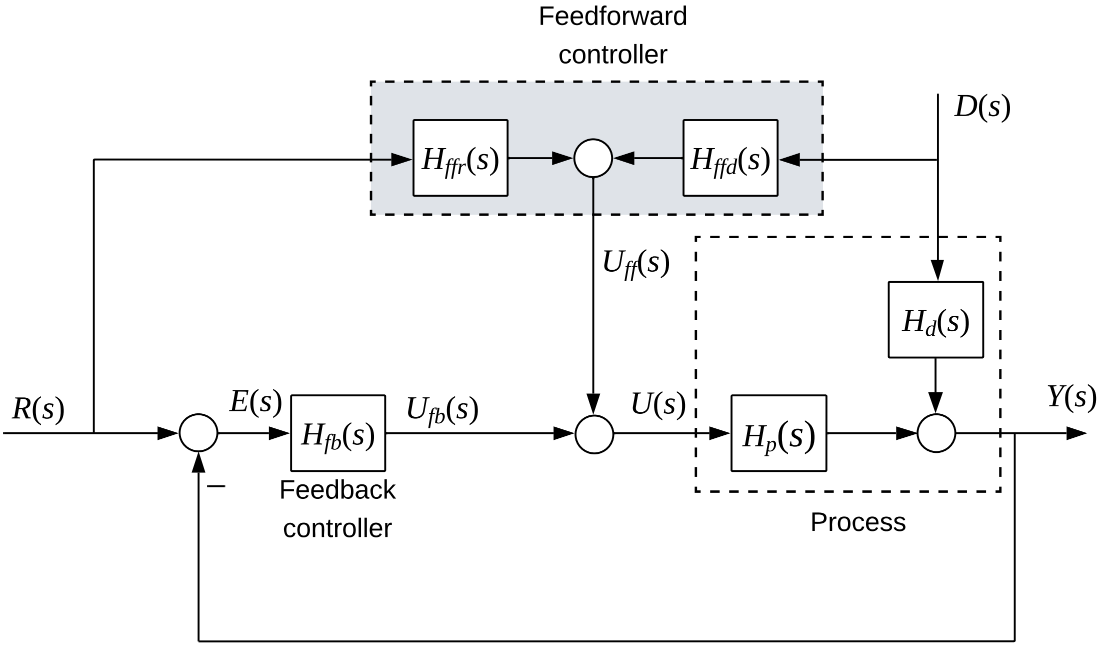


Figure 17.6: Control system with both feedforward control and feedback control.

Example 17.2 *Designing a feedforward based on a transfer function process model*

Assume a process as in Figure 17.5 with the following transfer functions, which are "time constant" transfer functions:

$$H_p(s) = \frac{K_p}{t_{cp}s + 1} \tag{17.15}$$

$$H_d(s) = \frac{K_d}{t_{cd}s + 1} \tag{17.16}$$

where K_p and K_d are gains, and t_{cp} and t_{cd} are time constants.

The individual, additive parts of the feedforward control signal given by (17.12) are:

- Feedforward from reference:

$$U_{\text{ffr}}(s) = \frac{1}{H_p(s)}R(s) = \frac{t_{cp}s + 1}{K_p}R(s) = \frac{t_{cp}}{K_p}sR(s) + \frac{1}{K_p}R(s) \tag{17.17}$$

- Feedforward from disturbance:

$$U_{\text{ffd}}(s) = -\frac{H_d(s)}{H_p(s)}D(s) = -\frac{K_d}{K_p}\frac{t_{cp}s+1}{t_{cd}s+1}D(s) \tag{17.18}$$

The total feedforward control signal is

$$U_{\text{ff}}(s) = U_{\text{ffr}}(s) + U_{\text{ffd}}(s) \tag{17.19}$$

Comments to these results:

- The transfer function (17.17) consists of a derivative part and a proportional part:
  - In the term $\frac{t_{cp}}{K_p}sR(s)$, the factor s represents time differentiation with respect to time, i.e. the term is $\frac{t_{cp}}{K_p}$times the time derivative $r'(t)$ in the time domain.
  - The term $\frac{1}{K_p}R(s)$ is a proportional term, which is $\frac{1}{K_p}r(t)$ in the time domain.
- The transfer function (17.18) is denoted a lead-lag transfer function. The term "lead-lag" conveys the frequency response properties of the transfer function. "Lead" means positive phase in the frequency response, while "lag" means negative phase in the frequency response.
  - If $t_{cp} = t_{cd}$, (17.18) is just a gain, which means that it produce a feedforward control signal being proportional to the (measured) disturbance.
  - If $t_{cp} > t_{cd}$, (17.18) is "lead dominated", which means it reacts quickly (almost like a highpass filter) to changes in the disturbance.
  - If $t_{cp} < t_{cd}$, (17.18) is "lag dominated", which means it reacts slowly (almost like a lowpass filter) to changes in the disturbance.

Implementation of the feedforward controller as a computer algorithm

Above, we have have designed the feedforward controller in terms of transfer functions, see (17.17) and (17.17). If your control software allows you to implement these transfer functions, you are done. But if you are to implement the feedforward controller from scratch in a computer program, you must have an algorithm to program. Let's see how we can obtain such an algorithm. We take the feedforward from reference first, and then the feedforward from disturbance.

Feedforward from reference:

In the time domain, (17.17) is

$$u_{\text{ffr}}(t) = \frac{t_{cp}}{K_p}r'(t) + \frac{1}{K_p}r \tag{17.20}$$

We may calculate r' numerically using backward differentiation with time step t_s. The algorithm ready to be programmed, is then

$$u_{\text{ffr},k} = \frac{t_{cp}}{K_p} \cdot \frac{r_k - r_{k-1}}{t_s} + \frac{1}{K_p}r_k \tag{17.21}$$

It may be wise to let the reference pass through a lowpass filter before it is used in the feedforward controller, cf. the comments about such filtering in Section 17.3.

Feedforward from disturbance:

We need to find out what (17.18) means in the time domain. We can start by cross-multiplying in (17.18) to get

$$K_p \left(t_{cd} s + 1 \right) U_{\text{ffd}}(s) = -K_d \left(t_{cp} s + 1 \right) D(s) \tag{17.22}$$

Dissolving the parenthesis gives

$$K_p t_{cd} s U_{\text{ffd}}(s) + K_p U_{\text{ffd}}(s) = -K_d t_{cp} s D(s) + K_d D(s) \tag{17.23}$$

In the time domain, (17.23) means

$$K_p t_{cd} \, {u_{\text{ffd}}}'(t) + K_p u_{\text{ffd}}(t) = -K_d t_{cp} d'(t) + K_d d(t) \tag{17.24}$$

Appying backward differentiation in (17.24) gives

$$K_p t_{cd} \frac{u_{\text{ffd},k} - u_{\text{ffd},k-1}}{t_s} + K_p u_{\text{ffd},k} = -K_d t_{cp} \frac{d_k - d_{k-1}}{t_s} + K_d d_k \tag{17.25}$$

Solving for $u_{\text{ffd},k}$ gives the algorithm ready to be programmed:

$$u_{\text{ffd},k} = \left[\left(K_p t_{cd}/t_s \right) u_{\text{ffd},k-1} - K_d t_{cp} \left(d_k - d_{k-1} \right)/t_s + K_d d_k \right] / \left(K_p t_{cd}/t_s + K_p \right) \tag{17.26}$$

Total feedforward control signal

The total feedforward control signal is the sum of (17.21) and (17.26):

$$u_{\text{ff},k} = u_{\text{ffr},k} + u_{\text{ffd},k} \tag{17.27}$$

[End of Example 17.2]

17.5 Designing feedforward control from experimental data

Feedforward control can be designed from experimental data as follows:

- Decide a proper set of N different values of the disturbance d on which the feedforward control will be based, for example $N = 6$ different values of the disturbance.
- For each of these N distinct disturbance values, find (experimentally or by simulation) the value of the control signal u which corresponds to zero steady state control error. This can (should) be done with PI or PID feedback control. (Typically feedback control is used together with feedforward control, so no extra effort is needed to run the feedback control here.)

- The set of N corresponding values of d and u can be represented by a table, cf. Table 17.1, or in a coordinate system, cf. Figure 17.7.
- For any given (measured) value of the disturbance, the feedforward control signal u_f is calculated using interpolation, for example linear interpolation as shown in Figure 17.7. In practice, this linear interpolation can be implemented using a table lookup function.[2]

Table 17.1: N corresponding values of u and u

u	d
u_1	d_1
u_2	d_2
u_3	d_3
u_4	d_4
u_5	d_5
u_6	d_6

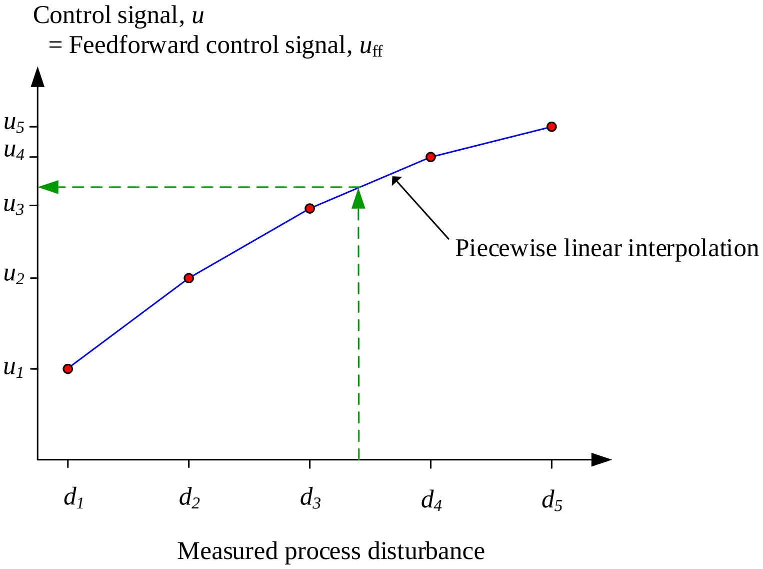


Figure 17.7: Calculation of feedforward control signal from known disturbance value

Note: This feedforward design method is based on *steady state* data. Therefore, the feedforward control will not be ideal or perfect. However, it is easy to implement, and it may give substantial better control compared to only feedback control.

Example 17.3 *Temperature control with feedforward from flow*

Figure 17.8 shows a lab process consisting of a heated air tube where the air temperature will be controlled. (The air heater is presented in more detail in Appendix 39.5.) The

[2] Both MATLAB/SIMULINK and LabVIEW have functions that implement linear interpolation between tabular data.

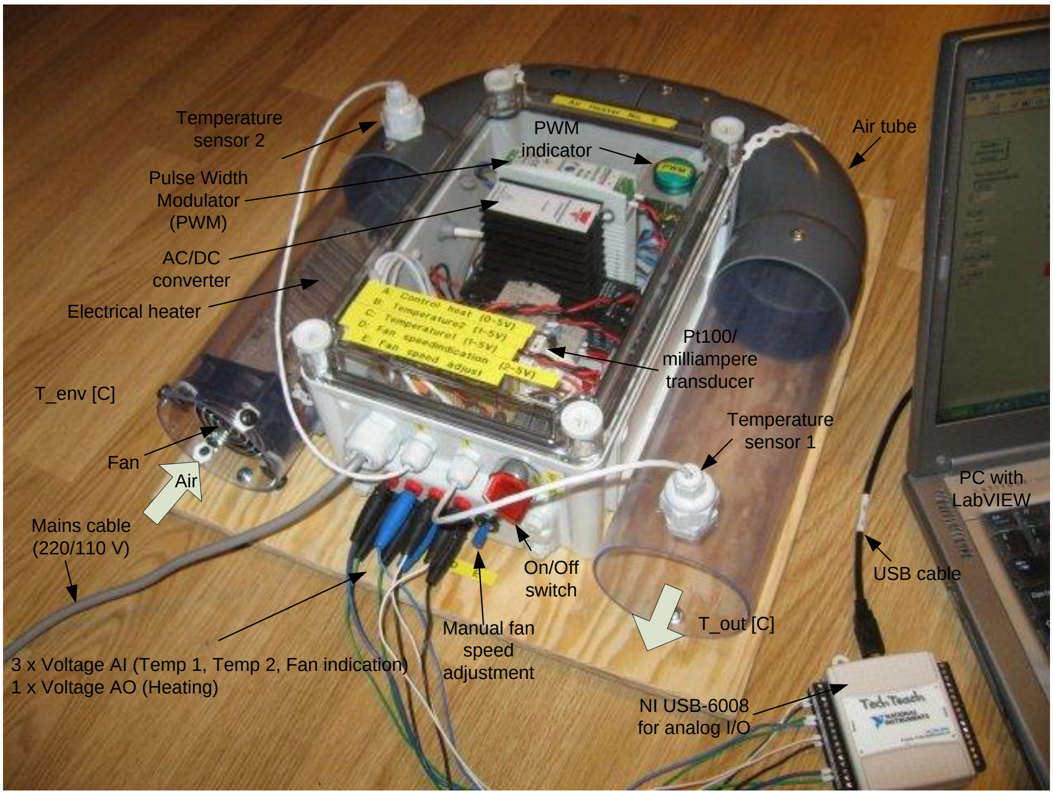


Figure 17.8: Laboratory air heater.

control signal adjusts the power to the heater. The air temperature is measured by the primary Pt100 element. The feedback PID control is based on this temperature measurement. (The control system is implemented with a LabVIEW program running on a laptop PC.)

Variations of the air flow act as disturbances to the process. The feedback controller tries to compensate for such variations using the temperature measurement. Can we obtain improved control by also basing the control signal on measured air flow, which is here available as the fan speed indication? First, ordinary PID control without feedforward is tried. The fan speed was changed from minimum to maximum, and then back again. The temperature reference was 40 %. Figure 17.9 shows the fan speed and the response in the temperature (which is represented in % with the range [0–100%] corresponding to [20–70^{o}C]).

The maximum control error was 1.0 %.

Will there be any improvement by using feedforward control from the fan speed (air flow)? A number of corresponding values of fan speed and control signal was found experimentally. The feedforward control signal, u_f, was calculated by linear interpolation with Interpolate 1D Array function in LabVIEW, and was added to the PID control signal to make up the total control signal: $u = u_{PID} + u_f$. Figure 17.10 shows the fan speed and the response in the temperature. Also, the set of 6 corresponding values of control signal and fan speed, on which the feeedforward control signal is based, is shown.

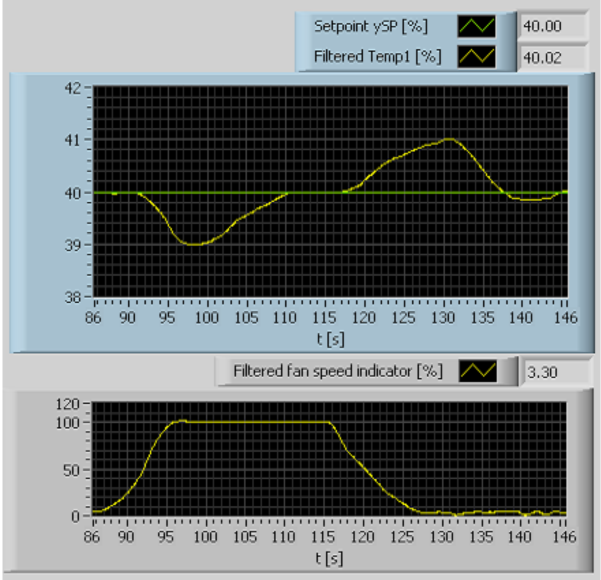
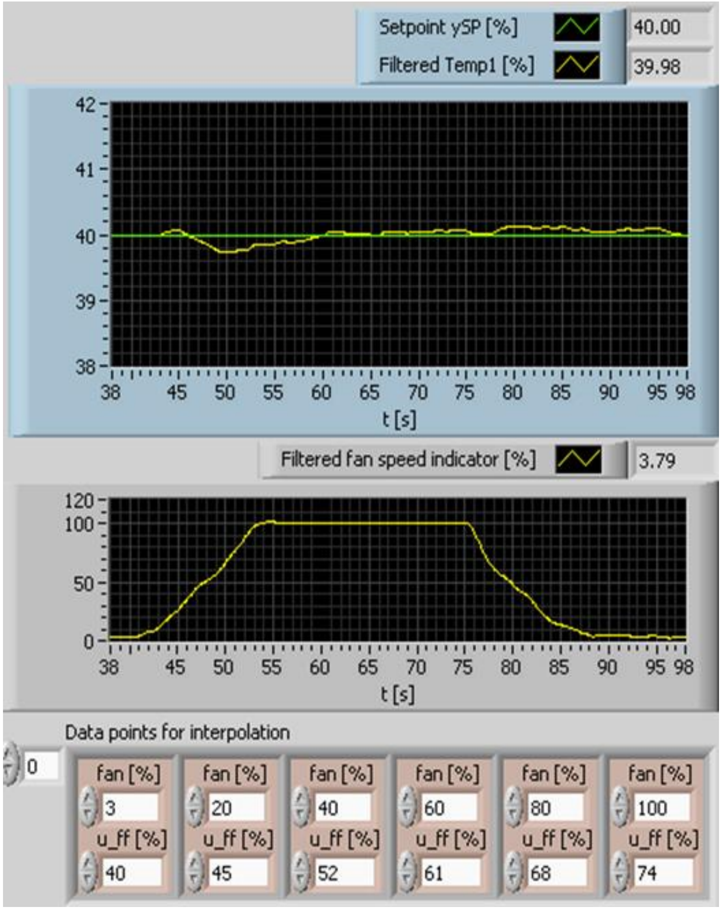


Figure 17.9: Example 17.3: The response in the temperature after a change in the fan speed. Only feedback control (no feedforward) control is used.

In this case the maximum control error was 0.27, which is a large improvement compared to using no feedforward control!

[End of Example 17.3]



Setpoint ySP [%] 40.00

Filtered Temp1 [%] 39.98

Filtered fan speed indicator [%] 3.79

Data points for interpolation

0

fan [%]	fan [%]	fan [%]	fan [%]	fan [%]	fan [%]
3	20	40	60	80	100
u_ff [%]	u_ff [%]	u_ff [%]	u_ff [%]	u_ff [%]	u_ff [%]
40	45	52	61	68	74

Figure 17.10: Example 17.3: The response in the temperature after a change in the fan speed. Feedforward from fan speed (air flow) is used together with feedback control. u_ff is the feedforward control signal, u_f.

17.6 *Problems for Chapter 17*

Problem 17.1 *Deriving a feedforward controller*

Given the following model of a process to be controlled:

$$Ty' = -y + K_1 u + K_2 d$$

where y is the output variable, u is the control variable, d is the process disturbance variable, and T, K_1 and K_2 are parameters. The reference of y is r.

1. Derive a feedforward controller for this process.
2. What quantities must be known for the feedforward controller to be realizable?
3. In general, why is it typically necessary in practice to combine feedforward control with feedback control?

Problem 17.2 *Feedforward controller for ship position control*

A mathematical model of a ship is presented in Ch. 39.9. The model is shown below, for convience:

In mathematical terms:

$$my'' = F_p + D_h \left(u_c - y'\right) \left|u_c - y'\right| + D_w \left(V_w - y'\right) \left|V_w - y'\right| \tag{17.28}$$

F_p is the control variable. Assume that the positional reference is r [m].

1. Design a feedforward controller for the ship. What information is needed to implement the feedforward controller? Is it realistic to get this information?

2. Challenging: Program a simulator, using elementary Python code, of the ship with both feedback control with a PID controller, and feedforward control. You can tune the PID controller using the SIMC method with closed loop time constant $T_c = 200$ s, cf. Section 14.8.5. Compare feedback control (and no feedforward control) and feedback + feedforward control. You can let the position reference be a smoothly varying signal, for example

$$\begin{aligned} r(t) &= 20\left[1 - \cos\left(\frac{2\pi}{1000}t\right)\right], \ 0 \leq t < 500 \text{ s} \\ &= 20, \ t \geq 500 \text{ s} \end{aligned} \tag{17.29}$$

and let the wind disturbance change as a step, for example from 0 to –20 m/s at $t = 2500$ s.

Problem 17.3 *Feedforward controller for heated tank*

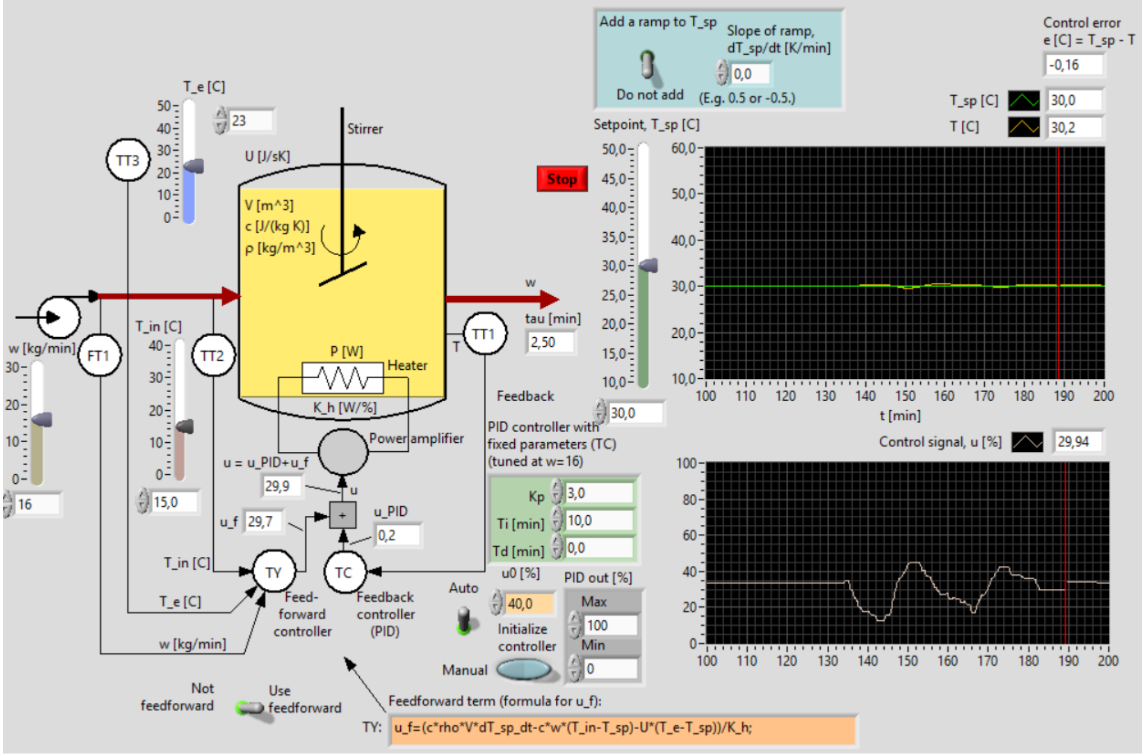
Figure 17.11 shows a heated liquid tank where the temperature T shall be controlled using feedback with PID controller in combination with feedforward control.

We assume the following process model, which is based on energy balance:

$$c\rho V T'(t) = \underbrace{K_h u(t)}_{P} + cw\left[T_{in}(t) - T(t)\right] + U\left[T_e(t) - T(t)\right] \tag{17.30}$$

where T [K] is the temperature of the liquid in the tank, T_{in} [K] is the inlet temperature, T_e [K] is environmental temperature, c [J/(kg K)] is specific heat capacity, w [kg/s] is mass flow (same in as out), V [m^3] is the liquid volume, ρ [kg/m^3] is the liquid density, U [(J/s)/K] is the total heat transfer coefficient, $P = K_h u$ [J/min] is supplied power via heating element where K_h is a parameter (gain) and u [%] is the control signal applied to the heating element. $c\rho VT$ is the (temperature dependent) energy of the liquid in the tank. We can consider T_{in} and T_e as disturbances, but the derivation of the feedforward function F_f is not dependent of such a classification.

Derive the feedforward function from the process model (17.30). The temperature reference is r_T [K]. Which parameters and variables must have known values to implement the feedforward control?



Slope of ramp, dT_sp/dt [K/min] 0,0 (E.g. 0.5 or -0.5.)

Control error e [C] = T_sp - T -0,16

T_sp [C] 30,0

T [C] 30,2

T_e [C] 23

Setpoint, T_sp [C] 30,0

tau [min] 2,50

w [kg/min] 16

T_in [C] 15,0

u = u_PID+u_f 29,9

u_f 29,7

u_PID 0,2

PID controller with fixed parameters (TC) (tuned at w=16)

Kp 3,0

Ti [min] 10,0

Td [min] 0,0

u0 [%] 40,0

PID out [%] Max 100 Min 0

Control signal, u [%] 29,94

Feedforward term (formula for u_f):

TY: u_f=(c*rho*V*dT_sp_dt-c*w*(T_in-T_sp)-U*(T_e-T_sp))/K_h;

Figure 17.11: Liquid tank with temperature control system

Problem 17.4 *Interpolation in feedforward controller*

Assume that you will implement the feedforward controller as a set of linear functions (lines) between the data points (fan, u_{ff}). The linear functions are on the form

$$u_{\text{ff}} = a \cdot \text{fan} + b \tag{17.31}$$

Let us select the points (40,52) and (60,61). Calculate a and b of the line between these two points.

17.7 *Solutions to problems for Chapter 17*

Solution to Problem 17.1

1. Solving the model for the control signal, u, and substituting y with y_{sp} gives the feedforward controller:
$$u_{\text{ff}} = [Tr' + r – K_2 d]/K_1 \tag{17.32}$$

2. All quantities on the right side of (17.32) must be known.

3. The feedforward controller is based on a mathematical model of the process, and on the measurements (or otherwise known values) of the disturbances which appears in the model. In practice, there will always be errors in the model (the model is not perfect) and inaccuracies in the measurements. Therefore, the practical feedforward controller will not calculate the perfect control signal, causing the control error to become different from zero. The feedback controller implements error-driven control, and will therefore reduce this error. In particular, assuming that the feedback controller has integral action, the feedback controller will ensure that the steady state control error is zero (despite model and measurement errors).

Solution to Problem 17.2

1. Firstly, we substitute y by its reference r. Secondly, we solve for the control variable F_p which we denote the feedforward control variable $F_{p,\text{ff}}$. The result is:

$$F_{p,\text{ff}} = mr'' - D_h \left(u_c - r'\right)\left|u_c - r'\right| - D_w \left(V_w - r'\right)\left|V_w - r'\right| \tag{17.33}$$

Requirements of implementing the feedforward controller: r must be known – no problem! Parameters m, D_h and D_w must be known, and we may assume they are known approximately by the ship designer. Water current u_c may be measured. However, in real DP applications (Dynamic Positioning) u_c is estimated using a state estimator algorithm named Kalman Filter. Wind speed W_w must be known. In real DP applications it is measured with wind sensors. (The sensor for measuring wind direction and speed is placed on the top of the ship.)

2. Python program 17.1 implements the simulator.

http://techteach.no/control/python/dynpos.py

Listing 17.1: dynpos.py

```
"""
Simulator of a dynamic positioning system
Selectable: (1) Feedback with PID. (2) Feedback + feedforward.
Author: Finn Aakre Haugen, finn@techteach.no
Updated 2023 09 02
"""

#%% Importing packages
```

```
import matplotlib.pyplot as plt
import numpy as np

#%% Model parameters

m = 71164*1000  # [kg]
Dh = 8.4*1000  # [N/(m/s)^2]
Dw = 0.177*1000  # [N/(m/s)^2]
Fp_max = 552000  # [N]
Fp_min = -467000  # [N]

#%% PID control params (using the Skogestad method)

Tc = 200  # Closed loop time constant [s]
Kii = 1/m  # Gain of double integrator process
Kc = 2/(Kii*Tc*Tc)  # Gain [N/m]
Ti = 4*Tc  # Integral time [s]
Td = 1*Tc  # Derivative gain [s]
u_man = 0  # Manual control signal [N]

#%% Simulation time settings

dt = 0.1  # [s]
t_start = 0  # [s]
t_stop = 5000  # [s]
N_sim = int((t_stop - t_start)/dt) + 1

#%% Preallocation of arrays for plotting

t_array = np.linspace(t_start, t_stop, N_sim)
k_array = np.arange(0, N_sim)
r_array = np.zeros(N_sim)
x1_array = np.zeros(N_sim)
x2_array = np.zeros(N_sim)
Fp_array = np.zeros(N_sim)
Vw_array = np.zeros(N_sim)
Fw_array = np.zeros(N_sim)
uff_array = np.zeros(N_sim)

#%% Initialization

x1_init = 0
x2_init = 0
x1_k = x1_init
x2_k = x2_init
e_km1 = 0
ui_km1 = 0
r_km1 = 0
x2_r_km1 = 0

#%% Simulation settings

K_ff = 1  # 0 = Only feedback. 1 = feedback + feedforward.
```

```
r0 = 40  # Constant pos ref [m]
A_r = 20  # Pos ref sine ampl [m]
P_r = 1000  # Pos ref sine period  [s]
Vw = -20  # Wind speed [m/s]

#%% Simulation loop

for k in k_array:

    t_k = dt*k

    # Excitations:
    if t_start <= t_k < 500:
        r_k = A_r*(1-np.cos((2*np.pi/P_r)*t_k))
        uc_k = 0
        Vw_k = 0
    if 500 <= t_k < 2500:
        r_k = r0
        uc_k = 0
        Vw_k = 0
    if 2500 <= t_k <= t_stop:
        r_k = r0
        uc_k = 0
        Vw_k = Vw

    # Calculation of references of speed and acceleration:
    x2_r_k = (r_k - r_km1)/dt
    dx2_r_k = (x2_r_k - x2_r_km1)/dt

    # Feedforward control signal:
    Fh_ff_k = Dh*(uc_k - x2_r_k)*np.abs(uc_k - x2_r_k)
    Fw_ff_k = Dw*(Vw_k - x2_r_k)*np.abs(Vw_k - x2_r_k)

    uff_k = m*dx2_r_k - Fh_ff_k - Fw_ff_k

    # PID controller + feedforward:
    e_k = r_k - x1_k
    up_k = Kc*e_k
    ui_k = ui_km1 + (Kc/Ti)*dt*e_k
    ud_k = Kc*Td*(e_k - e_km1)/dt
    u_k = u_man + up_k + ui_k + ud_k + K_ff*uff_k
    u_k = np.clip(u_k, Fp_min, Fp_max)

    # Forces:
    Fp_k = u_k
    Fh_k = Dh*(uc_k - x2_k)*np.abs(uc_k - x2_k)
    Fw_k = Dw*(Vw_k - x2_k)*np.abs(Vw_k - x2_k)

    # Arrays for plotting:
    t_array[k] = t_k
    r_array[k] = r_k
    x1_array[k] = x1_k
    x2_array[k] = x2_k
    Fp_array[k] = Fp_k
    Vw_array[k] = Vw_k
    Fw_array[k] = Fw_k
    uff_array[k] = uff_k

    # Time derivatives:
    dx1_k = x2_k
    dx2_k = (1/m)*(Fp_k + Fh_k + Fw_k)

    # State prediction (Euler step):
    x1_kp1 = x1_k + dx1_k*dt
    x2_kp1 = x2_k + dx2_k*dt

    # Time shift:
    x1_k = x1_kp1
    x2_k = x2_kp1
    ui_km1 = ui_k
    e_km1 = e_k
    r_km1 = r_k
    x2_r_km1 = x2_r_k

#%% Plotting

plt.close(’all’)  # Closes all figures before plotting
plt.figure(1)

plt.subplot(2, 1, 1)
plt.plot(t_array, r_array, ’r’, label=’r’)
plt.plot(t_array, x1_array, ’b’, label=’y’)
plt.legend()
plt.grid()
plt.ylabel(’[m]’)

plt.subplot(2, 1, 2)
plt.plot(t_array, Fp_array/1000, ’g’, label=’Fp’)
plt.plot(t_array, Fw_array/1000, ’m’, label=’Fw’)
plt.legend()
plt.grid()
plt.ylabel(’[kN]’)
plt.xlabel(’t [s]’)

#%% Saving the plot figure as pdf file

if K_ff == 0:
    plt.savefig(’plot_sim_dynpos_feedback.pdf’)
else:
    plt.savefig(’plot_sim_dynpos_feedforward.pdf’)

plt.show()
```

The PID settings are:

$$K_c = 3558 \tag{17.34}$$

$$T_i = 800 \text{ s} \tag{17.35}$$

$$T_d = 200 \text{ s} \tag{17.36}$$

Figure 1.23 shows a simulation with feedback PID control.

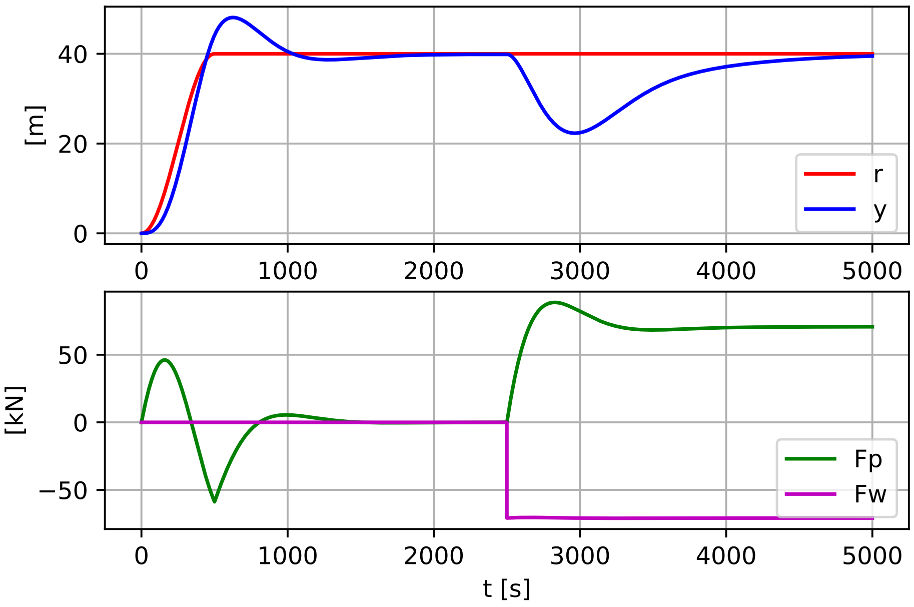


Figure 17.12: Simulation of position control of a ship (dynamic positioning).

Figure 17.13 shows a simulation of the ship with position control based on feedforward control combined with feedback control.

The feedback + feedforward control system behaves excellently:

- The position reference is tracked precisely (the position reference and the position measurement can hardly be distinguished).
- The wind disturbance (there is a wind gust at $t = 2500$ s) is compensated for effectively. Notice that the propeller force counteracts exactly the wind force, causing the ship to stay still despite the heavy wind gust. This demonstrates very well the behaviour of the feedforward controller.

Solution to Problem 17.3

We substitute the temperature T by the temperature reference r_T (the time argument t is omitted for simplicity, but it should not be omitted if the model contain time delay terms):

$$c\rho V r_T' = K_h u + cw\left(T_{in} - r_T\right) + U\left(T_e - r_T\right) \tag{17.37}$$

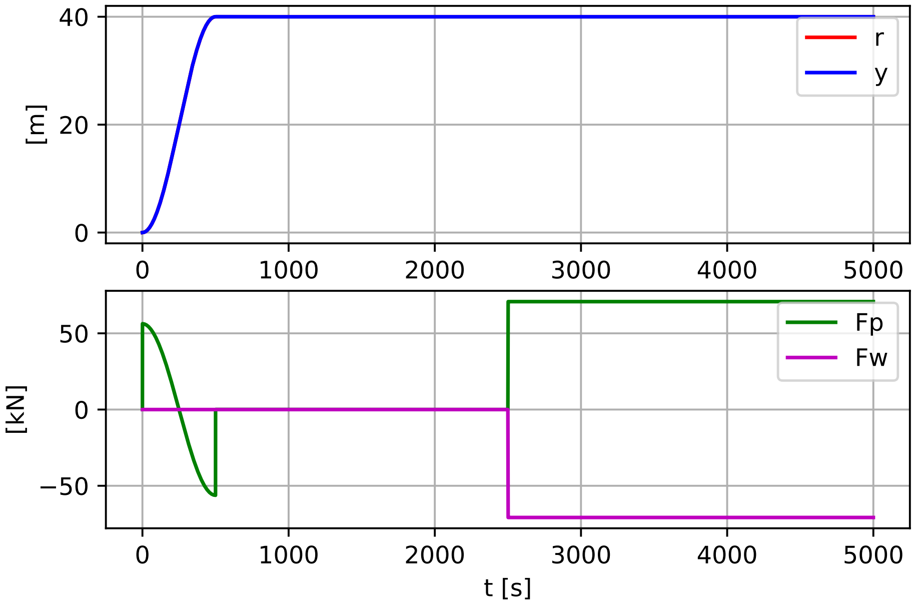


Figure 17.13: Simulation of the ship with position control based on feedforward control combined with feedback control.

We solve (17.37) for the control variable u to get the feedforward control variable u_f:

$$u_f = \frac{1}{K_h}\left[c\rho V r_T' - cw\left(T_{in} - r_T\right) - U\left(T_e - r_T\right)\right] \tag{17.38}$$

$$= \underbrace{\frac{1}{K_h}\left[c\rho V r_T' + cwr_T + Ur_T\right]}_{u_{fSP}} + \underbrace{\frac{1}{K_h}\left[-cwT_{in} - UT_e\right]}_{u_{fd}} \tag{17.39}$$

Implementation of feedforward control signal u_f requires measurement or knowledge of the following five quantities: c, ρ, V, h, w, K_h and T_{in}, in addition to the reference time-derivative, r_T'.

Solution to Problem 17.4

With the two data points given, we have

$$52 = a \cdot 40 + b \tag{17.40}$$

and

$$61 = a \cdot 60 + b \tag{17.41}$$

From these two equations we get (the mathematics to solve these two equations is not shown here)

$$a = 0.45 \tag{17.42}$$

$$b = 34 \tag{17.43}$$

Chapter 18

Sequential control

Sequential control is used to implement e.g. chemical batch process control and automated mechanical operations.

A sequential control procedure can be represented graphically by

- **Sequential function chart** (SFC), or
- **State diagram**.

SFCs have the following elements:

- **Steps** with a number of associated actions to be executed when the step is active. A step is either active or passive. One particular step is defined as the initial step. It can be symbolized with e.g. a double-lined box, while ordinary steps are represented with single-lined boxes.
- **Actions** are control actions made by the control device (typically a PLC or a PC), e.g. opening a valve, setting a PID controller into automatic mode, starting a motor, lighting a lamp on, etc. The action box indicates the changes of control actions from previous steps (control actions which are not changed, does not have to be listed in the action box).
- **Transitions** from one active step to another taking place when the transition condition is satisfied. Transition conditions are in the form of logical expressions – simple or complicated – having value either TRUE or FALSE. Here is one example: T_Step1_Step2: Level > 0.9 m.

  A transition may take place *unconditionally*, that is, it takes place automatically after the actions of the presently active state have been accomplished. The transition condition of an unconditional transition has permanent value TRUE, and it may be expressed for example as T_Step2_Step3: TRUE.

Figure 18.1 shows how these basic elements (step, action, transition) appear in a Sequential function chart. The := symbol is the assignment operator. The == symbol is the equality check operator.

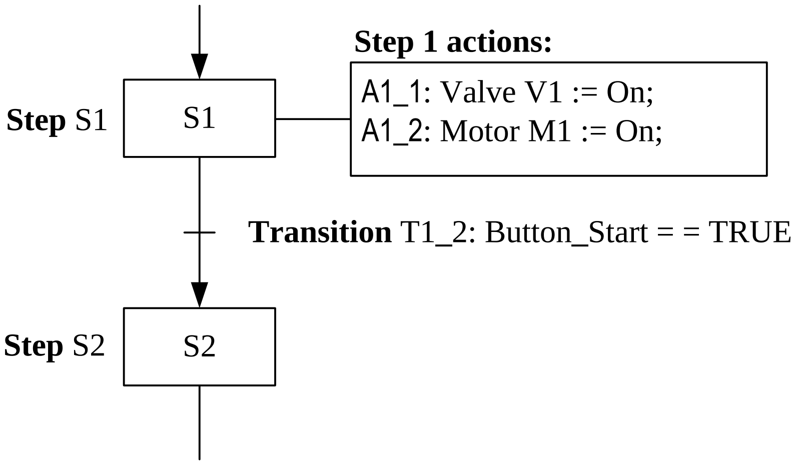


Figure 18.1: Elements of a Sequential Function Chart (SFC): Step, action, and transition

Sequential function charts may also contain *branches* defining simultaneous or alternative parts of the chart to be executed.

State diagrams can be drawn similar to Sequential function charts. In State diagram the term *state* is used instead of *step*. The control system represented with a state diagram is denoted a *state machine*.[1]

SFC is standardized in the IEC 61131-3 standard about PLC programming[2], and is available as a progamming tool in most PLC systems, e.g. Mitsubishi PLCs, Simatic PLCs, etc. State diagrams are supported in e.g. the Statchart Module of LabVIEW, the Stateflow Toolbox of Matlab/Simulink, and the Graph7 programming tool of Simatic PLCs.

Example 18.1 *Sequential control of a batch process*

Figure 18.2 shows a simple batch process which is to be controlled by sequential control.

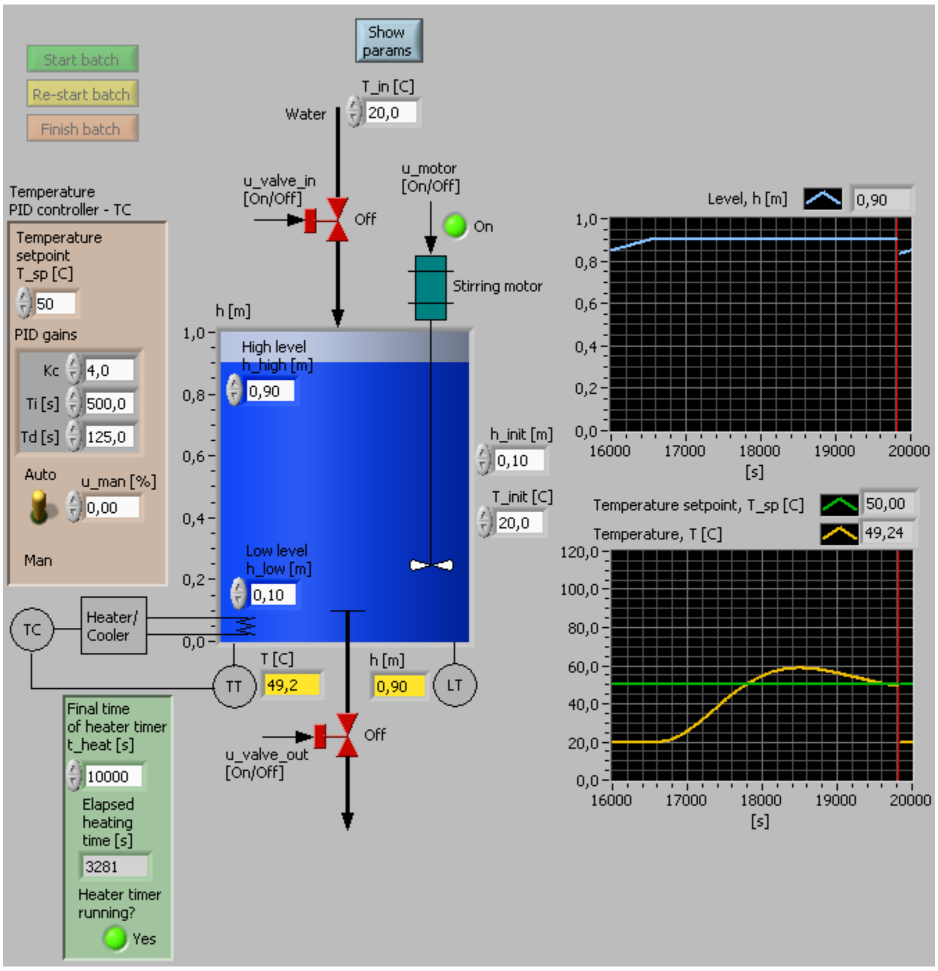
The simulations in this example are made with the following SimView simulator:

http://techteach.no/simview/sequential_control

The tank is filled with water up to a certain level. The water is then heated using temperature control (with a PID controller) for a specific time defined in timer, and

[1] The state machine is a very useful concept in programming, independently of the programming tool you are using. You can structure a programming task that contains several alternative paths dependent on certain conditions with a state diagram. After the structure has been made, the implementation with programming code is more or less straightforward.

[2] PLC = Programmable Logic Controller



T_in [C] 20,0

u_valve_in [On/Off] Off

u_motor [On/Off] On

Temperature setpoint T_sp [C] 50

Kc 4,0

Ti [s] 500,0

Td [s] 125,0

u_man [%] 0,00

High level h_high [m] 0,90

Low level h_low [m] 0,10

h_init [m] 0,10

T_init [C] 20,0

T [C] 49,2

h [m] 0,90

u_valve_out [On/Off] Off

Final time of heater timer t_heat [s] 10000

Elapsed heating time [s] 3281

Heater timer running? Yes

Level, h [m] 0,90

Temperature setpoint, T_sp [C] 50,00

Temperature, T [C] 49,24

Figure 18.2: A batch process to be controlled by sequential control

stirred[3]. Finally the heated water is discharged from the tank. Then the batch can be restarted, or ended.

The *control signals* are

- u_valve_in (boolean, i.e. having value TRUE or FALSE, or ON or OFF)
- u_valve_out (boolean)
- u_motor (boolean)
- u_heat (continuous having any value between 0% and 100%)

The *measurements* are

- Temperature T of the water in the tank

[3] The motor is assumed to ensure homogeneous thermal conditions in the water in the tank.

- Level h of the water in the tank

Figure 18.3 shows a Sequential Function Chart defining the control function.

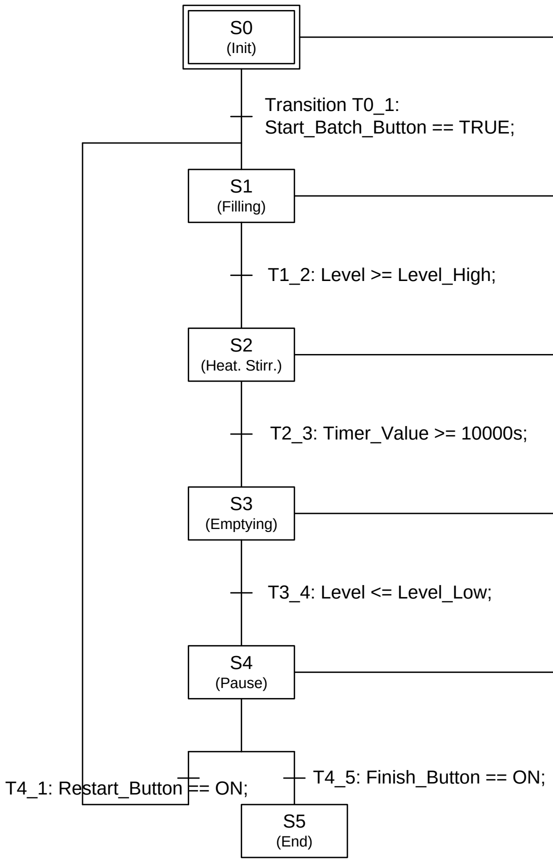


Figure 18.3: Sequential Function Chart (SFC) defining the control function

[End of Example 18.1]

18.1 *Problems for Chapter 18*

Problem 18.1 *Sequential control of drilling machine*

Figure 18.4 shows a simple drilling machine.

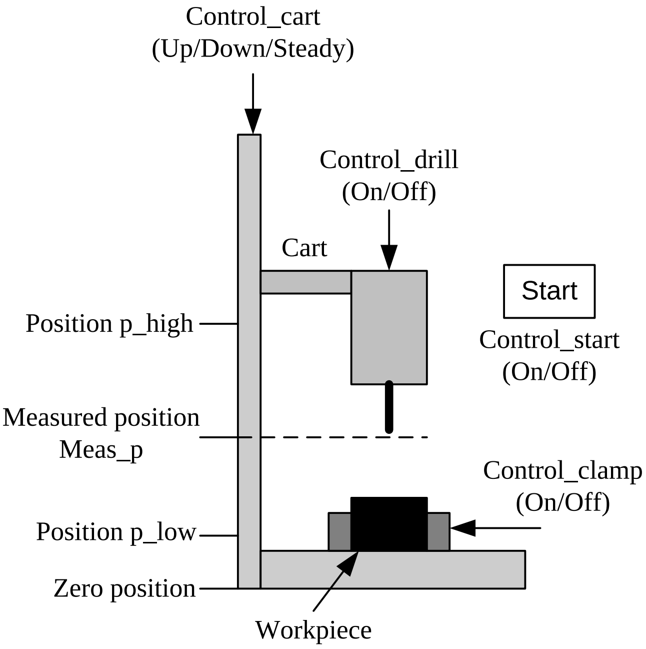


Figure 18.4: Drilling machine.

The machine operates as follows: The drilling operation is started with the Start button which sets the control signal Control_start to value On. Just after the button has been pressed, it pops up automatically and Control_start is automatically set back to Off (this reset is not a part of the control task in this problem). When the drilling operation has been started, the clamps are activated by setting the control signal Control_clamp to On, the drill starts rotating with Control_drill set to On, and the cart is moved downwards with Control_cart set to Down until the measured drill position Meas_p becomes p_low. Then, the cart is automatically moved upwards with Control_cart set to Up. When the Meas_p has become p_high, the cart is stopped with Control_cart set to Steady, the clamp is released with Control_clamp set to Off, and the drill is stopped with Control_drill set to Off. Then the drill is idle, waiting until the Start button is again pressed.

Placing the workpiece in the correct position is not a part of this control task.

Draw a Sequential Function Chart (SFC) with steps, actions and transitions solving the control task given above.

18.2 *Solutions to problems for Chapter 18*

Solution to Problem 18.1

Figure 18.5 shows the Sequential Function Chart (SFC) solving the given control task.

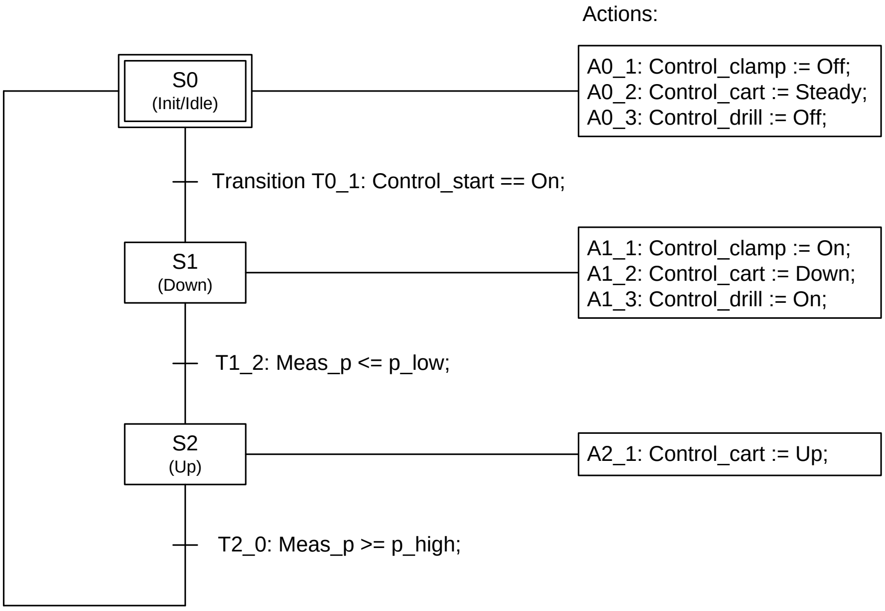

Actions:
S0
(Init/Idle)
A0_1: Control_clamp := Off;
A0_2: Control_cart := Steady;
A0_3: Control_drill := Off;
Transition T0_1: Control_start == On;
S1
(Down)
A1_1: Control_clamp := On;
A1_2: Control_cart := Down;
A1_3: Control_drill := On;
T1_2: Meas_p <= p_low;
S2
(Up)
A2_1: Control_cart := Up;
T2_0: Meas_p >= p_high;

Figure 18.5: Sequential Function Chart

Part V

ANALYSIS OF CONTINUOUS-TIME FEEDBACK SYSTEMS

Chapter 19

Stability analysis using poles

19.1 Introduction

In some situations we want to determine if a dynamic system is stable or unstable. Particularly in the control theory, stability analysis is important, since feedback control systems may become unstable if the controller parameters have been given erroneous values. In this chapter, stability properties will be defined in terms of the placement of the poles or eigenvalues of the system in the complex plane. Then, in Chapter 20, we will use this stability analysis method to analyse the stability of feedback systems.

19.2 Stability properties and impulse response

This section defines the different stability properties that a dynamic system can have in terms of impulse response of the system. Then, the corresponding transfer function pole locations in the complex plane are derived. In Section 20, and subsequent sections, these results are applied to feedback (control) systems, which are a special case of dynamic systems.The different stability properties can be defined in several ways. I have chosen to use the *impulse response* of the system as the basis for definition of the stability properties. The impulse response is the time-response in output variable of the system due to an impulse on the input. Using the impulse response makes it relatively simple to relate stability properties to the *poles* of the system (this is because the impulse response and the poles are closely connected), as you will see soon.

Some words about the impulse signal: It is a time-signal which in principle has infinitely short duration and infinite amplitude, but so that the integral of the signal – the integral is equal to the area under the time-function of the signal – is finite. This area is also called the *strength* of the impulse. An impulse of strength one is called a unit impulse, $\delta(t)$. The square pulse in Figure 19.1 approaches an impulse function as Δ goes to zero.

Below are the stability definitions based on the impulse response of the system. The

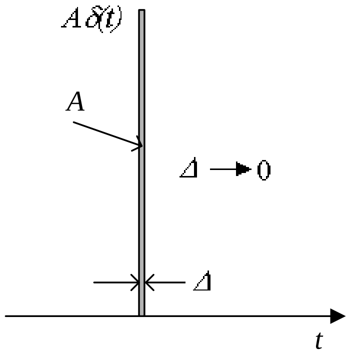


Figure 19.1: The square pulse approaches an impulse function as Δ goes to zero.

following notation is used:

$$h(\infty) \triangleq \lim_{t\to\infty} h(t) \tag{19.1}$$

- **Asymptotically stable system**: The steady state impulse response is zero:

$$h(\infty) = 0 \tag{19.2}$$

- **Marginally stable system**: The steady state impulse response is different from zero, but limited:

$$0 < |h(\infty)| < \infty \tag{19.3}$$

- **Unstable system**: The steady state impulse response is unlimited:

$$|h(\infty)| = \pm\infty \tag{19.4}$$

A system is *stable* if it is either asymptotically stable or marginally stable. Figure 19.2 illustrates the different stability properties in terms of the impulse response of the system.

One problem with the ideal impulse function is it can not be generated fully in practice, but in practice there is hardly ever a need to perform impulse response experiments to determine stability properties. It is more useful as a conceptual definition of stability, as explained in the following section.

19.3 Stability properties and poles

In many cases it would be quite impractical if the only way to determine the stability property of a system was to do experiments or to run simulations to obtain the impulse response, from which we could conclude about the stability property. Fortunately, we can find the impulse response, and therefrom conclude about the stability property, by just analyzing the mathematical model of the system.

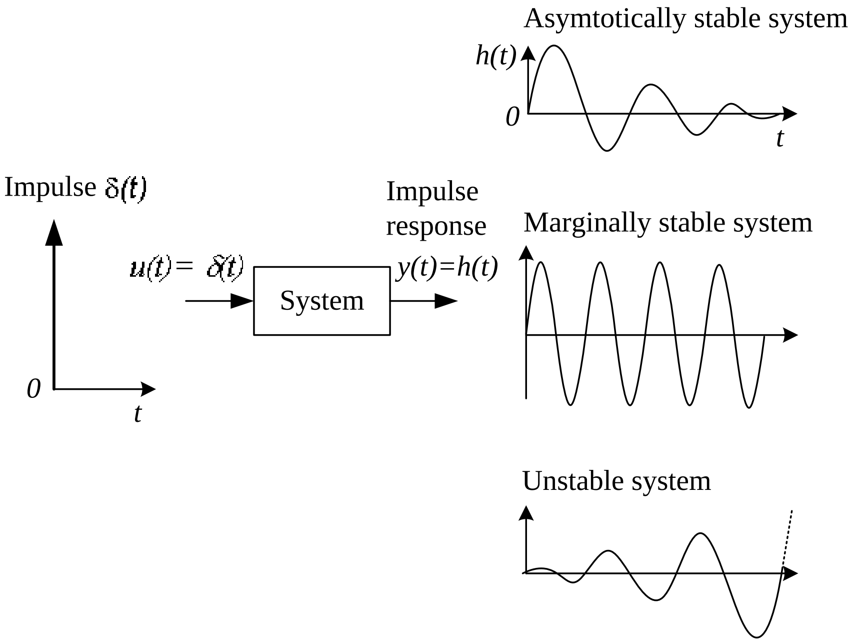


Figure 19.2: Different stability properties in terms of the impulse response.

Let us assume that the mathematical model of the system is a transfer function, $H_{Y,U}(s)$, from input signal u to output signal y:

$$Y(s) = H_{Y,U}(s)U(s) \tag{19.5}$$

The input u will be a unit impulse, that is, $u(t) = \delta(t)$. It can be shown that the Laplace transform of $\delta(t)$ is 1, i.e.

$$\mathcal{L}\left\{\delta(t)\right\} = 1 \tag{19.6}$$

Let us denote the impulse response by $h(t)$. The Laplace transform of $h(t)$ is

$$H(s) = \mathcal{L}\left\{h(t)\right\} = H_{Y,U}(s)\cdot\mathcal{L}\left\{\delta(t)\right\} = H_{Y,U}(s)\cdot 1 = H_{Y,U}(s) \tag{19.7}$$

Thus, *the Laplace transform of the impulse response equals the transfer function of the system.*[1]

I have found it convenient to simplify the notation: In the presentation that follows, I use symbol $H(s)$ instead of $H_{Y,U}(s)$ for the transfer function. You can then regard $H(s)$ either as the transfer function of the system *or* as the Laplace transform of the impulse response of the system – it is the same $H(s)$ in any case.

To analyze the stability property of the system we need to know the impulse response $h(t)$. It can be found with inverse Laplace transform:

$$h(t) = \mathcal{L}^{-1}\left\{H(s)\right\} \tag{19.8}$$

[1] I guess this is why $h(t)$ is the common symbol of the impulse response of a system having transfer function $H(s)$.

Although we can proceed with a rigorous analysis here, I want instead to make a simplified analysis, and give comments about how the results that we see, can be generalized.

Let's assume that the transfer function of the system is

$$H(s) = \frac{1}{s-p} \tag{19.9}$$

Note that p is the *pole* of the system.[2] Now, what is $h(t)$ when $H(s)$ is (19.9)? From the Laplace transform pair (44.10) in Appendix 44, we get

$$h(t) = \mathcal{L}^{-1}\left\{H(s)\right\} = \mathcal{L}^{-1}\left\{\frac{1}{s-p}\right\} = e^{-pt} \tag{19.10}$$

Let's now assume a number of specific values of p, namely $p = -1$, 0, and 1. (We could have chosen other values, but the given values will do.)

Pole $p = -1$

The transfer function (19.9) is

$$H(s) = \frac{1}{s+1} \tag{19.11}$$

The impulse response (19.10) is

$$h(t) = e^{-t} \tag{19.12}$$

Figure 19.3 shows the pole p and the impulse response $h(t)$.

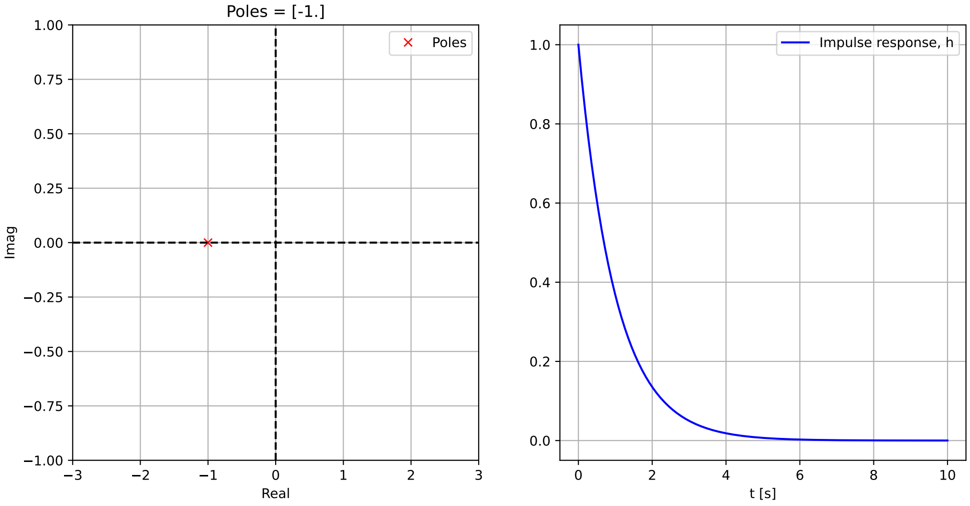


Figure 19.3: Pole p and the impulse response $h(t)$ when $p = -1$.

The diagrams in Figure 19.3 are generated with the following Python program:

[2] Recall, a pole is a value of s which is a root of the characteristic polynomial (i.e. the denominator of the transfer function).

http://techteach.no/control/python/stab_analysis_impulse_response_and_poles.py

Since $h(\infty) = 0$, the system is asymptotically stable. The system has pole $p = -1$, which is in the left half plane (LHP). We can conclude as follows: *For systems with one pole, the system is asymptotically stable if the pole is in LHP.*

Above we assumed $p = -1$. But any value of p in the left half plane, for example $p = -1.23$, would have given the same conclusion – an asymptotically stable system.

Pole $p = 0$

The transfer function (19.9) is

$$H(s) = \frac{1}{s+0} = \frac{1}{s} \tag{19.13}$$

The impulse response (19.10) is

$$h(t) = e^{-0 \cdot t} = 1 \tag{19.14}$$

Figure 19.4 shows the pole and the impulse response $h(t)$.

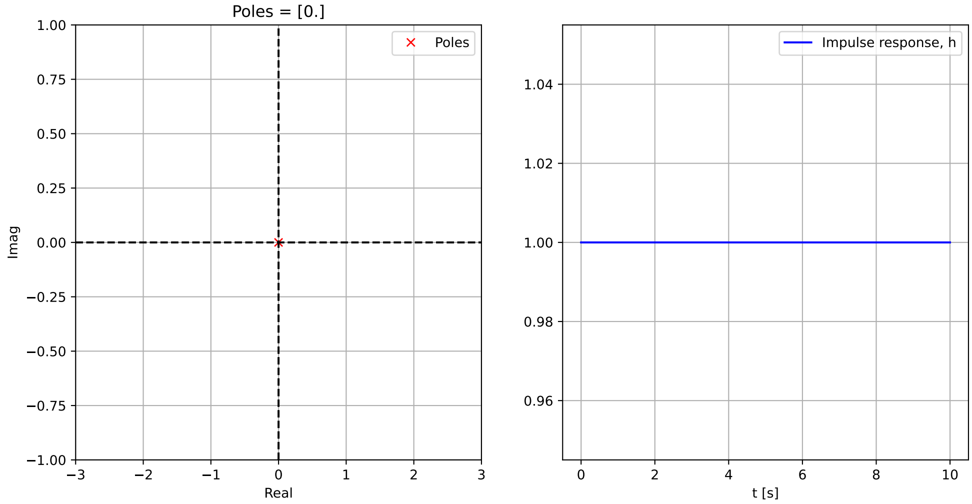


Figure 19.4: Pole p and the impulse response $h(t)$ when $p = 0$.

Since $h(\infty)$ has a limited value (namely 1 in this example), the system is marginally stable.

The system has pole $p = 0$, which is on the imaginary axis. We can conclude as follows: *For systems with one pole, the system is marginally stable if the pole is on the imaginary axis.*

Pole $p = 1$

The transfer function (19.9) is

$$H(s) = \frac{1}{s-1} \tag{19.15}$$

The impulse response (19.10) is

$$h(t) = e^t \tag{19.16}$$

Figure 19.5 shows the pole p and the impulse response $h(t)$ (with a limited time axis, of course). But actually, $h(\infty) = \infty$.

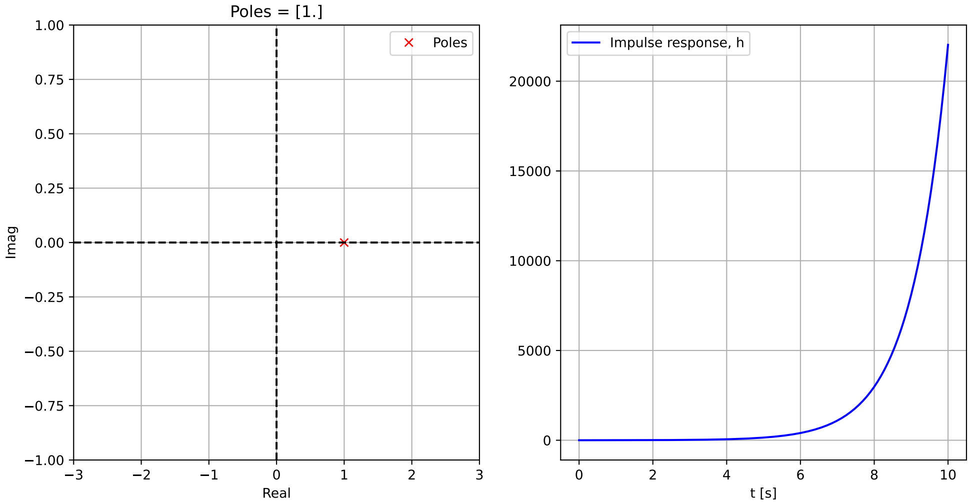


Figure 19.5: Pole p and the impulse response $h(t)$ when $p = 1$.

Since $h(\infty) = 0$, the system is unstable. The system has pole $p = 1$, which is in the right half plane (RHP). We can conclude as follows: *For systems with one pole, the system is unstable if the pole is in the RHP.*

Now, let's look at some systems having more than only one pole.

Poles $p_1 = -1$ and $p_2 = -2$

Let's assume that the transfer function (19.9) is

$$H(s) = \frac{1}{(s+1)(s+2)} \tag{19.17}$$

which has the poles

$$p_1 = -1 \text{ and } p_2 = -3 \tag{19.18}$$

which are both in the LHP.

To calculate the impulse response $h(t)$, we can start by writing the partial fraction decomposition of (19.17):

$$H(s) = \frac{1}{(s+1)(s+2)} = \frac{1}{s+1} - \frac{1}{s+2} \tag{19.19}$$

Due to the linear property of the Laplace transform, the impulse response becomes

$$h(t) = e^{-t} - e^{-2t} \tag{19.20}$$

Each of the two exponentials in (19.20) goes towards 0 as $t \to \infty$, and therefore $h(\infty) = 0$, and the system is asymptotically stable.

The conclusion of this example is: *For systems with two poles, the system is asymptotically if both poles are in the LHP.*

Poles $p_1 = -1$ and $p_2 = 2$

Let's assume that the transfer function (19.9) is

$$H(s) = \frac{1}{(s+1)(s-2)} \tag{19.21}$$

which has the poles

$$p_1 = -1 \text{ and} p_2 = 2 \tag{19.22}$$

Pole p_1 is in LHP, while pole p_2 is in RHP.

To calculate the impulse response $h(t)$, we can start by writing the partial fraction decomposition of (19.21):

$$H(s) = \frac{1}{(s+1)(s-2)} = -\frac{1/3}{s+1} + \frac{1/3}{s-2} \tag{19.23}$$

Due to the linear property of the Laplace transform, the impulse response becomes

$$h(t) = -\frac{1}{3}e^{-t} + \frac{1}{3}e^{2t} \tag{19.24}$$

The first exponential in (19.24) goes towards 0 as $t \to \infty$, while the second exponential goes to ∞. Therefore $h(\infty) = \infty$, and the system is unstable. Of course, the system would have been unstable also if both poles where in the RHP.

The conclusion of this example is: *For systems with two poles, the system is asymptotically if at least one of the poles is in the RHP.*

Complex poles

So far, the poles have been only real, i.e. the imaginary parts are zero. Let's now assume that the system has a number of complex poles with non-zero imaginary part. From mathematics we know that complex roots appear in complex conjugate pairs. Say such a pole pair is

$$p = a \pm jb \tag{19.25}$$

where a is the real part and $\pm b$ is the imaginary part. It can be shown that such a pole pair will create an additive part of the total impulse response of the following form:

$$h_i(t) = Ke^{at}\sin(bt + \phi) \tag{19.26}$$

where K and ϕ are some constants. The sine term has value between 1 and -1. Therefore, it is the exponential term e^{at} that determines the steady state ($t \to \infty$) value of $h_i(t)$:

- If $a < 0$, i.e. the poles are in LHP, $e^{at} \to 0$ as $t \to \infty$, and therefore $h_i(t) \to 0$ as $t \to \infty$.
- If $a = 0$, i.e. the poles are on the imaginary axis, $e^{at} = 1$, and therefore $0 < h_i(\infty) < \infty$.
- If $a > 0$, i.e. the poles are in RHP, $e^{at} \to \infty$ as $t \to \infty$, and therefore $|h_i(\infty)| = \infty$.

Conclusion about stability analysis based on pole placement

From the results earlier in this section, we can conclude the stability analysis as follows:

Stability analysis based on poles

- *Asymptotically stable system:* Each of the poles of the transfer function lies in the LHP.
- *Marginally stable system:* One or more poles lies on the imaginary axis. These poles must be distinct; Otherwise the system is unstable, see the item below.
- *Unstable system:* At least one pole lies in the RHP. Also systems with multiple poles on the imaginary axis are unstable, e.g. the transfer function $H(s) = 1/s^2$, which is a double integrator.

Figure 19.6 gives a illustration of the relation between stability property and pole placement.

Example 19.1 *Stability property of a mass-spring-damper*

Figure 19.7 shows a mass-spring-damper system.

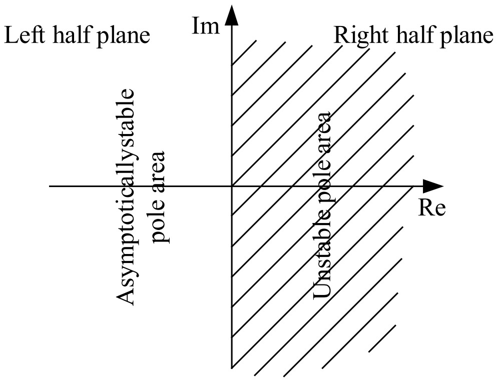


Figure 19.6: The relation between stability property and pole placement.

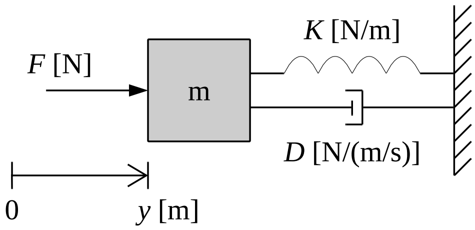


Figure 19.7: Mass-spring-damper.

y is position. F is applied force. D is damping constant. K is spring constant. Newton's 2. Law gives the following mathematical model:

$$my'' = F - Dy' - Ky \tag{19.27}$$

Taking the Laplace transform of (19.27)gives

$$ms^2Y(s) = F^*(s) - DsY(s) - KY(s) \tag{19.28}$$

The transfer function from the force $F^*(s)$ to position $Y(s)$ is

$$\frac{Y(s)}{F^*(s)} = H(s) = \frac{1}{ms^2 + Ds + K} \tag{19.29}$$

Assume that $m = 20$ kg, $D = 4$ N/(m/s), and $K = 2$ N/m. What is the stability property of the system? The characteristic polynomial becomes

$$a(s) = ms^2 + Ds + K = 20s^2 + 4s + 2 \tag{19.30}$$

which has roots

$$p_1,\ p_1 = \frac{-4 \pm \sqrt{4^2 - 4 \cdot 20 \cdot 2}}{2 \cdot 20} = -0.1 \pm j0.3 \tag{19.31}$$

which are the poles of $H(s)$. Both these poles have strictly negative real parts (-0.1). The system is therefore asymptotically stable.

The following Python program calculates and plot the poles and simulates the impulse response:

http://techteach.no/control/python/impulse_resp_and_poles_mfd.py

Figure 19.8 shows the poles and the impulse response.

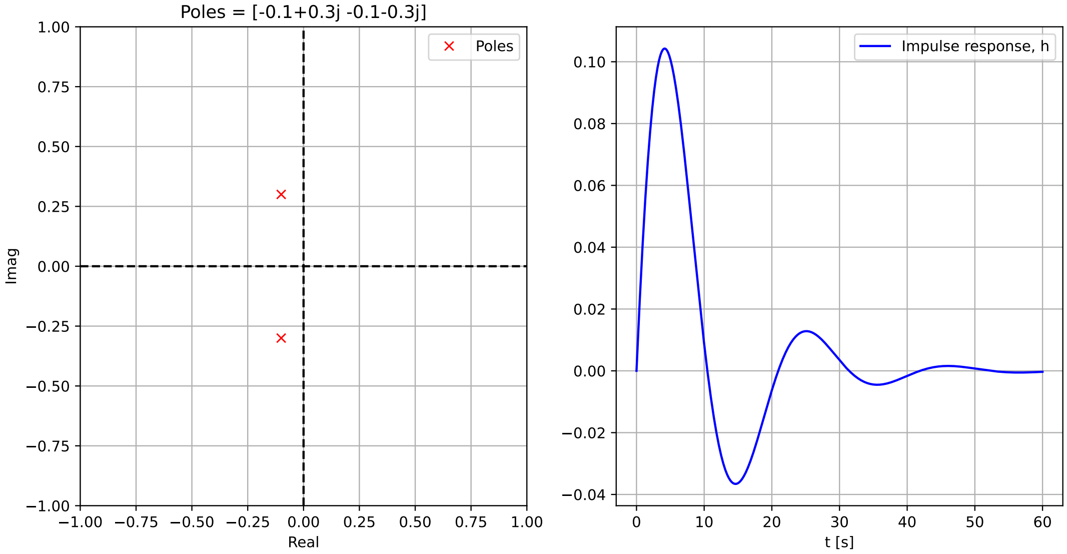


Figure 19.8: Example 19.1: The poles and the impulse response of the mass-spring-damper system.

[End of Example 19.1]

19.4 Stability analysis of state space models

In Section 19.3, the different stability properties of a transfer function were related to the poles of the transfer function. If the model originally is given as a state space model,

$$x' = Ax + Bu \tag{19.32}$$

$$y = Cx + Du \tag{19.33}$$

we can determine the stability properties of that system by finding the corresponding transfer function from u to y, and calculating the poles of the transfer function. Then, these poles determines the stability of the state space system!

What is this transfer function, then? In Section 8.10, we found that it is

$$H(s) = \frac{Y(s)}{U(s)} = C(sI - A)^{-1}B + D \equiv C\frac{\text{adj}(sI - A)}{\det(sI - A)}B + D \tag{19.34}$$

The poles of $H(s)$ are the roots of the denominator:

$$a(s) = \det(sI - A) = 0 \tag{19.35}$$

Thus, we can conclude about the stability property of the state space model from the roots of (19.35).

But (19.35) also defines the *eigenvalues* of the system matrix A of the state space model. [3] Therefore, the poles of the transfer function are the same as the eigenvalues of A. Consequently, we can *conclude about the stability of the state space model from the eigenvalues* – you can just substitute "pole" by "eigenvalue" in the criteria for asymptotic stability, marginal stability and instability in Section 19.3.

Note: In some state space models factors of type $(s - p_i)$ in the denominator can be cancelled against factors $(s - z_i)$ in the numerator of the transfer function. Such pole/zero-cancellations implies that some of the poles (and zeros) "disappears" from the transfer function. Consequently, the set of poles will then be just a subset of the set of eigenvalues. Thus, there may exist eigenvalues which are not poles, so that stability analysis based on eigenvalues placement (in the complex plane) may give a different result than stability analysis based on pole placement.

Example 19.2 *Stability property of a mass-spring-damper*

See Example 19.1. A state space model of the system is, cf. Example 6.4,

$$\underbrace{\begin{bmatrix} {x_1}' \\ {x_2}' \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\frac{K}{m} & -\frac{D}{m} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}}_{B} u \tag{19.36}$$

$$y = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \end{bmatrix}}_{D_1} u \tag{19.37}$$

The Python program below calculates the eigenvalues of the state space model.

http://techteach.no/control/python/eigenvalues_ss_model.py

The eigenvalues are:

$$\lambda_1,\ \lambda_1 = -0.1 \pm 0.3j \tag{19.38}$$

which are the same as the poles, cf. Example 19.1.

[End of Example 19.2]

[3] In mathematics literature it is more common to use the symbol λ instead of s for eigenvalues.

19.5 *Problems for Chapter 19*

Problem 19.1 *Stability property of a transfer function*

Determine the stability property of the transfer function (19.39) by calculating its pole and also by studying its impulse response, $h(t)$ (make a rough sketch of it).

$$\frac{Y(s)}{U(s)} = H(s) = \frac{1}{s+1} \tag{19.39}$$

To calculate $h(t)$, you can use the following Laplace transform:

$$\frac{k}{t_c s + 1} \iff \frac{k e^{-t/t_c}}{t_c} \tag{19.40}$$

Problem 19.2 *Stability properties of several transfer functions*

Determine the stability property of the following transfer functions:

$$H_1(s) = \frac{1}{s+1} \tag{19.41}$$

$$H_2(s) = \frac{1-s}{1+s} \tag{19.42}$$

$$H_3(s) = \frac{1}{1-s} \tag{19.43}$$

$$H_4(s) = \frac{1}{(s+1)(s-1)} \tag{19.44}$$

$$H_5(s) = \frac{1}{s} \tag{19.45}$$

$$H_6(s) = \frac{1}{s^3} \tag{19.46}$$

$$H_7(s) = \frac{e^{-s}}{s+1} \tag{19.47}$$

$$H_8(s) = -\frac{1}{s+1} \tag{19.48}$$

$$H_9(s) = \frac{1}{s^2+s+1} \tag{19.49}$$

$$H_{10}(s) = \frac{1}{s^2+1} \tag{19.50}$$

$$H_{11}(s) = \frac{1}{(s+1)s} \tag{19.51}$$

Problem 19.3 *Stability property of state space model*

Determine the stability property of the following state space model:

$$\begin{bmatrix} {x_1}' \\ {x_2}' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{19.52}$$

19.6 *Solutions to problems for Chapter 19*

Solution to Problem 19.1

The pole of $H(s)$ is the root of

$$(s+1) = 0 \tag{19.53}$$

The pole is

$$p = -1$$

which is in the left half plane. Therefore, the system is asymtptotically stable.

The Laplace transform of the impulse response is

$$h(t) = Y(s) = H(s)\underbrace{U(s)}_{=1} = H(s) = \frac{1}{s+1} \tag{19.54}$$

Using (44.10) with $k = 1$ and $T = 1$ we get

$$h(t) = \frac{ke^{-t/t_c}}{t_c} = e^{-t} \tag{19.55}$$

Figure 19.9 shows $h(t)$.

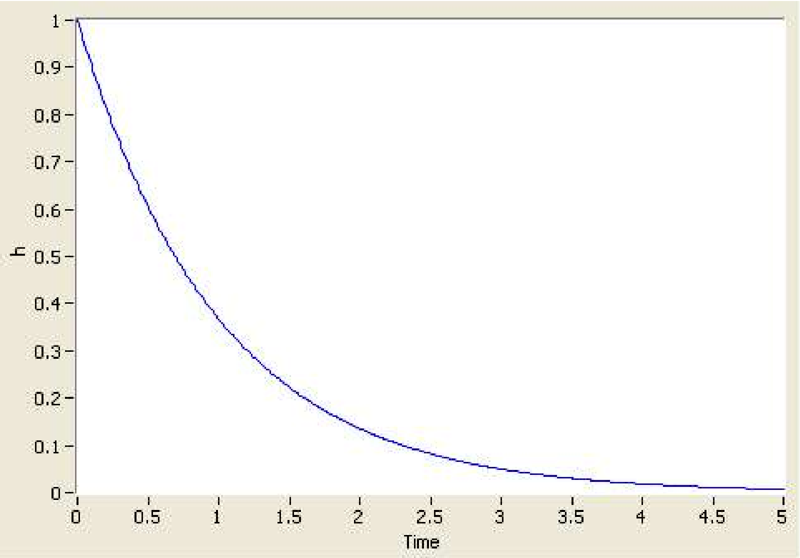


Figure 19.9: Impulse response.

Since $h(t)$ goes to zero as time goes to infinity, the system is $\underline{\underline{\text{asymtptotically stable}}}$.

Solution to Problem 19.2

The transfer function

$$H_1(s) = \frac{1}{s+1} \tag{19.56}$$

is asymptotically stable since the pole $p = -1$ is in the left half plane.

The transfer function

$$H_2(s) = \frac{1-s}{1+s} \tag{19.57}$$

is asymptotically stable since the pole $p = -1$ is in the left half plane. The value of the zero, which is 1, does not determine the stability.

The transfer function

$$H_3(s) = \frac{1}{1-s} \tag{19.58}$$

is unstable since the pole $p = 1$ is in the right half plane.

The transfer function

$$H_4(s) = \frac{1}{(s+1)(s-1)} \tag{19.59}$$

is unstable since one of the poles, $p_1 = 1$, is in the right half plane.

The transfer function

$$H_5(s) = \frac{1}{s} \tag{19.60}$$

is marginally stable since the pole $p = 0$ is in origin, which is on the imaginary axis, and this pole is single (there are no multiple poles).

The transfer function

$$H_6(s) = \frac{1}{s^3} \tag{19.61}$$

is unstable since there are multiple poles, $p_{1,2,3} = 0$, on the imaginary axis.

The transfer function

$$H_7(s) = \frac{e^{-s}}{s+1} \tag{19.62}$$

is asymptotically stable since the pole $p = -1$ is in the left half plane.

The transfer function

$$H_8(s) = -\frac{1}{s+1} \tag{19.63}$$

is asymptotically stable since the pole $p = -1$ is in the left half plane.

The transfer function

$$H_9(s) = \frac{1}{s^2+s+1} \tag{19.64}$$

is asymptotically stable since both the poles

$$p_{1,2} = \frac{-1 \pm \sqrt{1^2 - 4 \cdot 1 \cdot 1}}{4 \cdot 1} = \frac{-1 \pm j\sqrt{3}}{4} \tag{19.65}$$

are in the left half plane.

The transfer function

$$H_{10}(s) = \frac{1}{s^2+1} \tag{19.66}$$

is marginally stable since the poles

$$p_{1,2} = \pm j \tag{19.67}$$

are on the imaginary axis and they are single (not multiple).

The transfer function

$$H_{11}(s) = \frac{1}{(s+1)s} \tag{19.68}$$

has poles

$$p_{1,2} = 0, -1 \tag{19.69}$$

One pole is on the imaginary axis, and the other is in the left half plane. The system is marginally stable.

Solution to Problem 19.3

The stability property is determined by the system eigenvalues, which are the roots of the characteristic equation:

$$\begin{aligned}
\det(sI - A) &= \det\left(s \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix} \right) && (19.70)\\
&= \det\left(\begin{bmatrix} s & -1 \\ 0 & s+2 \end{bmatrix} \right) && (19.71)\\
&= s\,(s+2) + 0 && (19.72)
\end{aligned}$$

The roots are

$$s_{1,2} = 0, \quad -2 \tag{19.73}$$

One pole is in the origin, and the other pole is in the left half plane. Thererfore, the system is marginally stable.

Chapter 20

Stability analysis of feedback systems using poles

20.1 Introduction

In Chapter 19 we analysed the stability of a dynamic system using the poles of the transfer function of the system, or using the eigenvalues of the state space model of the system. In the present chapter we will use this stability analysis method to analyse the stability of feedback systems, with particular focus on feedback *control* systems.

Traditionally, the Routh's stability criterion has been taught as a tool for stability analysis. Routh's criterion is a method for determining the stability properties directly from the parameters of a transfer function model of the system, *without* calculating the poles or eigenvalues of the system. Although this sounds great, I think it is often sufficient to know how to determine the stability from the calculated poles or eigenvalues. Therefore, I have decided not to include that method in this book.

There exists a graphical method — the Nyquist's stability criterion — for stability analysis of feedback (control) systems based on the frequency response. This method is presented in Chapter 22.3.

Note: If the feedback system is a continuous-time model containing a time delay, you can not analyse the stability of the system using poles (or eigenvalues). In such cases, you may proceed with the analysis in alternative ways:

- Replace the irrational transfer function of the time delay, i.e. $e^{-t_d s}$, with some rational transfer function, typically a Padé approximation, cf. Section 9.5.1.
- Do the analysis using a discrete time model corresponding to the continuous time model. The time delay can be represented with a z transfer function as explained in Section 41.5.1.4. The part of the continuous time model not containing the time delay can be discretized as explained in Section 27.7.

20.2 Stability analysis of feedback systems

Figure 20.1 shows a general transfer function-based block diagram of a feedback control system. The stability of the feedback system can be determined from the poles of the transfer function from reference $R(s)$ to process measurement $Y(s)$. This transfer function is the tracking transfer function, $H_t(t)$, defined in Section 12.4. (Actually, we can determine the stability from any input to the control system to any output of the control system – the conclusion about stability will be the same.)

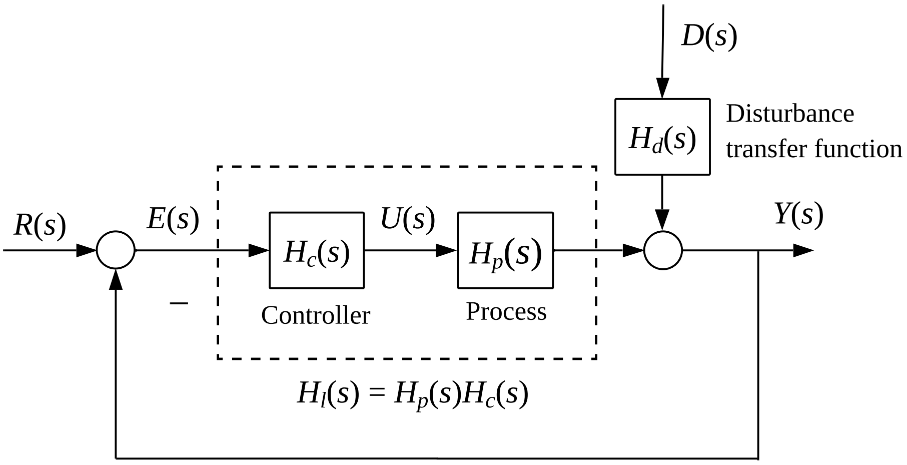


Figure 20.1: Transfer function-based block diagram of a feedback control system.

$H_t(s)$ is given by (12.27), but repeated here for convience:

$$H_t(s) = \frac{H_l(s)}{1 + H_l(s)} \tag{20.1}$$

The stability of the feedback control system is determined by the poles of $H_t(s)$.

These poles are the roots of the characteristic polynomial of $H_t(s)$. We will now derive that polynomal. First, we write $H_l(s)$ in terms of its numerator and denominator polynomials:

$$H_t(s) = \frac{n_l(s)}{d_l(s)} \tag{20.2}$$

We can then write $H_t(s)$ as:

$$H_t(s) = \frac{H_l(s)}{1 + H_l(s)} = \frac{\frac{n_l(s)}{d_l(s)}}{1 + \frac{n_l(s)}{d_l(s)}} = \frac{n_l(s)}{d_l(s) + n_l(s)} \tag{20.3}$$

So, the characteristic polynomial of the tracking transfer function is:

$$a(s) = d_l(s) + n_l(s) \tag{20.4}$$

$K_c = 1$ (asympt. stable)	$K_c = 2$ (marg. stable)	$K_c = 4$ (unstable)
$p_1 = -1.75$	$p_1 = -2$	$p_1 = -2.31$
$p_2 = -0.12 + j0.74$	$p_2 = j$	$p_2 = 0.16 + j1.31$
$p_3 = -0.12 - j0.74$	$p_3 = -j$	$p_3 = 0.16 - j1.31$

Table 20.1: Poles of the tracking transfer function for various values of K_c.

Thus, the stability of the feedback control system is determined by the roots of characteristic polynomial (20.4).

Example 20.1 *Stability analysis of a feedback control system*

Assume the following specific transfer functions in Figure 20.1:

$$H_c(s) = K_c \quad \text{(proportional controller)} \tag{20.5}$$

$$H_p(s) = \frac{1}{(s+1)^2 s} \tag{20.6}$$

(The transfer function $H_d(s)$ in Figure 20.1 is not used in the stability analysis.) The loop transfer function is:

$$\begin{aligned} H_t(s) &= H_p(s)H_c(s) \\ &= \frac{K_c = n_l(s)}{(s+1)^2 s = d_l(s)} \end{aligned}$$

We get:

$$H_t(s) = \frac{H_l(s)}{1 + H_l(s)} = \frac{n_l(s)}{d_l(s) + n_l(s)} = \frac{K_c}{s^3 + 2s^2 + s + K_c} \tag{20.7}$$

Thus, the characteristic polynomial of $H_t(s)$ is:

$$a(s) = s^3 + 2s^2 + s + K_c \tag{20.8}$$

The following Python program calculates the poles and simulates the step response of the control system.

http://techteach.no/control/python/poles_and_sim_control_system.py

I have used three different values of K_c. Table 20.1 shows the poles, and Figures 20.4-20.4 show the pole placements in the complex plane, and the step responses.

There is also a SimView simulator of this control system:

http://techteach.no/simview/stability_poles

[End of Example 20.1]

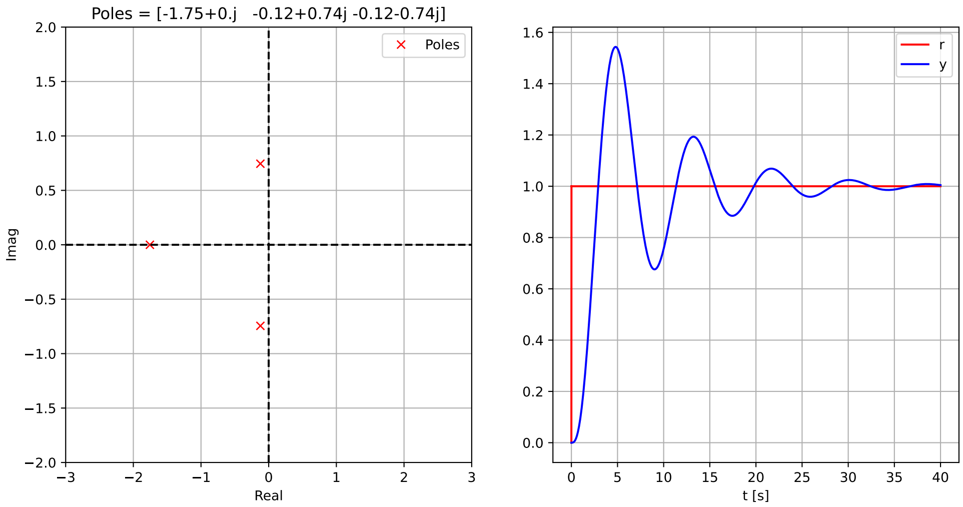


Figure 20.2: Example 20.1: Poles and step response of the control system with $K_c = 1$. The control system is asymptotically stable.

20.3 Problems for Chapter 20

Problem 20.1 *Control system stability with P controller*

Given a feedback control system where the process is an integrator:

$$H_p(s) = \frac{1}{s} \tag{20.9}$$

and the controller is a P controller:

$$H_c(s) = K_c \tag{20.10}$$

1. What is the stability property of $H_p(s)$, i.e. of the process itself – also called the open-loop system?
2. For which values of the controller gain K_c is the control system asymptotically stable?
3. Has this problem demonstrated that it is possible to obtain an asymptotically stable feedback system even though the process itself is asymptotically stable?

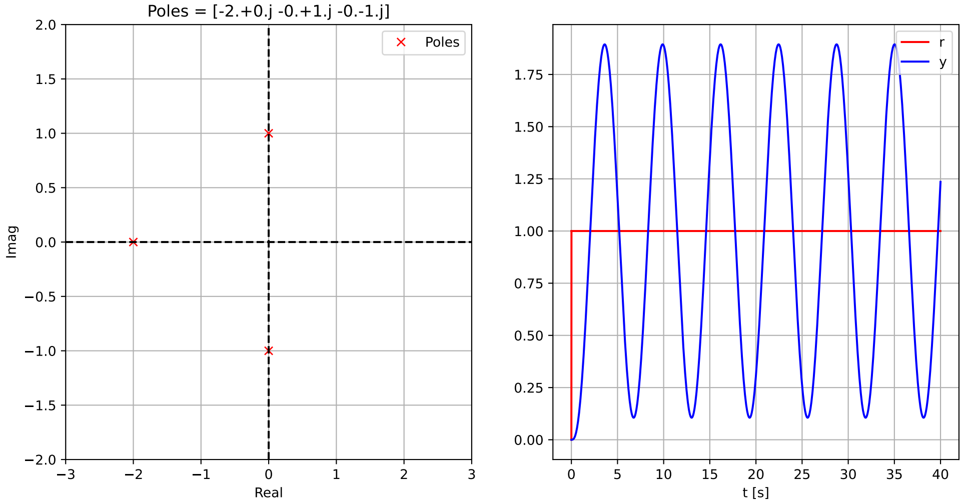


Figure 20.3: Example 20.1: Poles and step response of the control system with $K_c = 2$. The control system is marginally stable.

Solution to Problem 20.1

1. $H_p(s)$ has pole equal to 0. Consequently the system is marginally stable.

2. The transfer function of the controller is

$$H_c(s) = K_c \tag{20.11}$$

The loop transfer function is

$$H_l(s) = H_p(s)H_c(s) = \frac{K_c}{s} = \frac{n_l(s)}{d_l(s)} \tag{20.12}$$

The characteristic polynomial is

$$a(s) = d_l(s) + n_l(s) = s + K_c \tag{20.13}$$

The pole of the control system is the root of (20.13):

$$p = -K_c \tag{20.14}$$

The control system is asymptotically stable with $\text{Re}(p) < 0$, i.e.

$$K_c > 0 \tag{20.15}$$

3. Sure!

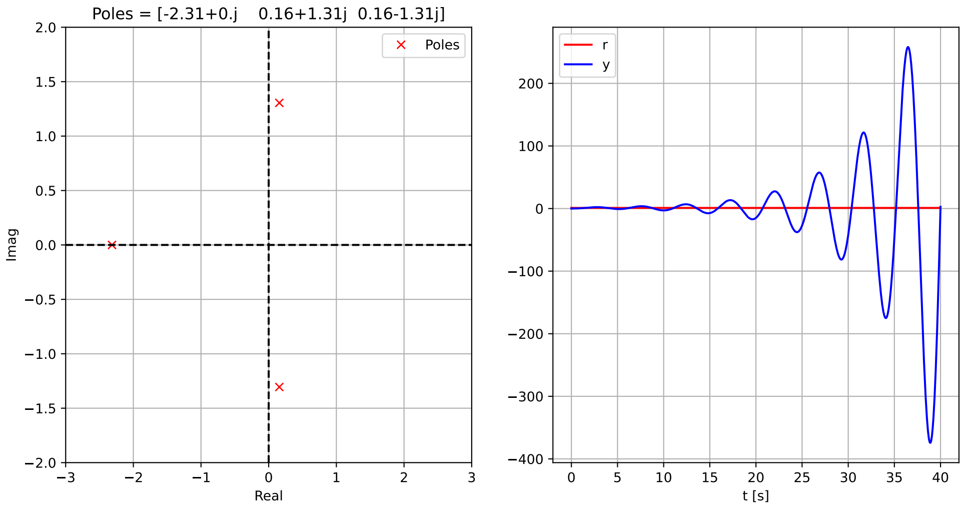


Figure 20.4: Example 20.1: Poles and step response of the control system with $K_c = 4$. The control system is unstable.

Chapter 21

Frequency response

21.1 Introduction

The *frequency response* of a system is a frequency dependent function which expresses how a sinusoidal signal of a given frequency on the system input is transferred through the system. Time-varying signals – at least periodical signals – which excite systems, as the reference signal or a disturbance in a control system or measurement signals which are inputs signals to signal filters, can be regarded as consisting of a sum of *frequency components*. Each frequency component is a sinusoidal signal having a certain amplitude and a certain frequency. (The Fourier series expansion or the Fourier transform can be used to express these frequency components quantitatively.) The frequency response expresses how each of these frequency components is transferred through the system. Some components may be amplified, others may be attenuated, and there will be some phase lag through the system.

The frequency response is an important tool for analysis and design of signal filters (as lowpass filters and highpass filters), and for analysis, and to some extent, design, of control systems. Both signal filtering and control systems applications are described (briefly) later in this chapter.

The definition of the frequency response – which will be given in the next section – *applies only to linear models*, but this linear model may very well be the local linear model about some operating point of a non-linear model.

The frequency response can found experimentally or from a transfer function model. It can be presented graphically or as a mathematical function.

21.2 How to calculate frequency response from sinusoidal input and output

We can find the frequency response of a system by exciting the system with a sinusoidal signal of amplitude U and frequency ω [rad/s] and observing the response in the output

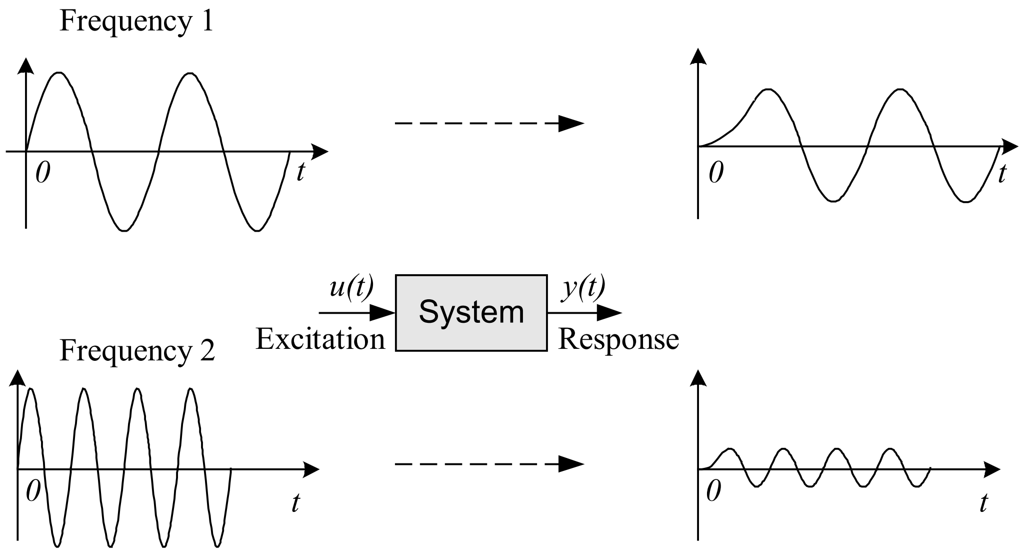


Figure 21.1: Sinusoidal signals in the input and the resulting responses on the output for two different frequencies

variable of the system. [1] Mathematically, we set the input signal to

$$u(t) = U \sin \omega t \tag{21.1}$$

See Figure 21.1. This input signal will give a transient response (which will die, eventually) and a *steady state* response, $y_s(t)$, in the output variable:

$$\begin{aligned} y_s(t) &= Y \sin(\omega t + \phi) && (21.2) \\ &= \underbrace{UA}_{Y} \sin(\omega t + \phi) && (21.3) \end{aligned}$$

A is the *(amplitude)gain*, and ϕ (phi) is *the phase lag* in radians. The frequency of $y_s(t)$ will be the same as in $u(t)$. Figure 21.2 shows in detail $u(t)$ and $y(t)$ for a simulated system. The system which is simulated is

$$Y(s) = \frac{1}{s+1} U(s) \tag{21.4}$$

i.e., a first order system with gain 1 and time constant 1. The input signal $u(t)$ has frequency $\omega = 3$ rad/s and amplitude $U = 1$. The simulation is done with the following Python program:

http://techteach.no/control/python/sim_freqresp_first_order_sys.py

[1] The correspondance between a given frequency ω in rad/s and the same same frequency f in Hz is $\omega = 2\pi f$.

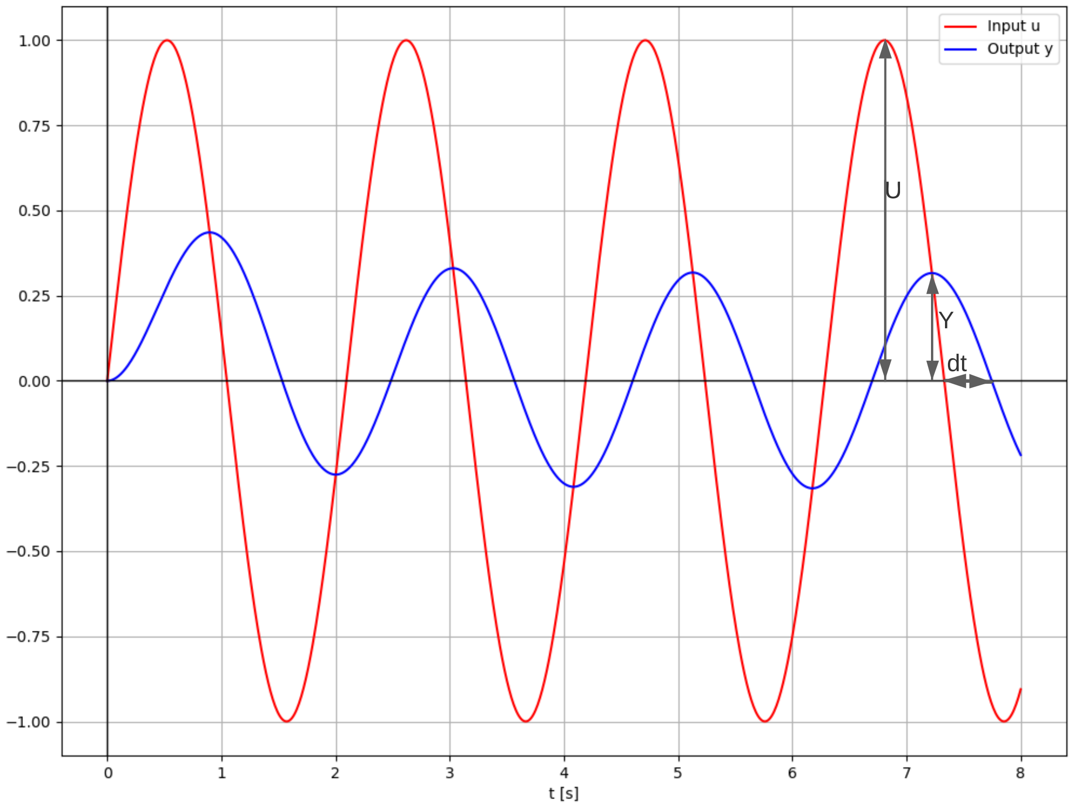


Figure 21.2: The input signal $u(t)$ and the resulting (sinusoidal) response $y(t)$ for a simulated system. $u(t)$ has frequency $\omega = 3$ rad/s and amplitude $U = 1$. The system is given by (21.4).

A is the ratio between the amplitudes of the output signal and the input signal (in steady state):

$$A = \frac{Y}{U} \tag{21.5}$$

For the signals shown in Figure 21.2,

$$A = \frac{Y}{U} = \frac{0.32}{1} = 0.32 \tag{21.6}$$

ϕ can be calculated by first measuring the time-lag dt between $u(t)$ and $y_s(t)$, and then calculating ϕ as follows:

$$\phi = -\omega \cdot dt \quad [\text{rad}] \tag{21.7}$$

In Figure 21.2 we can read off $dt = 0.42$ sec, which gives

$$\phi = -\omega \cdot dt = -3 \cdot 0.42 = -1.26 \text{ rad} \tag{21.8}$$

The gain A and the phase-lag ϕ are functions of the frequency. We can use the following terminology: $A(\omega)$ is the *gain function*, and $\phi(\omega)$ is the *phase shift function* (or more simply: phase function). We say that $A(\omega)$ and $\phi(\omega)$ expresses the *frequency response* of the system.

21.3 Bode diagram

It is common to present $A(\omega)$ and $\phi(\omega)$ graphically in a *Bode diagram*, which consists of two subdiagrams, one for $A(\omega)$ and one for $\phi(\omega)$. It is common, but not mandatory, to set up the Bode diagrams as follows:

- The frequency is plotted in unit rad/s or Hz along a logarithm axis (abscissa). Note: In case you need to switch between rad/s and Hz, the following relation applies:

  $$\omega = 2\pi f \tag{21.9}$$

  where ω is in rad/s and f is in Hz.

- The amplitude gain is plotted in unit decibel (dB):

  $$A = 20 \log_{10} A \text{ dB} \tag{21.10}$$

  Some examples:

  - $0.1 = 20 \log_{10} 0.1 = 20 \cdot (-1) = -20$ dB
  - $1 = 20 \log_{10} 1 = 20 \cdot 0 = 0$ dB
  - $10 = 20 \log_{10} 10 = 20 \cdot 1 = 20$ dB

- Phase values are in degrees. An example of conversion from rad to deg:

  - 0.5 rad $= 0.5 {\cdot} 180/\pi = 28.6$ deg

Figure 21.3 shows a Bode diagram of the frequency response of the system (21.4). The curves may be an (interpolated) plot of a number of A-values and ϕ-values found in experiments (or simulations) with a sinusoidal input signal of various frequencies. As an example, In Figure 21.2, the red points are the A-value in dB and ϕ-value in degrees of the system at the frequency 3 rad/s of the sinusoidal input shown in Figure 21.2. The frequency is logarithmic. Let's calculate the values of these points in the pertinent units:

- The 10-logaritm of 3 rad/s is $\log_{10} 3 = 0.48$ rad/s.

- From the simulations with frequency $\omega = 3$ rad/s shown in Figure 21.2 we earlier found the A-value as 0.32, see (21.6), which in dB is

  $$A(3) = 20 \log_{10} A \text{ dB} = 20 \log_{10} 0.32 \text{ dB} = -10 \text{ dB} \tag{21.11}$$

  which is the red point in the upper diagram of Figure 21.3.

- From the simulations mentioned above we found ϕ as -1.26 rad, see (21.8), which in degrees is

  $$\phi = -1.26 \cdot 180/\pi = -72.2 \text{ deg} \tag{21.12}$$

  which is the red point in the lower diagram of Figure 21.3.

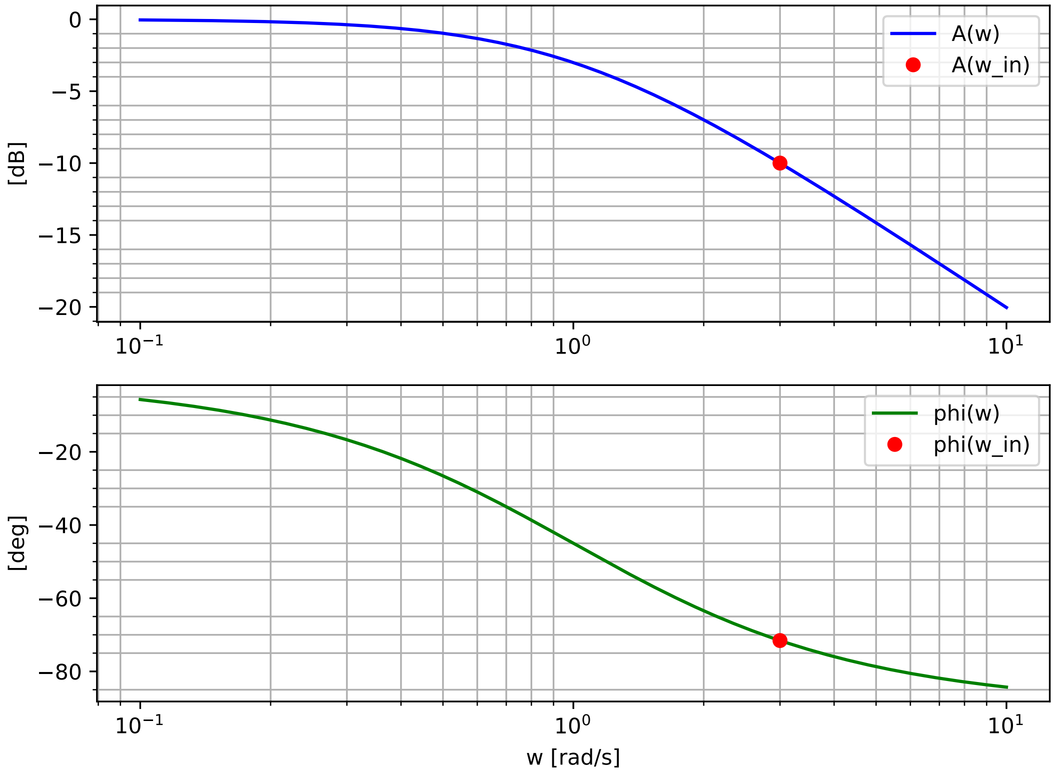


Figure 21.3: The frequency response of the system given by (21.4) presented in a Bode diagram.

21.4 How to calculate frequency response from transfer function

In Section 21.2 we saw how to find the frequency response from experiments on the system, namely the gain function, $A(\omega)$, and the phase function, $\phi(\omega)$. No model was assumed in Section 21.2. However, if we know a transfer function model of the system, we can actually *calculate the frequency response from the transfer function*, as explained below, without a need of experiments. So, it is an experiment-free method. The method is only presented here, not derived, but it can derived using Laplace transform theory. As you will see in the examples of this section, you need knowledge about complex numbers to calculate the frequency response from a given transfer function.

Assume given a system with transfer function $H(s)$ from input u to output y, that is,

$$Y(s) = H(s)U(s) \tag{21.13}$$

By setting

$$s = j\omega \tag{21.14}$$

where j is the imaginary unit, into $H(s)$, we get $H(j\omega)$, which is a complex function as it will have complex value. $H(j\omega)$ is the *frequency response* (function). *The gain function*, which we already know from Section 21.2, is

$$A(\omega) = |H(j\omega)| \tag{21.15}$$

and the *phase shift function*, also known from Section 21.2, is the angle or argument of $H(j\omega)$:

$$\phi(\omega) = \arg H(j\omega) \tag{21.16}$$

Example 21.1 *Frequency response calculated from a transfer function*

We will find the frequency response for the transfer function

$$H(s) = \frac{K}{t_c s + 1} \tag{21.17}$$

The frequency response becomes

$$H(j\omega) = H(s)|_{s=j\omega} = \frac{K}{t_c j\omega + 1} = \frac{K}{\underbrace{1}_{\text{Re}} + j\underbrace{t_c\omega}_{\text{Im}}} \tag{21.18}$$

which we write on polar form:

$$H(j\omega) = \frac{K}{\sqrt{1^2 + (t_c\omega)^2} e^{j\arctan\left(\frac{t_c\omega}{1}\right)}} \tag{21.19}$$

$$= \frac{1}{\sqrt{1 + (t_c\omega)^2}} e^{j[-\arctan(t_c\omega)]} \tag{21.20}$$

$$= |H(j\omega)|\, e^{j\arg H(j\omega)} \tag{21.21}$$

Thus, the gain function is

$$|H(j\omega)| = A(\omega) = \frac{K}{\sqrt{1 + (t_c\omega)^2}} \tag{21.22}$$

and the phase function is

$$\arg H(j\omega) = \phi(\omega) = -\arctan(t_c\omega) \quad \text{[rad]} \tag{21.23}$$

A Bode diagram of $H(j\omega)$ with the given parameters, namely $K = 1$ and $t_c = 1$, is identical to the Bode diagram in Figure 21.3, and is therefore not repeated here. The Bode diagram is generated with the following program:

http://techteach.no/control/python/bode_plot_first_order_sys.py

Comments to the above program:

- The program uses functions in Python Control Package, cf. Appendix 41.
- The frequency response is obtained with the control.freqresp function.

To illustrate the use of (21.22) and (21.23), let us calculate $|H(j\omega)|$ and $\arg H(j\omega)$ at frequency $\omega = 3$ rad/s, which are the respective two red points in the Bode diagram in Figure 21.3. (21.22) gives

$$|H(j3)| = \frac{1}{\sqrt{1+3^2}} = \frac{1}{\sqrt{10}} = 0.316 = -20\log_{10}\left(\frac{1}{\sqrt{10}}\right) = -10.0 \text{ dB} \tag{21.24}$$

(21.23) gives

$$\arg H(j3) = -\arctan{(3)} = -1.25 \text{ rad} = -71.6 \text{ degrees} \tag{21.25}$$

These values are corresponds to the values read off from simulations with $\omega = 3$ rad/s in Section 21.2. The phase functions differs a little from the value -72.2 read off from simulations. The accurate value is -71.6 found with (21.25).

[End of Example 21.1]

The frequency response of a compound transfer function

The following example shows how the frequency response can be found of a transfer function which consists of several factors in the numerator and/or the denominator.

Example 21.2 *Frequency response of a (more complicated) transfer function*

Given the transfer function

$$H(s) = K\frac{t_1 s + 1}{(t_2 s + 1)\, s}e^{-t_d s} \tag{21.26}$$

The term $e^{-t_d s}$ represents actually a time delay of t_d sec time units (typically sec.). We set $s = j\omega$ in $H(s)$, and then express the relevant factors on *polar form*. And finally we combine these factors to end up with a polar form of $H(j\omega)$:

$$H(j\omega) = K\frac{t_1 j\omega + 1}{(t_2 j\omega + 1)\, j\omega}e^{-t_d j\omega} \tag{21.27}$$

$$= K\frac{\sqrt{1^2 + (t_1\omega)^2}e^{j\arctan\left(\frac{t_1\omega}{1}\right)}}{\left[\sqrt{1^2 + (t_2\omega)^2}e^{j\arctan\left(\frac{t_2\omega}{1}\right)}\right]\left[\sqrt{0^2+\omega^2}e^{j\frac{\pi}{2}}\right]}e^{-t_d j\omega} \tag{21.28}$$

$$= \underbrace{\frac{K\sqrt{1 + (t_1\omega)^2}}{\sqrt{1 + (t_2\omega)^2}\omega}}_{|H(j\omega)|} e^{j\underbrace{\left[\arctan{(t_1\omega)} - \arctan{(t_2\omega)} - \frac{\pi}{2} - t_d\omega\right]}_{\arg H(j\omega)}} \tag{21.29}$$

So, the amplitude gain function is

$$|H(j\omega)| = \frac{K\sqrt{1+(t_1\omega)^2}}{\sqrt{1+(t_2\omega)^2}\omega} \tag{21.30}$$

and the phase shift function is

$$\arg H(j\omega) = \arctan(t_1\omega) - \arctan(t_2\omega) - \frac{\pi}{2} - t_d\omega \tag{21.31}$$

[End of Example 21.2]

21.5 Filters

21.5.1 Filter types

A *signal filter* – or just *filter* – is used to attenuate (ideally: remove) a certain frequency interval of frequency components from a signal. These frequency components are typically noise. For example, a lowpass filter is used to attenuate high-frequent components (low-frequent components will pass).

Knowledge about filtering functions is crucial in signal processing, but it is useful also in control engineering because control systems can be regarded as filters in the sense that the controlled process variable can follow only a certain range or interval of frequency components in the reference signal, and it will be only a certain frequency range of process disturbances that the control system can compensate for effectively. Furthermore, knowledge about filters can be useful in the analysis and design of physical processes. For example, a stirred tank in a process line can act as a lowpass filter since it attentuates low-frequent components in the inflow to the tank.

In this section we will particularly study *lowpass filters*, which is the most commonly used filtering function, but we will also take a look at *highpass filters*, *bandpass filters* and *bandstop filters*.

Figure 21.4 shows the gain function for ideal filtering functions and for practical filters (the phase lag functions are not shown).

The *passband* is the frequency interval where the gain function has value 1, ideally (thus, frequency components in this frequency interval passes through the filter, unchanged). The *stopband* is the frequency interval where the gain function has value 0, ideally (thus, frequency components in this frequency interval are stopped through the filter).[2]

It can be shown that transfer functions for ideal filtering functions will have infinitely large order. Therefore, ideal filters can not be realized, neither with analog electronics nor with a filtering algorithm in a computer program.

[2] It is a pity that lowpass filters in the beginning were not called highstop filters instead because the main purpose of a lowpass filter is to stop high-frequency components, and not to let low-frequency components pass. Similarly, highpass filters should have been called lowstop filters. It is too late to make a change now.

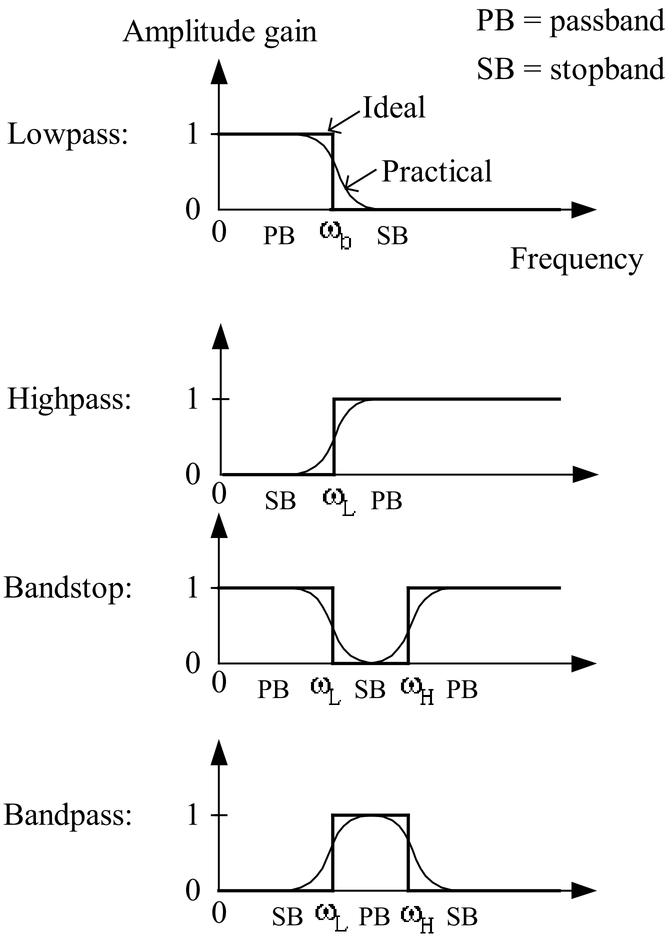


Figure 21.4: The gain functions for ideal filters and for practical filters of various filter types.

21.5.2 First order lowpass filters

The most commonly used signal filter is the first order lowpass filter. For example, it is often used as measurement filter in feedback control systems, cf. the time constant filter presented in Section 3.4.6.

The transfer function of a first order lowpass filter with input variable u and output variable y is usually written on the form

$$H(s) = \frac{Y(s)}{U(s)} = \frac{1}{\frac{s}{\omega_b} + 1} = \frac{K}{t_f s + 1} \tag{21.32}$$

where ω_b [rad/s] is the *bandwidth* of the filter. (21.32) is a first order transfer function with gain

$$K = 1 \tag{21.33}$$

and (filter) time constant

$$t_f = \frac{1}{\omega_b} \tag{21.34}$$

The frequency response of (21.32) is

$$H(j\omega) = \frac{1}{\frac{j\omega}{\omega_b} + 1} \tag{21.35}$$

$$= \frac{1}{\sqrt{\left(\frac{\omega}{\omega_b}\right)^2 + 1}\, e^{j \arctan \frac{\omega}{\omega_b}}}$$

$$= \frac{1}{\sqrt{\left(\frac{\omega}{\omega_b}\right)^2 + 1}} e^{j\left(-\arctan \frac{\omega}{\omega_b}\right)} \tag{21.36}$$

The gain function is

$$|H(j\omega)| = \frac{1}{\sqrt{\left(\frac{\omega}{\omega_b}\right)^2 + 1}} \tag{21.37}$$

and the phase function is

$$\arg H(j\omega) = -\arctan \frac{\omega}{\omega_b} \tag{21.38}$$

The bandwidth defines the upper limit of the passband. It is common to say that the bandwidth is the frequency where the filter gain is $1/\sqrt{2} = 0.71 \approx -3$ dB (above the bandwidth the gain is less than $1/\sqrt{2}$). This bandwidth is therefore referred to as the "-3 dB-bandwidth". Now, what is the -3 dB-bandwidth of a first order lowpass filter? It is the ω-solution of the equation

$$|H(j\omega)| = \frac{1}{\sqrt{\left(\frac{\omega}{\omega_b}\right)^2 + 1}} = \frac{1}{\sqrt{2}} \tag{21.39}$$

The solution is $\omega = \omega_b$. Therefore, ω_b given in (21.32) is the -3 dB-bandwidth in rad/s.

Figure 21.5 shows the Bode diagram (the upper two plots in the figure) of (21.32) with bandwidth $\omega_b = 1$ rad/s. In the figure, the cyan dashed line represents the bandwidth.

The plots in Figure 21.5 are generated with the following program:

http://techteach.no/control/python/bode_plot_sim_lowpass_filter.py

Figure 21.5 also shows the result of an example of a simulation. In the simulation, the filter input (the red curve) is a sine function with amplitude $U = 1$ and frequency $\omega_{\text{in}} = 5$ rad/s. The response (the blue curve) has amplitude $Y \approx 0.2$. Let's check if this value of Y is in accordance with what we can find from the gain curve of the Bode diagram. At $\omega = 5$ rad/s, the gain curve has the value -14 dB, which is equal to

$$10^{-14/20} = 0.20 \tag{21.40}$$

So, the simulation is in accordance with the frequency response. In other words, from the Bode diagram, you can predict the sinusoidal response, eliminating the need for practical experiments or simulations.

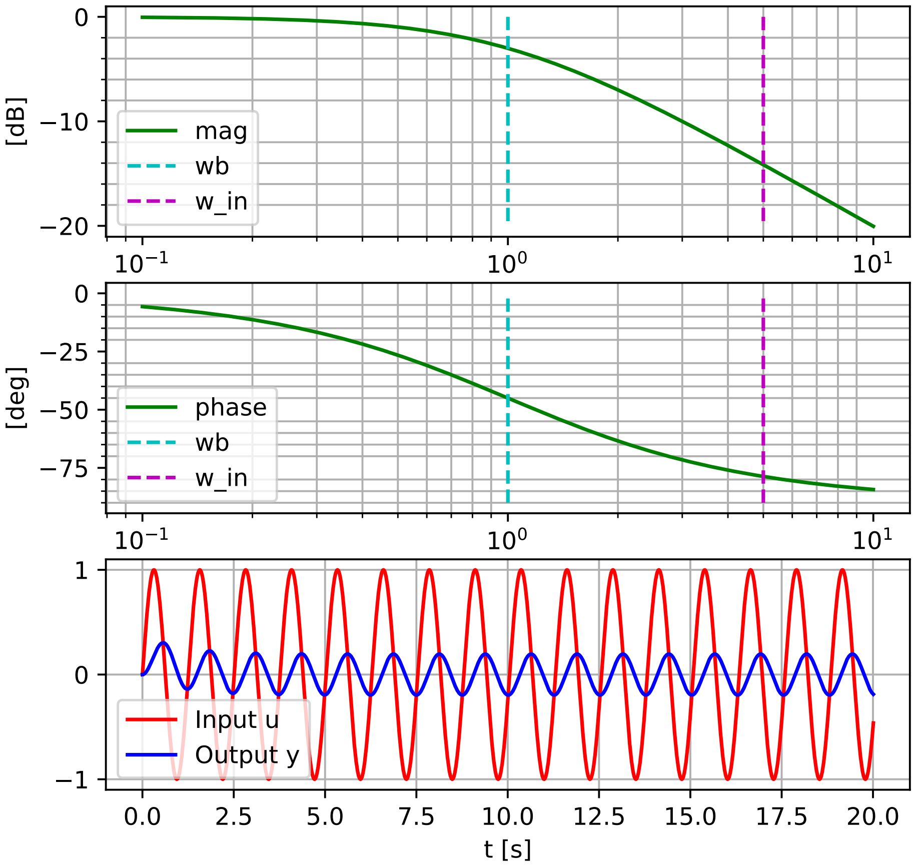


Figure 21.5: Bode diagram (the upper two plots) and simulated response of a first order lowpass filter.

Example 21.3 *The RC-circuit as a lowpass filter*

Figure 21.6 shows an RC-circuit (the circuit contains the resistor R and the capacitor C). The RC-circuit is frequently used as an analogue lowpass filter: Signals of *low* frequencies *passes* approximately unchanged through the filter, while signals of high frequencies are approximately filtered out (stopped). v_1 is the signal source or input voltage to be filtered, while v_2 is the resulting filtered output voltage.

We will now find a mathematical model relating v_2 to v_1. First we apply the Kirchhoff's voltage law in the circuit which consists the input voltage terminals, the resistor, and the capacitor (we consider the voltage drops to be positive clockwise direction):

$$-v_1 + v_R + v_2 = 0 \tag{21.41}$$

(v_2 equals the voltage drop over the capacitor.) In (21.41) v_R is given by

$$v_R = Ri \tag{21.42}$$

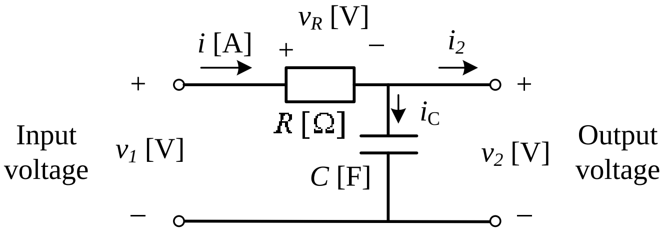


Figure 21.6: RC-circuit

We assume that there is no current going through the output terminals. (This is a common assumption, and not unrealistic, since it it typical that the output terminals are connected to a subsequent circuit which has approximately infinite input impedance, causing the current into it to be approximately zero. An operational amplifier is an example of such a load-circuit.) Therefore,

$$i = i_C = Cv_2' \tag{21.43}$$

The final model is achieved by using i as given by (21.43) in (21.42) and then using v_R as given by (21.42) for v_R in (21.41). The model becomes

$$RCv_2' = v_1 - v_2 \tag{21.44}$$

The transfer function from the input voltage v_1 to the output voltage v_2 becomes

$$\frac{V_2(s)}{V_1(s)} = H(s) = \frac{1}{RCs + 1} = \frac{1}{\frac{s}{\omega_b} + 1} \tag{21.45}$$

Thus, the RC-circuit is a first order lowpass filter with bandwidth

$$\omega_b = \frac{1}{RC} \text{ rad/s} \tag{21.46}$$

If for example $R = 1$ kΩ and $C = 10$ μF, the bandwidth is $\omega_b = 1/RC = 100$ rad/s. (21.46) can be used to design the RC-circuit (i.e. calculating the R- and C-values).

[End of Example 21.3]

21.6 *Problems for Chapter 21*

Problem 21.1 *Frequency response measures from sinusoidal input and output*

Figure 21.7 shows the input signal and the corresponding output signal of a system.

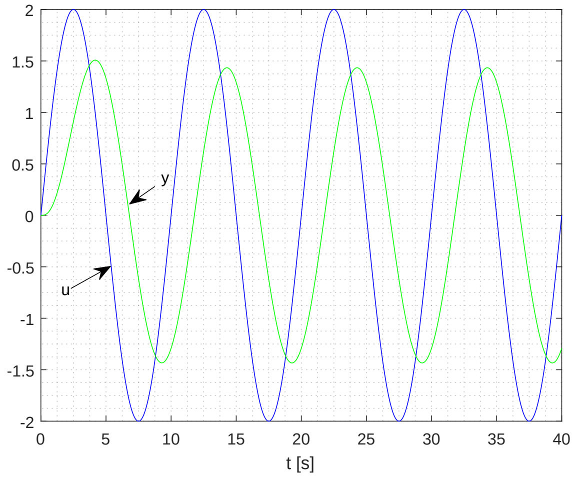


Figure 21.7: Input signal u and output signal y

1. What is the frequency of the signal in Hz and in rad/s?
2. Calculate the amplitude gain and the phase lag at the frequency found in Problem 1 above. What is the amplitude gain in dB?

Problem 21.2 *Steady-state sinusoidal response found from Bode diagram*

Figure 21.8 shows a Bode diagram of a system.

Assume that the input signal u is a sine function with amplitude $U = 0.8$ and frequency $\omega = 1.0$ rad/s. Write the corresponding steady state output response $y_s(t)$.

Problem 21.3 *Frequency response from transfer function*

Derive the frequency response functions $A(\omega)$ and $\phi(\omega)$ of the transfer function

$$H(s) = \frac{K}{(t_1 s + 1)(t_2 s + 1)} e^{-t_d s} \tag{21.47}$$

Problem 21.4 *Frequency response and simulation with Python*

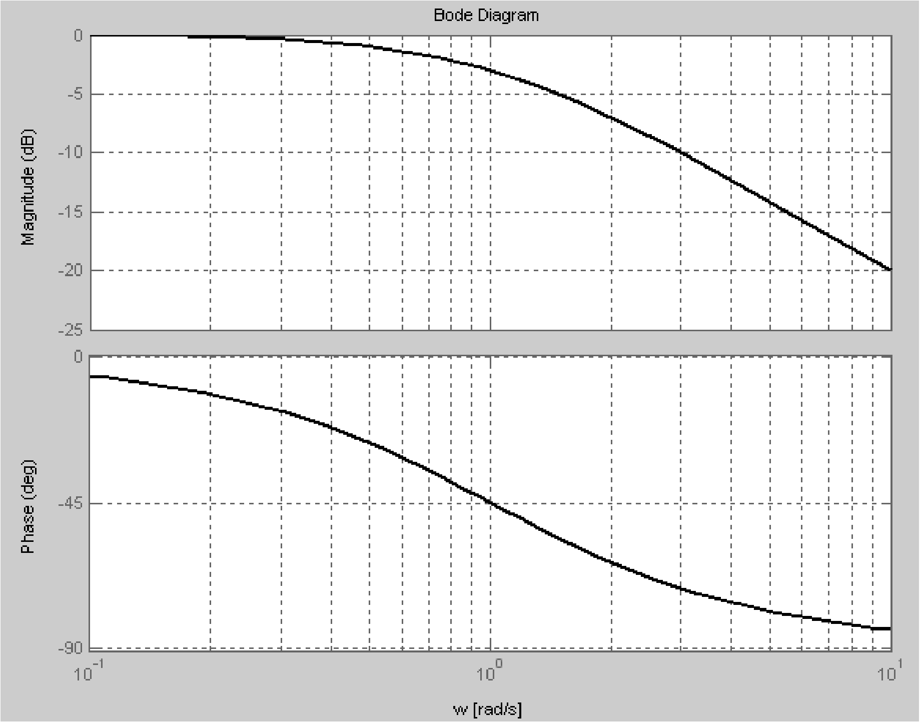


Figure 21.8: Bode diagram

The following program was used to generate the Bode diagram and to run the simulation shown in Figure 21.10.

http://techteach.no/control/python/bode_plot_sim_lowpass_filter.py

Modify the program as follows:

- The filter bandwidth is set to $\omega_b = 2$ rad/s.
- The input signal has frequency $\omega_{\text{in}} = 2$ rad/s and amplitude $U = 5$. So, in this problem, ω_{in}happens to be set equal to the bandwidth.

Now, run the program. From the Bode diagram generated by the program, predict the amplitude, Y, of the response due to the sinusoidal input. Is this value of Y the same as you can read off from the simulation?

Problem 21.5 *Design of RC filter from bandwidth*

Assume that it is specified that a given RC filter shall have bandwidth 100 Hz. Find proper values of the resistance R the capacitance C. (Tip: C can be selected between 10^{-4} and 10^{-6} F because this gives a practical size of the capacitor, e.g. 10^{-5} F.)

21.7 *Solutions to problems for Chapter 21*

Solution to Problem 21.1

1. Figure 21.9 shows the signals with amplitudes and time-lag indicated.

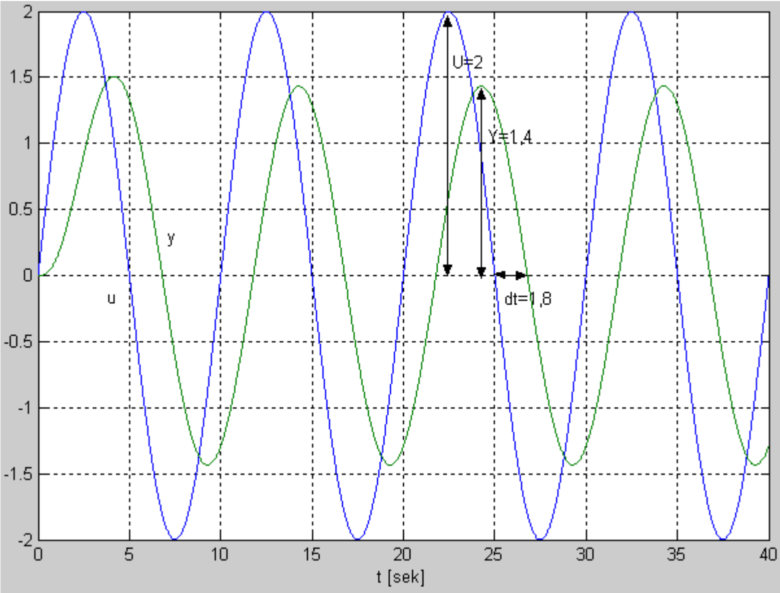


Figure 21.9: Input signal u and output signal y

From the figure we see that the period of the input signal is

$$t_p = 10 \text{ sec} \tag{21.48}$$

which corresponds to the frequency

$$f_1 = \frac{1}{t_p} = \frac{1}{10} = 0.1 \text{ Hz} \tag{21.49}$$

or, alternatively,

$$\omega_1 = 2\pi f_1 = 2 \cdot \pi \cdot 0.1 = 0.63 \text{ rad/s} \tag{21.50}$$

2. The amplitude gain (at frequency f_1) is

$$A = \frac{Y}{U} = \frac{1.4}{2} = 0.7 = 20 \cdot \log_{10}(0.7) \text{ dB} = -3.1 \text{ dB} \tag{21.51}$$

The phase lag ϕ can be calculated by firstly measuring the time lag Δt between input $u(t)$ and output $y(t)$ and then calculating ϕ with

$$\phi = -\omega\Delta t \text{ [rad]} \tag{21.52}$$

From Figure 21.9 you can find

$$\Delta t = 1.8 \text{ s} \tag{21.53}$$

Hence,

$$\phi = -\omega \Delta t = -0.63 \cdot 1.8 = -1.13 \text{ rad} = -1.13 \cdot \frac{180}{\pi} = -65 \text{ deg} \tag{21.54}$$

Alternatively, we can calculate ϕ from the following ratio (360 degrees corresponds to one period):

$$\phi = -\frac{\Delta t}{t_p} \cdot 360 \text{ deg} = -\frac{1.8 \text{ s}}{10 \text{ s}} \cdot 360 \text{ deg} = -65 \text{ deg} \tag{21.55}$$

Solution to Problem 21.2

The steady state response is

$$y_s(t) = UA \sin(\omega t + \phi) \tag{21.56}$$

The amplitude of the input signal is $U = 0.8$. The amplitude gain A is read off from the upper curve in the Bode diagram at frequency $\omega = 1.0$ rad/s:

$$A = -3 \text{ dB} \tag{21.57}$$

which is

$$A = 10^{-3/20} = 0.71 \tag{21.58}$$

The phase lag ϕ at frequency $\omega = 1.0$ rad/s is read off from the lower curve in the Bode diagram:

$$\phi = -45 \text{ deg} = -45 \cdot \frac{\pi}{180} \text{ rad} = -0.79 \text{ rad} \tag{21.59}$$

Hence,

$$y_s(t) = 0.8 \cdot 0.71 \cdot \sin(1.0 \cdot t - 0.79) = 0.57 \sin(t - 0.79) \tag{21.60}$$

Solution to Problem 21.3

We set $s = j\omega$ in $H(s)$, and then turn the factors into *polar forms*, which we then combine to finally get a polar form of $H(j\omega)$:

$$H(j\omega) = \frac{K}{(1 + t_1 j\omega)(1 + t_2 j\omega)} e^{-j\omega t_d} \tag{21.61}$$

$$= \frac{K}{\left[\sqrt{1^2 + (t_1\omega)^2} e^{j \arctan\left(\frac{t_1\omega}{1}\right)}\right] \left[\sqrt{1^2 + (t_2\omega)^2} e^{j \arctan\left(\frac{t_2\omega}{1}\right)}\right]} e^{-j\omega t_d} \tag{21.62}$$

$$= \underbrace{\frac{K}{\sqrt{1 + (t_1\omega)^2}\sqrt{1 + (t_2\omega)^2}}}_{|H(j\omega)|} e^{j \underbrace{[-\arctan(t_1\omega) - \arctan(t_2\omega) - \omega t_d]}_{\arg H(j\omega)}} \tag{21.63}$$

So, we have

$$A(\omega) = |H(j\omega)| = \frac{K}{\sqrt{1+(t_1\omega)^2}\sqrt{1+(t_2\omega)^2}} \tag{21.64}$$

and

$$\phi(\omega) = \arg H(j\omega) = -\arctan(t_1\omega) - \arctan(t_2\omega) \tag{21.65}$$

Solution to Problem 21.4

The following program is a solution:

http://techteach.no/control/python/solution_bode_plot_sim_lowpass_filter.py

The program generates the Bode diagram and the simulation shown in Figure 21.10.

At $\omega = 2$ rad/s, the gain curve has the value -3 dB, which is equal to

$$10^{-3/20} = 0.71 \tag{21.66}$$

From this, we can predict that the sinusoidal output has amplitude $Y = U \cdot 0.71 = 3.55$. From Figure 21.10 we can actually read off $Y = 3.55$. So, the simulation and the frequency response are in accordance.

Solution to Problem 21.5

The bandwidth is

$$f_b \text{ [Hz]} = \frac{\omega_b \text{ [rad/s]}}{2\pi} = \frac{1}{2\pi} \cdot \frac{1}{RC} \tag{21.67}$$

With

$$C = 10^{-5} \text{ F} \tag{21.68}$$

we get

$$R = \frac{1}{2\pi f_b C} = \frac{1}{2\pi \cdot 100 \text{ Hz } \cdot 10^{-5} \text{ F}} = 159\ \Omega \tag{21.69}$$

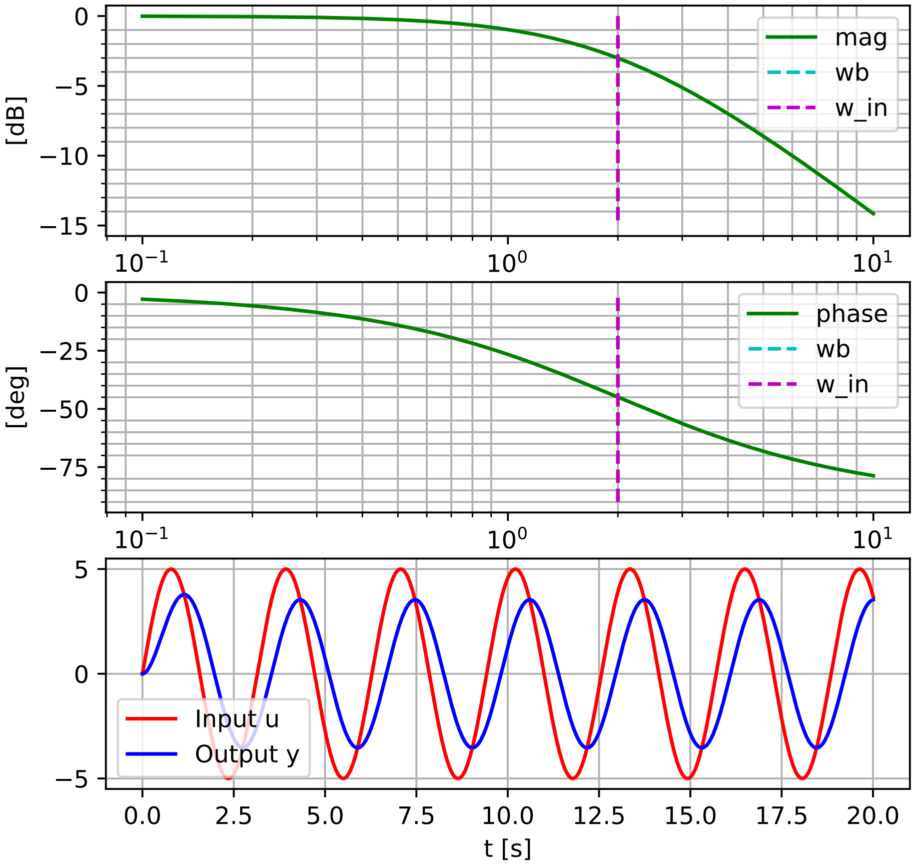


Figure 21.10: Bode diagram (the upper two plots) and simulated responses of a first order lowpass filter.

Chapter 22

Frequency response analysis of feedback systems

22.1 Introduction

With frequency response – using Bode diagrams – we can analyse dynamic properties of feedback. Our focus is feedback *control* systems. These dynamic properties refer to

- dynamic reference tracking, and
- dynamic disturbance compensation.

By definition, in frequency response analysis all signals in the system are assumed to be sine functions. This seems to limit the usefulness of such analysis because in real systems signals are rarely sinusoids. Still, the frequency response analysis provides useful insight about the dynamic properties of a control system because varying signals can be decomposed into certain frequency components.

Frequency response analysis assumes a *linear* model of the control system. However, practical control systems are nonlinear due to phenomena as saturation, hysteresis, stiction, nonlinear signal scaling etc. Such nonlinearities can influence largely the dynamic behaviour of the control system. To perform "linear" analysis of a non-linear model, this model must be linearized about some operating point. Thus, the results of the analysis will be valid at or close to the operation point where the linearization was made. This fact limits the usefulness of a theoretical analysis of a given nonlinear control system using linear systems methods, but the results may still be useful, particularly if the system most of the time operates close to the chosen or specified operating point.

Although a "linear" analysis of a given nonlinear control system may have limited practical value, you may enhance your general understanding about the behaviour of control systems through analysis of examples of linear control systems.

Note: Once you have a mathematical model of a given control system, I recommend that you *run simulations* as a part of the analysis. This applies for both linear and nonlinear control systems. Simulations will give you information about dynamics responses, whether the static control error is zero or not, the impact of process disturbances, the effects of measurement noise, effects of parameter variations, etc.

22.2 Analysis of reference tracking and disturbance compensation

22.2.1 Introduction

Frequency response analysis of control systems expresses the tracking and compensation properties under the assumption that the reference and the disturbance are *sinusoidal signals* or *frequency components* in a compound signal. We assume that the control system has a transfer function-based block diagram as shown in Figure 22.1.

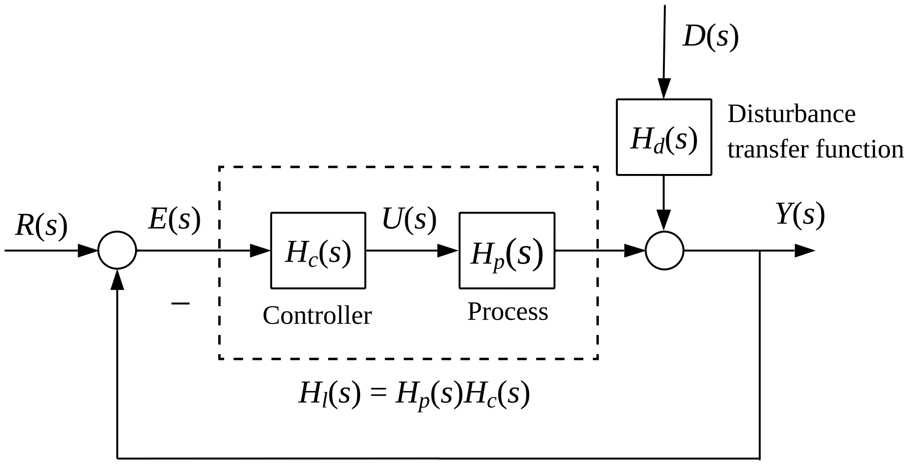


Figure 22.1: Transfer function-based block diagram of a control system.

The following transfer functions where defined in Chapter 12, and repeated here for convenience:

The loop transfer function:

$$H_l(s) = H_p(s)H_c(s) \tag{22.1}$$

The sensitivity transfer function:

$$H_s(s) = \frac{1}{1 + H_l(s)} \tag{22.2}$$

The tracking transfer function:

$$H_t(s) = \frac{H_l(s)}{1 + H_l(s)} = \tag{22.3}$$

$H_t(s)$ and $H_s(s)$ are related:

$$H_t(s) = 1 - H_s(s) \text{ or } H_t(s) + H_s(s) = 1 \tag{22.4}$$

22.2.2 Frequency response analysis of reference tracking

Tracking analysis using $H_s(s)$

As shown in Section 12.3.2, the response in the control error due to the reference is

$$E_r(s) = H_s(s)R(s) \tag{22.5}$$

By plotting the *frequency response* $H_s(j\omega)$ we can easily calculate how large the error is for a given frequency component in the reference: Assume that the reference is a sine function with amplitude A_r and frequency ω. According to frequency response theory, cf. Section 21.4, the steady state response in the control error is

$$e_r(t) = A_r \left| H_s(j\omega) \right| \sin \left[\omega t + \arg H_s(j\omega) \right] \tag{22.6}$$

Thus, the error is small and consequently the tracking property is good if $|H_s(j\omega)| \ll 1$, while the error is large and the tracking property poor if $|H_s(j\omega)| \approx 1$.

Tracking analysis using $H_t(s)$

Alternatively, the tracking property can be analysed using the tracking transfer function $H_t(s)$. The response in the process output measurement due to the reference is

$$Y(s) = H_t(s)R(s) \tag{22.7}$$

Assume that the reference is a sine function with amplitude A_r and frequency ω. According to frequency response theory, cf. Chapter 21, and in particular, Section 21.4, the steady state response in the process output due to the reference is:

$$y(t) = A_r \left| H_t(j\omega) \right| \sin \left[\omega t + \arg H_t(j\omega) \right] \tag{22.8}$$

Thus, $|H_t(j\omega)| \approx 1$ indicates that the control system has good tracking property, while $|H_t(j\omega)| \ll 1$ indicates poor tracking property.

Since both $H_s(s)$ and $H_t(s)$ are functions of the loop transfer function $H_l(s)$, cf. (22.2) and (22.3), there is a relation between $H_l(s)$ and the tracking property of the control system. Using (22.2) and (22.2) we can conclude as follows:

$$\text{Good setpoint tracking: } |H_s(j\omega)| \ll 1, \ |H_t(j\omega)| \approx 1, \ |H_l(j\omega)| \gg 1 \tag{22.9}$$

$$\text{Poor setpoint tracking: } |H_s(j\omega)| \approx 1, \ |H_t(j\omega)| \ll 1, \ |H_l(j\omega)| \ll 1 \tag{22.10}$$

Analysis with Bode plots

Figure 22.2 shows typical Bode plots of $|H_s(j\omega)|$, $|H_t(j\omega)|$ and $|H_l(j\omega)|$. Usually we are interested in the amplitude gains, not the phase lags. Therefore plots of $\arg H_s(j\omega)$, $\arg H_t(j\omega)$ and $\arg H_l(j\omega)$ are not shown nor discussed here.

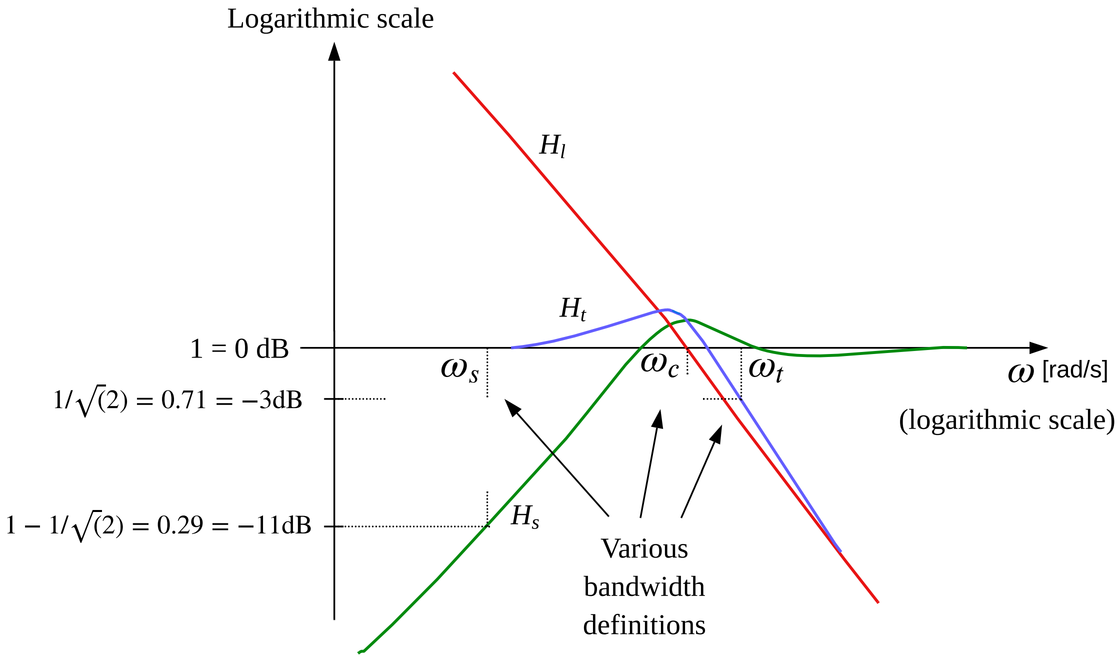


Figure 22.2: Typical Bode plots of $|H_s(j\omega)|$, $|H_t(j\omega)|$ and $|H_l(j\omega)|$.

The *bandwidth* of a control system is the frequency which divides the frequency range of good tracking and the frequency range of poor tracking. From (22.9) and (22.10) and Figure 22.2 we can list the following three candidates for a definition of the bandwidth:

- ω_t, which is the frequency where the amplitude gain of the tracking transfer function has value $1/\sqrt{2} \approx 0.71 = -3$ dB. This definition is in accordance with the usual bandwidth definition of lowpass filters. The ω_t bandwidth is also called the -3 dB bandwidth $\omega_{-3\text{dB}}$.
- ω_c, which is the frequency where the amplitude gain of the loop transfer function has value $1 = -0$ dB. ω_c is called the *crossover frequency* of H_l.
- ω_s, which is the frequency where the amplitude gain of the sensitivity transfer function has value $1 - 1/\sqrt{2} \approx 1 - 0.71 \approx 0.29 \approx -11$ dB. This definition is derived from the -3 dB bandwidth of the tracking transfer function: Good tracking corresponds to tracking gain between $1/\sqrt{2}$ and 1. Now recall that the sensitivity transfer function is the transfer function from reference to control error, cf. (22.5).

> Expressed in terms of the control error, we can say that good tracking corresponds to sensitivity gain $|H_s|$ less than $1 - 1/\sqrt{2} \approx -11$ dB $\approx 0{,}29$. The frequency where $|H_s|$ is -11 dB is denoted the *sensitivity bandwidth*, ω_s.

Of the three bandwidth candidates defined above, the sensitivity bandwidth ω_s is the one that is most closely related to the control error. Therefore, we may say that ω_s is the most convenient bandwidth definition as far as the tracking property of a control system is concerned. In addition, ω_s is a convenient bandwidth related to the compensation property of a control system (we will look into this in more detail below). However, the crossover frequency ω_c and the -3 dB bandwidth are the commonly used bandwidth definitions.

As indicated in Figure 22.2, the numerical values of the various bandwidth definitions will be different (and this is demonstrated in Example 22.1).

If you need a (possibly rough) estimate of the *response time* t_r of a control system, which is time it takes for a step response to reach 63% of its steady state value, you can use

$$t_r \approx \frac{k}{\omega_t} \text{ [s]} \tag{22.11}$$

where ω_t is the -3 dB bandwidth in rad/s.[1] k can be set to some value between 1.5 and 2.0, say 2.0 if you want to be conservative.

Example 22.1 *Frequency response analysis of reference tracking of level control system of wood chips tank*

In this example, we will make a frequency response analysis of the level control system of the wood chips tank presented in Appendix 39.2. The mathematical process model presented in the appendix is repeated here for convenience:

$$\rho A h'(t) \quad = \quad u(t - t_d) - F_{\text{out}}(t) \tag{22.12}$$

h [m] is wood chips level, u [kg/s] is control signal to the feed screw, F_{out} [kg/s] is outflow flow rate, $\rho = 145$ kg/m^3 is density, $A = 13.4$ m^2 is the tank cross-sectional area, and $t_d = 250$ s is time delay on the conveyor belt.

We start by deriving the process transfer functions $H_p(s)$ and $H_d(s)$ from the model (22.12): Taking the Laplace transform, the model becomes

$$\rho A s H^*(s) \quad = \quad e^{-t_d s} U(s) - F^*_{\text{out}}(s) \tag{22.13}$$

From (22.13), we get

$$H^*(s) = \underbrace{\frac{1}{\rho A s} e^{-t_d s}}_{H_p(s)} U(s) + \underbrace{\left(-\frac{1}{\rho A s}\right)}_{H_d(s)} F^*_{\text{out}}(s) \tag{22.14}$$

[1] How can you find the exact value of the response time? Answer: Simulate!

In this example, we will study the reference tracking properties of the control system. Therefore, we can assume the disturbance is zero and therefore neglect $H_d(s)$. So, the process transfer function of interest here, is

$$H_p(s) = \frac{1}{\rho A s} e^{-t_d s} \tag{22.15}$$

We also need the transfer function $H_c(s)$ of the PI controller. It is developed in Section 11.2, but is repeated here for convenience:

$$H_c(s) = K_c + \frac{K_c}{T_i s} \tag{22.16}$$

We assume that the PI level controller has been tuned with the SIMC (Skogestad 2003) PI tuning method for "integrator with time delay" processes (cf. Example 14.6):

$$K_c = 3.89\ [\%/\text{m}] \tag{22.17}$$

$$T_i = 1000\ \text{s} \tag{22.18}$$

The following Python program implements the following, using functions of the Python Control package:

- Creates $H_c(s)$.
- Creates $H_p(s)$.
- Creates $H_l(s)$, $H_t(s)$, and $H_s(s)$.
- Calculates and plots $|H_l(j\omega)|$, $|H_t(j\omega)|$, and $|H_s(j\omega)|$ in a Bode diagram, see Figure 22.3.
- Simulates the responses in the level $h(t)$ and the control error $e(t) = r(t) - h(t)$ due to a sinusoidal $r(t)$, see Figure 22.4.

http://techteach.no/control/python/freq_resp_tracking_level_control_woodchips_tank.py

In the simulation shown in Figure 22.4, the reference $r(t)$ is a sine function with amplitude 1 m and frequency $\omega_r = 0.00075$ rad/s. I have chosen that frequency to be equal to the sensitivity bandwidth ω_s, see Figure 22.3. The expected amplitude of the control error should therefore be $0.29 \cdot 1$ m (cf. the analysis presented before the present example), and this is actually in accordance with the simulation: The plot of $e(t)$ in Figure 22.4 has indeed amplitude equal to 0.29 m.

An interesting observation: Look at the $h(t)$ plot in Figure 22.4. Its steady state amplitude can be seen to be 1.24 m, while the amplitude of $r(t)$ is 1.00 m. It is tempting to say that the maximum amplitude of the control error therefore is $1.24 - 1.00 = 0.24$ m, while that is

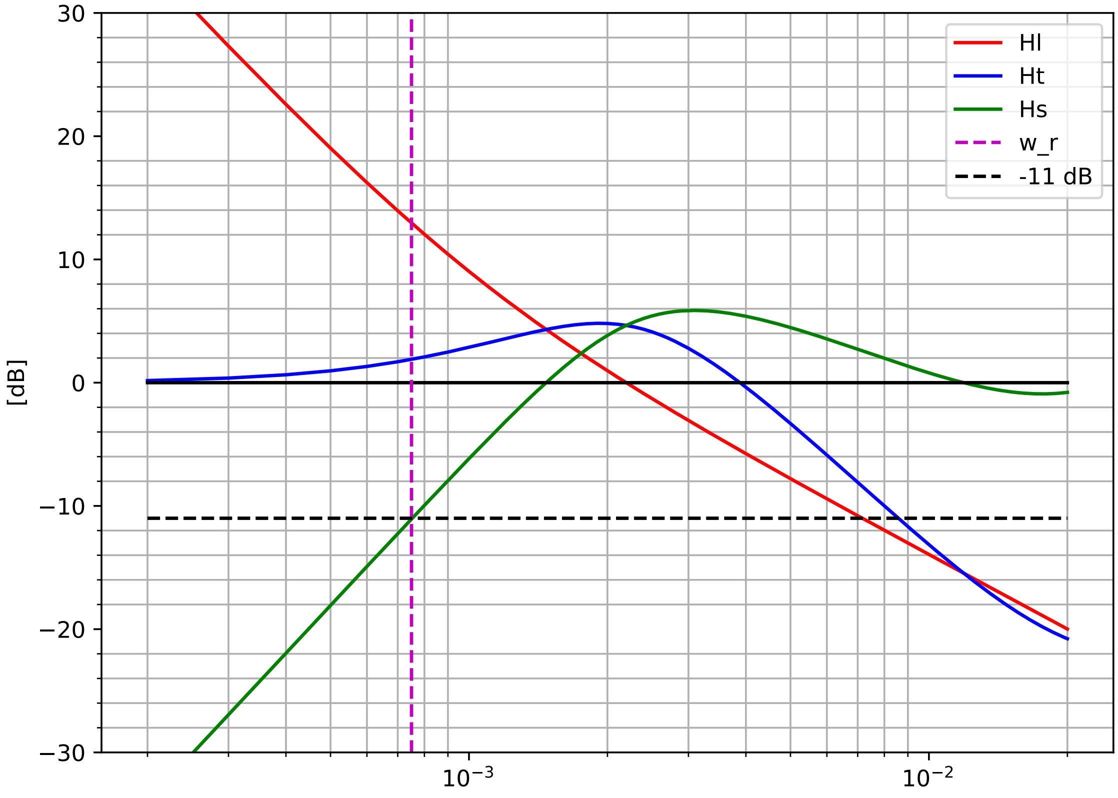


Figure 22.3: Bode plots of $|H_l(j\omega)|$, $|H_l(j\omega)|$, and $|H_l(j\omega)|$ of level control system.

actually wrong. Correct is 0.29 m. This confirms that frequency tracking properties should be analyzed using the sensitivity transfer function $H_s(s)$ rather than the tracking transfer function $H_t(s)$.

Finally, let's compare the values of the alternative bandwidths. From Figure 22.3 we read off the following values:

- -3 dB bandwidth: $\omega_t \approx 0.004$ rad/s.
- Crossover frequency: $\omega_c \approx 0.002$ rad/s.
- Sensitivity bandwidth: $\omega_s = 0.00075$ rad/s.

These values are actually quite different. As commented in the text above this example, I think that the ω_s bandwidth is the most expressive one among the alternative bandwidth definitions.

[End of Example 22.1]

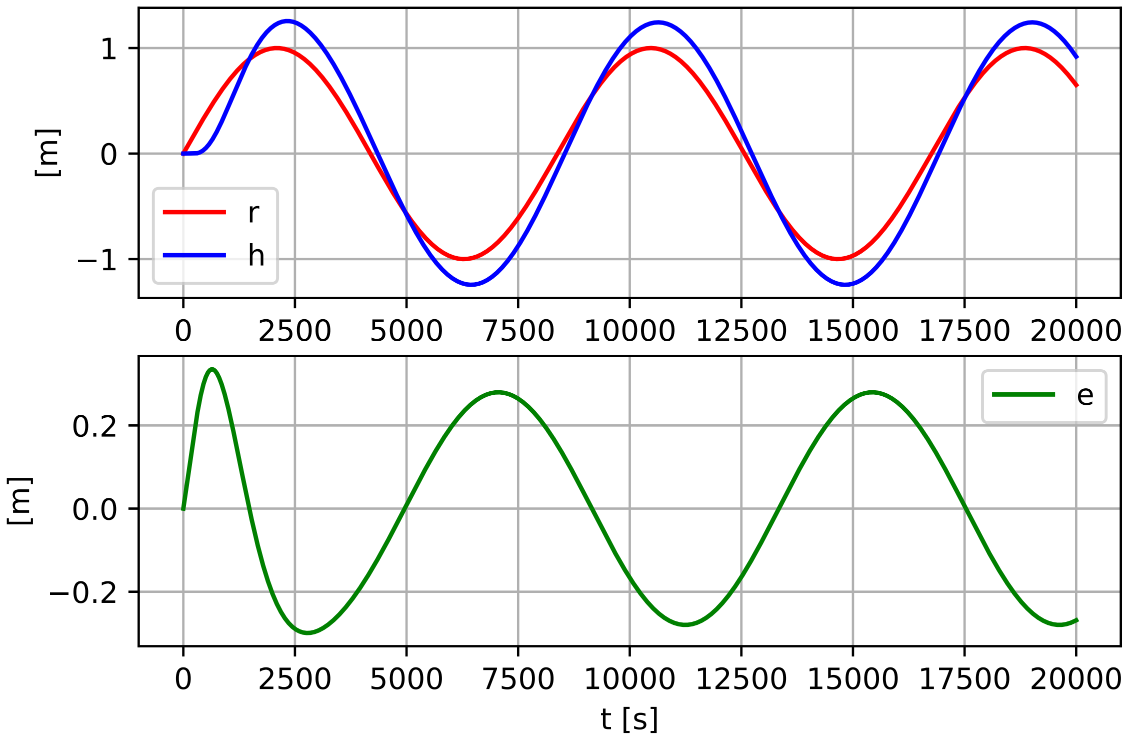


Figure 22.4: Simulated responses in $h(t)$ and $e(t)$ of level control system.

22.2.3 Frequency response analysis of disturbance compensation

As shown in Section 12.3.2, the response in the control error $E_d(s)$ due to the disturbance $D(s)$ is

$$E_d(s) = -H_s(s)H_d(s)D(s) \tag{22.19}$$

Thus, the sensitivity transfer function $H_s(s)$ is a factor in the transfer function from $D(s)$ til $E(s)$ for the control system. However, $H_s(s)$ has an additional meaning related to the compensation of a disturbance, namely it expresses the degree of the reduction of the control error due to using closed loop control, as explained below.

With feedback (i.e. closed loop system) the response in the control error due to the disturbance is

$$E_d(s) = -H_s(s)H_d(s)D(s) \tag{22.20}$$

Without feedback (open loop) this response is

$$E_d(s) = -H_d(s)D(s) \tag{22.21}$$

The ratio between these responses is

$$\frac{E_d(s)_{\text{with feedback}}}{E_d(s)_{\text{without feedback}}} = \frac{-H_s(s)H_d(s)D(s)}{-H_d(s)D(s)} = H_s(s) \tag{22.22}$$

Assuming that the disturbance is a sine function with frequency ω rad/s, (22.22) with $s = j\omega$, that is $H_s(j\omega)$, expresses the ratio between the amplitudes of the respective sinusoidal responses.

Again, effective control, which here means effective disturbance compensation, corresponds to a small value of $|H_s|$ (value zero or close to zero), while ineffective control corresponds to $|H_s|$ close to or greater than 1. We can define the *bandwidth* of the control system with respect to its compensation property. Here are two alternate bandwidth definitions:

- The bandwidth ω_s – the sensitivity bandwidth – is the upper limit of the frequency range of effective compensation. One possible definition is

  $$|H_s(j\omega_s)| \approx 0.29 \approx -11 \text{ dB} \tag{22.23}$$

  which means that the amplitude of the error *with* feedback control is less than 29% of amplitude *without* feedback control. The number 0.29 is chosen to have the same bandwidth definition regarding disturbance compensation as regarding reference tracking, cf. page 580.

- The bandwidth ω_c is the crossover frequency of the loop transfer functions ω_c, that is,

  $$|H_l(j\omega_c)| = 0 \text{ dB} \approx 1 \tag{22.24}$$

Note: The feedback does not reduce the control error due to a sinusoidal disturbance if its frequency is above the bandwidth. But still the disturbance may be well attenuated through the (control) system. This attenuation is due to the typical inherent lowpass filtering characteristic of physical systems (processes). Imagine a liquid tank, which attenuates high-frequent temperature variations existing in the inflow fluid temperature or in the environmental temperature. This inherent lowpass filtering is *self regulation.*

In the present edition of the book, I limit the presentation to giving the theory regarding disturbance compensation, and there is no example.

22.3 Stability analysis of feedback systems

22.3.1 Introduction

In Chapter 19 we analyzed the stability property of a feedback control systems in terms of poles of the tracking transfer function. I will now show how to analyse the stability of a control system from the frequency response of the loop transfer function[2], $L(j\omega)$.

There is an algebraic stability analysis method named Routh's stability criterion which is based on the coefficients of the characteristic polynomial of the control system. I have decided to not present this method since I think it has quite limited practical importance, and the mathematical operations become quite complicated except for simple models. A reference to Routh's stability criterion is e.g. Seborg et al. (2004).

[2] the product of all the transfer functions in the control loop, cf. (12.1)

22.3.2 Nyquist's stability criterion

Let us start with a quick review from Section 20: The stability of a feedback control system is determined by the placement of the roots of the characteristic polynomial, $a(s)$, in the complex plane. $a(s)$ is:

$$a(s) = d_l(s) + n_l(s) \tag{22.25}$$

where $d_l(s)$ is the denominator polynomial, and $n_l(s)$ is the numerator polynomial of the loop transfer function:

$$H_l(s) = \frac{d_l(s)}{n_l(s)} \tag{22.26}$$

To continue with deriving the Nyquist's stability criterion, we start with a rewriting. The roots of (22.25) are the same as the roots of:

$$\frac{d_l(s) + n_l(s)}{d_l(s)} = 1 + \frac{n_l(s)}{d_l(s)} = 1 + H_l(s) = 0 \tag{22.27}$$

Therefore, we can denote also (22.27) as the characteristic equation of the feedback control system. (22.27) is the equation from which the Nyquist's stability criterion will be derived. In the derivation we will use the so-called Argument variation principle:

Argument variation principle: Given a function $f(s)$ where s is a complex number. Then $f(s)$ is a complex number, too. As with all complex numbers, $f(s)$ has an angle or argument. If s follows a closed contour Γ (gamma) in the complex s-plane which encircles a number of poles and a number of zeros of $f(s)$, see Figure 22.5,

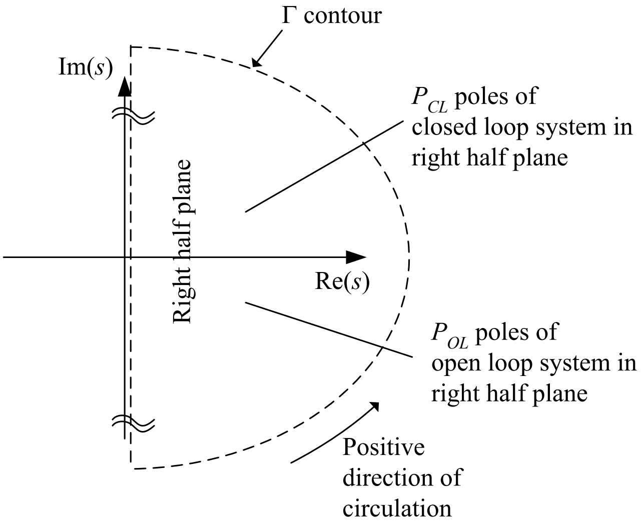

Γ contour
Im(s)
P_CL poles of closed loop system in right half plane
Right half plane
Re(s)
P_OL poles of open loop system in right half plane
Positive direction of circulation

Figure 22.5: s shall follow the Γ contour once in positive direction (counter clockwise).

then the following applies:

$$\arg_{\Gamma} f(s) = 360^\circ \cdot (\text{number of zeros minus number of poles of } f(s) \text{ inside } \Gamma) \tag{22.28}$$

where $\arg_\Gamma f(s)$ means the change of the angle of $f(s)$ when s has followed Γ once in positive direction of circulation (i.e. clockwise).

For our purpose, let the function $f(s)$ in the Argument variation principle be

$$f(s) = 1 + H_l(s) \tag{22.29}$$

The Γ contour must encircle the entire right half s-plane, so that we are certain that all poles and zeros of $1 + H_l(s)$ are encircled. From the Argument Variation Principle we have:

$$\arg_{\Gamma}[1 + L(s)] = \arg_{\Gamma} \frac{d_l(s) + n_l(s)}{d_l(s)} \tag{22.30}$$

$$= 360^\circ \cdot (\text{number of roots of } (d_l + n_l) \text{ in RHP}$$
$$\text{minus number roots of } d_l \text{ in RHP}) \tag{22.31}$$
$$= 360^\circ \cdot (\text{number poles of closed loop system in RHP}$$
$$\text{minus number poles of open system in RHP})$$
$$= 360^\circ \cdot (P_{\text{CL}} - P_{\text{OL}}) \tag{22.32}$$

where RHP means right half plane. By “open system” we mean the (imaginary) system having transfer function $H_l(s) = n_l(s)/d_l(s)$, i.e., the original feedback system with the feedback broken. The poles of the open system are the roots of $d_l(s) = 0$.

Finally, we can formulate the Nyquist’s stability criterion. But before we do that, we should remind ourselves what we are after, namely to be able to determine the number poles P_{CL} of the closed loop system in RHP. It those poles which determines whether the closed loop system (the control system) is asymptotically stable or not. *If $P_{\text{CL}} = 0$ the closed loop system is asymptotically stable.*

Nyquist’s stability criterion: Let P_{OL} be the number of poles of the open system in the right half plane, and let $\arg_{\Gamma} H_l(s)$ be the angular change of the vector $H_l(s)$ as s have followed the Γ contour once in positive direction of circulation. Then, the number poles P_{CL} of the closed loop system in the right half plane, is

$$P_{\text{CL}} = \frac{\arg_{\Gamma} H_l(s)}{360^\circ} + P_{\text{OL}} \tag{22.33}$$

If $P_{\text{CL}} = 0$, the closed loop system is asymptotically stable.

Let us take a closer look at the terms on the right side of (22.33): P_{OL} are the roots of $d_l(s)$, and there should not be any problem calculating these roots. To determine the angular change of the vector $1 + H_l(s)$. Figure 22.6 shows how the vector (or complex number) $1 + H_l(s)$ appears in a *Nyquist diagram* for a typical plot of $H_l(s)$. A Nyquist diagram is simply a Cartesian diagram of the complex plane in which H_l is plotted. $1 + H_l(s)$ is the vector from the point $(-1,\ 0j)$, which is denoted the *critical point*, to the Nyquist curve of $H_l(s)$.

More about the Nyquist curve of $H_l(j\omega)$ Let us take a more detailed look at the Nyquist curve of H_l as s follows the Γ contour in the s-plane, see Figure 22.5. In practice, the denominator polynomial of $H_l(s)$ has higher order than the numerator polynomial. This implies that $H_l(s)$ is mapped to the origin of the Nyquist diagram when $|s| = \infty$. Thus, the whole semicircular part of the Γ contour is mapped to the origin.

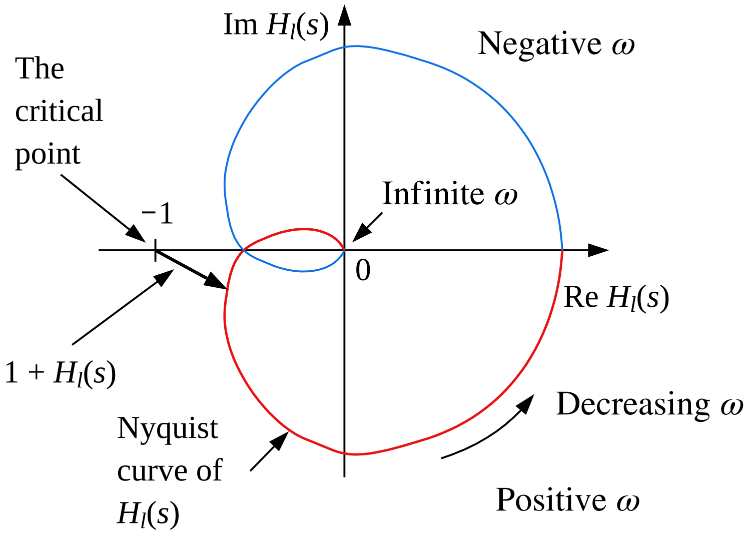


Figure 22.6: Typical Nyquist curve of $H_l(s)$. The vector $1 + H_l(s)$ is drawn.

The imaginary axis constitutes the rest of the Γ contour. How is the mapping of $H_l(s)$ as s runs along the imaginary axis? On the imaginary axis $s = j\omega$, which implies that $H_l(s) = H_l(j\omega)$, which *is the frequency response of* $H_l(s)$. A consequence of this is that we can in principle determine the stability property of a feedback system by just looking at the frequency response of the open system, $H_l(j\omega)$.

ω has negative values when $s = j\omega$ is on the negative imaginary axis. For $\omega < 0$ the frequency response has a mathematical meaning. From general properties of complex functions,

$$|H_l(-j\omega)| = |H_l(j\omega)| \tag{22.34}$$

and

$$\angle H_l(-j\omega) = -\angle H_l(j\omega) \tag{22.35}$$

Therefore the Nyquist curve of $H_l(s)$ for $\omega < 0$ will be identical to the Nyquist curve of $\omega > 0$, but mirrored about the real axis. Thus, we only need to know how $H_l(j\omega)$ is mapped for $\omega \geq 0$. The rest of the Nyquist curve then comes by itself! Actually we need not draw more of the Nyquist curve (for $\omega > 0$) than what is sufficient for determining if the critical point is encircled or not.

We must do some extra considerations if some of the poles in $H_l(s)$, which are the poles of the open loop system, lie in the origin. This corresponds to pure integrators in control loop, which is a common situation in feedback control systems because the controller usually has integral action, as in a PI or PID controller. If $H_l(s)$ contains integrators, the Γ contour must go *outside* the origo. But to the left or to the right? We choose to the right, see Figure 22.7.

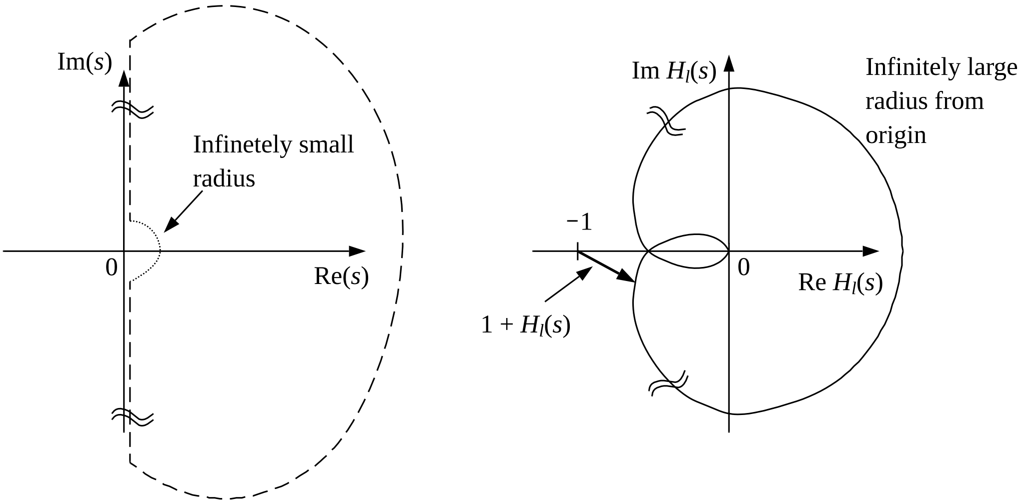


Figure 22.7: Left diagram: If $H_l(s)$ has a pole in origin, the Γ contour must pass the origin along an arbitrarily small semicircle to the right. Right diagram: A typical Nyquist curve of H_l.

(We have thereby decided that the origin belongs to the left half plane. This implies that P_{OL} does not count these poles.) The radius of the semicircle around origin is arbitrarily small. The Nyquist curve then becomes as shown in the diagram to the right in the same figure. The arbitrarily small semicircle in the s-plane is mapped to an infinitely large semicircle in the L-plane. The is because as $s \to 0$, the loop transfer function is approximately

$$H_l(s) \approx \frac{K}{s}$$

(if we assume one pole in the origin). On the small semicircle,

$$s = re^{j\theta} \tag{22.36}$$

which gives

$$H_l(s) \approx \frac{K}{r}e^{-j\theta} \tag{22.37}$$

When $r \to 0$ and when simultaneously θ goes from $+90°$ via $0°$ to $-90°$, the Nyquist curve becomes an infinitely large semicircle, as shown.

The Nyquist's stability criterion for non-rational transfer functions The Nyquist's stability criterion gives information about the poles of feedback systems. So far it has been assumed that the loop transfer function $H_l(s)$ is a rational transfer function. What if $H_l(s)$ is irrational? Here is one example:

$$H_l(s) = \frac{1}{s}e^{-t_d s} \tag{22.38}$$

where $e^{-t_d s}$ represents time delay. In such cases the tracking ratio $H_t(s)$ will also be irrational, and the definition of poles does not apply to such irrational transfer functions.

Actually, the Nyquist's stability criterion can be used as a graphical method for determining the stability property on basis of the frequency response $H_l(j\omega)$.

Nyquist's special stability criterion In most cases the open system is stable, that is, $P_{\rm OL} = 0$. (22.33) then becomes

$$P_{\rm CL} = \frac{\arg_\Gamma[L(s)]}{360^\circ} \tag{22.39}$$

This implies that the feedback system is asymptotically stable if the Nyquist curve does not encircle the critical point. This is the *Nyquist's special stability criterion* or the *Nyquist's stability criterion for open stable systems.*

The Nyquist's special stability criterion can also be formulated as follows: *The feedback system is asymptotically stable if the Nyquist curve of H_l has the critical point on its left side for increasing ω.*

Another way to formulate Nyquist's special stability criterion involves the *amplitude crossover frequency* ω_c and the *phase crossover frequency* ω_{180}. ω_c is the frequency at which the $H_l(j\omega)$ curve crosses the unit circle, while ω_{180} is the frequency at which the $H_l(j\omega)$ curve crosses the negative real axis. In other words:

$$|H_l(j\omega_c)| = 1 \tag{22.40}$$

and

$$\arg H_l(j\omega_{180}) = -180^\circ \tag{22.41}$$

See Figure 22.8. Note: The Nyquist diagram contains no explicit frequency axis.

We can now determine the stability properties from the relation between these two crossover frequencies:

- Asymptotically stable closed loop system: $\omega_c < \omega_{180}$
- Marginally stable closed loop system: $\omega_c = \omega_{180}$
- Unstable closed loop system: $\omega_c > \omega_{180}$

The frequency of the sustained oscillations There are sustained oscillations in a marginally stable system. *The frequency of these oscillations is $\omega_c = \omega_{180}$.*This can be explained as follows: In a marginally stable system, $H_l(\pm j\omega_{180}) = H_l(\pm j\omega_c) = -1$. Therefore, $d_l(\pm j\omega_{180}) + n_l(\pm j\omega_{180}) = 0$, which is the characteristic equation of the closed loop system with $\pm j\omega_{180}$ inserted for s. Therefore, the system has $\pm j\omega_{180}$ among its poles. The system usually have additional poles, but they lie in the left half plane. The poles $\pm j\omega_{180}$ leads to sustained sinusoidal oscillations. Thus, ω_{180} (or ω_c) is the frequency of the sustained oscillations in a marginally stable system.

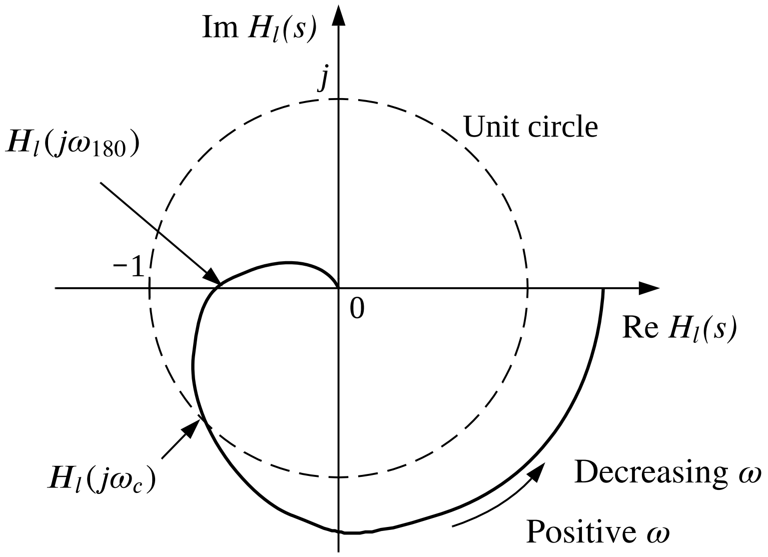


Figure 22.8: Definition of amplitude crossover frequency ω_c and phase crossover frequency ω_{180}.

22.3.3 Stability margins

22.3.3.1 Stability margins in terms of gain margin and phase margin

An asymptotically stable feedback system may become marginally stable if the loop transfer function changes. The *gain margin* GM and the *phase margin* PM [radians or degrees] are *stability margins* which in their own ways expresses how large parameter changes can be tolerated before an asymptotically stable system becomes marginally stable. Figure 22.9 shows the stability margins defined in the Nyquist diagram.

GM is the (multiplicative, not additive) increase of the gain that H_l can tolerate at ω_{180} before the H_l curve (in the Nyquist diagram) passes through the critical point. Thus,

$$|H_l(j\omega_{180})| \cdot \text{GM} = 1 \tag{22.42}$$

which gives

$$\text{GM} = \frac{1}{|H_l(j\omega_{180})|} = \frac{1}{|\text{Re}H_l(j\omega_{180})|} \tag{22.43}$$

(The latter expression in (22.43) is because at ω_{180}, $\text{Im}H_l = 0$ so that the amplitude is equal to the absolute value of the real part.)

If we use decibel as the unit (like in the Bode diagram which we will soon encounter), then

$$\text{GM [dB]} = -\,|H_l(j\omega_{180})|\ \text{[dB]} \tag{22.44}$$

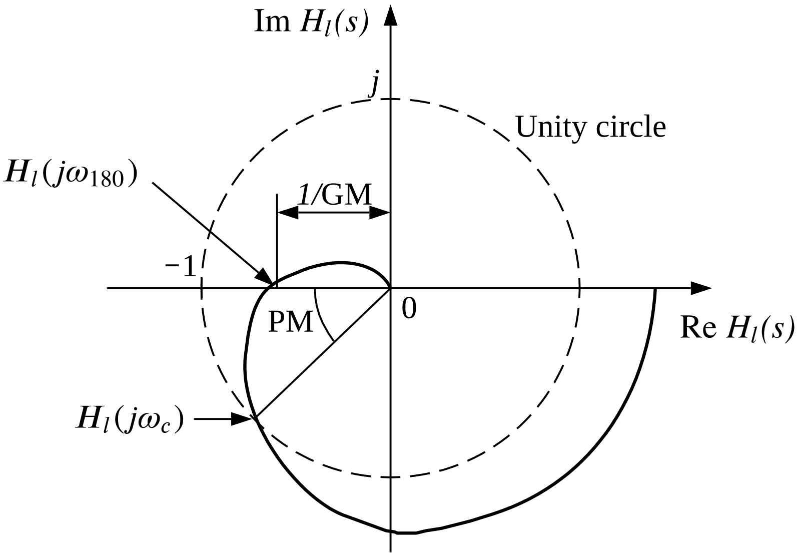


Figure 22.9: Gain margin GM and phase margin PM defined in the Nyquist diagram.

The phase margin PM is the phase reduction that the H_l curve can tolerate at ω_c before the H_l curve passes through the critical point. Thus,

$$\arg H_l(j\omega_c) - \text{PM} = -180^\circ \tag{22.45}$$

which gives

$$\text{PM} = 180^\circ + \arg H_l(j\omega_c) \tag{22.46}$$

We can now state as follows: The feedback (closed) system is asymptotically stable if

$$\text{GM} > 0\text{dB} \ = 1 \text{ and PM} > 0^\circ \tag{22.47}$$

This criterion is often denoted the *Bode-Nyquist stability criterion.*

Reasonable ranges of the stability margins are

$$2 \approx 6\text{dB} \leq \text{GM} \leq 4 \approx 12\text{dB} \tag{22.48}$$

and

$$30^\circ \leq \text{PM} \leq 60^\circ \tag{22.49}$$

The larger values, the better stability, but at the same time the system becomes more sluggish, dynamically. If you are to use the stability margins as design criterias, you can use the following values (unless you have reasons for specifying other values):

$$\text{GM} \geq 2.5 \approx 8\text{dB and PM} \geq 45^\circ \tag{22.50}$$

For example, the controller gain, K_c, can be adjusted until one of the inequalities becomes an equality.[3]

[3] But you should definitely check the behaviour of the control system by simulation, if possible.

It can be shown[4] that for PM $\leq 70°$, the damping of the feedback system approximately corresponds to that of a second order system with relative damping factor

$$\zeta \approx \frac{\text{PM}}{100°} \tag{22.51}$$

For example, PM $= 50° \sim \zeta = 0.5$.

22.3.3.2 Stability margins in terms of maximum sensitivity amplitude

An alternative quantity of a stability margin is the minimum distance from the $H_l(j\omega)$ curve to the critical point. This distance is $|1 + H_l(j\omega)|$, see Figure 22.10.

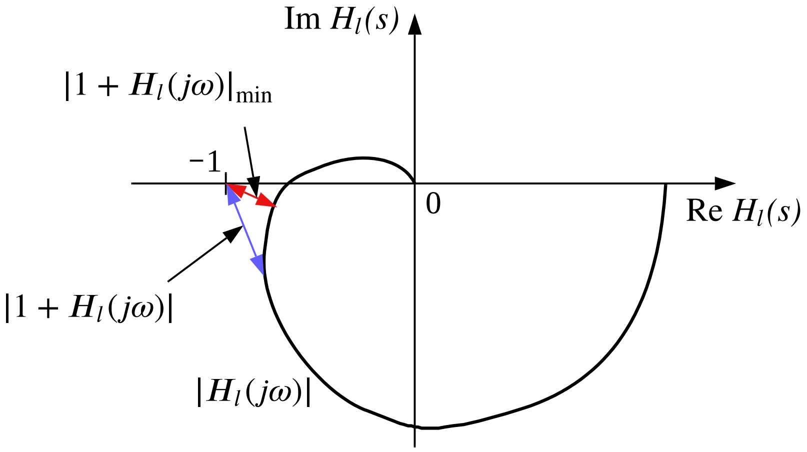


Figure 22.10: The distance between the $H_l(j\omega)$ curve and the critical point is $|1 + H_l|$. The minimum of this distance is related to the stability margin.

We can use the minimal value of $|1 + H_l(j\omega)|$ as a stability margin. However, it is more common to take the inverse of the distance: Thus, a stability margin is the *maximum* value of $1/|1 + H_l(j\omega)|$. And since $1/[1 + H_l(s)]$ is the sensitivity transfer function $H_s(s)$, then $|H_s(j\omega)|_{\text{max}}$ represents a stability margin. Reasonable values are in the range

$$1.5 \approx 3.5 \text{ dB} \leq |H_s(j\omega)|_{\text{max}} \leq 3.0 \approx 9.5 \text{ dB} \tag{22.52}$$

If you use $|H_s(j\omega)|_{\text{max}}$ as a criterion for adjusting controller parameters, you can use the following criterion (unless you have reasons for some other specification):

$$|H_s(j\omega)|_{\text{max}} = 2.0 \approx 6 \text{ dB} \tag{22.53}$$

[4] The result is based on the assumption that the loop transfer function is $L(s) = \omega_0^2 / [(s + 2\zeta\omega_0)s]$ which gives tracking transfer function $T(s) = L(s)/[1 + L(s)] = \omega_0^2 / \left[s^2 + 2\zeta\omega_0 s + \omega_0^2\right]$. The phase margin PM can be calculated from $L(s)$.

22.3.4 Stability analysis in a Bode diagram

It is most common to use a Bode diagram for frequency response based stability analysis of closed loop systems. The Nyquist's Stability Criterion says: The closed loop system is marginally stable if the Nyquist curve (of L) goes through the critical point, which is the point $(-1, 0)$. But where is the critical point in the Bode diagram? The critical point has phase (angle) $-180°$ and amplitude $1 = 0$dB. The critical point therefore constitutes two lines in a Bode diagram: The 0dB line in the amplitude diagram and the $-180°$ line in the phase diagram. Figure 22.11 shows typical L curves for an asymptotically stable closed loop system. In the figure, GM, PM, ω_c and ω_{180} are indicated.

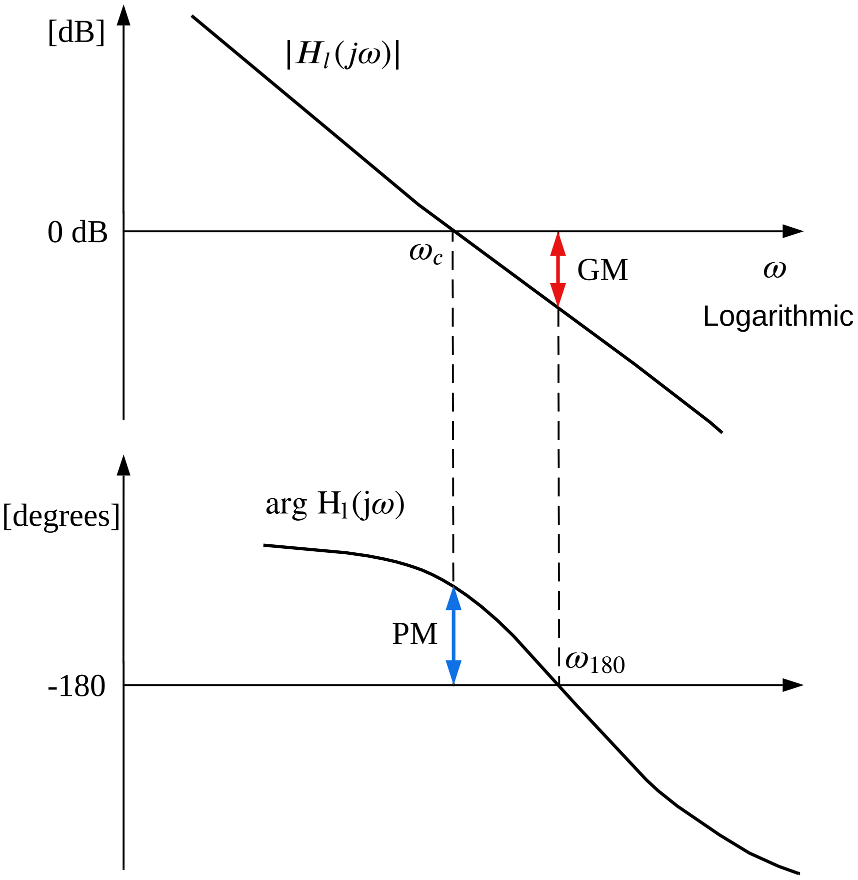


Figure 22.11: Typical H_l curves of an asymptotically stable closed loop system with GM, PM, ω_c and ω_{180} indicated.

Example 22.2 *Stability analysis of level control system of wood chips tank*

In this example, we will analyse the stability of the by now hopefully well-known level control system of the wood chips tank.

The example has the same process model and the same controller with the same PI settings as in Example 22.1. However, for your convenience, I repeat here the relevant transfer models and the PI controller settings:

Process transfer function (from control signal to level):

$$\frac{H^*(s)}{U(s)} = H_p(s) = \frac{1}{\rho A s} e^{-t_d s} \tag{22.54}$$

Transfer function $H_c(s)$ of the PI controller:

$$H_c(s) = K_c + \frac{K_c}{T_i s} \tag{22.55}$$

PI settings (according to the SIMC (Skogestad 2003) PI tuning method, cf. Example 14.6):

$$K_c = 3.89 \text{ [\%/m]} \tag{22.56}$$

$$T_i = 1000 \text{ s} \tag{22.57}$$

The stability analysis is done with the following Python program:

http://techteach.no/control/python/freq_resp_stab_analysis_level_control_woodchips_tank.py

The program implements the following:

The following Python program implements the following, using functions of the Python Control package:

- Creates $H_c(s)$, $H_p(s)$, $H_l(s)$ and $H_t(s)$.
- Calculates and plots $|H_l(j\omega)|$ in a Bode diagram, see Figure 22.3. GM, PM, ω_c, and ω_{180} are automatically shown in the figure, but with limited accuracy. More accurate values should be calculated, see items below.
- Calculates ω_c and ω_{180}, see below, and Figure 22.3.
- Calculates GM and PM, see below, and Figure 22.3.
- Simulates the response in the level $h(t)$ due to a step change of the reference $r(t)$, see Figure 22.13. The purpose of this simulation is to illustrate the stability property of the control system in the time domain (GM and PM express stability properties in the frequency domain).

Results of the program:

```
GM [1 (not dB)] = 2.74e+00
PM [deg] = 3.41e+01
w180 [rad/s] = 2.20e-03
wc [rad/s] = 5.57e-03
```

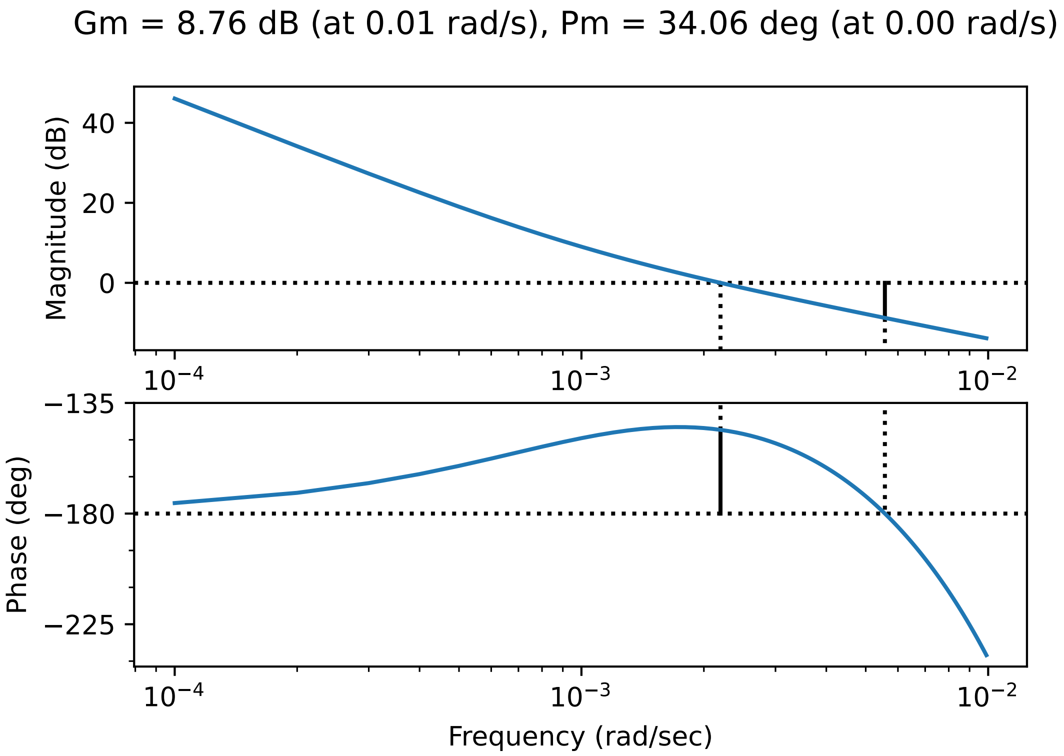


Figure 22.12: The loop transfer function $|H_l(j\omega)|$ plotted in a Bode diagram.

Note that these values may me a little inaccurate. Why? Because we have approximated the time delay with a Padé approximation.

Are the stability margins within acceptable limits? Acceptable intervals for GM and PM are given in Seborg et al. (2004) as15.2

$$1.7 \leqslant \text{GM} \leqslant 4.0 \tag{22.58}$$

$$30^\circ \leqslant \text{PM} \leqslant 45^\circ \tag{22.59}$$

Yes, both GM and PM are within their respective acceptable intervals, so we conclude that the stability is acceptable.

The simulation shown in Figure 22.13 confirms that the stability is ok.

In Example 15.2 we found GM and PM experimentally from simulations, and manual readings from simulated responses. In that example, we found GM = 2.33 and PM = 30°. These values were found from manual readings in simulated responses. They don't differ too much from the more accurate values we have found in the present example.S

[End of Example 22.2]

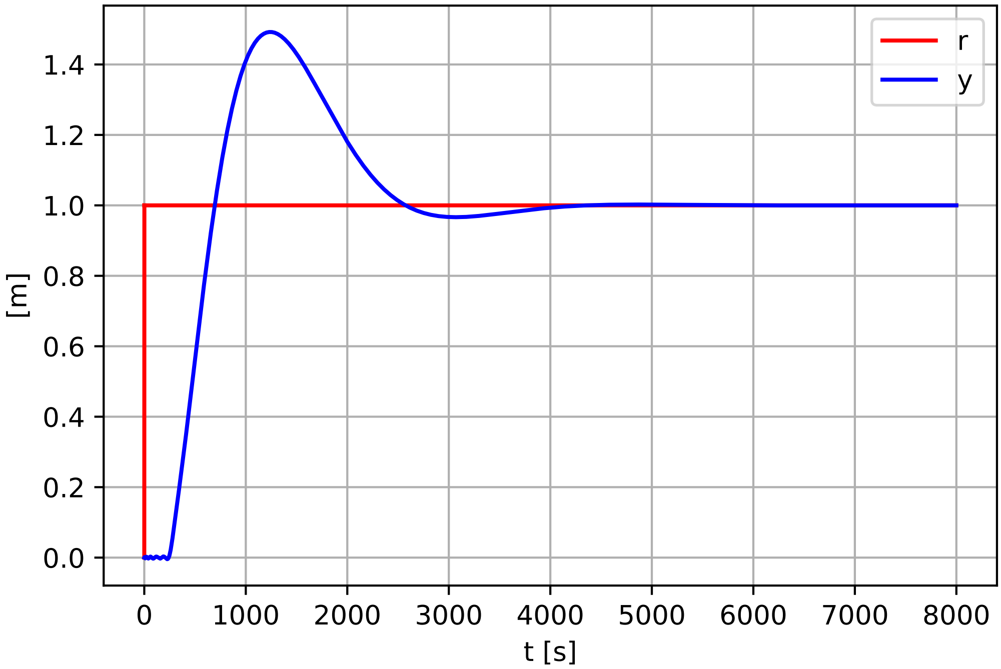


Figure 22.13: Simulated response in level $h(t)$ due to a step change of reference $r(t)$.

22.3.5 Robustness in term of stability margins

Per definition the stability margins expresses the robustness of the feedback control system against certain parameter changes in the loop transfer function:

- *The gain margin,* GM, is how much the loop gain, K, can increase before the system becomes unstable. For example, is GM $= 2$ when $K = 1.5$, the control system becomes unstable for K larger than $1.5 \cdot 2 = 3.0$.
- *The phase margin,* PM, is how much the phase lag function of the loop can be reduced before the loop becomes unstable. One reason of reduced phase is that the time delay in control loop is increased. A change of the time delay by Δt_d introduces the factor $e^{-\Delta t_d s}$ in $H_l(s)$ and contributes to $\arg H_l$ with $-\Delta t_d \cdot \omega$ [rad] or $-\Delta t_d \cdot \omega \frac{180^\circ}{\pi}$ [deg]. $|H_l|$ is however not influenced because the amplitude function of $e^{-t_d s}$ is 1, independent of the value of t_d. The system becomes unstable if the time delay have increased by $\Delta t_{d\max}$ such that[5]

$$\text{PM} = \Delta t_{d\max} \cdot \omega_c \frac{180^\circ}{\pi} \text{ [deg]} \tag{22.60}$$

which gives the following maximum change of the time delay:

$$\Delta t_{d\max} = \frac{\text{PM}}{\omega_c} \frac{\pi}{180^\circ} \tag{22.61}$$

[5] Remember that PM is found at ω_c.

If you want to calculate how much the phase margin PM is reduced if the time delay is increased by Δt_d, you can use the following formula which stems from (22.60):

$$\Delta \text{PM} = \Delta t_d \cdot \omega_c \frac{180^\circ}{\pi} \text{ [deg]} \tag{22.62}$$

For example, assume that a given control system has $\omega_c = 0.2$ rad/min and PM $= 50^\circ$. If the time delay increases by 1 min, the phase margin is reduced by $\Delta\text{PM} = 1 \cdot 0.2 \frac{180^\circ}{\pi} = 11.4^\circ$, i.e. from 50° to 38.6°.

22.4 *Problems for Chapter 22*

Problem 22.1 *Frequence response of control system*

Figure 22.14 shows the amplitude gain curves of the loop transfer function $H_l(j\omega)$, the tracking transfer function $H_t(j\omega)$ and the sensitivity transfer function $H_s(j\omega)$ of a feedback control system.

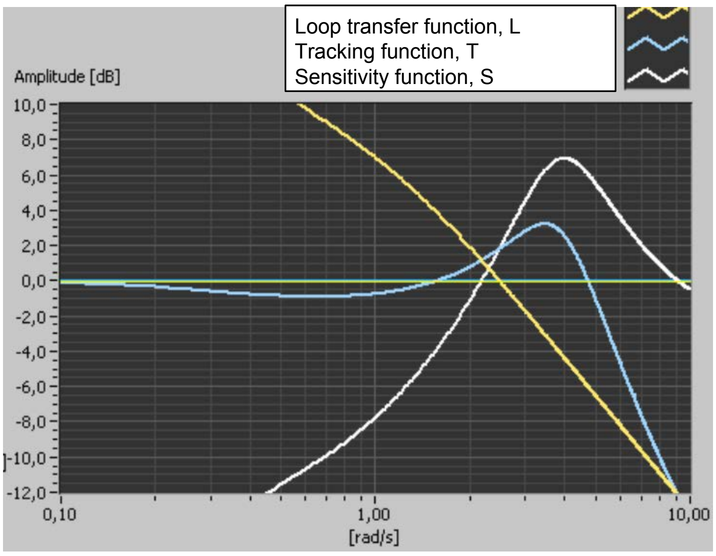


Figure 22.14: Amplitude gain curves of the loop transfer function $H_l(j\omega)$, the tracking transfer function $H_t(j\omega)$ and the sensitivity transfer function $H_s(j\omega)$ of a feedback control system.

1. Read off from the frequency response curves the following three alternative bandwidths:
   - The crossover frequency ω_c.
   - The -3 dB frequency ω_t of the tracking transfer function
   - The -11dB frequency ω_s of the sensitivity transfer function
2. Assume that the reference is a sinusoid of amplitude $A_r = 4$ and frequency 1 rad/s. What is the amplitude, A_y, of the steady state sinusoidal process output variable? What is the amplitude, A_e, of the steady state sinusoidal control error?
3. Assume that the process disturbance is a sinusoid of frequency 1 rad/s. Assume that this disturbance creates a steady state sinusoidal response in the process output variable of amplitude $A_{y_{\text{OL}}}0.5$ when the process is controlled with a constant control system, i.e. in open loop control. What is the amplitude, $A_{y_{\text{CL}}}$, of the process output variable using feedback control, i.e. in closed loop control?
4. Estimate the response time, t_r, of the response on the process output variable due to a step change of the reference.

Problem 22.2 *Frequency response of temperature control system*

1. The diagram to the left of Figure 22.15 shows a non-controlled thermal process which is a liquid tank with throughput and heating.

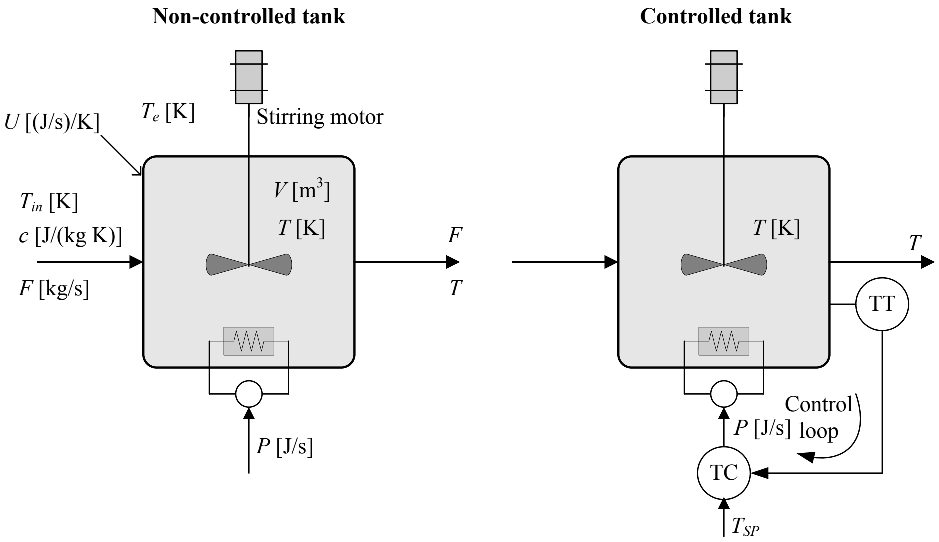


Figure 22.15: To the left: A non-controlled thermal process which is a liquid tank with throughput and heating.

Assume that the amplitude gain of the frequency response of the transfer function from inlet temperature T_{in} to outlet temperature T is as shown in the Bode diagram in Figure 22.16.

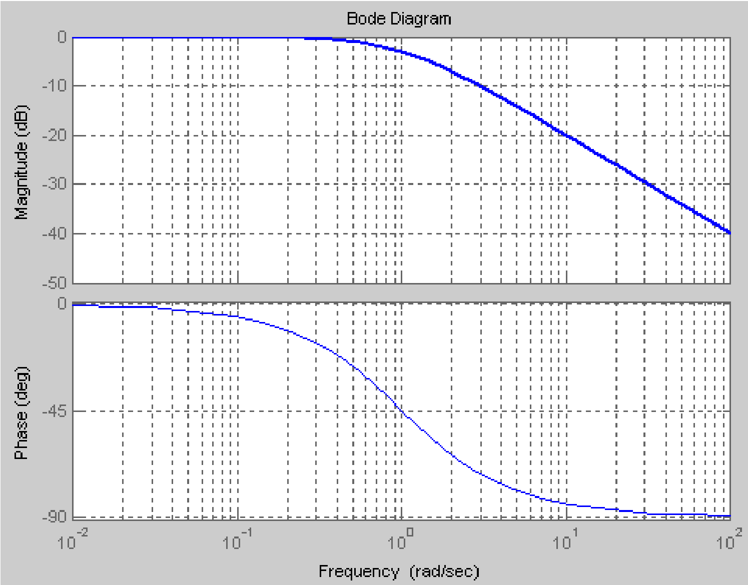


Figure 22.16: Frequency response of open-loop system from T_{in} to T.

Assume that T_{in} contains a frequency component of amplitude $A_{T_{\text{in}}}$ of frequency 0.1 rad/s. Calculate the amplitude A_T of the corresponding steady state response in T.

2. The diagram to the right of Figure 22.15 shows a temperature control system of the process. Assume that the amplitude gain of the frequency response of the transfer function from inlet temperature T_{in} to outlet temperature T of the control system is as shown in the Bode diagram in Figure 22.17.

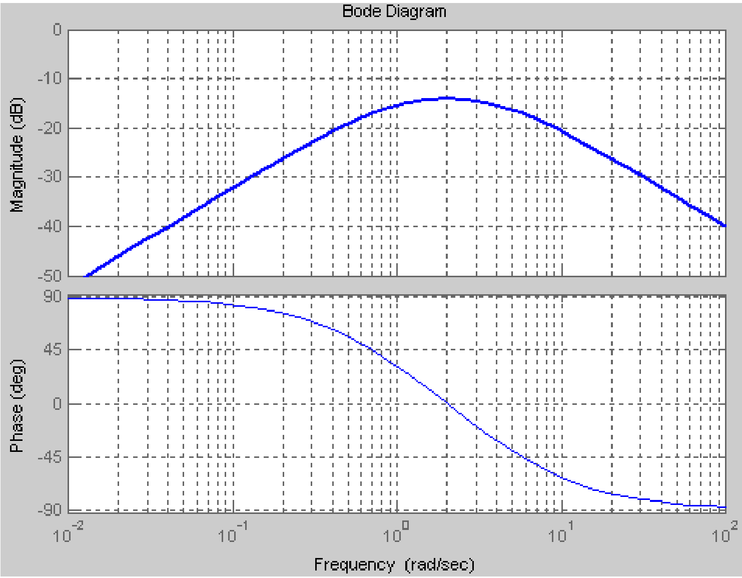


Figure 22.17: Frequency response of closed loop (feedback control) system from T_{in} to T.

Assume that that T_{in} contains a frequency component of amplitude $A_{T_{\text{in}}}$ of frequency 0.1 rad/s. What is the amplitude A_T of the corresponding steady state response in T? Compare the answer with problem 1 above. Is there any improvement by using control?

Problem 22.3 *Stablity analysis for various gains in Nyquist diagram*

Given a control system with loop transfer function

$$H_l(s) = \frac{K_c}{(s+1)^3 s} \tag{22.63}$$

Figure 22.18 shows the Nyquist curve of H_l with $K_c = 0.4$.

(The curve actually encircles the whole right half plane.)

Use Nyquist's stability criterion to calculate the values of K_c that makes the control system become

- Asymptotically stable.
- Marginally stable.
- Unstable. In this case, what is the number of poles in the right half plane?

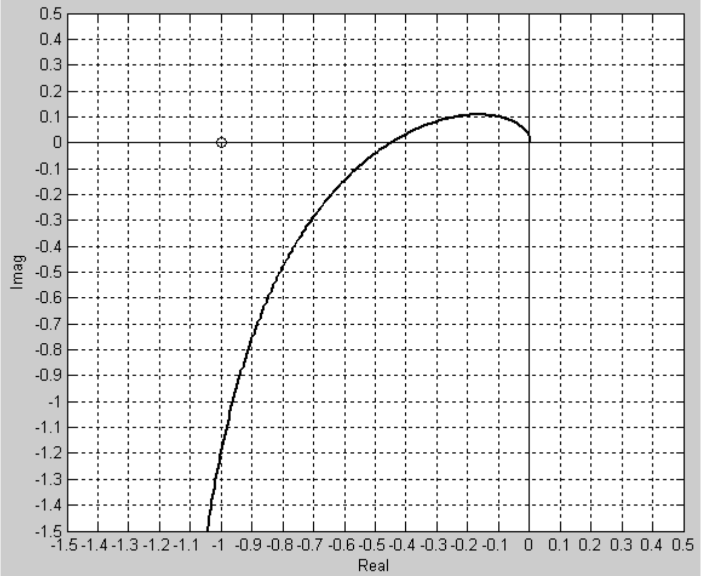


Figure 22.18: Nyquist curve of H_l with $K_c = 0.4$.

Problem 22.4 *Designing a stable control system in Nyquist diagram*

Given a closed loop system having the following loop transfer function:

$$H_l(s) = \frac{K}{s-1} \tag{22.64}$$

1. Show that the corresponding open loop system is unstable by calculating the pole of the system.

2. Figure 22.19 shows the Nyquist curve of $H_l(j\omega)$ with $K = 2$. Find using the Nyquist stability criterion for which values of K the closed loop system is asymptotically stable. Confirm the answer by calculating the pole of the closed loop system.

Problem 22.5 *Nyquist curve for conditionally stable system*

Figure 22.20 shows the Nyquist curve of $H_l(j\omega)$ of a feedback system which is open stable.

The loop gain is then $K = 1$. For which values of K is the feedback system asymptotically stable?

Problem 22.6 *Gain margin and sensitivity from Nyquist diagram*

Figure 22.21 shows the Nyquist curve of the loop transfer function $H_l(j\omega)$ of an asymptotically stable control system.

What is the gain margin GM and the maximum sensitivity gain $|H_s(j\omega)|_{\max}$?

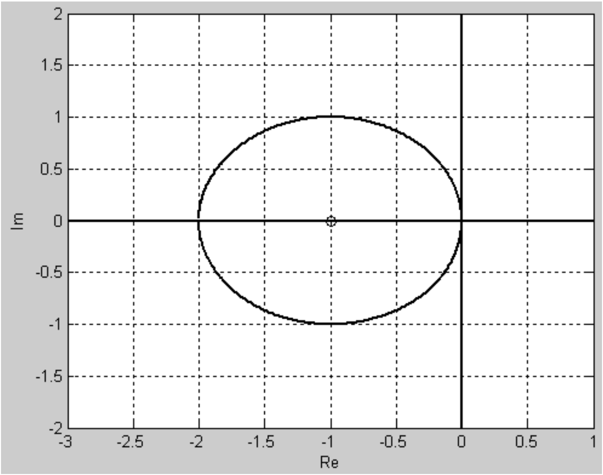


Figure 22.19: Nyquist curve of $H_l(j\omega)$ with $K = 2$.

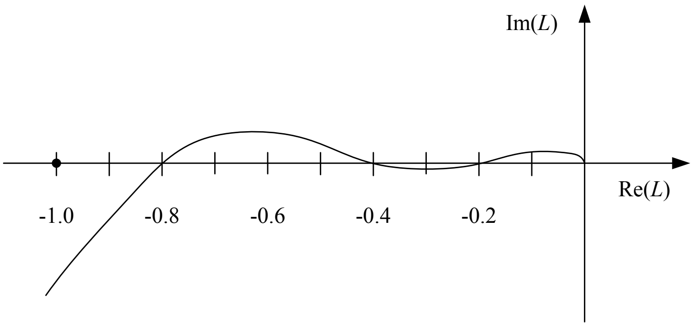


Figure 22.20: Nyquist curve of $H_l(j\omega)$.

Problem 22.7 *Stability margin in terms of sensitivity*

Figure 22.22 shows the amplitude gain curves of the loop transfer function $H_l(j\omega)$, the tracking transfer function $H_{lt}(j\omega)$ and the sensitivity transfer function $H_t(j\omega)$ of a feedback control system.

Determine the stability margin in terms of $|H_s(j\omega)|_{\max}$. Is the value in the range of reasonable values of this stability margin?

Problem 22.8 *Stability margins in Bode diagram*

Figure 22.23 shows the Bode diagram of the loop transfer function of a given control system.

1. Read off the stability margins GM and PM, and the crossover frequencies ω_c and ω_{180} in the Bode diagram.
2. How large increase of the loop gain will bring the system to the stability limit? What is the period t_p of the steady state oscillations existing in the system when the system is at the stability limit.

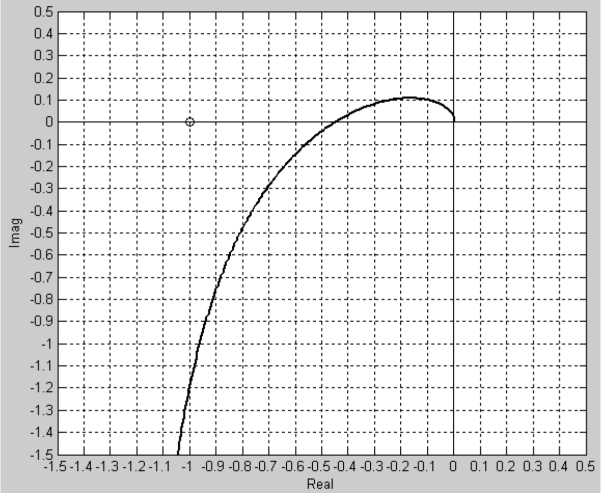


Figure 22.21: Nyquist curve of $H_l(j\omega)$.

Problem 22.9 *Marginal stability at increased time delay*

Given a feedback control system with time delay $t_d = 4.2$ min. The control system has phase margin

$$\text{PM} = 45^\circ \tag{22.65}$$

and crossover frequency

$$\omega_c = 0.2 \text{ rad/min} \tag{22.66}$$

Assume that the time delay increases, but the controller parameters are not changed. With which value of the time delay t_d is the control system marginally stable?

Problem 22.10 *When Ziegler-Nichols PID tuning method can not be used*

Ziegler-Nichols closed-loop method is based on bringing the closed loop to marginal stability with a P controller with a proper controller gain value. Explain in terms of frequency response why the Ziegler-Nichols closed-loop method can not be used for tuning a PID controller for the following processes. It is assumed that the parameters of the transfer functions have positive values.

$$H_1(s) = \frac{K}{s} \text{ (integrator)} \tag{22.67}$$

$$H_2(s) = \frac{K}{t_c s + 1} \text{ (1. order system)} \tag{22.68}$$

$$H_3(s) = \frac{K{\omega_0}^2}{s^2 + 2\zeta\omega_0 + {\omega_0}^2} \text{ (2. order system)} \tag{22.69}$$

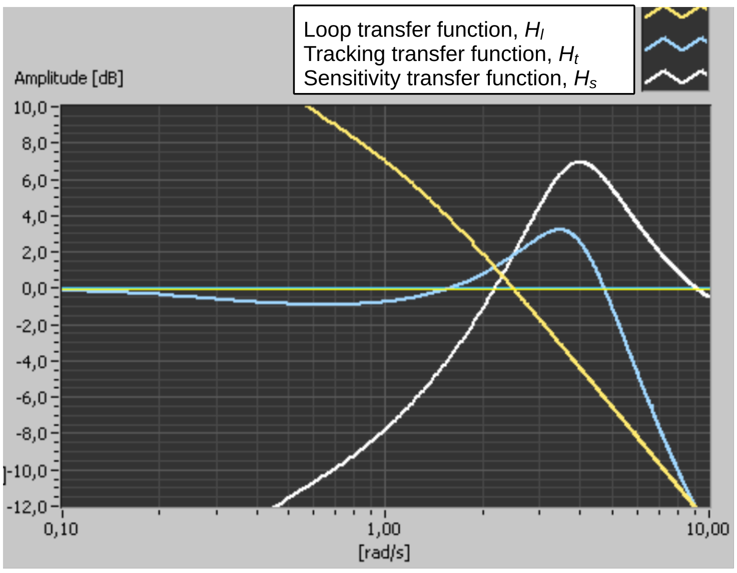


Figure 22.22: Amplitude gain curves.

22.5 *Solutions to problems for Chapter 22*

Solution to Problem 22.1

1. From Figure 22.14 we read off:

$$\omega_c = 2.5 \text{ rad/s} \tag{22.70}$$

$$\omega_t = 5.5 \text{ rad/s} \tag{22.71}$$

$$\omega_s = 0.55 \text{ rad/s} \tag{22.72}$$

(Hence, there is quite large difference between the different bandwidths in this example.)

2. The calculation from dB is made with the formula $x = 10^{x[\text{dB}]/20}$.

$$A_{y_m} = \overbrace{|H_t(j1\text{rad/s})|}^{=-0.8\text{dB}=0.91} A_r = 0.91 A_r \tag{22.73}$$

$$A_e = \overbrace{|H_s(j1\text{rad/s})|}^{=-8\text{dB}=0.40} A_r = 0.40 A_r \tag{22.74}$$

3.

$$A_{y_{\text{CL}}} = \overbrace{|H_s(j1\text{rad/s})|}^{=-8\text{dB}=0,40} A_{y_{\text{OL}}} = 0.40 \cdot 0.5 = 0.2 \tag{22.75}$$

4.

$$t_r \approx \frac{2}{\omega_t} = \frac{2}{5.5} = 0.36 \text{ s} \tag{22.76}$$

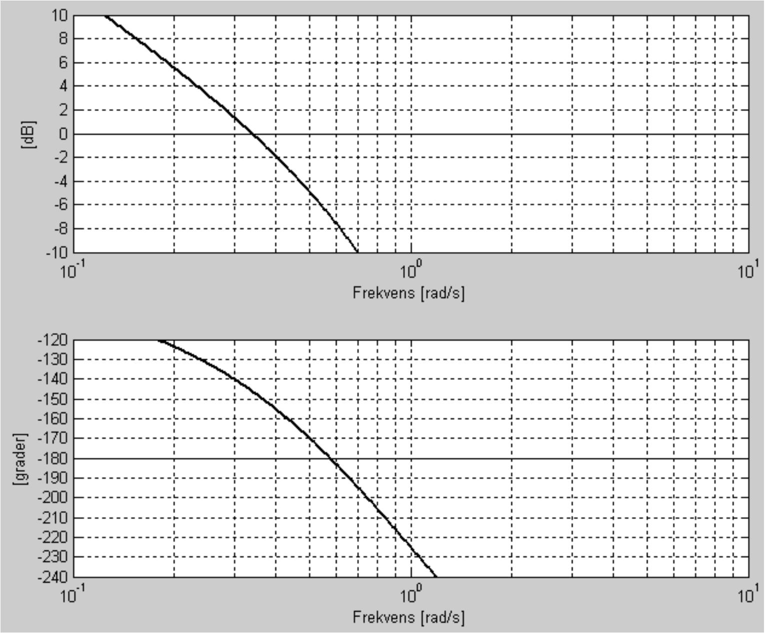


Figure 22.23: Bode diagram.

Solution to Problem 22.2

1. In Figure 22.16, which applies to the non-controlled system, we read off that the amplitude gain at frequency 0.1 rad/s is

$$G_{\text{NC}}(0.1) \approx 0 \text{ dB} = 1 \tag{22.77}$$

2. In Figure 22.17, which applies to the controlled system, we read off that the amplitude gain at frequency 0.1 rad/s is

$$G_{\text{C}}(0.1) \approx -32 \text{ dB} = 0.025 \tag{22.78}$$

Hence, with control the response in T due to T_{in} is about $\underline{\underline{\text{40 times less}}}$ than the response in the non-controlled system.

Solution to Problem 22.3

We start by determining the stability property with $K_c = 0.4$. The number of right half plane poles of the control system is

$$P_{\text{CL}} = \frac{\arg[1 + H_l(s)]}{360°} + P_{\text{OL}} \tag{22.79}$$

To determine P_{CL} we need to know $\arg H_l$ and P_{OL}. We have

$$H_l(s) = \frac{K_c}{(s+1)^3 s} \tag{22.80}$$

which does not have any poles in the right half plane (the pole in the origin belongs to the left half plane in this context). Hence,

$$P_{\mathrm{OL}} = 0 \tag{22.81}$$

$\arg H_l$ is found from the Nyquist diagram shown in Figure 22.18. $H_l(j\omega)$ does not encircle the critical point, and therefore the vector H_l has zero net change of angle, and hence $\arg H_l = 0$. So, (22.79) becomes

$$P_{\mathrm{CL}} = \frac{0}{360^\circ} + 0 = 0 \tag{22.82}$$

Consequently, the control system is asymptotically stable with $K_c = 0.4$.

From Figure 22.18 we see that the H_l curve with $K_c = 0.4$ passes through the negative real axis at -0.45. This implies that if K_c is increased by a factor of $1/0.45 = 2.22$, or in other words: if K_c is increased from 0.4 to $0.4 \cdot 2.22 = 0.89$, the H_l curve will pass through the critical point.

Consequently, the control system is marginally stable with $K_c = 0.89$.

From the results above we can conclude that the control system is asymptotically stable with (positive) $K_c < 0.89$.

If $K_c > 0.89$, the H_l curve encircles the critical point and $\arg H_l = 720^\circ$, giving

$$P_{\mathrm{CL}} = \frac{720^\circ}{360^\circ} + 0 = 2 \tag{22.83}$$

Therefore, the control system is unstable with $K_c > 0.89$, and it has two poles in the right half plane.

The control system is unstable also with $K_c < 0$. In this case $\arg H_l = 360^\circ$, and $P_{CL} = 1$, and the control system has one pole in the right half plane.

Solution to Problem 22.4

1. The open loop system is unstable because $H_l(s)$ has a pole in the right half plane. (The pole is $p = 1$.)

2. K must be halved to make the H_l curve pass through the critical point:

$$K_{\text{critical}} = \frac{2}{2} = 1 \tag{22.84}$$

Since the open loop system has one pole in the right half plane, $P_{\mathrm{OL}} = 1$ in the Nyquist criterion. To make $P_{\mathrm{CL}} = 0$ (asymptotically stable closed loop system) $\arg H_l$ must be 360°, which implies that the critical point must be encircled once. This is achieved with

$$K > K_{\text{critical}} = 1 \tag{22.85}$$

The pole of the closed loop is

$$p = 1 - K \tag{22.86}$$

This pole is in the left half plane with $K > 1$, which confirms the result of the Nyquist stability criterion above.

Solution to Problem 22.5

The Nyquist curve passes through the critical point with the following values of K:

$$K = 1/0.8 = 1.25 \tag{22.87}$$

$$K = 1/0.4 = 2.5 \tag{22.88}$$

$$K = 1/0.2 = 5 \tag{22.89}$$

The control system is asymptotically stable with

$$0 < K < 1.25 \ \text{ and } \ 2.5 < K < 5 \tag{22.90}$$

Solution to Problem 22.6

See Figure 22.24.

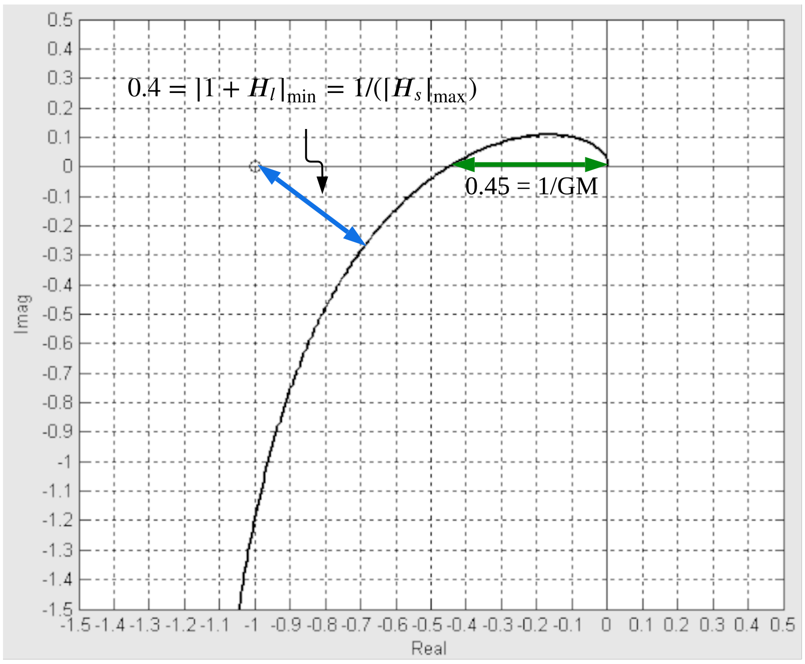


Figure 22.24: Nyquist curve.

As indicated in the figure,

$$0.4 = |1 + H_l|_{\text{min}} = \frac{1}{|H_s|_{\text{max}}} \tag{22.91}$$

which gives

$$|H_s|_{\text{max}} = \frac{1}{0.4} = 2.5 = 8.0 \text{ dB} \tag{22.92}$$

Furthermore,

$$\frac{1}{\text{GM}} = 0.45 \tag{22.93}$$

which gives

$$\text{GM} = \frac{1}{0.45} = 2.22 = 6.9 \text{ dB} \tag{22.94}$$

Solution to Problem 22.7

From Figure 22.22 we read off

$$|H_s(j\omega)|_{\max} = 7 \text{ dB} \tag{22.95}$$

which is in the "reasonable" range of

$$3.5 \text{ dB} \le |H_s(j\omega)|_{\max} \le 9.5 \text{ dB} \tag{22.96}$$

Solution to Problem 22.8

1. See Figure 22.25.

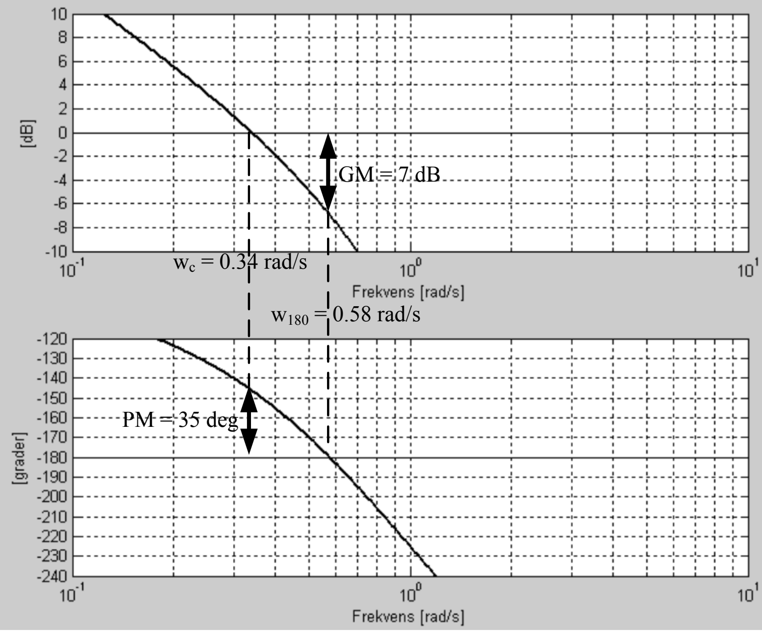


Figure 22.25: Bode plot.

From the Bode diagram:

$$\text{GM} = 7 \text{ dB} = 2.2 \tag{22.97}$$

$$\text{PM} = 35^\circ \tag{22.98}$$

$$\omega_c = 0.34 \text{ rad/s} \tag{22.99}$$

$$\omega_{180} = 0.58 \text{ rad/s} \tag{22.100}$$

2. An increase of the loop gain by a factor of

$$\text{GM} = 2.2 \tag{22.101}$$

will bring the system to the stability limit.

The period is

$$t_p = \frac{2\pi}{\omega_{180}} = \frac{2\pi}{0.58} = 10.8 \text{ s} \tag{22.102}$$

Solution to Problem 22.9

When the control system is marginally stable, the phase margin PM is zero. The maximum change of the time delay that is allowed before the system is marginally stable is

$$\Delta\tau = \frac{\text{PM}}{\omega_c} \cdot \frac{\pi}{180^\circ} = \frac{45^\circ}{0.2} \cdot \frac{\pi}{180^\circ} = 3.9 \text{ min} \tag{22.103}$$

Since the time delay *before* the change is 4.2 min, the total value of the time delay at marginally stability is

$$t_d = 4.2 + 3.9 = 8.1 \text{ min} \tag{22.104}$$

Solution to Problem 22.10

For $H_1(s)$ the loop transfer function with a P controller is

$$H_{l1}(s) = H_p(s)H_c(s) = H_1(s)K_c = \frac{K}{s}K_c \tag{22.105}$$

The phase (angle) of $H_{l1}(j\omega)$ converges to -90° as the frequency goes to infinity. Therefore, the closed loop system will not become marginally stable with any controller gain, and hence, the Ziegler-Nichols method can not be used.

For $H_2(s)$ the loop transfer function with a P controller is

$$H_{l2}(s) = H_2(s)H_c(s) = H_2(s)K_c = \frac{K}{t_c s + 1}K_c \tag{22.106}$$

The phase (angle) of $L_2(j\omega)$ converges to -90° as the frequency goes to infinity. Therefore, the closed loop system will not become marginally stable with any controller gain, and hence, the Ziegler-Nichols method can not be used.

For $H_3(s)$ the loop transfer function with a P controller is

$$H_{l3}(s) = H_3(s)H_c(s) = H_3(s)K_c = \frac{K{\omega_0}^2}{s^2 + 2\zeta\omega_0 + {\omega_0}^2}K_c \tag{22.107}$$

The phase (angle) of $H_{l3}(j\omega)$ converges to -180° as the frequency goes to infinity. Therefore, the closed loop system will not become marginally stable with any limited controller gain, and hence, the Ziegler-Nichols method can not be used.

Part VI

ANALYSIS OF DISCRETE-TIME FEEDBACK SYSTEMS

Chapter 23

Discrete-time signals

Assume that an AD-converter (analog-digital) at discrete points of time converts an analog signal $y_a(t)$, which can be a voltage signal from a temperature or speed sensor, to an equivalent digital signal, $y_d(t_k)$, in the form of a number to be used in operations in the computer, see Figure 23.1.

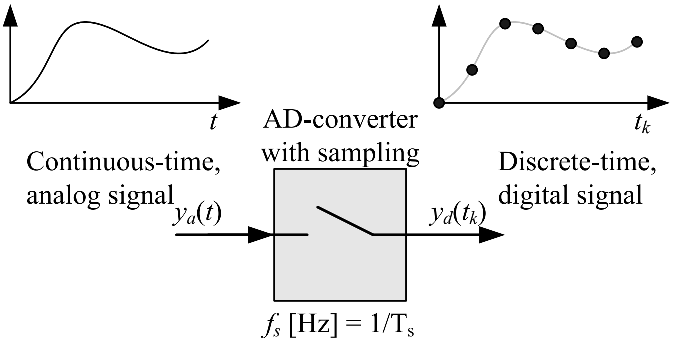


Figure 23.1: Sampling. t_s is the time step between the samplings, or the sampling interval.

(The AD-converter is a part of the interface between the computer and the external equipment, e.g. sensors.) As indicated in Figure 23.1 the resulting *discrete-time signal* is a sequence or a series of signal values defined in discrete points of time. t_s is the time step between the samplings, or the sampling interval. Figure 23.2 shows this signal in more detail.

The discrete points of time may be denoted t_k where k is an integer *time index*. The time series can be written in various ways:

$$\{x(t_k)\} = \{x(kT_s)\} = \{x(k)\} = x(0),\ x(1),\ x(2), \ldots \tag{23.1}$$

To make the notation simple, we can write the signal in one of the following ways:

$$x(t_k) \tag{23.2}$$

$$x(kt_s) \tag{23.3}$$

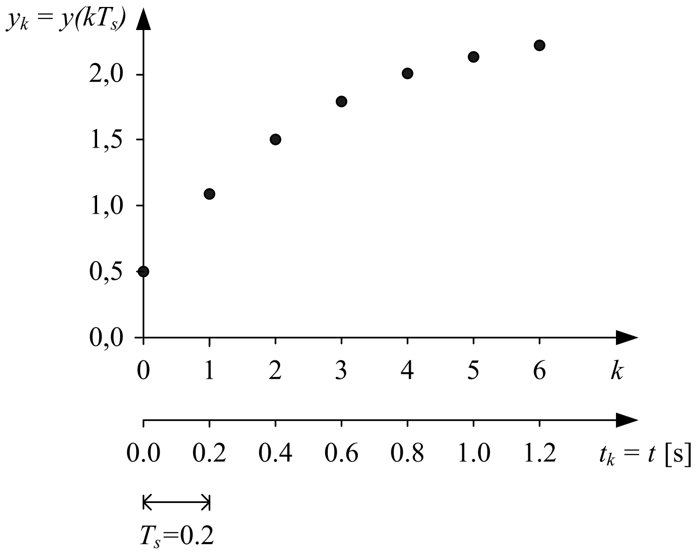


Figure 23.2: Discrete-time signal.

$$x(k) \tag{23.4}$$

$$x_k \tag{23.5}$$

In this book, I mostly use (23.5).

In the example above, the discrete-time signal originated from *sampling* of a continuous-time signal. However, discrete-time signals exists in many other circumstances, for example,

- the output signal from a discrete-time (computer-based) *signal filter*, for example a lowpass filter,
- the output from a discrete-time (computer-based) *controller* which controls a physical process,
- the response in a dynamic system as calculated by a (computer-based) *simulator*.

23.1 *Problems for Chapter 23*

Problem 23.1 *Discrete signal*

Given the following continuous-time signal (a ramp):

$$x_c(t) = 2t \tag{23.6}$$

where t is time in seconds.

1. Assume that the signal is sampled with sampling time (time-step) $t_s = 0.5$ s. Express x_d as a function of the discrete time t_k. Write the discrete signal or sequence (time series) x_d from time 0 to 2. Plot x with both discrete time t_k and time index k along the abscissa.

2. Repeat Problem 1 above, but now with $t_s = 0.1$ s.

23.2 *Solutions to problems for Chapter 23*

Solution to Problem 23.1

1.

$$x_{d,k} = 2t_k \tag{23.7}$$

$$x_d = \{0, 0.5, 1.0, 1.5, 2.0\} \tag{23.8}$$

Figure 23.3 shows $x_{d,k}$ with $t_s = 0.5$ s.

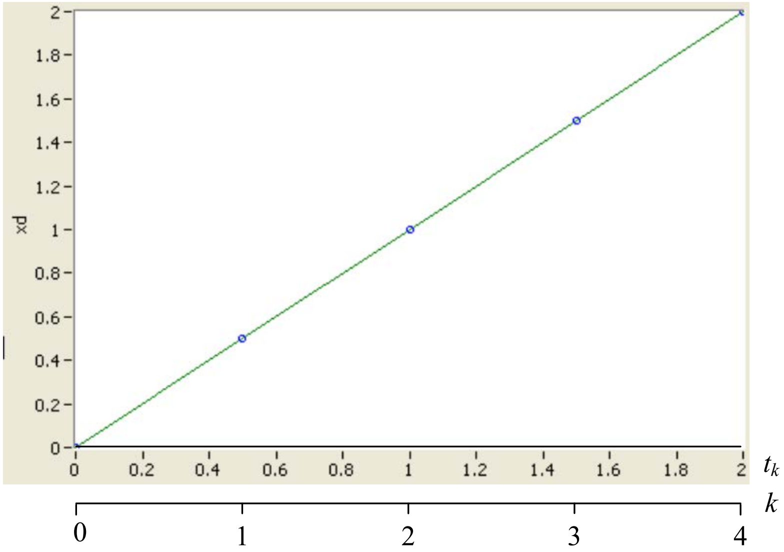


Figure 23.3: $x_{d,k}$ with $t_s = 0.5$ s.

2.

$$x_{d,k} = 2t_k \tag{23.9}$$

$$x_d = \{0, 0.1, 0.2, 0.3, \ldots, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0\} \tag{23.10}$$

Figure 23.4 shows $x_{d,k}$ with $t_s = 0.1$ s.

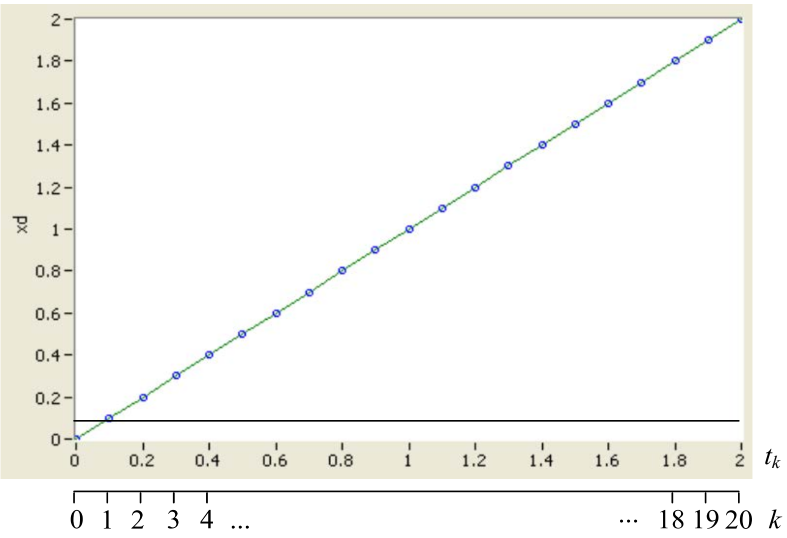


Figure 23.4: $x_{d,k}$ with $t_s = 0.1$ s

Chapter 24

Difference equations

24.1 Difference equation models

The basic model type of continuous-time dynamic systems is the differential equation. Analogously, the basic model type of discrete-time dynamic systems is the *difference equation*. Here is an example of a linear second order difference equation with u as input variable and y as output variable:

$$y_{k+2} + a_1 y_{k+1} + a_0 y_k = b_0 u_k \tag{24.1}$$

where a_i and b_j are *coefficients* of the difference equation, or model *parameters*. Note that this difference equation has unique coefficients since the coefficient of y_{k+2} is 1.

One equivalent form (24.1) is

$$y_{(k)k} + a_1 y_{k-1} + a_0 y_{k-2} = b_0 u_{k-2} \tag{24.2}$$

where there are *no time delayed* terms (no negative time indexes), only time advanced terms (positive or zero time indexes). This form can be obtained from (24.1) by increasing each time index in (24.1) by 2.

In most cases we want to write the difference equation as a formula for the output variable. In our example the formula for the output $y(k)$ can be obtained by solving for y_k from (24.2):

$$y_k = -a_1 y_{k-1} - a_0 y_{k-2} + b_0 u_{k-2} \tag{24.3}$$

(24.3) says that the output y_k is given as a linear combination of the output one time step back in time, y_{k-1}, the output two time steps back in time, y_{k-2}, and the input two time steps back in time, u_{k-2}.

24.2 Calculating responses from difference equation models

For example, (24.3) is a formula for calculating dynamic (time-varying) responses in the output, $y(k)$. The formula must be calculated once per time step, and it can be implemented in a While loop or a For loop in a computer program. Assume as an example that $y(1)$, $y(0)$ and $u(0)$ are zero. Then (24.3) gives

$$y_2 = -a_1 y_1 - a_0 y_0 + b_0 u_0 \tag{24.4}$$

$$y_3 = -a_1 y_2 - a_0 y_1 + b_0 u_1 \tag{24.5}$$

$$y_4 = -a_1 y_3 - a_0 y_2 + b_0 u_2 \tag{24.6}$$

and so on.

The *static response* – which is the (steady state) response of the system when all variables are assumed to have constant values – can be calculated from the static version of the difference equation. The static version is found by neglecting all time-dependencies in the difference equation, and setting $y_k = y_s$, $y_{k-1} = y_s$ etc. where subindex s is for static. For example, the static version of (24.3) is

$$y_s = -a_1 y_s - a_0 y_s + b_0 u_s \tag{24.7}$$

The static response is

$$y_s = \frac{b_0}{1 + a_1 + a_0} u_s \tag{24.8}$$

24.3 *Problems for Chapter 24*

Problem 24.1 *Difference equation form*

Given the following difference equation:

$$y_{k+3} + ay_{k+1} = b_1u_{k+2} + b_0u_k \tag{24.9}$$

Write the corresponding difference equation having only zero or negative time shifts.

Problem 24.2 *Block diagram of difference equation*

Particularly in the area of discrete-time (digital) signal processing the difference equations constituting the mathematical model of signal filters are represented with mathematical *block diagrams*, in the same way as differential equations of continuous-time systems are represented with block diagrams.

Figure 24.1 shows the most frequently used blocks – or the *elementary blocks* – used in block diagrams of difference equation models.

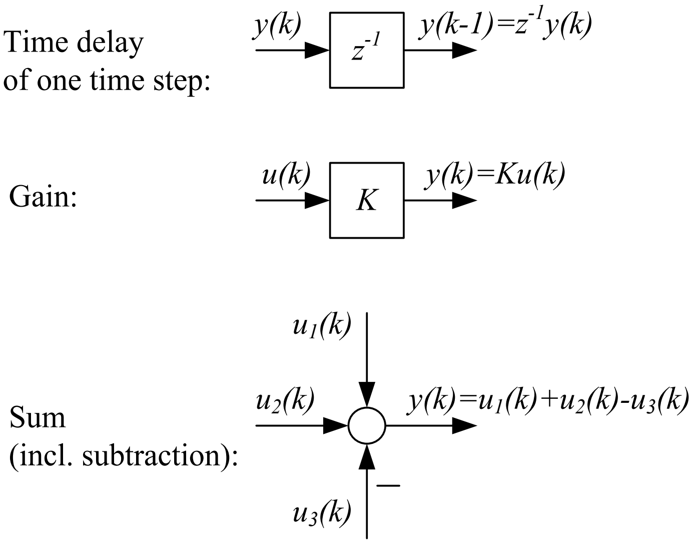


Figure 24.1: Elementary blocks for drawing block diagrams of difference equation models

A comment about the time delay block: The output y_k is equal to the time delayed input, y_{k-1}:

$$y_{k-1} = z^{-1}y_k \tag{24.10}$$

Or, equivalently:

$$y_k = z^{-1}y_{k+1} \tag{24.11}$$

The operator z^{-1} is here a time-step delay operator, which is actually the z-transfer function of the time-step delay. (z-transfer functions are described in Chapter 11 of the text-book.)

Draw a block diagram of the following difference equation:

$$y_{k+1} = ay_k + bu_k \tag{24.12}$$

where a and b are constant parameters. The block diagram shall have u_k as input and y_k as output.

Problem 24.3 *Response from discrete time filter*

Given the signal filter

$$y_k = \frac{1}{3}\left[u_k + u_{k-1} + u_{k-2}\right]$$

which is a moving average lowpass filter.

1. What is the steady state response in y when the input u is a constant? Does the filter let a constant input pass unchanged, in steady state?

2. Assume that the filter input u is a ramp:

$$\begin{aligned}\{u_k\} &= \{u_0,\ u_1,\ u_2,\ u_3,\ u_4\} && (24.13)\\ &= \{0,\ 0.5,\ 1.0,\ 1.5,\ 2.0\} && (24.14)\\ &= \{0.5k\} && (24.15)\end{aligned}$$

   Calculate the response y at $k = 0...4$. (You can assume that u is zero at negative k.)

   Also, calculate the general response y_k. Will there be a constant difference from zero between the output and the input as time index goes to infinity?

24.4 *Solutions to problems for Chapter 24*

Solution to Problem 24.1

Each of the time indexes is reduced by 3, giving

$$y_k + a y_{k-2} = b_1 u_{k-1} + b_0 u_{k-3} \tag{24.16}$$

Solution to Problem 24.2

See Figure 24.2.

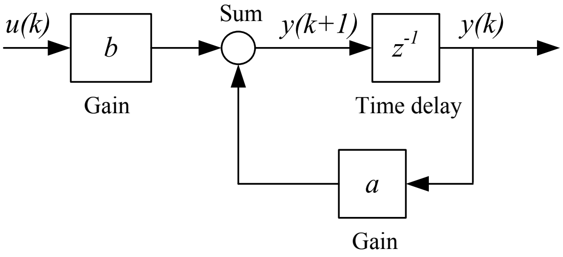


Figure 24.2: Block diagram

Solution to Problem 24.3

1. Let us assume that

$$u = U \text{ (constant)} \tag{24.17}$$

   In steady state,

$$y_s = \frac{1}{3}(U + U + U) = U$$

   Hence, the filter lets a constant input pass unchanged, in steady state.

2. The ramp response is

$$\begin{aligned}
y_0 &= \frac{1}{3}\left[u_0 + u(-1) + u(-2)\right] = \frac{1}{3}\left[0 + 0 + 0\right] = 0 \\
y_1 &= \frac{1}{3}\left[u_1 + u_0 + u(-1)\right] = \frac{1}{3}\left[0.5 + 0 + 0\right] = \frac{1}{6} = 0.17 \\
y_2 &= \frac{1}{3}\left[u_2 + u_1 + u_0\right] = \frac{1}{3}\left[1.0 + 0.5 + 0\right] = 0.5 \\
y_3 &= \frac{1}{3}\left[u_3 + u_2 + u_1\right] = \frac{1}{3}\left[1.5 + 1.0 + 0.5\right] = 1 \\
y_4 &= \frac{1}{3}\left[u_4 + u_3 + u_2\right] = \frac{1}{3}\left[2.0 + 1.5 + 1.0\right] = 1.5
\end{aligned}$$

The general response:

$$\begin{aligned}
y_k &= \frac{1}{3}\left[u_k + u_{k-1} + u_{k-2}\right] \\
&= \frac{1}{3}\left[0.5 \cdot k + 0.5 \cdot (k-1) + 0.5 \cdot (k-2)\right] \\
&= 0.5 \cdot k + \frac{1}{3}\left(0 - 0.5 - 1.0\right) \\
&= 0.5 \cdot k - 0.5 \\
&= u_k - 0.5
\end{aligned}$$

Hence, there is a constant difference equal to -0.5 between the output and the input in steady state.

Chapter 25

Discrete-time state space models

25.1 General form of discrete-time state space models

The general form of a discrete-time state space model is

$$x_{k+1} = f[x_k,\ u_k] \tag{25.1}$$

$$y_k = g[x_k,\ u_k] \tag{25.2}$$

where x is the state variable, u is the input variable which may consist of control variables and disturbances (in this model definition there is no difference between these two kinds of input variables). y is the output variable. f and g are functions – linear or nonlinear. x_{k+1} in (25.1) means the state one time-step ahead (relative to the present state x_k). Thus, the state space model expresses how the systems' state (variables) and output variables evolves along the discrete time axis.

The variables in (25.1) – (25.2) may actually be vectors, e.g.

$$x_k = \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ \vdots \\ x_{n,k} \end{bmatrix} \tag{25.3}$$

Therefore, f and/or g are generally vector evaluated functions.

25.2 Linear discrete-time state space models

A special case of the general state space model presented above is the *linear* state space model:

$$x_{k+1} = \underbrace{Ax_k + Bu_k}_{\equiv f[x_k, u_k]} \tag{25.4}$$

$$y_k = \underbrace{Cx_k + Du_k}_{\equiv g[x_k, u_k]} \tag{25.5}$$

where A is the transition matrix, B is the input gain matrix, C is the output gain matrix or measurement gain matrix and D is the direct output gain matrix (in most cases, $D = 0$).

25.3 Discretization of continuous-time state space models

Here are some situations where you need to discretize a continuous-time state space model:

- Creating a simulation algorithm from a process model. This was described in Section 7.2.
- Defining the process model to be used as the basis of a state estimator in form of a Kalman Filter, cf. Chapter 32.

The Forward Discretization method is the simplest, most commonly used, and the most flexible method. Only this method will be described here.

Given the following continuous-time state space model, which can be linear or non-linear,

$$x'(t) = f_c\left[x(t),\ u(t)\right] \tag{25.6}$$

$$y(t) = g_c\left[x(t),\ u(t)\right] \tag{25.7}$$

Approximating the time derivative in (25.6) with Forward differentiation gives

$$\frac{x_{k+1} - x_k}{t_s} = f_c\left[x_k,\ u_k\right] \tag{25.8}$$

where t_s is the time step. Solving for x_{k+1} gives

$$x_{k+1} = x_k + t_s f_c\left[x_k,\ u_k\right] \tag{25.9}$$

The dicrete-time version of (25.7) is

$$y_k = g_c\left[x_k,\ u_k\right] = g\left[x_k,\ u_k\right] \tag{25.10}$$

(25.9) and (25.10) constitute a discrete-time version of the original state space model (25.6)–(25.7).

25.4 *Problems for Chapter 25*

Problem 25.1 *Discrete-time state space model*

In Problem 6.1 a mathematical model of two coupled liquid tanks is presented. The model written as a continuous-time state space model is

$$h_1' = \frac{1}{A_1}\left(K_p u_1 - K_{v_1}\sqrt{\frac{\rho g h_1}{G}}\right) \tag{25.11}$$

$$h_2' = \frac{1}{A_2}\left(K_{v_1}\sqrt{\frac{\rho g h_1}{G}} - K_{v_2} u_2\sqrt{\frac{\rho g h_2}{G}}\right) \tag{25.12}$$

By applying Forward differentiation approximation to the time-derivatives, we get the following discrete-time model:

$$h_{1,k}' \approx \frac{h_{1,k+1} - h_{1,k}}{t_s} = \frac{1}{A_1}\left(K_p u_{1,k} - K_{v_1}\sqrt{\frac{\rho g h_{1,k}}{G}}\right) \tag{25.13}$$

$$h_{2,k}' \approx \frac{h_{2,k+1} - h_{2,k}}{t_s} = \frac{1}{A_2}\left(K_{v_1}\sqrt{\frac{\rho g h_{1,k}}{G}} - K_{v_2} u_{2,k}\sqrt{\frac{\rho g h_{2,k}}{G}}\right) \tag{25.14}$$

Assume that both levels are output variables. Write this model as a discrete-time state space model on the standard form

$$h_{1,k+1} = f_1\left[h_{1,k}, h_{2,k}, \cdots\right] \tag{25.15}$$

$$h_{2,k+1} = f_2\left[h_{1,k}, h_{2,k}, \cdots\right] \tag{25.16}$$

(which corresponds to the compact standard form $x_{k+1} = f\left[x_k, \cdots\right]$). Is the state space model linear or nonlinear?

Problem 25.2 *State space model on matrix-vector form*

Write the following difference equations model as a state space model on matrix-vector form.

$$x_{1,k+1} = -0.5x_{1,k} \tag{25.17}$$

$$x_{2,k+1} = 2u_k - x_{2,k} - 3x_{1,k} \tag{25.18}$$

$$y_k = x_{2,k} + 4u_k \tag{25.19}$$

25.5 *Solutions to problems for Chapter 25*

Solution to Problem 25.1

Solving (25.13) – (25.14) for $h_{1,k+1}$ and $h_{2,k+1}$, respectively, gives the discrete-time state space model:

$$h_{1,k+1} = h_1\left(t_k\right) + \overbrace{\frac{t_s}{A_1}\left(K_p u_{1,k} - K_{v_1}\sqrt{\frac{\rho g h_{1,k}}{G}}\right)}^{f_1} \tag{25.20}$$

$$h_{2,k+1} = h_{2,k} + \overbrace{\frac{t_s}{A_2}\left(K_{v_1}\sqrt{\frac{\rho g h_{1,k}}{G}} - K_{v_2}u_{2,k}\sqrt{\frac{\rho g h_{2,k}}{G}}\right)}^{f_2} \tag{25.21}$$

The outputs variables are

$$y_{1,k} = h_{1,k} \tag{25.22}$$

$$y_{2,k} = h_{2,k} \tag{25.23}$$

This state space model is nonlinear due to the square root functions in which the state variables are arguments.

Solution to Problem 25.2

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} -0.5 & 0 \\ -3 & -1 \end{bmatrix}}_{A}\begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 2 \end{bmatrix}}_{B}u_k \tag{25.24}$$

$$y_k = \underbrace{\begin{bmatrix} 0 & 1 \end{bmatrix}}_{C}\begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \underbrace{[4]}_{D}u_k \tag{25.25}$$

Chapter 26

The z-transform

26.1 Definition of the z-transform

The z-transform of discrete-time signals plays much the same role as the Laplace transform for continuous-time systems.

The z-transform of the discrete-time signal $\{y(k)\}$, or just y_k, is defined as follows:

$$\mathcal{Z}\{y_k\} = \sum_{k=0}^{\infty} y_k z^{-k} \tag{26.1}$$

I will use capital as symbol of the z-transformed time function, e.g. $Y(z)$ for $\mathcal{Z}\{y_k\}$.

Example 26.1 *z-transform of a constant*

Assume that the signal y_k has constant value A. This signal can be regarded a step of amplitude A at time-step 0. z-transforming y_k gives

$$Y(z) = \sum_{k=0}^{\infty} y_k z^{-k} = \sum_{k=0}^{\infty} A z^{-k} = \frac{A}{1 - z^{-1}} = \frac{Az}{z - 1} \tag{26.2}$$

[End of example 26.1]

26.2 Properties of the z-transform

Below are the most important properties of the z-transform. These properties can be used when calculating the z-transform of composite signals.

- **Linearity:**

$$k_1 Y_1(z) + k_2 Y_2(z) \Longleftrightarrow k_1 y_{1,k} + k_2 y_{2,k} \tag{26.3}$$

- **Time delay:** Multiplication by z^{-n} means time delay of n time-steps:

$$z^{-n}Y(z) \Longleftrightarrow y_{k-n} \tag{26.4}$$

- **Time advancing**: Multiplication by z^{n} means time advancing by n time-steps:

$$z^{n}Y(z) \Longleftrightarrow y_{k+n} \tag{26.5}$$

26.3 z-transform pairs

Below are several important z-transform pairs showing discrete-time functions and their corresponding z-transforms. The time functions are defined for $k \geq 0$.

$$\text{Unity impulse at time-step } k\text{: } \delta(k) \quad \Longleftrightarrow \quad z^{k} \tag{26.6}$$

$$\text{Unity impulse at time-step } k=0\text{: } \delta(0) \quad \Longleftrightarrow \quad 1 \tag{26.7}$$

$$\text{Unity step at time-step } k=0\text{: } 1 \Longleftrightarrow \quad \frac{z}{z-1} \tag{26.8}$$

$$\text{Time exponential: } a^{k} \quad \Longleftrightarrow \quad \frac{z}{z-a} \tag{26.9}$$

Example 26.2 *z-transformation of a composite signal*

Given the following discrete-time function:

$$y_k = Ba^{k-n} \tag{26.10}$$

(which is a time delayed time exponential). The inverse z-transform of $y(k)$ can be calculated using (26.9) together with (26.3) and (26.4). The result becomes

$$Y(z) = Bz^{-n}\frac{z}{z-a} = B\frac{z^{1-n}}{z-a} \tag{26.11}$$

[End of example 26.2]

26.4 Inverse z-transform

Inverse z-transformation of a given z evaluated function, say $Y(z)$, is calculating the corresponding time function, say y_k. The inverse transform may be calculated using a complex integral[1], but this method is not very practical. Another method is to find a proper combination of precalculated z-transformation pairs, possibly in combination with some of the z-transform properties defined above.

[1] $y(k) = \frac{1}{2\pi j}\oint Y(z)z^{k}\frac{dz}{z}$, where the integration path must be in the area of convergence of $Y(z)$.**?**

In most cases where you need to calculate a time signal y_k, its z-transform $Y(z)$ stems from a transfer function excited by some discrete-time input signal. You may then calculate y_k by first transfering the transfer function to a corresponding difference equation, and then calculating y_k iteratively from this difference equation as explained in Section 24.2.

26.5 *Problems for Chapter 26*

Problem 26.1 *z-transform of unit impulse*

Calculate the z-transform of δ_k which a unit impulse. This is a signal having amplitude 1 at discrete time $k = 0$ and zero at other points of time.

Problem 26.2 *Linearity property*

The Linearity property of the z-transform can be expressed as

$$k_1 Y_1(z) + k_2 Y_2(z) \Longleftrightarrow k_1 y_{1,k} + k_2 y_{2,k} \tag{26.12}$$

Show that the Linearity property holds for this example of a signal:

$$y_k = A + B = AS_k + BS_k \tag{26.13}$$

where A and B are constants, and S_k is the unity step functions, i.e. a step occuring at time zero. (Hint: Show that the $\mathcal{Z}$-transform of the right side of (26.12) is equal to the left side of (26.12) for the given signal.)

Problem 26.3 *z-transform of a step signal*

Calculate the z-transform of a step signal of amplitude 4 which occurs at time-index 2. You can use $S(k)$ to represent a unit step function, i.e. a step occuring at time zero.

Problem 26.4 *Inverse z-transform*

Calculate the inverse z-transform of

$$Y(z) = \frac{z}{z - 0.5} \tag{26.14}$$

26.6 *Solutions to problems for Chapter 26*

Solution to Problem 26.1

$$\mathcal{Z}\left\{y_k\right\} = Y(z) = \sum_{k=0}^{\infty} y_k z^{-k} = \sum_{k=0}^{\infty} \delta_k z \tag{26.15}$$

$$= 1 \cdot z^{-0} + 0 \cdot z^{-1} + 0 \cdot z^{-2} + \cdots + 0 \cdot z^{-n} + \cdots \tag{26.16}$$

$$= 1 \tag{26.17}$$

Solution to Problem 26.2

The $\mathcal{Z}$-transform of the right side of (26.12) becomes

$$\mathcal{Z}\{AS_k + BS_k\} = \mathcal{Z}\{(A+B)\,S_k\} = (A+B)\frac{z}{z-1} \tag{26.18}$$

The left side of (26.12) becomes

$$A \cdot \mathcal{Z}\{S_k\} + B \cdot \mathcal{Z}\{S_k\} = A\frac{z}{z-1} + B\frac{z}{z-1} = (A+B)\frac{z}{z-1} \tag{26.19}$$

Hence, the $\mathcal{Z}$-transform of the right side of (26.12) is equal to the left side of (26.12), and consequently the linear property holds.

Solution to Problem 26.3

The signal is

$$y_k = 4 \cdot S_{k-2} \tag{26.20}$$

Using the Linearity property and the Time delay property of the $\mathcal{Z}$-transform, we get

$$\mathcal{Z}\{y_k\} = Y(z) = 4z^{-2}\mathcal{Z}\{S_k\} = 4z^{-2}\frac{z}{z-1} = \frac{4}{z^2 - z} \tag{26.21}$$

Solution to Problem 26.4

Using the $\mathcal{Z}$-transform pair denoted "Time exponential" we get

$$y_k = 0.5^k \tag{26.22}$$

Chapter 27

Discrete-time (or z-) transfer functions

27.1 Introduction

Models in the form of difference equations can be z-transformed to z-transfer *functions*, which plays the same role in discrete-time systems theory as s transfer functions do in continuous-time systems theory. More specific:

- The combined model of systems in a serial connection can be found my simply multiplying the individual z-transfer functions.
- The frequency response can be calculated from the transfer function.
- The transfer function can be used to represent the system in a simulator or in computer tools for analysis and design (as SIMULINK, MATLAB or LabVIEW)

27.2 From difference equation to transfer function

As an example we will derive the discrete-time or z-transfer function from input u to output y from the difference equation (24.3), which is repeated here:

$$y_k = -a_1 y_{k-1} - a_0 y_{k-2} + b_0 u_{k-2} \tag{27.1}$$

First, we take the z-transform of both sides of the difference equation:

$$\mathcal{Z}\left\{y_k\right\} = \mathcal{Z}\left\{-a_1 y_{k-1} - a_0 y_{k-2} + b_0 u_{k-2}\right\} \tag{27.2}$$

Using the linearity property (26.3) and the time delay property (26.4) (27.2) can be written as

$$\mathcal{Z}\left\{y_k\right\} = -\mathcal{Z}\left\{a_1 y_{k-1}\right\} - \mathcal{Z}\left\{a_0 y_{k-2}\right\} + \mathcal{Z}\left\{b_0 u_{k-2}\right\} \tag{27.3}$$

and

$$Y(z) = -a_1 z^{-1} Y(z) - a_0 z^{-2} Y(z) + b_0 z^{-2} U(z) \tag{27.4}$$

which can be written as

$$Y(z) + a_1 z^{-1} Y(z) + a_0 z^{-2} Y(z) = b_0 z^{-2} U(z) \tag{27.5}$$

or

$$\left[1 + a_1 z^{-1} + a_0 z^{-2}\right] Y(z) = b_0 z^{-2} U(z) \tag{27.6}$$

$$Y(z) = \underbrace{\frac{b_0 z^{-2}}{1 + a_1 z^{-1} + a_0 z^{-2}}}_{H(z)} U(z) \tag{27.7}$$

$$= \underbrace{\frac{b_0}{z^2 + a_1 z^1 + a_0}}_{H(z)} U(z) \tag{27.8}$$

where $H(z)$ is the z-transfer function from u to y. Hence, z-transfer functions can be written both with positive and negative exponents of z.[1]

27.3 From transfer function to difference equation

In the above Section we derived a z-transfer function from a difference equation. We may go the opposite way – to derive a difference equation from a given z-transfer function. Some applications of this are

- Deriving a filtering algorithm from a filtering transfer function
- Deriving a control function from a given controller transfer function
- Deriving a simulation algorithm from the transfer function of the system to be simulated

The procedure will be illustrated via a concrete example. Assume given the following transfer function:

$$H(z) = \frac{b_0}{z^2 + a_1 z + a_0} = \frac{Y(z)}{U(z)} \tag{27.9}$$

We start by cross multiplying (27.9):

$$\left(z^2 + a_1 z + a_0\right) Y(z) = b_0 U(z) \tag{27.10}$$

which can be written as

$$z^2 Y(z) + a_1 z Y(z) + a_0 Y(z) = b_0 U(z) \tag{27.11}$$

[1] In signal processing theory transfer functions ares usually written with negative exponents of z, while in control theory they are usually written with positive exponents.

Taking the inverse transform of the above expression gives

$$\underbrace{z^2Y(z)}_{y_{k+2}} + \underbrace{a_1zY(z)}_{a_1y_{k+1}} + \underbrace{a_0Y(z)}_{a_0y_k} = \underbrace{b_0U(z)}_{b_0u_k} \tag{27.12}$$

Reducing each of the time indexes by 2 yields

$$y_k + a_1y_{k-1} + a_0y_{k-2} = b_0u_{k-2} \tag{27.13}$$

Usually it is practical to have the output variable alone on the left side:

$$y_k = -a_1y_{k-1} - a_0y_{k-2} + b_0u_{k-2} \tag{27.14}$$

27.4 Calculating time responses for discrete-time transfer functions

Assume given a transfer function, say $H(z)$, with input variable u and output variable y. Then,

$$Y(z) = H(z)U(z) \tag{27.15}$$

If $U(z)$ is given, the corresponding time response in y can be calculated in several ways:

1. By finding a proper transformation pair in Section 26.3, possibly combined with some of the z-transform properties in Section 26.2.
2. By deriving a differential equation corresponding to the transfer function and then calculating y_k iteratively according to the difference equation. The procedure of deriving a differential equation corresponding to a given transfer function is explained in Section 27.3, and the calculation of time responses for a difference equation is described in Section 24.2.

27.5 Static transfer function and static response

The static version H_s of a given transfer function $H(z)$ will now be derived. Using the static transfer function the static response can easily be calculated. Assume that the input variable u is a step of amplitude U. The stationary response can be found using the final value theorem:

$$\begin{aligned}
\lim_{k\to\infty} y_k = y_s &= \lim_{z\to 1}(z-1)Y(z) && (27.16)\\
&= \lim_{z\to 1}(z-1)H(z)U(z) && (27.17)\\
&= \lim_{z\to 1}(z-1)H(z)\frac{zU}{z-1} && (27.18)\\
&= H(1)U && (27.19)
\end{aligned}$$

Thus, we have the following static transfer function:

$$H_s = \frac{y_s}{u_s} = \lim_{z \to 1} H(z) = H(1) \tag{27.20}$$

Using the static transfer function the static response can be calculated by

$$y_s = H_s U \tag{27.21}$$

Example 27.1 *Static transfer function*

Let us consider the following transfer function:

$$H(z) = \frac{Y(z)}{U(z)} = \frac{az}{z - (1 - a)} \tag{27.22}$$

which is the transfer function of the lowpass filter presented in Ch. 3.4.6.2, repeated here:

$$y_k = (1 - a)\, y_{k-1} + a u_k \tag{27.23}$$

The corresponding static transfer function is

$$H_s = \frac{y_s}{u_s} = \lim_{z \to 1} H(z) = \lim_{z \to 1} \frac{a}{1 - (1 - a)\, z^{-1}} = \frac{a}{1 - (1 - a) \cdot 1} = 1 \tag{27.24}$$

Thus,

$$y_s = H_s u_s = u_s \tag{27.25}$$

Can we find the same correspondence between u_s and y_s from the difference equation (27.23)? Setting $y_k = y_{k-1} = y_s$ and $u_k = u_s$ gives

$$y_s = (1 - a)\, y_s + a u_s \tag{27.26}$$

giving

$$\frac{y_s}{u_s} = \frac{a}{1 - (1 - a)} = 1 \tag{27.27}$$

which is the same as (27.24).

[End of Example 27.1]

27.6 Poles and zeros

Poles and zeros of z-transfer functions are defined in the same way as for s transfer functions: The zeros of the transfer function are the z-roots of numerator polynomial, and the poles are the z-roots of the denominator polynomial.

One important application of poles is stability analysis, cf. Section 29.

Example 27.2 *Poles and zeros*

Given the following z-transfer function:

$$H(z) = \frac{(z-b)}{(z-a_1)(z-a_2)} \tag{27.28}$$

The poles are a_1 and a_2, and the zero is b.

[End of Example 27.2]

27.7 From s-transfer functions to z-transfer functions

In some cases you need to find a discrete-time z-transfer function from a given continuous-time s transfer function:

- In accurate model based design of a discrete controller for a process originally in the form of a continuous-time s transfer function, $H_p(s)$. The latter should be discretized to get a discrete-time process model before the design is started.
- Implementation of continuous-time control and filtering functions in a computer program.

There are several methods for discretization of an s transfer function. The methods can be categorized as follows, and they are described in the following sections:

1. **Discretization based on having a zero order hold (ZOH) element on the input of the system**. This method should be used in controller design of a process which has a sample and hold element on its input, as when a physical process is controlled by a computer via a DA converter (digital to analog). Zero order hold means that the input signal is held constant during the time-step or sampling interval. Figure 27.1 shows a block diagram of a continuous-time process with transfer function model $H(s)$ having a zero order hold element on its input.

   ZOH discretization gives a perfect z-transfer function in the sense that it produces exactly the same response as produced by the s-transfer function at the discrete points of time. (If we discretize using e.g. the Forward differentiation the responses will differ a little.) The ZOH discretization method is actually complicated to implement manually, but tools as Matlab and LabVIEW have functions that perform the discretization easily, and in most practical applications, you will be using such tools.

2. **Using an apropriate approximation to time-derivatives**, as Forward Difference method, or Backward Difference method, or Tustin's method. In such cases the input signal is a discrete-time signal with no holding (no ZOH element is assumed). The procedure has the following steps:

   - From the given the continuous-time s-transfer function $H_c(s)$, derive the corresponding differential equation.

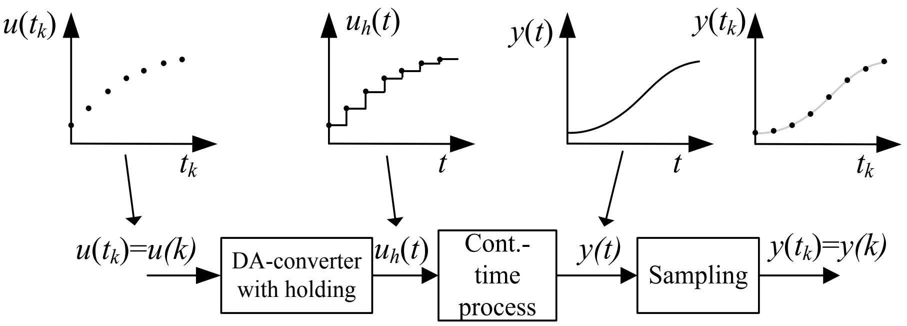


Figure 27.1: Block diagram of a process having a zero order hold element on its input. u_h is the piecewise constant (held) input signal.

- Apply some approximation to the time-derivatives of the differential equation. If you do not have any other preferences, use the Backward difference method. The result is a difference equation.
- Calculate the z-transfer function from the difference equation, cf. Section 27.2.

Below is an example of discretizing a s-transfer function manually using the Backward differentiation approximation.

Example 27.3 *Discretizing a first order transfer function*

We will discretize the following continuous-time transfer function:

$$H_c(s) = \frac{K}{t_c s + 1} = \frac{Y(s)}{U(s)} \tag{27.29}$$

1. Deriving the corresponding differential equation: Cross-multiplying gives

$$(t_c s + 1)\, Y(s) = KU(s) \tag{27.30}$$

   Resolving the parenthesis gives

$$t_c sY(s) + Y(s) = KU(s) \tag{27.31}$$

   Taking the inverse Laplace transform of both sides of this equation gives the following differential equation (because multiplying by s means time-differentiation in the time-domain):

$$t_c y'(t) + y(t) = Ku(t) \tag{27.32}$$

   Let us use t_k to represent the present point of time – or discrete time:

$$t_c y'_k + y_k = Ku_k \tag{27.33}$$

2. Applying the Backward differentiation approximation:

$$t_c \frac{y_k - y_{k-1}}{t_s} + y_k = Ku_k \tag{27.34}$$

Solving for y_k gives the following difference equation:

$$y_k = \frac{t_c}{t_c + t_s} y_{k-1} + \frac{t_s K}{t_c + t_s} u_k \tag{27.35}$$

3. Taking the z-transform of the difference equation:

$$y(z) = \frac{t_c}{t_c + t_s} z^{-1} Y(z) + \frac{t_s K}{t_c + t_s} U(z) \tag{27.36}$$

from which we obtain the following z-transfer function:

$$H(z) = \frac{Y(z)}{U(z)} = \left(z - \frac{t_c}{t_c + t_s} \right)^{-1} \frac{z t_s K}{t_c + t_s} \tag{27.37}$$

[End of Example 27.3]

27.8 *Problems for Chapter 27*

Problem 27.1 *Find the z-transfer function*

Given the signal filter

$$y_k = \frac{1}{3}\left[u_k + u_{k-1} + u_{k-2}\right]$$

which is a moving average lowpass filter. Find the z-transfer function from u to y.

Problem 27.2 *From z-transfer function to difference equation*

Assume that the following transfer function from input signal u to output signal y of a physical system is found by some system identification method:

$$H(z) = \frac{Y(z)}{U(z)} = \frac{a}{bz^2 + cz + d}$$

What is the corresponding difference equation relating u and y?

Problem 27.3 *Output response from a z-transfer function*

Assume that the input signal u to the following transfer function is an impulse of amplitude A.

$$H(z) = \frac{Y(z)}{U(z)} = \frac{z}{z-1} \tag{27.38}$$

Calculate the output response $y(k)$.

Problem 27.4 *Static z-transfer function*

In Problem 27.1 the following transfer function of a moving avarage lowpass filter was found:

$$H(z) = \frac{Y(z)}{U(z)} = \frac{1}{3}\left[1 + z^{-1} + z^{-2}\right] \tag{27.39}$$

1. Calculate the corresponding static transfer function H_s.
2. Assume that the filter input is a constant, $u_k = U$. Calculate the corresponding steady state filter output y_s from H_s.

Problem 27.5 *Poles and the zeros of z-transfer function*

Given the following transfer function:

$$H(z) = \frac{bz^{-2} + z^{-1}}{1 - az^{-1}} \tag{27.40}$$

Calculate the poles and the zeros of the transfer function.

Problem 27.6 *From s-transfer function to z-transfer function*

The s-transfer function of a (continuous-time) integrator is

$$H_{\text{cont}}(s) = \frac{Y(s)}{U(s)} = \frac{1}{s} \tag{27.41}$$

Derive a corresponding z-transfer function $H_{\text{disc}}(z)$ assuming Backward discretization. The time-step is t_s.

27.9 *Solutions to problems for Chapter 27*

Solution to Problem 27.1

z transformation gives

$$Y(z) = \frac{1}{3}\left[u(z) + z^{-1}u(z) + z^{-2}u(z)\right] = \frac{1}{3}\left[1 + z^{-1} + z^{-2}\right]U(z) \tag{27.42}$$

which gives the transfer function

$$\frac{Y(z)}{U(z)} = \frac{1}{3}\left[1 + z^{-1} + z^{-2}\right] \tag{27.43}$$

Solution to Problem 27.2

Cross-multiplication gives

$$\left[bz^2 + cz + d\right]Y(z) = aU(z)$$

or

$$bz^2Y(z) + czY(z) + dY(z) = aU(z) \tag{27.44}$$

which, inverse-transformed, gives this difference equation:

$$by(k+2) + cy\,(k+1) + dy(k) = au(k)$$

Solution to Problem 27.3

The $\mathcal{Z}$-transform of y becomes

$$Y(z) = H(z)U(z) = \frac{z}{z-1} \cdot A \tag{27.45}$$

y_k is given by the inverse transform of $Y(z)$:

$$y_k = \mathcal{Z}^{-1}\{Y(z)\} = \mathcal{Z}^{-1}\{A\frac{z}{z-1}\} = A \tag{27.46}$$

(which is a step of amplitude A at time zero).

Solution to Problem 27.4

1. The static transfer function is

$$H_s = H(z=1) = \frac{1}{3}\left[1 + 1^{-1} + 1^{-2}\right] = 1 \tag{27.47}$$

2. The steady state filter output is

$$y_s = H_sU = 1 \cdot U = U \tag{27.48}$$

Solution to Problem 27.5

It is convenient to start by rewriting the transfer function as follows:

$$H(z) = \frac{z^{-2}b + z^{-1}}{1 - az^{-1}} \cdot \frac{z^2}{z^2} = \frac{z+b}{z^2 - az} = \frac{z+b}{(z-a)\,z} \tag{27.49}$$

Thus, the zero z is

$$z = -b \tag{27.50}$$

and the poles p_i are

$$p_1 = a; \;\; p_2 = 0 \tag{27.51}$$

Solution to Problem 27.6

(27.41) can be written

$$sY(s) = U(s) \tag{27.52}$$

Inverse Laplace transform gives

$$y'(t) = u(t) \tag{27.53}$$

Applying Backward discretization to the time-derivative and introducing discrete time notation:

$$y'(t_k) \approx \frac{y_k - y_{k-1}}{t_s} = u_k \tag{27.54}$$

Solving for $y(t_k)$:

$$y_k = y_{k-1} + t_s u_k \tag{27.55}$$

Taking the $\mathcal{Z}$-transform:

$$Y(z) = z^{-1}Y(z) + t_s U(z) \tag{27.56}$$

The transfer function becomes

$$H_{\text{disc}}(z) = \frac{Y(z)}{U(z)} = \frac{t_s}{1 - z^{-1}} = \frac{zt_s}{z-1} \tag{27.57}$$

Chapter 28

Frequency response of discrete-time systems

As for continuous-time systems, the frequency response of a discrete-time system can be calculated from the transfer function: Given a system with z-transfer function $H(z)$. Assume that input signal exciting the system is the sinusoid

$$u_k = U\sin(\omega t_k) = U\sin(\omega k t_s) \tag{28.1}$$

where ω is the signal frequency in rad/s. The time-step is t_s.

It can be shown that the stationary response on the output of the system is

$$\begin{align}
y_k &= Y\sin(\omega k t_s + \phi) \tag{28.2}\\
&= UA\sin(\omega k t_s + \phi) \tag{28.3}\\
&= \underbrace{U\overbrace{\left|H(e^{j\omega t_s})\right|}^{A}}_{Y}\sin\left[\omega t_k + \underbrace{\arg H(e^{j\omega t_s})}_{\phi}\right] \tag{28.4}
\end{align}$$

where $H(e^{j\omega t_s})$ is the *frequency response* which is calculated with the following substitution:

$$H(e^{j\omega t_s}) = H(z)|_{z=e^{j\omega t_s}} \tag{28.5}$$

The *amplitude gain function* is

$$A(\omega) = |H(e^{j\omega t_s})| \tag{28.6}$$

The *phase lag function* is

$$\phi(\omega) = \arg H(e^{j\omega t_s}) \tag{28.7}$$

$A(\omega)$ and $\phi(\omega)$ can be plotted in a Bode diagram.

Figure 28.1 shows as an example the Bode plot of the frequency response of the following transfer function (time-step is 0.1s):

$$H(z) = \frac{b}{z-a} = \frac{0.0952}{z-0.9048} \tag{28.8}$$

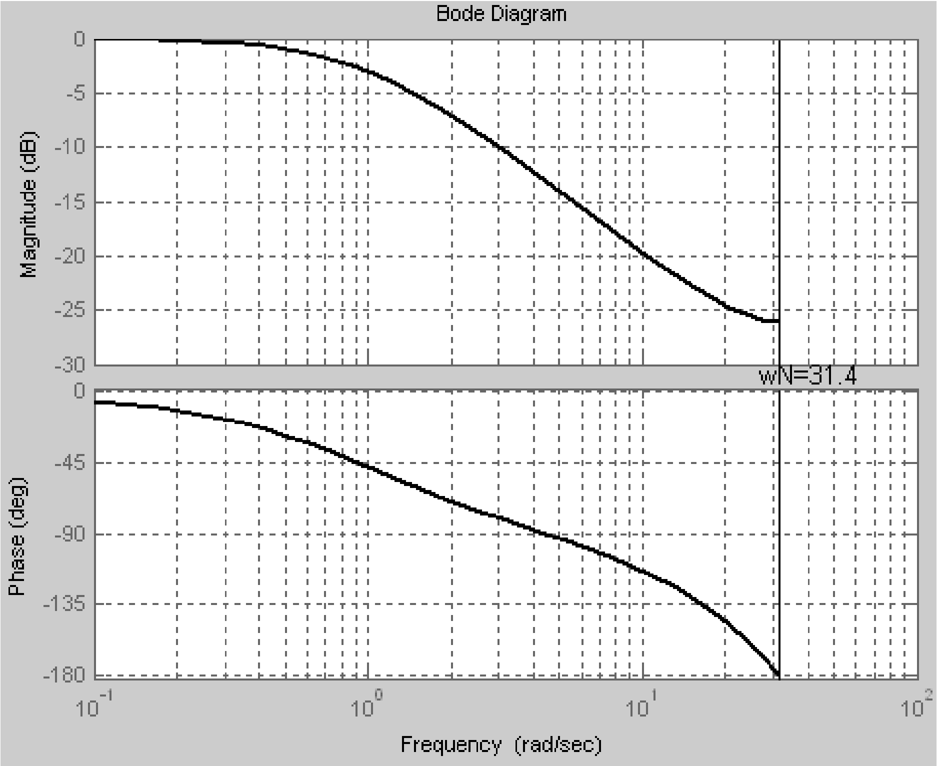


Figure 28.1: Bode plot of the transfer function (28.8). $\omega_N = 31.4$ rad/s is the Nyquist frequency (sampling time h is 0.1s).

Note that the plots in Figure 28.1 are drawn only up to the Nyquist frequency which in this case is

$$\omega_N = \frac{\omega_s}{2} = \frac{2\pi/t_s}{2} = \frac{\pi}{t_s} = \frac{\pi}{0.1} = 10\pi \approx 31.4 \text{ rad/s} \tag{28.9}$$

The plots are not drawn (but they exist!) above the Nyquist frequency because of symmetry of the frequency response, as explained in the following section.

Example 28.1 *Calculating the frequency response manually from the z-transfer function*

Given the z-transfer function

$$H(z) = \frac{b}{z-a} \tag{28.10}$$

The frequency response becomes

$$H(e^{j\omega t_s}) = \frac{b}{e^{j\omega t_s} - a} \tag{28.11}$$

$$= \frac{b}{\cos \omega t_s + j \sin \omega t_s - a} \tag{28.12}$$

$$= \frac{b}{\underbrace{(\cos \omega t_s - a)}_{\text{Re}} + j\underbrace{\sin \omega t_s}_{\text{Im}}} \tag{28.13}$$

$$= \frac{b}{\sqrt{(\cos \omega t_s - a)^2 + (\sin \omega t_s)^2} e^{j \arctan[(\sin \omega t_s)/(\cos \omega t_s - a)]}} \tag{28.14}$$

$$= \frac{b}{\sqrt{(\cos \omega t_s - a)^2 + (\sin \omega t_s)^2}} e^{j\left[-\arctan\left(\frac{\sin \omega t_s}{\cos \omega t_s - a}\right)\right]} \tag{28.15}$$

The amplitude gain function is

$$A(\omega) = |H(e^{j\omega t_s})| = \frac{b}{\sqrt{(\cos \omega t_s - a)^2 + (\sin \omega t_s)^2}} \tag{28.16}$$

and the phase lag function is

$$\phi(\omega) = \arg H(e^{j\omega t_s}) = -\arctan\left(\frac{\sin \omega t_s}{\cos \omega t_s - a}\right) \text{ [rad]} \tag{28.17}$$

[End of Example 28.1]

Even for the simple example above, the calculations are cumbersome, and prone to errors. Therefore you should use some computer tool for calculating the frequency response, as Python Control Package, MATLAB Control System Toolbox or LabVIEW Control Design Toolkit.

Symmetry of frequency response

It can be shown that the frequency response is symmetric as follows: $|H(e^{j\omega t_s})|$ and $\arg H(e^{j\omega t_s})$ are unique functions in the frequency interval $[0, \omega_N]$ where ω_N is the Nyquist frequency. In the following intervals $[m\omega_s, (m+1)\omega_s]$ (m is an integer) the functions are mirrored as indicated in Figure 28.2 which has a logarithmic frequency axis. (The Bode plots in this section are for the transfer function (28.8).)

The symmetry appears clearer in the Bode plots in Figure 28.3 where the frequency axis is linear.

Due to the symmetry of the frequency response, it is strictly not necessary to draw more of frequency response plots than of the frequency interval $[0, \omega_N]$.

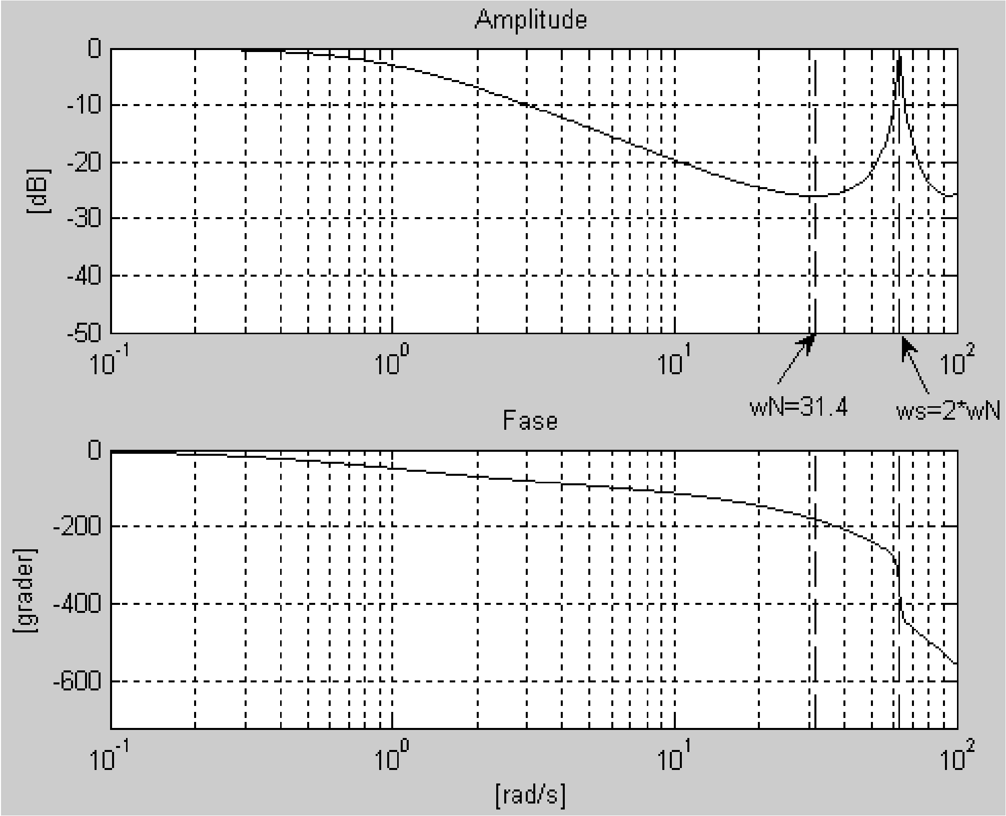


Figure 28.2: Bode plots of frequency response of (28.16). The frequency axis is logarithmic.

28.1 *Problems for Chapter 28*

Problem 28.1 *Gain and amplitude functions*

Given the following transfer function:

$$H(z) = \frac{1}{z} \tag{28.18}$$

What is the amplitude gain function and the phase lag function?

Problem 28.2 *Frequence response*

Given a continuous-time filter with transfer function

$$H_{\text{cont}}(s) = \frac{1}{t_f s + 1} \tag{28.19}$$

with

$$t_f = 1 \text{ sec} \tag{28.20}$$

Discretization of $H_{\text{cont}}(s)$ using the Backward method of discretization with time-step t_s gives the following discrete-time filter:

$$H_{\text{disc}}(z) = \frac{az}{z-(1-a)} \tag{28.21}$$

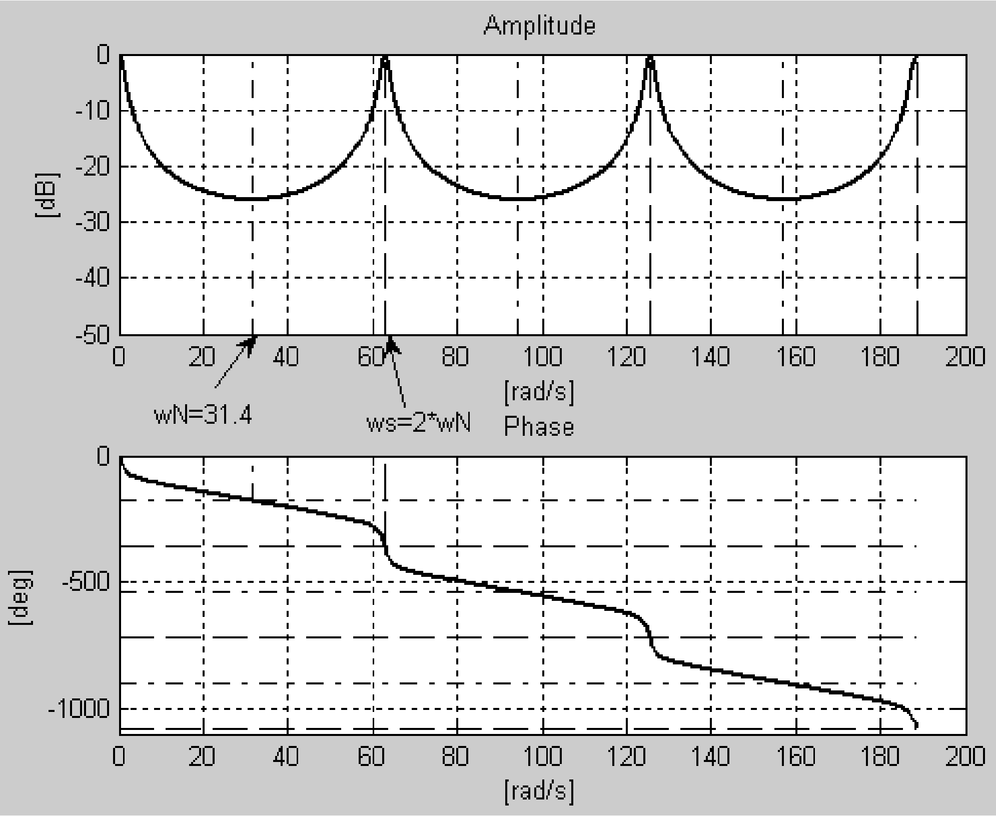


Figure 28.3: Bode plots of frequency response of (28.17). The frequency axis is linear to make the symmetries if the frequency responses clearer.

where

$$a = \frac{t_s}{t_f + t_s} \tag{28.22}$$

Figure 28.4 shows the frequency responses of $H_{\text{cont}}(s)$ and $H_{\text{disc}}(z)$ with time-step

$$t_s = 0.2 \text{ s} \tag{28.23}$$

1. Why is there a difference between the frequence responses of $H_{\text{cont}}(s)$ and $H_{\text{disc}}(z)$, and why is the difference more apparent at higher frequencies than at lower frequencies? (Qualitative answers are ok.)

2. The frequency response curves of $H_{\text{disc}}(z)$ are unique up to a certain frequency – which frequency? Express this frequency in Hz and rad/s. Is that frequency indicated in Figure 28.4?

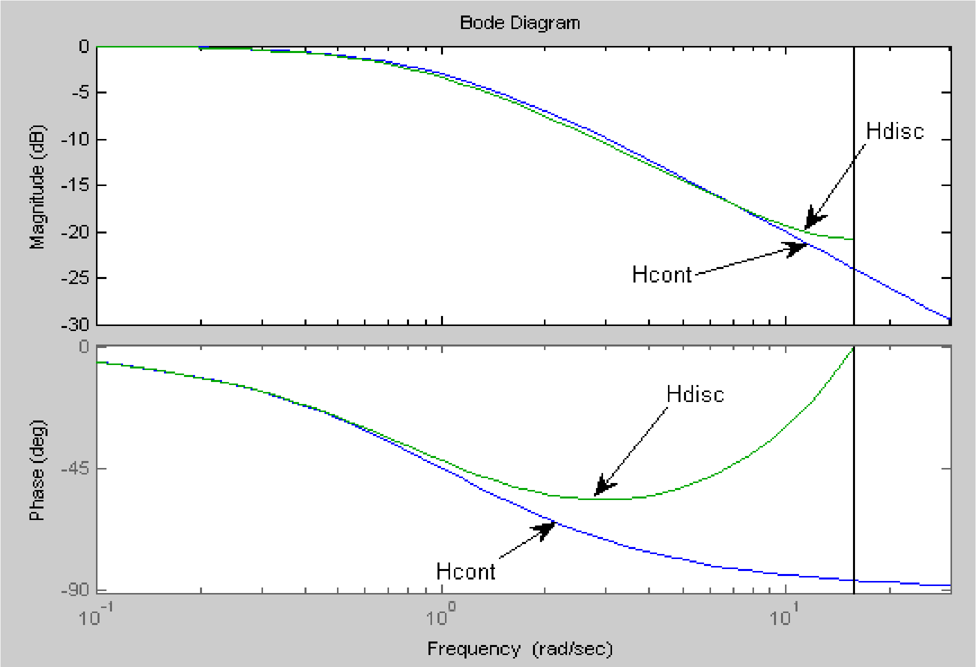


Figure 28.4: Frequency response

28.2 *Solutions to problems for Chapter 28*

Solution to Problem 28.1

The amplitude gain function is

$$A(\omega) = |H(e^{j\omega T_s})| = \left|\frac{1}{e^{j\omega t_s}}\right| = \frac{1}{|e^{j\omega t_s}|} = \frac{1}{1} = 1 \tag{28.24}$$

The phase lag function is

$$\phi(\omega) = \arg H(e^{j\omega t_s}) = \arg \frac{1}{e^{j\omega t_s}} = \arg e^{-j\omega t_s} = -\omega t_s = -\omega \cdot 0.05 \text{ [rad]} \tag{28.25}$$

Solution to Problem 28.2

1. There a difference between the frequence responses of $H_{\text{cont}}(s)$ and $H_{\text{disc}}(z)$ because the discrete-time filter is derived from the continuous-time filter using an *approximation* of the time-derivatives in the filter model when that model is written as a differential equation.

   An explanation of why the difference increases with increasing frequency is that the approximation of the time-derivative becomes less accurate if signals vary faster (i.e. have higher frequency).

2. The frequency response curves of $H_{\text{disc}}(z)$ are unique up to the Nyquist frequency:

$$f_N = \frac{1}{t_s} = \frac{1}{0.2} = 5 \text{ Hz} \tag{28.26}$$

$$\omega_N = \frac{\pi}{t_s} = \frac{\pi}{0.2} = 5\pi \text{ rad/s} = 15.7 \text{ rad/s} \tag{28.27}$$

which *is* the frequency indicated with a vertical line in Figure 28.4.

Chapter 29

Stability analysis of discrete-time dynamic systems

29.1 Definition of stability properties

Assume given a dynamic system with input u and output y. The stability property of a dynamic system can be defined from the *impulse response*[1] of a system:

- **Asymptotic stable system:** The steady state impulse response, h, is zero:

$$\lim_{k\to\infty} h_k = 0 \tag{29.1}$$

- **Marginally stable system:** The steady state impulse response is different from zero, but limited:

$$0 < \lim_{k\to\infty} h_k < \infty \tag{29.2}$$

- **Unstable system:** The steady state impulse response is unlimited:

$$\lim_{k\to\infty} h_k = \infty \tag{29.3}$$

The impulse response for the different stability properties are illustrated in Figure 29.1.

(The simulated system is defined in Example 29.1.)

29.2 Stability analysis of transfer function models

In the following we will base the analysis on the following fact: *The transfer function is the z-transformed impulse response.* Here is the proof of this fact: Given a system with transfer

[1] An impulse $\delta(0)$ is applied at the input.

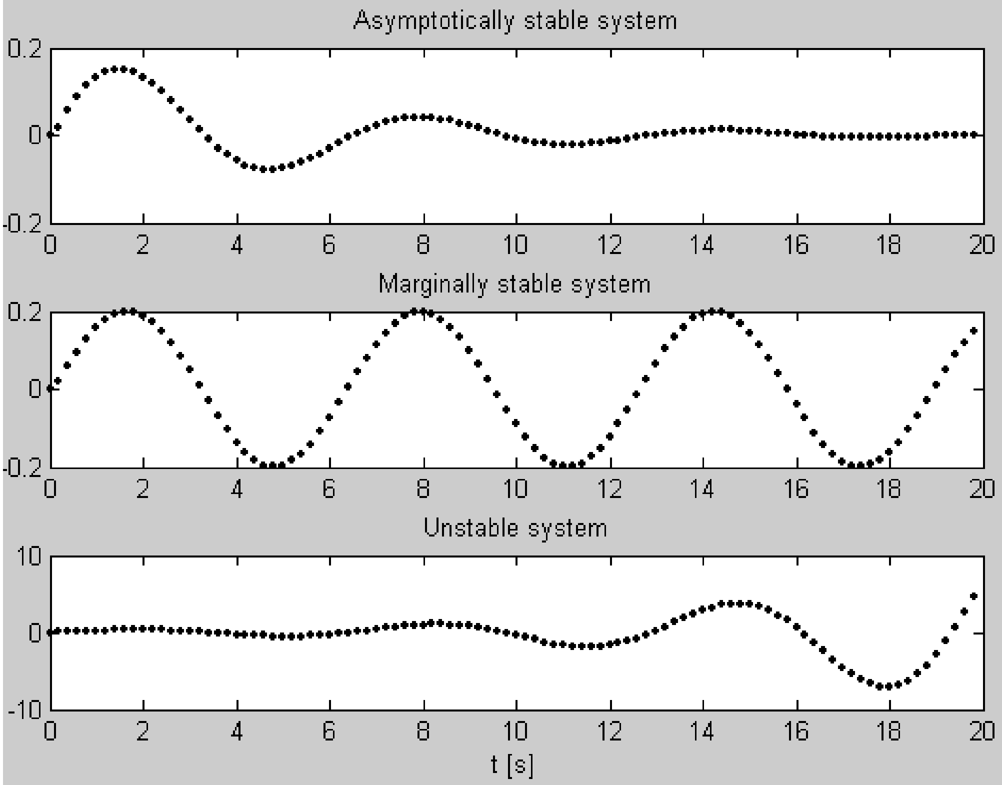


Figure 29.1: Impulse response and stability properties

function $H(z)$. Assume that the input u is an impulse, which is a signal having value 1 at time index $k = 0$ and value zero at other points of time. According to (26.7) $U(z) = 1$. Then the z-transformed impulse response is

$$Y(z) = H(z)U(z) = H(z) \cdot 1 = H(z) \tag{29.4}$$

(as stated).

Now, we proceed with the stability analysis of transfer functions. The impulse response h_k, which defines the stability property of the system, is determined by the poles of the system's poles and zeros since the impulse responses is the inverse z-transform of the transfer function:

$$h_k = \mathcal{Z}^{-1}\{H(z)\} \tag{29.5}$$

Consequently, the stability property is determined by the poles and zeros of $H(z)$. However, we will soon see that only the poles determine the stability.

We will now derive the relation between the stability and the poles by studying the impulse response of the following system:

$$H(z) = \frac{Y(z)}{U(z)} = \frac{bz}{z-p} \tag{29.6}$$

The pole is p. Do you think that this system is too simple as a basis for deriving general conditions for stability analysis? Actually, it is sufficient because we can always think that a given z-transfer function can be partial fractionated in a sum of partial transfer functions or terms each having one pole. Using the superposition principle we can conclude about the stability of the original transfer function.

In the following, cases having of multiple (coinciding) poles will be discussed, but the results regarding stability analysis will be given.

The system given by (29.6) has the following impulse response calculated below. It is assumed that the pole in general is a complex number which may be written on polar form as

$$p = me^{j\theta} \tag{29.7}$$

where m is the magnitude and θ the phase. The impulse response is

$$\begin{align} h_k &= \mathcal{Z}^{-1}\left\{\frac{bz}{z-p}\right\} \tag{29.8}\\ &= \mathcal{Z}^{-1}\left\{\frac{p}{1-pz^{-1}}\right\} \tag{29.9}\\ &= \mathcal{Z}^{-1}\left\{b\sum_{k=0}^{\infty}p^k z^{-k}\right\} \tag{29.10}\\ &= bp^k \tag{29.11}\\ &= b|m|^k e^{jk\theta} \tag{29.12}\end{align}$$

From (29.12) we see that it is the *magnitude* m which determines if the steady state impulse response converges towards zero or not. From (29.12) we can now state the following relations between stability and pole placement (the statements about multiple poles have however not been derived here):

- **Asymptotic stable system:** All poles lie inside (none is on) the unit circle, or what is the same: all poles have magnitude less than 1.
- **Marginally stable system:** One or more poles – but no multiple poles – are on the unit circle.
- **Unstable system:** At least one pole is outside the unit circle. Or: There are multiple poles on the unit circle.

The “stability areas” in the complex plane are shown in Figure 29.2.

Let us return to the question about the relation between the *zeros* and the stability. We consider the following system:

$$H_1(z) = \frac{Y(z)}{U(z)} = \frac{b(z-c)}{z-p} = (z-c)H(z) \tag{29.13}$$

where $H(z)$ is it the “original” system (without zero) which were analyzed above. The zero is c. $H_1(z)$ can be written as

$$\begin{align} H_1(z) &= \frac{bz}{z-p} + \frac{-bc}{z-p} \tag{29.14}\\ &= H(z) - cz^{-1}H(z) \tag{29.15}\end{align}$$

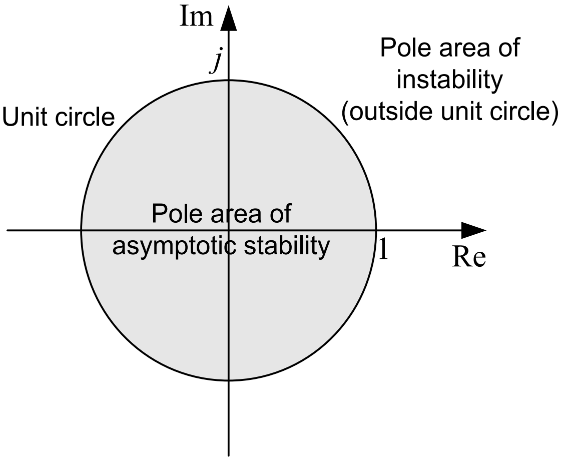


Figure 29.2: The different stability property areas of the complex plane.

The impulse response of $H_1(z)$ becomes

$$h_{1,k} = h_k - ch_{k-1} \tag{29.16}$$

where h_k is the impulse response of $H(z)$. We see that the zero does not influence wether the steady state impulse response converges towards to zero or not. We draw the conclusion that the zeros of the transfer function do not influence the stability of the system.

Example 29.1 *Stability analysis of discrete-time system*

The three responses shown in Figure 29.1 are actually the impulse responses in three systems each having a transfer function on the form

$$\frac{Y(z)}{U(z)} = H(z) = \frac{b_1 z + b_0}{z^2 + a_1 z + a_0} \tag{29.17}$$

The parameters of the systems are given below:

1. Asymptotically stable system: $b_1 = 0.019$, $b_0 = 0.0190$, $a_1 = -1.885$ and $a_0 = 0.923$. The poles are

   $$z_{1,\,2} = 0.94 \pm j0.19 \tag{29.18}$$

   They are shown in Figure 29.3 (the zero is indicated by a circle). The poles are inside the unity circle.

2. Marginally stable system: $b_1 = 0.020$, $b_0 = 0.020$, $a_1 = -1.96$ and $a_0 = 1.00$. The poles are

   $$z_{1,\,2} = 0.98 \pm j0.20 \tag{29.19}$$

   They are shown in Figure 29.3. The poles are on the unity circle.

3. Unstable system: $b_1 = 0.021$, $b_0 = 0.021$, $a_1 = -2.04$ and $a_0 = 1.08$. The poles are

   $$z_{1,\,2} = 1.21 \pm j0.20 \tag{29.20}$$

   They are shown in Figure 29.3. The poles are outside the unity circle.

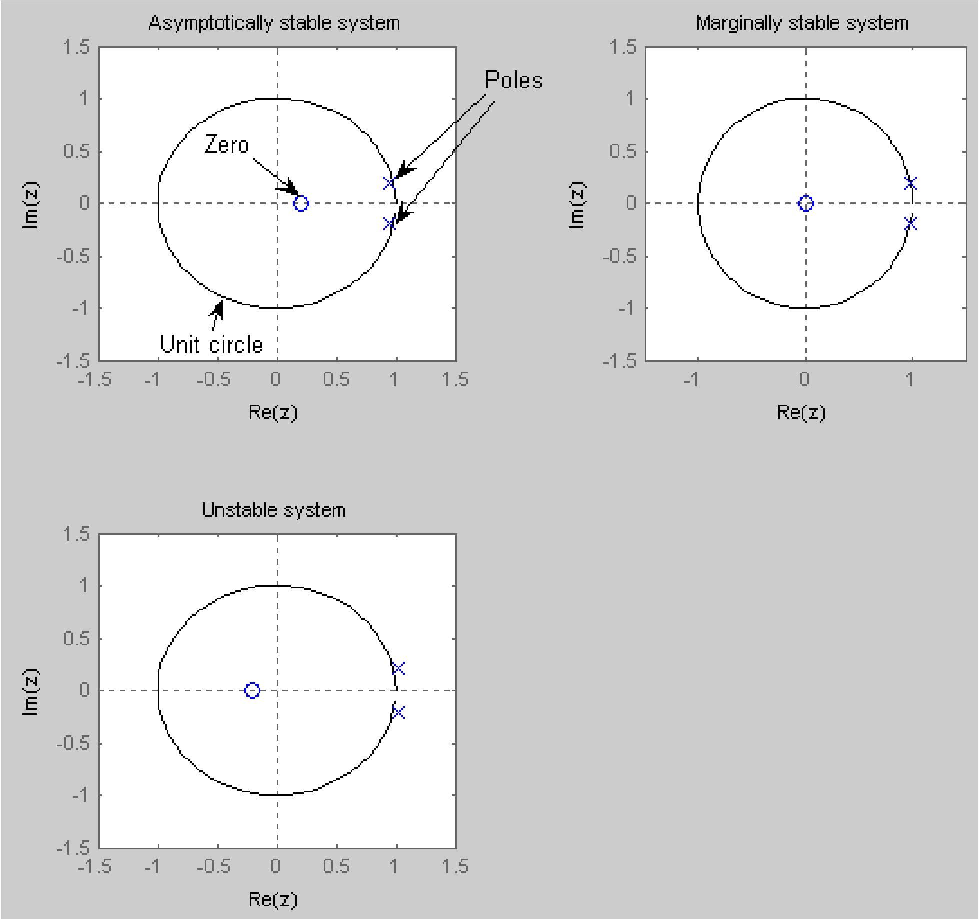


Figure 29.3: Example 29.1: Poles (and zeros) for the three systems each having different stability property

[End of Example 29.1]

29.3 Stability analysis of state space models

Assume that the system has the following state space model:

$$x_{k+1} = Ax_k + Bu_k \tag{29.21}$$

$$y_k = Cx_k + Du_k \tag{29.22}$$

We can determine the stability by finding the corresponding transfer function from u to y, and then calculating the poles from the transfer function, as we did in the previous section. Let us derive the transfer function: Take the $\mathcal{Z}$-transform of eqs. (29.21) – (29.22) to get (I is the identity matrix of equal dimension as of A)

$$zIX(z) = AX(z) + BU(z) \tag{29.23}$$

$$Y(z) = CX(z) + DU(z) \tag{29.24}$$

Solving (29.23) for $X(z)$ gives

$$X(z) = (zI - A)^{-1} BU(z) \tag{29.25}$$

Inserting this $X(z)$ into (29.24) gives

$$Y(z) = \left[C(zI - A)^{-1} B + D\right] U(z) \tag{29.26}$$

So, the transfer function is

$$H(z) = \frac{Y(z)}{U(z)} = C(zI - A)^{-1} B + D \equiv C \frac{\text{adj}(zI - A)}{\det(zI - A)} B + D \tag{29.27}$$

The stability property can now be determined from the poles of this transfer function. The poles are the roots of the characteristic equation:

$$\det(zI - A) = 0 \tag{29.28}$$

But (29.63) actually defines the *eigenvalues* of A, eig(A)! The eigenvalues are the z-solutions to eq. (29.28). Therefore, *the poles are equal to the eigenvalues*, and the relation between stability properties and eigenvalues are the same relation as between stability properties and poles, cf. Section 29.2. To make it clear:

- **Asymptotic stable system:** All eigenvalues (poles) lie inside (none is on) the unit circle, or what is the same: All eigenvalues have magnitude less than 1.
- **Marginally stable system:** One or more eigenvalues – but no multiple eigenvalues – are on the unit circle.
- **Unstable system:** At least one eigenvalue is outside the unit circle. Or: There are multiple eigenvalues on the unit circle.

The "stability areas" in the complex plane are as shown in Figure 29.2.

Example 29.2 *Stability analysis of a state space model*

Given the following state space model:

$$x_{k+1} = \underbrace{\begin{bmatrix} 0.7 & 0.2 \\ 0 & 0.8 \end{bmatrix}}_{A} x_k + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u_k \tag{29.29}$$

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k + \begin{bmatrix} 0 \end{bmatrix} u_k \tag{29.30}$$

It can be shown that the eigenvalues of A are 0.7 and 0.8. Both lies inside the unit circle, and hence the system is asymptotically stable.

[End of Example 29.2]

29.4 *Problems for Chapter 29*

Problem 29.1 *Impulse response*

Given the following transfer function:

$$H(z) = \frac{Y(z)}{U(z)} = \frac{1}{z - 0.5} \tag{29.31}$$

Calculate the first four values of the impulse response. Determine the stability property from the impulse response.

Problem 29.2 *Properties of z-transfer functions*

Determine the stability property of the following transfer functions:

$$H_1(s) = \frac{1}{z - 0.5} \tag{29.32}$$

$$H_2(s) = \frac{1}{z + 0.5} \tag{29.33}$$

$$H_3(s) = \frac{z - 2}{z - 0.5} \tag{29.34}$$

$$H_4(s) = \frac{1}{z - 1} \tag{29.35}$$

$$H_5(s) = \frac{1}{z - 2} \tag{29.36}$$

$$H_6(s) = \frac{1}{(z - 1)^2} \tag{29.37}$$

$$H_7(s) = \frac{1}{z} \tag{29.38}$$

$$H_8(z) = \frac{1}{z^2 - 2.5z + 1} \tag{29.39}$$

Problem 29.3 *Stability of z-transfer function*

Discretizing the continuous-time transfer function

$$H_{\text{con}}(s) = \frac{Y(s)}{U(s)} = \frac{K}{t_c s + 1} \tag{29.40}$$

using the Forward discretization method with time-step t_s yields the following discrete-time transfer function:

$$H_{\text{dis}}(z) = \frac{Y(z)}{U(z)} = \frac{\frac{Kt_s}{t_c}}{z - \left(1 - \frac{t_s}{t_c}\right)} \tag{29.41}$$

For which (positive) values of t_s is the discrete-time system asymptotically stable? (You can assume that t_c is positive.)

Problem 29.4 *Stability of state space model*

Determine the stability property of the following state space model.

$$x_{k+1} = \begin{bmatrix} 1 & 0.5 \\ 0 & 0.9 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k \tag{29.42}$$

29.5 *Solutions to problems for Chapter 29*

Solution to Problem 29.1

The z-transform of the impulse response is equal to the transfer function:

$$\mathcal{Z}\{h_k\} = H(z) = \frac{1}{z-0.5} \tag{29.43}$$

From the z transform pair (26.9),

$$h_k = 0.5^k \tag{29.44}$$

The first four values of the impulse response is

$$\{0.5^0, 0.5^1, 0.5^2, 0.5^3, 0.5^4\} = \{1.0, 0.5, 0.25, 0.125, 0.0625\} \tag{29.45}$$

Obviously, the impulse response converges towards zero as time goes to infinity. Hence, the transfer function is asymptotically stable.

Solution of Problem 29.2

The transfer function

$$H_1(s) = \frac{1}{z-0.5} \tag{29.46}$$

is asymptotically stable since the pole $p = 0.5$ is inside the unit circle.

The transfer function

$$H_2(s) = \frac{1}{z+0.5} \tag{29.47}$$

is asymptotically stable since the pole $p = -0.5$ is inside the unit circle.

The transfer function

$$H_3(s) = \frac{z-2}{z-0.5} \tag{29.48}$$

is asymptotically stable since the pole $p = 0.5$ is inside the unit circle. (The zero is 2, but the stability property is independent of the value of the zero.)

The transfer function

$$H_4(s) = \frac{1}{z-1} \tag{29.49}$$

is marginally stable since the pole $p = 1$ is on the unit circle, and that pole is single.

The transfer function

$$H_5(s) = \frac{1}{z-2} \tag{29.50}$$

is unstable since the pole $p = 2$ is outside the unit circle.

The transfer function

$$H_6(s) = \frac{1}{(z-1)^2} \tag{29.51}$$

is unstable since there are multiple (two) poles, namely $p_{1,2} = 1$, on the unit circle.

The transfer function

$$H_7(s) = \frac{1}{z} \tag{29.52}$$

is asymptotically stable since the pole $p = 0$ is inside the unit circle.

The transfer function

$$H_8(z) = \frac{1}{z^2 - 2.5z + 1} \tag{29.53}$$

is unstable since one of the poles is outside the unit circle. The poles are $p_{1,2} = 0.5,\ 2.0$.

Solution to Problem 29.3

The pole of (29.41) is

$$p = 1 - \frac{t_s}{t_c} \tag{29.54}$$

The system is asymptotically stable if

$$\left| p = 1 - \frac{t_s}{t_c} \right| < 1 \tag{29.55}$$

If

$$1 - \frac{t_s}{t_c} > 0 \tag{29.56}$$

(29.55) becomes

$$1 - \frac{t_s}{t_c} < 1 \tag{29.57}$$

which gives

$$\underline{t_s > 0} \tag{29.58}$$

which is always satisfied.

If

$$1 - \frac{t_s}{t_c} < 0 \tag{29.59}$$

(29.55) becomes

$$-\left(1 - \frac{t_s}{t_c}\right) < 1 \tag{29.60}$$

which gives

$$t_s < \frac{t_c}{2} \tag{29.61}$$

So, the system is asymptotically stable if

$$t_s < \frac{t_c}{2} \tag{29.62}$$

Solution to Problem 29.4

The stability property is determined by the system eigenvalues, which are the roots of the characteristic equation:

$$\det(zI - A) = \det\left(z\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0.5 \\ 0 & 0.9 \end{bmatrix}\right) \tag{29.63}$$

$$= \det\left(\begin{bmatrix} z-1 & 0.5 \\ 0 & z-0.9 \end{bmatrix}\right) \tag{29.64}$$

$$= (z-1)\,(z-0.9) - 0\cdot 0.5 \tag{29.65}$$

$$= (z-1)\,(z-0.9) \tag{29.66}$$

The roots are

$$z_{1,2} = 1,\ 0.9 \tag{29.67}$$

One pole is on the unit circle, and the other pole is inside the unit circle. Thererfore, the system is marginally stable.

Chapter 30

Stability analysis of discrete-time feedback systems

You can analyse the dynamics (frequency response) and the stability of discrete-time feedback systems in the same way as you can analyse the dynamics and stability of continuous-time feedback systems, cf. Chapters 22 and 19, respectively. I assume that you already have knowledge about these topics.

Here is a summary of the relevant differences between analysis of continuous-time and discrete-time feedback systems:

- In the block diagrams etc. every s-transfer function is replaced by an equivalent z-transfer function, using a proper discretization method, e,g, the ZOH method.
- The stability property of any discrete-time system is given by the placement of the z-poles (or eigenvalues) in the complex plane. A discrete-time feedback system is asymptotically stable if all the poles of the closed-loop system lie inside the unit circle. These closed-loop poles are the poles of the tracking transfer function, $H_t(z)$.
- The stability property of a discrete-time feedback system can be analyzed in a Nyquist diagram or a Bode diagram stability based on the frequency response of the loop transfer function $H_l(z)$, which is the product of all the individual transfer functions of the feedback loop. Definitions of crossover frequencies and stability margins are as for continuous-time systems.

Here are some examples:

Example 30.1 *Pole based stability analysis of feedback system*

Assume given a control system where the P controller

$$H_c(z) = K_c \tag{30.1}$$

controls the process (which is actually an integrating process)

$$H_p(z) = \frac{K_i t_s}{z-1} \tag{30.2}$$

We assume that $K_i = 1$ and $t_s = 1$. The loop transfer function becomes

$$H_l(z) = H_p(z)H_c(z) = \frac{K_c}{z-1} = \frac{n_l(z)}{d_l(z)} \tag{30.3}$$

We will calculate the range of values of K_c that ensures asymptotic stability of the control system.

The characteristic polynomial is, cf. (20.4),

$$a(z) = d_l(z) + n_l(z) = z - 1 + K_c \tag{30.4}$$

The pole is

$$p = 1 - K_c \tag{30.5}$$

The feedback system is asymptotically stable if p is inside the unity circle or has magnitude less than one:

$$|p| = |1 - K_c| < 1 \tag{30.6}$$

which is satisfied with

$$0 < K_c < 2 \tag{30.7}$$

Assume as an example that $K_c = 1.5$. Figure 30.1 shows the step response in y due to a step in reference r for this value of K_c.

[End of Example 30.1]

Example 30.2 *Stability analysis in Nyquist diagram*

Given the following continuous-time process transfer function:

$$H_{p,\text{cont}}(s) = \frac{Y(s)}{U(s)} = \frac{K}{\left(\frac{s}{\omega_0}\right)^2 + 2\zeta\frac{s}{\omega_0} + 1} e^{-t_d s} \tag{30.8}$$

with parameter values

$$K = 1;\ \zeta = 1;\ \omega_0 = 0.5 \text{ rad/s};\ t_d = 1 \text{ s} \tag{30.9}$$

The process is controlled by a discrete-time PI-controller having the following z-transfer function, which can be derived by taking the z-transform of the PI control function presented in Ch. 1.3.4.3:

$$H_c(z) = \frac{K_c\left(1 + \frac{T_s}{T_i}\right)z - K_c}{z-1} \tag{30.10}$$

where the time-step (or sampling interval) is

$$t_s = 0.2 \text{ s} \tag{30.11}$$

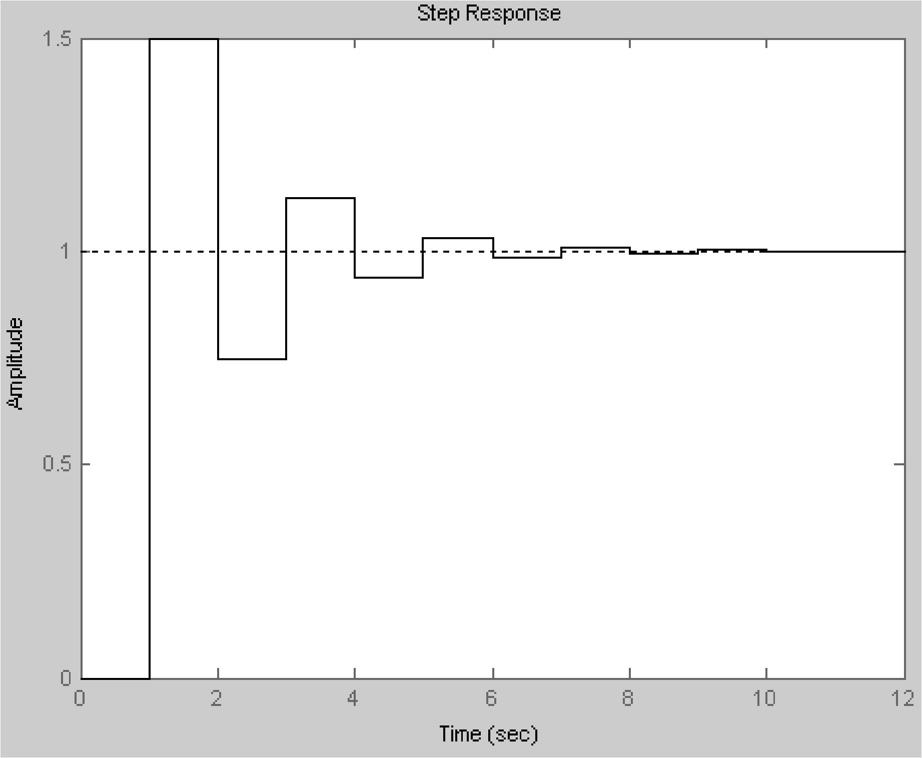


Figure 30.1: Example 30.1: Step resonse in y. There is a step in reference r.

Tuning the controller with the Ziegler-Nichols closed-loop method in a simulator gave the following controller parameter settings:

$$K_c = 2.0;\ T_i = 5.6 \text{ s} \tag{30.12}$$

To perform the stability analysis of the discrete-time control system, $H_{p,\text{cont}}(s)$ is discretized assuming zero order hold (using e.g. Python, or Matlab, or LabVIEW). The result is

$$H_p(z) = \frac{0.001209z + 0.001169}{z^2 - 1.902z + 0.9048} z^{-10} \tag{30.13}$$

The loop transfer function is

$$H_l(z) = H_p(z)H_c(z) \tag{30.14}$$

Figure 30.2 shows the Nyquist plot of $H_l(z)$.

From the Nyquist diagram we read off

$$\omega_{180} = 0.835 \text{ rad/s} \tag{30.15}$$

and

$$\text{Re}H_l(e^{j\omega_{180}T_s}) = -0.558 \tag{30.16}$$

which gives the following gain margin, cf. (22.43),

$$\text{GM} = \frac{1}{|\text{Re}H_l(e^{j\omega_{180}T_s})|} = \frac{1}{|-0.558|} = 1.79 = 5.1 \text{ dB} \tag{30.17}$$

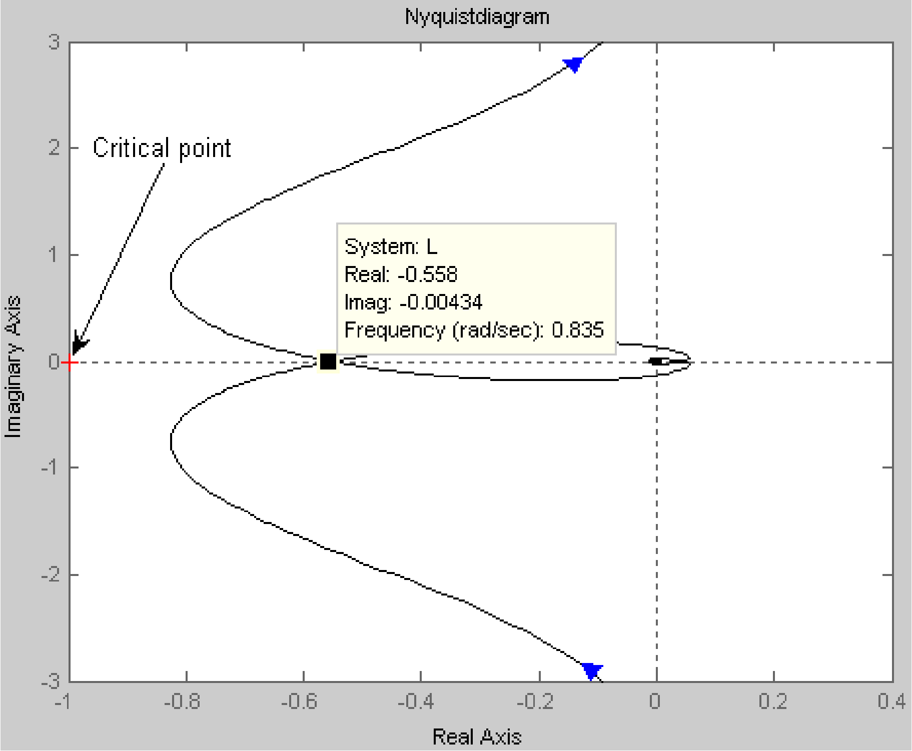


Figure 30.2: Example 30.2: Nyquist diagram of $H_l(z)$.

The phase margin can be found to be

$$\text{PM} = 35^\circ \tag{30.18}$$

Figure 30.3 shows the step response in y_m (unity step in reference r).

[End of Example 30.2]

Example 30.3 *Stability analysis in Bode diagram*

See Example 30.2. Figure 30.4 shows a Bode plot of $H_l(e^{j\omega T_s})$.

The stability margins are shown in the figure. They are

$$\text{GM} = 5.12\text{dB} = 1.80 \tag{30.19}$$

$$\text{PM} = 35.3^\circ \tag{30.20}$$

which is in accordance with Example 30.2.

[End of Example 30.3]

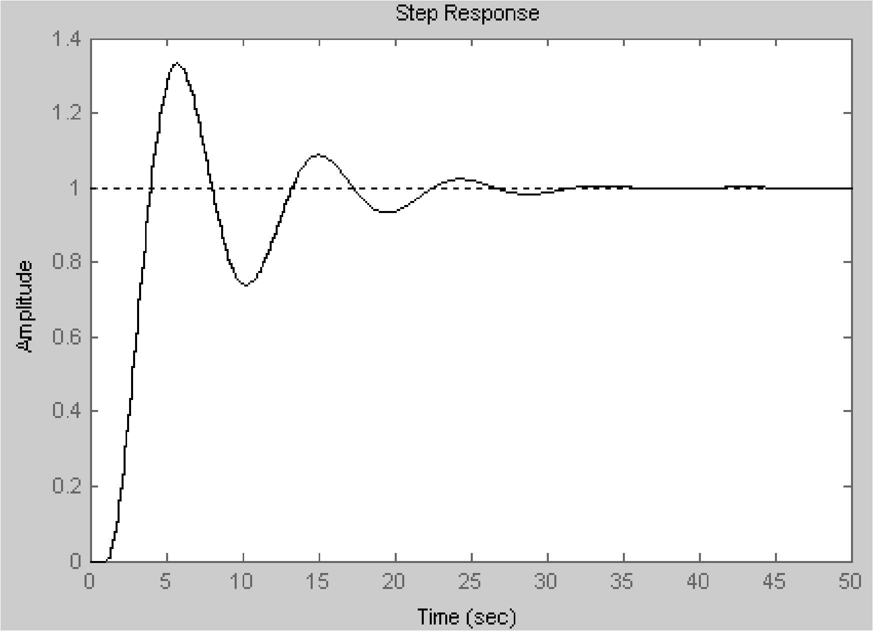


Figure 30.3: Example 30.2: Step response in y after a unity step in reference r.

30.1 *Problems for Chapter 30*

Problem 30.1

Given a feedback control system where the controller (a P controller)

$$H_c(z) = K_c \tag{30.21}$$

controls a process. The transfer function of the combined process and sensor (these systems are in series) is

$$H_{pm}(z) = \frac{z^{-1}}{1 - z^{-1}}$$

Calculate for which values of the controller gain K_c the control system is asymptotically stable.

Problem 30.2

Assume that Figure 30.5 shows the Bode diagram of the loop transfer function of some discrete-time control system.

Read off the stability margins GM and PM, and the crossover frequencies ω_c and ω_{180} in the Bode diagram.

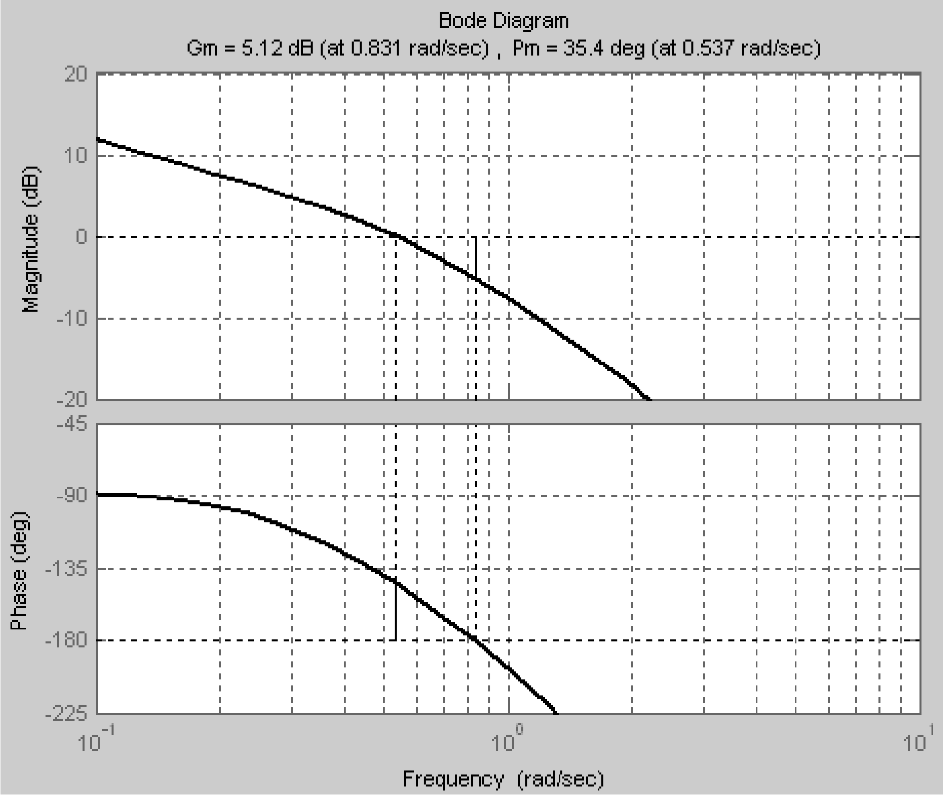


Figure 30.4: Example 30.3: Bode plot of H_l.

30.2 *Solutions to problems for Chapter 30*

Solution to Problem 30.1

The loop transfer function is

$$H_l(z) = H_p(z)H_c(z) = \frac{z^{-1}}{1 - z^{-1}}K_c \tag{30.22}$$

The tracking transfer function is

$$H_t(z) = \frac{H_l(z)}{1 + H_l(z)} = \frac{K_c\frac{z^{-1}}{1-z^{-1}}}{1 + K_c\frac{z^{-1}}{1-z^{-1}}} = \frac{K_c}{z - 1 + K_c} = \frac{K_c}{z - (1 - K_c)} \tag{30.23}$$

The pole of $H_t(z)$ is

$$p = 1 - K_c \tag{30.24}$$

The control system is asymptotically stable if the absolute value of the pole is less than one:

$$|p| = |1 - K_c| < 1 \tag{30.25}$$

which is obtained with

$$0 < K_c < 2 \tag{30.26}$$

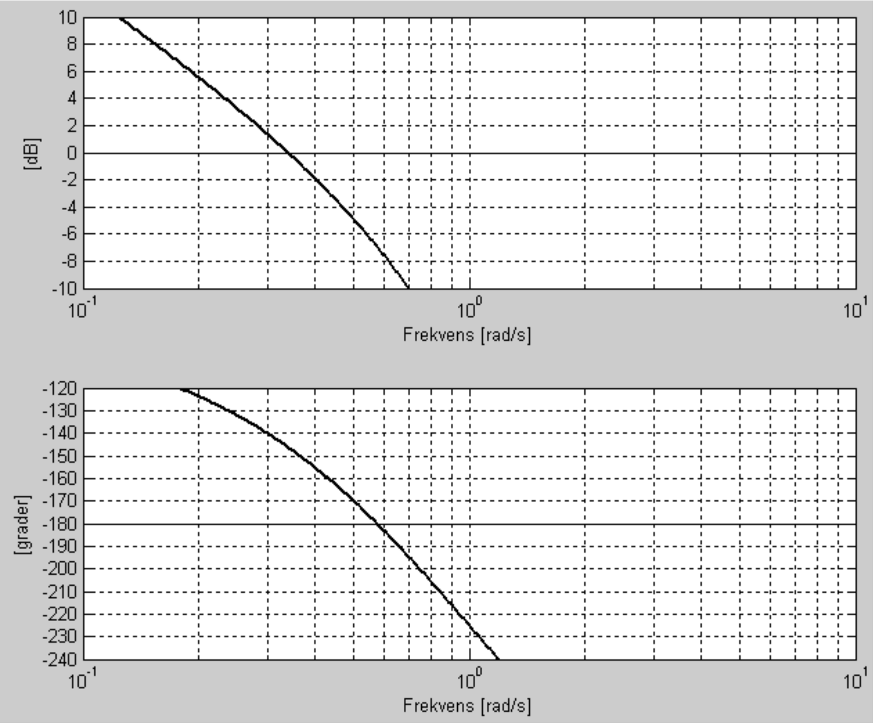


Figure 30.5: Bode diagram.

Solution to Problem 30.2

From the Bode diagram:

$$\text{GM} = 7 \text{ dB} = 2.2 \tag{30.27}$$

$$\text{PM} = 35^\circ \tag{30.28}$$

$$\omega_c = 0.34 \text{ rad/s} \tag{30.29}$$

$$\omega_{180} = 0.58 \text{ rad/s} \tag{30.30}$$

Part VII

STATE ESTIMATION

Chapter 31

Stochastic signals

31.1 Introduction

In practical systems there are signals that vary more or less randomly. For example, measurement signals contain random noise, and process disturbances have some random component. Consequently, control signals and controlled variables, i.e. process output variables, have some random behaviour. The future value of a random signal can not be predicted precisely, i.e. such signals are non-deterministic, while steps, ramps and sinusoids are deterministic. Instead, random signals can be described with statistical measures, typically *expectation (or mean) value* and *standard deviation or variance (standard deviation is square root of variance).*

Random signals may be denoted *stochastic signals.* Characteristics of assumed random process disturbances and random measurement noise are used in design of state estimators with Kalman Filters in Chapter 32.

31.2 How to characterize stochastic signals

31.2.1 Realizations of stochastic processes

A *stochastic process* may be characterized by its mean and standard deviation or variance. The stochastic process can be observed via one or more *realizations* of the process in the form of a *sequence* or time-series of samples, say $\{x_0,\, x_1, x_2 \ldots\}$. Another realization of the same stochastic process will certainly show different sample values, but the mean value and the variance will be almost the same (the longer the realization sequence is, the more equal the mean values and the variances will be). Figure 31.1 shows as an example *two different* realizations (sequences) of the same stochastic process, which in this case is Gaussian (normally) distributed with expectation (mean) value 0 and standard deviation 1. We see that the sequences are not equal.

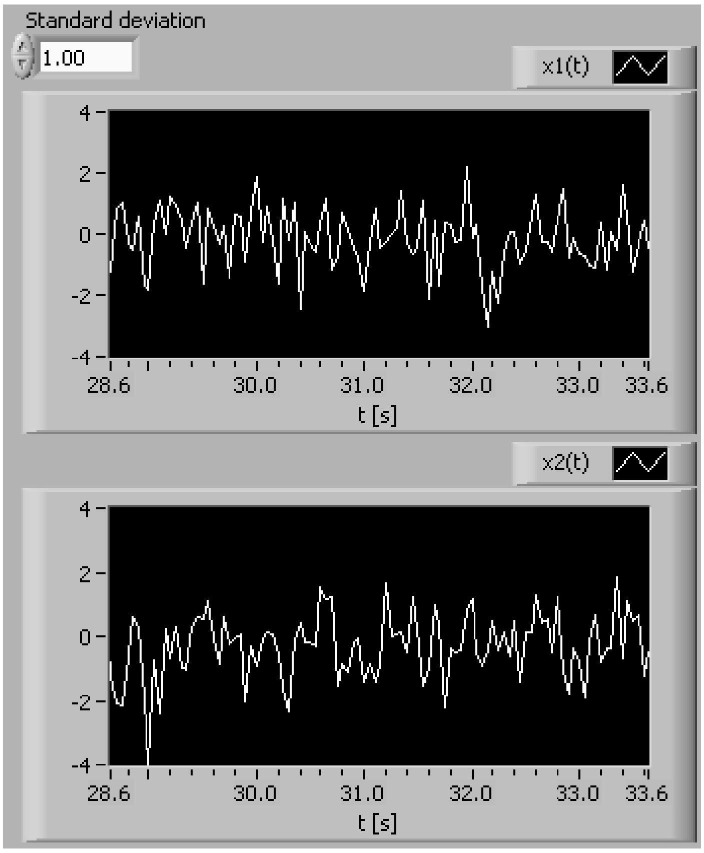


Figure 31.1: Two different realizations of the same stochastic process, which in this case is Gaussian (normally) distributed with expectation value 0 and standard deviation 1 . (Created with the Gaussian White Noise function in LabVIEW.)

31.2.2 Probability distribution of a stochastic variable

As known from statistics a stochastic variable can be described by its *probability distribution function*, PDF. Figure 31.2 shows two commonly used PDFs, the *Normal* (Gaussian) PDF and the *Uniform* PDF.

With the Normal PDF the probability that the variable has value in the range $\{-\sigma, +\sigma\}$ where σ is the standard deviation is approximately 68% (the standard deviation is defined below). With the Uniform PDF the probability that the variable has a value the $\{-A, +A\}$ range is uniform (constant) and the variable can not take any value outside this range.

A stochastic process is *stationary* if the PDF is time independent (constant), or in other words, if the statistical properties are time independent.

31.2.3 The expectation value and the mean value

The *expectation value* , $E(x)$, of the stochastic variable x is the mean (or average) value of x calculated from an infinite number of samples of x. For a limited number N of samples

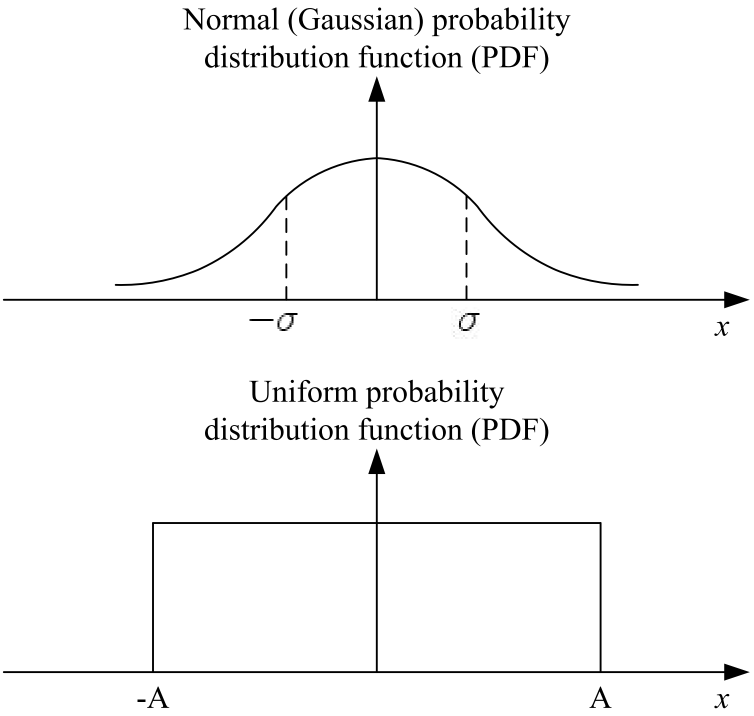

Normal (Gaussian) probability
distribution function (PDF)
−σ
σ
x
Uniform probability
distribution function (PDF)
-A
A
x

Figure 31.2: The Normal (Gaussian) PDF and the Uniform PDF

the mean value can be calculated from

Mean value:

$$m_x = \frac{1}{N} \sum_{k=0}^{N-1} x_k \tag{31.1}$$

Often these two terms (expectation value and mean value) are used interchangeably.

If x is a vector, say

$$x_k = \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} \tag{31.2}$$

then the mean value of x has the form

$$m_x = \begin{bmatrix} m_{x_1} \\ m_{x_2} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum_{k=0}^{N-1} x_{1,k} \\ \\ \frac{1}{N} \sum_{k=0}^{N-1} x_{2,k} \end{bmatrix} \tag{31.3}$$

31.2.4 Variance. Standard deviation

The *variance* of a stochastic variable is the mean or expected value of the squared difference between the value and its mean value:

$$\text{Var}(x) = E\left\{[x_k - m_x]^2\right\} \tag{31.4}$$

The variance can be calculated from a sequence of samples as follows:

$$\mathrm{Var}(x) = \frac{1}{N-1} \sum_{k=0}^{N-1} [x_k - m_x]^2 \tag{31.5}$$

(Statistically it is better to divide by $N-1$ and not by N since the estimate of the statistical variance becomes unbiased using $N-1$.) The variance is some times denoted the *power* of the signal.

The *standard deviation* may give a more meaningful value of the variation of a signal. The standard deviation is the square root of the variance:

$$\sigma = \sqrt{\mathrm{Var}(x)} \tag{31.6}$$

In many situations σ^2 is used as a symbol for the variance.

31.2.5 Auto-covariance. Cross-covariance

Sometimes it is useful to express how a stochastic variable, say x_k, varies along the time-index axis (k). This type of variance can be expressed by the *auto-covariance* :

Auto-covariance:

$$R_x(L) = E\{[x_{k+L} - m_x][x_k - m_x]\} \tag{31.7}$$

where L is the *lag*. Note that the argument of the auto-covariance function is the lag L. Figure 31.3 shows $R_x(L)$ for a signal x where the covariance decreased as the lag increases (this is typical). As indicated in Figure 31.3 the auto-covariance usually has a peak value at lag $L = 0$.

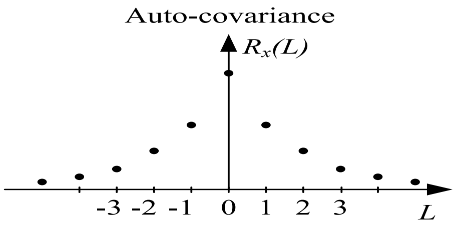


Figure 31.3: The auto-covariance for a signal x where the covariance decreased as the lag increases (this is typical).

If $L = 0$ the auto-covariance becomes the variance:

$$\begin{aligned} R_x(0) &= E\{[x_{k+0} - m_x][x_k - m_x]\} && (31.8) \\ &= E\{[x_k - m_x]^2\} = \mathrm{Var}(x) = \sigma^2 && (31.9) \end{aligned}$$

In some applications x is a *vector*, say

$$x_k = \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} \tag{31.10}$$

What does the auto-covariance look like in this case? For simplicity, assume that each of the four variables above have zero mean. The auto-covariance then becomes

$$\begin{align} R_x(L) &= E\{[x_{k+L}][x_k]^T\} \tag{31.11} \\ &= E\left\{ \begin{bmatrix} x_{1,k+L} \\ x_{2,k+L} \end{bmatrix} \begin{bmatrix} x_{1,k} & x_{2,k} \end{bmatrix} \right\} \tag{31.12} \\ &= \begin{bmatrix} E[x_{1,k+L}x_{1,k}] & E[x_{1,k+L}x_{2,k}] \\ E[x_{2,k+L}x_{1,k}] & E[x_{2,k+L}x_{2,k}] \end{bmatrix} \tag{31.13} \end{align}$$

If $L = 0$, the auto-covariance becomes

$$R_x(0) = \begin{bmatrix} \underbrace{E\left\{[x_{1,k}]^2\right\}}_{=\mathrm{Var}(x_1)} & E[x_{1,k}x_{2,k}] \\ E[x_{2,k}x_{1,k}] & \underbrace{E\left\{[x_{2,k}]^2\right\}}_{=\mathrm{Var}(x_2)} \end{bmatrix} \tag{31.14}$$

Hence, the *variances are on the diagonal.*

The *cross-covariance* between two *different* scalar signals, say x and y, is

Cross-covariance:

$$R_{xy}(L) = E\{[x_{k+L} - m_x][y_k - m_y]\} \tag{31.15}$$

The cross-covariance can be *estimated* from sequences of sample values of x and y of length N with

$$\begin{align} &R_{xy}(L) \\ &= S \sum_{k=0}^{N-1-|L|} [x_{k+L} - m_x][y_k - m_y], \;\; L = 0, 1, 2, \ldots \tag{31.16} \\ &= S \sum_{k=1}^{N-1-|L|} [y_{k-L} - m_y][x_k - m_x] = R_{yx}(-L), \;\; L = -1, -2, \ldots \tag{31.17} \end{align}$$

where S is a scaling factor which is defined below:

- **Raw estimate:**

$$S = 1 \tag{31.18}$$

- **Normalized estimate**:

$$S = \frac{1}{R_{xy}(0) \text{ in the raw estimate}} \tag{31.19}$$

This gives $R_{xy}(0) = 1$.

- **Unbiased estimate:**

  $$S = \frac{1}{N-L} \tag{31.20}$$

  This calculates $R_{xy}(L)$ as an average value of the product $[x(k+L)-m_x][y(k)-m_y]$. However, $R_{xy}(L)$ may be very "noisy" if L is large since then the summation is calculated with only a few additive terms (it is assumed that there are noise or random components in x and/or y).

- **Biased estimate:**

  $$S = \frac{1}{N} \tag{31.21}$$

  With this option $R_{xy}(L)$ is not an average value of the product $[x(k+L)-m_x][y(k)-m_y]$ since the sum of terms is divided by N no matter how many additive terms there are in the summation. Although this makes $R_{xy}(L)$ become "biased" it reduces the "noise" in $R_{xy}(L)$ because the "noisy" terms are weighed by $1/N$ instead of $1/(N-L)$. Unless you have reasons for some other selection, you may use biased estimate as the default option.

Correlation (auto/cross) is the same as covariance (auto/cross) except that the mean value, as m_x, is removed from the formulas. Hence, the cross-correlation is, cf. (31.15),

$$r_{xy}(L) = E\{x_{k+L}y_k\} \tag{31.22}$$

31.3 White and coloured noise

31.3.1 White noise

An important type of stochastic signals are the so-called *white noise signals (or processes)*. "White" is because in some sense white noise contains equally much of all frequency components, analogously to white light which contains all colours. White noise has zero mean value:

Mean value of white noise:

$$m_x = 0 \tag{31.23}$$

There is no co-variance or relation between sample values at different time-indexes, and hence the auto-covariance is zero for all lags L except for $L=0$. Thus, the auto-covariance is the pulse function shown in Figure 31.4.

Mathematically the auto-covariance function of white noise is

$$R_x(L) = \mathrm{Var}(x)\delta(L) = \sigma^2\delta(L) = V\delta(L) \tag{31.24}$$

Here, the short-hand symbol V has been introduced for the variance. $\delta(L)$ is the *unit pulse* defined as follows:

Unit pulse:

$$\delta(L) = \begin{cases} 1 \text{ when } L = 0 \\ 0 \text{ when } L \neq 0 \end{cases} \tag{31.25}$$

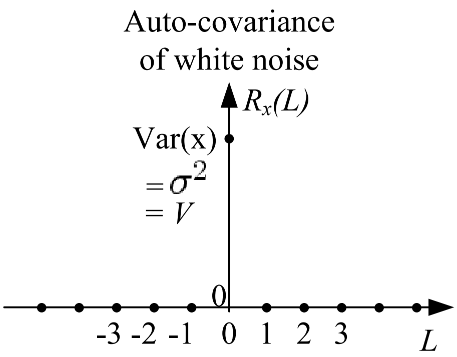


Figure 31.4: White noise has auto-correlation function like a pulse function.

White noise is an important signal in estimation theory because the random noise which is always present in measurements, can be represented by white noise. For example, the variance of the assumed white measurement noise is used as an input parameter in the Kalman Filter design, cf. Chapter 32.

If you calculate the auto-covariance of a white noise sequence of finite length, the auto-covariance function will not be exactly as the ideal function shown in Figure 31.4, but the main characteristic showing a relatively large value at lag $L = 0$ is there.

Example 31.1 *White noise*

Figure 31.5 shows a simulated white noise signal x and its auto-covariance $R_x(L)$ (normalized) calculated from the most recent $N = 50$ samples of x.[1]

The white noise characteristic of the signal is clearly indicated by $R_x(L)$.

[End of Example 31.1]

31.3.2 Coloured noise

As opposite to white noise, *coloured noise* does not vary completely randomly. In other words, there is a co-variance between the sample values at different time-indexes. As a consequence, the auto-covariance $R_x(L)$ is non-zero for lags $L \neq 0$. $R_x(L)$ will have a maximum value at $L = 0$, and $R_x(L)$ will descrease for increasing L.

You may generate coloured noise from white noise by sending the white noise through a dynamic system, typically a lowpass filter. Such a system is denoted *shaping filter*. The output signal of the shaping filter will be coloured noise. You can tune the colour of the coloured noise by adjusting the parameters of the shaping filter.

Example 31.2 *Coloured noise*

[1] Implemented in LabVIEW.

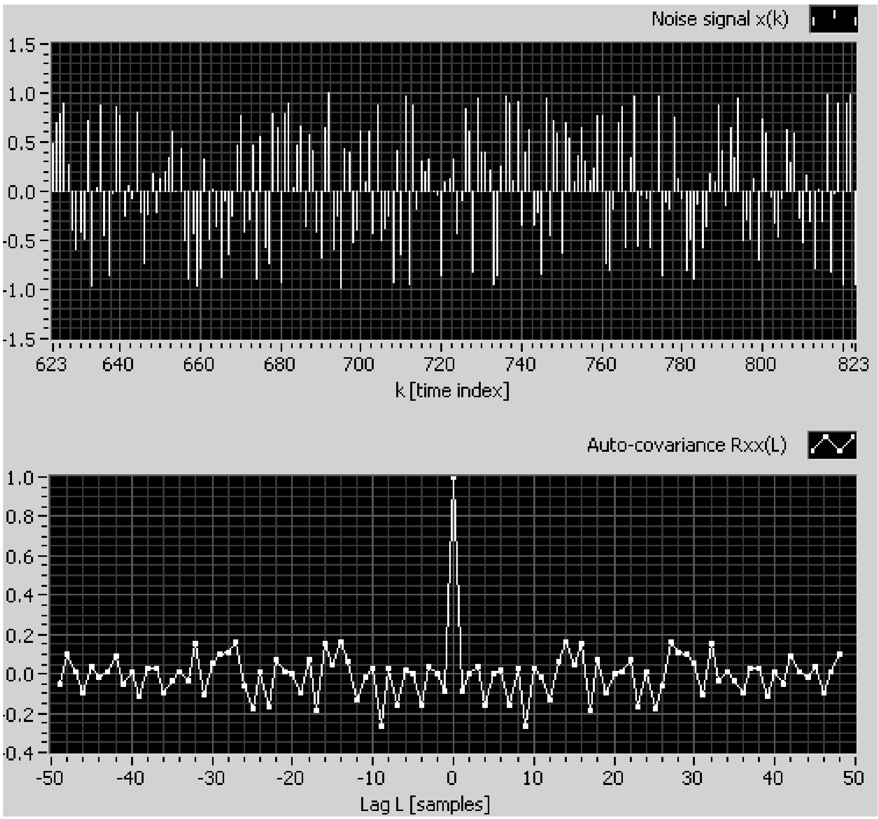
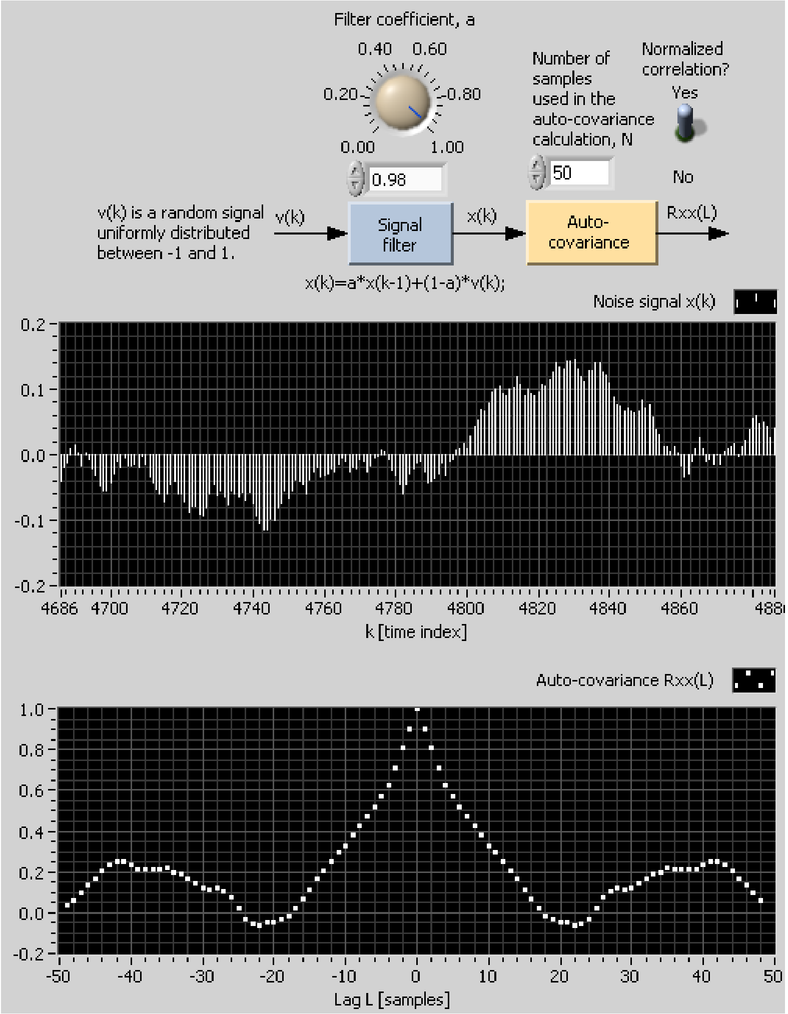


Figure 31.5: Example 31.1: Simulated white noise signal x and its auto-covariance $R_x(L)$ (normalized) calculated from the most recent $N = 50$ samples of x.

Figure 31.6 shows a simulated coloured noise signal x and its auto-covariance $R_x(L)$ (normalized) calculated from the most recent $N = 50$ samples of x.[2]

[2] Implemented in LabVIEW.



Filter coefficient, a: 0.98

Number of samples used in the auto-covariance calculation, N: 50

Normalized correlation? Yes

x(k)=a*x(k-1)+(1-a)*v(k);

v(k) is a random signal uniformly distributed between -1 and 1.

Figure 31.6: Example 31.1: Simulated coloured noise signal x and its auto-covariance $R_x(L)$ (normalized) calculated from the most recent $N = 50$ samples of x.

The coloured noise is the output of this shaping filter:

$$x_k = ax_{k-1} + (1-a)v_k \tag{31.26}$$

which is a discrete-time first order lowpass filter. The filter input v_k is white noise. The filter parameter is $a = 0.98$. (If the filter parameter is 0 the filter performs no filtering, and the output is just white noise.)

The coloured noise characteristic of the signal is shown both in the plot of the signal x_k in the upper diagram of Figure 31.6 and in the auto-covariance $R_x(L)$ shown in the lower diagram of Figure 31.6.

[End of Example 31.2]

31.4 Propagation of mean value and co-variance through static systems

If a stochastic ("random") signals excites a static or dynamic system, the mean value and the co-variance of the output signal is different from those of the input. In this section we will concentrate on *static* systems. The results are useful e.g. in calculating the system gain needed to obtain a random signal of a specified variannce when the source signal is a random signal of fixed variance.

The theory of the propagation of mean value and co-variance through *dynamic* systems is certainly important if you are going to analyse and design signal filters, controllers and state estimators assuming they are excited by random signals. However, it is my experience that this theory is not needed to be able to use the tools that exist for such applications (e.g. the Kalman Filter for state estimation). Therefore, I have omitted the topic of propagation of mean value and co-variance through dynamic systems in this book.

Assume given the following static linear system:

$$y_k = Gv_k + C \tag{31.27}$$

where v is a stationary stochastic input signal with mean value m_v and co-variance $R_v(L)$. y is the output of the system. G is the gain of the system, and C is a constant. In a multivariable system G is a matrix and C is a vector, but in the following we will assume that G and C are scalars, which is the most usual case.

Let us calculate the mean value and the auto-covariance of the output y. The mean value becomes

$$\begin{align} m_y = E[y_k] &= GE\left[v_k\right] + C \tag{31.28} \\ &= Gm_v + C \tag{31.29} \end{align}$$

The auto-covariance of the output becomes

$$\begin{align} R_y(L) &= E\{[y_{k+L} - m_y][y_k - m_y]\} \tag{31.30} \\ &= E\left\{([Gv_{k+L} + C] - [Gm_v + C])\left([Gv_k + C] - [Gm_v + C]\right)\right\} \tag{31.31} \\ &= E\left\{(Gv_{k+L} - Gm_v)\left(Gv_k - Gm_v\right)\right\} \tag{31.32} \\ &= E\left\{(G\left[v_{k+L} - m_v\right])\left(G\left[v_k - m_v\right]\right)\right\} \tag{31.33} \\ &= G^2 \underbrace{E\left\{([v_{k+L} - m_v])\left([v_k - m_v]\right)\right\}}_{=R_v(L)} \tag{31.34} \\ &= G^2 R_v(L) \tag{31.35} \end{align}$$

If the system (31.27) is multivariable, that is, if v and y are vectors, G is a matrix and C is a vector. In this case we will get

$$R_y(L) = GR_v(L)G^T \tag{31.36}$$

Let us sum it up: For a scalar system (31.27):

- Mean of output of stochastic static system:

$$m_y = Gm_v + C \tag{31.37}$$

- Co-variance of output of stochastic static system:

$$R_y(L) = G^2 R_v(L) \tag{31.38}$$

- The variance, which is equal to $R_y(0)$:

$$\sigma_y^2 = G^2 \sigma_v^2 \tag{31.39}$$

- The standard deviation:

$$\sigma_y = G\sigma_v \tag{31.40}$$

Example 31.3 *Mathematical operations to achieve specified output mean and variance*

Assume that you have signal generator available in a computer tool that can generate a white noise signal v having mean $m_v = 0$ and variance $\sigma_v^2 = 1$, and that you want to generate a signal y of mean $m_y = M$ and variance $\sigma_y^2 = V$. Find proper mathematical operations on v that create this y.

The gain G can be calculated from (31.39):

$$G = \sqrt{\frac{\sigma_y^2}{\sigma_v^2}} = \sqrt{\frac{V}{1}} = \sqrt{V} = \sigma_y \tag{31.41}$$

The constant C can be calculated from (31.37):

$$C = m_y - Gm_v = M - G \cdot 0 = M \tag{31.42}$$

So, the mathematical operation is

$$y_k = Gv_k + C = \sqrt{V} \cdot v_k + M = \sigma_y v_k + M \tag{31.43}$$

In words: Multiply the input by the specified standard deviation and add the specified mean value to this result.

[End of Example 31.3]

31.5 *Problems for Chapter 31*

Problem 31.1 *Statistical measures of a signal*

Given the following sequence of measurement values:

$$\{x_k\} = \{x_0,\ x_1,\ x_2\} = \{0.73,\ 1.23,\ 0.89\} \tag{31.44}$$

1. Calculate the mean value, m_x.
2. Calculate the variance, σ_x^2.
3. Calculate the standard deviation, σ_x.
4. Calculate the auto-covariance, $R_x(L)$ with $L = 0$ and $L = 1$ for the following options:
   (a) Raw estimate
   (b) Normalized estimate
   (c) Unbiased estimate
   (d) Biased estimate

Problem 31.2 *Characteristics of uniformly distributed noise*

Given a random sequence, $\{x_k\}$, uniformly distributed between $-A$ and $+A$.

1. Calculate the mean value or expectation value m_x using this formula:

   $$m_x = \int_{-\infty}^{\infty} xP(x)dx = \int_{-A}^{A} xP(x)dx \tag{31.45}$$

   where $P(x)$ is the probability density. For a uniformly distributed signal,

   $$P(x) = \frac{1}{2A} \tag{31.46}$$

   since the area under the $P(x)$-curve must be 1 (the width of the curve is 2A, so the height is $1/(2A)$ for the area to be 1).

2. Show that the standard deviation of the sequence is

   $$\sigma_x = \frac{A}{\sqrt{3}} \tag{31.47}$$

Problem 31.3 *Auto covariance of white noise*

Draw by hand the principal auto covariance of white noise with variance 4. What is the standard deviation σ_x of this signal?

Problem 31.4 *Shaping filter*

The following discrete-time first order lowpass filter is one example of a shaping filter:

$$x_k = ax_{k-1} + (1-a)v_k \tag{31.48}$$

which is a discrete-time first order lowpass filter. x is the filter output, and v_k is the filter input, which is assumed to be white noise.

1. Explain (no calculations are needed) why $a = 0$ makes the output become white, and hence, there is no "shaping" through the filter.
2. Explain (no calculations are needed) why a between 0 and 1 makes the output become "coloured".

Problem 31.5 *Generating a random signal*

Assume that you in a computer tool are to generate a random signal y having mean value 3 and variance 4, and that the tool has a function that generates a random signal u of mean value 0 and variance 1. How can you obtain y from u? (Express y as a mathematical function of u.)

Problem 31.6 *Programming a uniformly distributed random signal in Python*

In Python, the random generator numpy.random.uniform() can be used to generate a signal as a sequence of n uniformly distributed random values between amplitude limits $\pm A$ with zero mean value:

signal = numpy.random.uniform(–A, A, n)

The relation between the standard deviation and the amplitude is given by (31.47).

Make a Python program that generates and plots a uniformly distributed random signal, x, with the following specifications:

- Sampling time: 0.1 s.
- Start time: 0 s.
- Stop time: 10 s.
- Standard deviation: 0.1.
- Mean value: 1.

In the same diagram, plot the mean(x) and mean(x)$\pm A$.

What is the value of A?

31.6 *Solutions to problems for Chapter 31*

Solution to Problem 31.1

$$\{x_k\} = \{x_0, x_1, x_2\} = \{0.73, 1.23, 0.89\} \tag{31.49}$$

1. Mean value:

$$m_x = \frac{1}{N}\sum_{k=0}^{N-1} x_k \tag{31.50}$$

$$= \frac{1}{3}\left[x_0 + x_1 + x_2\right] \tag{31.51}$$

$$= \frac{1}{3}\left[0.73 + 1.23 + 0.89\right] \tag{31.52}$$

$$= 0.95 \tag{31.53}$$

2. Variance:

$$\sigma_x^2 = \frac{1}{N-1}\sum_{k=0}^{N-1} \left[x_k - m_x\right]^2 \tag{31.54}$$

$$\frac{1}{3-1}\sum_{k=0}^{3-1} \left[x_k - m_x\right]^2 \tag{31.55}$$

$$= \frac{1}{2}\left\{\begin{array}{l} \left[x_0 - m_x\right]^2 \\ + \left[x_1 - m_x\right]^2 \\ + \left[x_2 - m_x\right]^2 \end{array}\right\} \tag{31.56}$$

$$= \frac{1}{2}\left\{\begin{array}{l} \left[0.73 - 0.95\right]^2 \\ + \left[1.23 - 0.95\right]^2 \\ + \left[1.13 - 0.95\right]^2 \end{array}\right\} \tag{31.57}$$

$$= 0.0652 \tag{31.58}$$

3. Standard deviation:

$$\sigma_x = \sqrt{\sigma_x^2} = \sqrt{0.0652} = 0.255 \tag{31.59}$$

4. Calculate the auto-covariance, $R_x(L)$ with $L = 0$ and $L = 1$ for the following options:

   (a) Raw estimate of auto-covariance: General formula:

$$R_x(L) = \sum_{k=0}^{N-1-|L|} \left[x_{k+L} - m_x\right]\left[x_k - m_x\right] = R_x(L)_{\text{Raw}} \tag{31.60}$$

$L = 0$:

$$\begin{aligned}
R_x(0) &= \sum_{k=0}^{3-1-|0|=2} [x_{k+0} - m_x][x_k - m_x] && (31.61)\\
&= \sum_{k=0}^{2} [x_k - m_x]^2 && (31.62)\\
&= [x_0 - m_x]^2 && (31.63)\\
&+ [x_1 - m_x]^2 && (31.64)\\
&+ [x_2 - m_x]^2 && (31.65)\\
&= [0.73 - 0.95]^2 && (31.66)\\
&+ [1.23 - 0.95]^2 && (31.67)\\
&+ [0.89 - 0.95]^2 && (31.68)\\
&= 0.1304 && (31.69)\\
&= R_x(0)_{\text{Raw}} && (31.70)
\end{aligned}$$

$L = 1$:

$$\begin{aligned}
R_x(1) &= \sum_{k=0}^{3-1-|1|=1} [x_{k+1} - m_x][x_k - m_x] && (31.71)\\
&= [x_1 - m_x][x_0 - m_x] && (31.72)\\
&+ [x_2 - m_x][x_1 - m_x] && (31.73)\\
&= [1.23 - 0.95][0.73 - 0.95] && (31.74)\\
&+ [0.89 - 0.95][1.23 - 0.95] && (31.75)\\
&= -0.0784 && (31.76)\\
&= R_x(1)_{\text{Raw}} && (31.77)
\end{aligned}$$

(b) Normalized estimate: General formula:

$$\begin{aligned}
R_x(L) &= \frac{1}{R_x(0)_{\text{Raw}}} \sum_{k=0}^{N-1-|L|} [x_{k+L} - m_x][x_k - m_x]\\
&= \frac{1}{R_x(0)_{\text{Raw}}} R_x(L)_{\text{Raw}} && (31.78)
\end{aligned}$$

$L = 0$:

$$R_x(0) = \frac{1}{R_x(0)_{\text{Raw}}} R_x(0)_{\text{Raw}} = 1 \qquad (31.79)$$

$L = 1$:

$$R_x(1) = \frac{1}{R_x(0)_{\text{Raw}}} R_x(1)_{\text{Raw}} = \frac{1}{0.1304}(-0.0784) = -0.6012 \qquad (31.80)$$

(c) Unbiased estimate: General formula:

$$R_x(L) = \frac{1}{N-L} R_x(L)_{\text{Raw}} \qquad (31.81)$$

$L = 0$:

$$R_x(0) = \frac{1}{3-0} R_x(0)_{\text{Raw}} = \frac{1}{3} \cdot 0.1304 = 0.0435 \tag{31.82}$$

$L = 1$:

$$R_x(1) = \frac{1}{3-1} R_x(1)_{\text{Raw}} = \frac{1}{2} (-0.0784) = -0.0392 \tag{31.83}$$

(d) Biased estimate: General formula:

$$R_x(L) = \frac{1}{N} R_x(L)_{\text{Raw}} \tag{31.84}$$

$L = 0$:

$$R_x(0) = \frac{1}{3} R_x(0)_{\text{Raw}} = \frac{1}{3} \cdot 0.1304 = 0.0435 \tag{31.85}$$

$L = 1$:

$$R_x(1) = \frac{1}{3} R_x(1)_{\text{Raw}} = \frac{1}{3} (-0.0784) = -0.0261 \tag{31.86}$$

Solution to Problem 31.2

1. Mean value:

$$\begin{aligned} m_x &= \int_{-A}^{A} x P(x) dx && (31.87) \\ &= \int_{-A}^{A} x \cdot \frac{1}{2A} \cdot dx && (31.88) \\ &= \frac{1}{2A} \left[\frac{x^2}{2} \right]_{-A}^{A} && (31.89) \\ &= \frac{1}{2A} \left[\frac{A^2}{2} - \frac{(-A)^2}{2} \right] && (31.90) \\ &= 0 && (31.91) \end{aligned}$$

2. Variance:

$$\begin{aligned} \sigma_x^2 &= \int_{-A}^{A} (x - m_x)^2 P(x) dx && (31.92) \\ &= \int_{-A}^{A} (x-0)^2 \frac{1}{2A} dx && (31.93) \\ &= \frac{1}{2A} \int_{-A}^{A} x^2 dx && (31.94) \\ &= \frac{1}{2A} \left[\frac{x^3}{3} \right]_{-A}^{A} && (31.95) \\ &= \frac{1}{2A} \left[\frac{A^3}{3} - \frac{(-A)^3}{3} \right] && (31.96) \\ &= \frac{A^2}{3} && (31.97) \end{aligned}$$

Standard deviation:

$$\sigma_x = \frac{A}{\sqrt{3}} \tag{31.98}$$

Solution to Problem 31.3

See Figure 31.7.

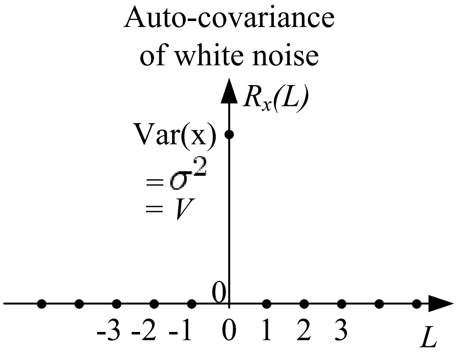


Figure 31.7: Auto-covariance

The standard deviation is

$$\sigma_x = \sqrt{\sigma_x^2} = \sqrt{4} = 2 \tag{31.99}$$

Solution to Problem 31.4

1. With $a = 0$ the filter model is

$$x_k = v_k \tag{31.100}$$

   So, if the input is white, the output is white.

2. If a is between 0 and 1, the filter output $x(k)$ depends not only on the input $v(k)$ but also on $x(k-1)$ which is x at the previous time step. Therefore, $x(k)$ will not vary purely randomly (it will not become purely white) – it is "coloured".

Solution to Problem 31.5

y expressed as a function of u:

$$y = Gu + C \tag{31.101}$$

where G and C are calculated from the following formulas:

$$m_y = Gm_u + C \tag{31.102}$$

and

$$\sigma_y^2 = G^2\sigma_u^2 \tag{31.103}$$

where

$$m_u = 0 \tag{31.104}$$

$$m_y = 3 \tag{31.105}$$

$$\sigma_u^2 = 1 \tag{31.106}$$

$$\sigma_y^2 = 4 \tag{31.107}$$

Now, from (31.103) we get

$$G = \sqrt{\frac{\sigma_y^2}{\sigma_u^2}} = \sqrt{\frac{4}{1}} = 2 \tag{31.108}$$

and from (31.102) we get

$$C = m_y - Gm_u = 3 - 2 \cdot 0 = 3 \tag{31.109}$$

Solution to Problem 31.6

From (31.47) we get amplitude:

$$A = \sqrt{3}\sigma_x = \sqrt{3} \cdot 0.1 = 0.173 \tag{31.110}$$

Program 31.1 generates and plots x.

http://techteach.no/control/python/prog_plot_sim_uniform.py

Listing 31.1: prog_plot_sim_uniform.py

```
import numpy as np
import matplotlib.pyplot as plt

# %% Generating time signal:

Ts = 0.1
t_start = 0
t_stop = 10
t_array = np.arange(t_start, t_stop+Ts, Ts)
n = len(t_array)

# %% Parameters of uniform():

mean_x = 1
sigma_x = 0.1
A = np.sqrt(3)*sigma_x

# %% Generating array of noise:

x = mean_x + np.random.uniform(-A, A, n)

# %% Plotting:

plt.close('all')
fig1 = plt.figure(num=1, figsize=(12, 9))

plt.plot(t_array, x, 'b-o')
plt.plot(t_array, np.zeros(n) + mean_x, 'r--')
plt.plot(t_array, np.zeros(n) + mean_x + A, 'g--')
plt.plot(t_array, np.zeros(n) + mean_x - A, 'g--')
plt.grid()
plt.xlabel('t [s]')
plt.legend(labels=('x', 'mean_x',
                   'mean_x - A', 'mean_x + A'))
plt.show()

plt.savefig('plot_unif_noise.pdf')
```

The signal x is plotted in Figure 31.8

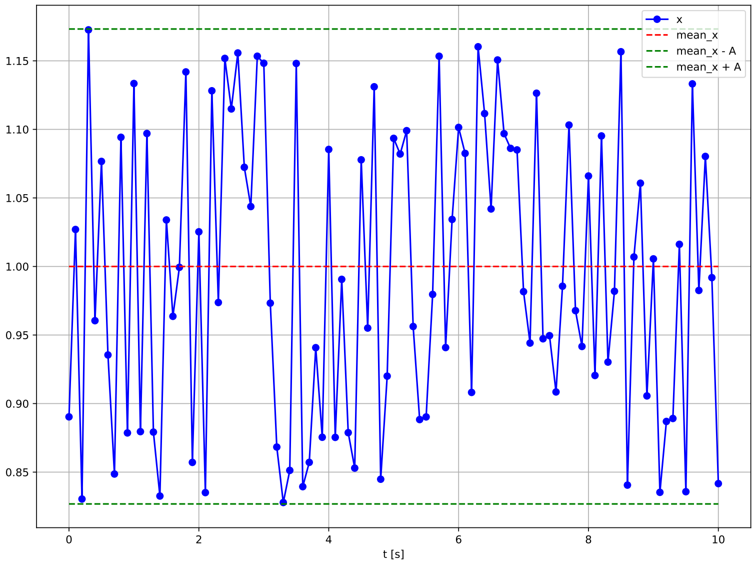


Figure 31.8: Uniformly distributed random signal, x.

Chapter 32

State estimation with Kalman Filter

32.1 Introduction

The Kalman Filter is a commonly used algorithm to estimate the values of unknown state variables of a dynamic system. The Kalman Filter algorithm was developed by Rudolf E. Kalman (Kalman 1960). There is a continuous-time version of the Kalman Filter and several discrete-time versions. Here the *predictor-corrector* version of the discrete-time Kalman Filter will be described. I think this version is the most commonly used version.

Figure 32.1 illustrates the principle of the Kalman Filter. The Kalman Filter is basically a process simulator and a sensor simulator which uses a process model and a sensor model, respectively, to estimate the states and the measurements. The difference between the real measurement and the simulated (estimated) measurement is used to correct the present estimate. That difference is called the innovation variable, or the innovation "process". The correction is proportional to the innovation variable. The proportional gain, which is actually a matrix of time-varying elements, is called the Kalman Filter gain, K. Overall, the Kalman Filter uses process knowledge in terms of model and measurements to calculate the state estimate.

Although the Kalman Filter is a *state* estimator, also process disturbances and process parameters can be estimated. The clue is to model the disturbances or parameters as state variables, and then the Kalman Filter algorithm can be used to estimate them.

Since a Kalman Filter is a software algorithm that produces a "soft measurement", the Kalman Filter is (an example) of a soft sensor.

Why can state estimates be useful?

- ***Monitoring***: State estimates can provide valuable information about important variables in a physical process, for example feed composition to a reactor, environmental forces acting on a ship, load torques acting on a motor, etc.

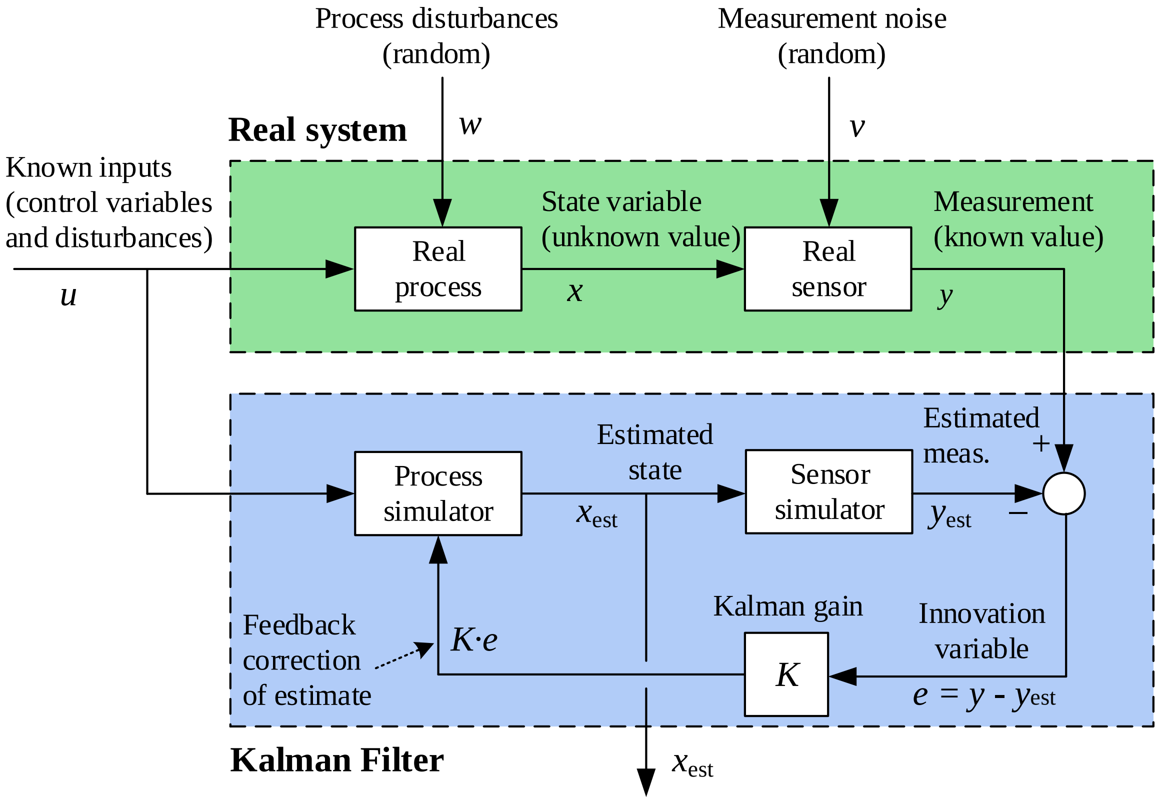


Figure 32.1: The principle of the Kalman Filter.

- ***Control***: The estimated states can be used, as if they were real measurements, by a controller, see Figure 32.2. The controller can be e.g. a feedforward controller implementing feedforward from estimated disturbances, or a model predictive controller (MPC) using estimated states and estimated disturbances. Actually, it is common that MPCs have an "integrated" Kalman Filter.

The Kalman Filter algorithm was originally developed for systems assumed to be represented with a *linear* state space model. However, in many applications the system model is *nonlinear*. Furthermore the linear model is just a special case of a nonlinear model. Therefore, I have decided to present the Kalman Filter for nonlinear models, but comments are given about the linear case. The Kalman Filter for nonlinear models is denoted the *Extended Kalman Filter* because it is an extended use of the original Kalman Filter. However, for simplicity we can just denote it the Kalman Filter, dropping "extended" in the name. I will present the Kalman Filter without detailed derivation.

As with every model-based algorithm you should *test* your Kalman Filter with a simulated process before applying it to the real system. In the testing, you should start with testing the Kalman Filter with the nominal model, i.e. assumed correct model, in the simulator, including process and measurement noise. This is the model on which you are basing the Kalman Filter. If you have designed and implemented your Kalman Filter correctly, the estimated states should reproduce the simulated states (but with less noise in the estimates).

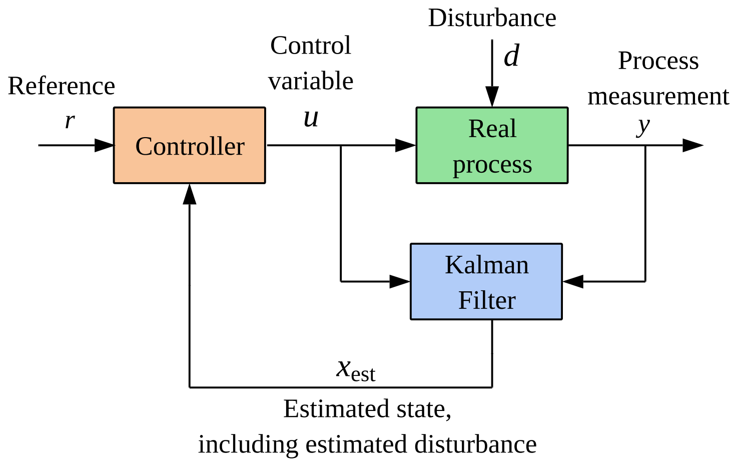


Figure 32.2: Control system where the controller uses estimated state variables, x_{est}, for control.

This chapter is meant to provide sufficient information about how to design and implement a standard Kalman Filter. More theory about state estimation and about various Kalman Filter forms can be found in e.g. (Simon 2006).

32.2 Observability of discrete-time systems

A necessary condition for the Kalman Filter to work correctly is that the system for which the states are to be estimated, is *observable,* which can be checked numerically. You should check for observability before applying the Kalman Filter. (There may still be other problems that prevent the Kalman Filter from producing accurate state estimates, as a faulty or inaccurate mathematical model.)

The observability check presented here applies only to *linear* state space models, which may stem from a linearized nonlinear model.

Observability of discrete-time systems can be defined as follows: The discrete-time system

$$x_{k+1} = Ax_k + Bu_k \tag{32.1}$$

$$y_k = Cx_k + Du_k \tag{32.2}$$

is observable if there is a finite number of time steps k so that knowledge about the input sequence $u_0, \ldots, u_{k-1}$ and the output sequence $y_0, \ldots, y_{k-1}$ is sufficient to determine the initial state state of the system, x_0.

Let us derive a criterion for the system to be observable. Since the influence of input u on state x is known from the model, let us for simplicity assume that $u_k = 0$. From the model

(32.1) – (32.2) we get

$$y_0 = Cx_0 \tag{32.3}$$
$$y_1 = Cx_1 = CAx_0 \tag{32.4}$$
$$\vdots$$
$$y_{n-1} = CA^{n-1}x_0 \tag{32.5}$$

which can be expressed compactly as

$$\underbrace{\begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}}_{M_{\text{obs}}} x_0 = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}}_{Y} \tag{32.6}$$

Let us make a definition:

Observability matrix:

$$M_{\text{obs}} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{32.7}$$

(32.6) has a unique solution only if the rank of M_{obs} is n. Therefore:

Observability Criterion:

The system (32.1) – (32.2) is observable if and only if the observability matrix has rank equal to n where n is the order of the system model (the number state variables).

The rank can be checked by calculating the determinant of M_{obs}. If the determinant is non-zero, the rank is full, and hence, the system is observable. If the determinant is zero, the system is non-observable.

Non-observability has several concequences:

- There are state variables or linear combinations of state variables which do not make any response in the estimated measurement, and therefore, their estimates can not be corrected by the innovation process.
- The value of the Kalman Filter gain may diverge.
- The transfer function from the input variable u to the output variable y has an order that is less than the number of state variables (n).

Observability may be checked manually or with a pertinent function in e.g. Python (Control Package), Matlab, and LabVIEW. In Example 32.1, observability is checked manually and with Python.

Example 32.1 *Observability*

Given the following state space model:

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B} u_k \tag{32.8}$$

$$y_k = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \underbrace{[0]}_{D} u_k \tag{32.9}$$

The observability matrix is $(n = 2)$

$$M_{\text{obs}} = \begin{bmatrix} C \\ CA^{2-1} = CA \end{bmatrix} = \left[\begin{array}{c} \begin{bmatrix} 1 & 0 \end{bmatrix} \\ \hdashline \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \end{array} \right] = \begin{bmatrix} 1 & 0 \\ 1 & a \end{bmatrix} \tag{32.10}$$

The determinant of M_{obs} is

$$\det\left(M_{\text{obs}}\right) = 1 \cdot a - 1 \cdot 0 = a \tag{32.11}$$

From (32.11) we can conclude that the system is observable only if $a \neq 0$.

Does this result make sense?

- Assume that $a \neq 0$ which means that the first state variable, x_1, contains some non-zero information about the second state variable, x_2. Hence, x_2 can be "observed" from the observed (measured) x_1, and the system is observable.
- Assume that $a = 0$ which means that x_1 contains no information about x_2. In this case the system (or specifically x_2) is non-observable despite x_1 being observed (measured), and we say that the system is non-observable.

The Python program 32.1 implements an observability check both with the rank and with the determinant of the observability matrix.

http://techteach.no/control/python/observability_check.py

Listing 32.1: observability_check.py

```
import numpy as np
import control

# %% Model matrices as 2D arrays:
```

```
a = 0  # Model param

A = np.array([[1, a],
              [0, 1]])
B = np.array([[0],
              [1]])
C = np.array([[1, 0]])
D = np.array([[0]])

# %% Creating and checking the observability matrix:

M_obs = control.obsv(A, C)

#Rank check:
rank_M_obs = np.linalg.matrix_rank(M_obs)
print('rank_M_obs =', rank_M_obs)

# Determinant check:
det_M_obs = np.linalg.det(M_obs)
print('det_M_obs =', det_M_obs)
```

With $a = 0$ (non-observability) the results are:

```
rank_M_obs = 1
det_M_obs = 0.0
```

[End of Example 32.1]

32.3 The Kalman Filter algorithm

32.3.1 The assumed process model

The Kalman Filter presented below assumes that the system model consists of the following discrete-time state space model:

$$x_{k+1} = f\left(x_k,\, u_k\right) + G w_k \tag{32.12}$$

and this measurement model:

$$y_k = g\left(x_k,\, u_k\right) + v_k \tag{32.13}$$

Some times, but rarely, the additive term $H w_k$ is included in (32.13), but I skip this term here, which is equivalent to setting $H = [0]$, a matrix of zeros.

Typically, the discrete-time model (32.12) – (32.13) is the discretized version of the following continuous-time model, where we for simplicity have disregarded random disturbance and random measurement noise (but such terms are included in the discrete-time model:

$$x' = f_{\text{cont}}(x, u) \tag{32.14}$$

$$y = g(x, u) \tag{32.15}$$

Often, Euler Forward discretization is used to discretize (32.14):

$$x_{k+1} = \underbrace{x_k + t_s f_{\text{cont}}(x_k,\, u_k)}_{f(x_k, u_k)} \tag{32.16}$$

(32.13) is just the discrete-time version of (32.15).

A *linear* model is just a special case:

$$x_{k+1} = \underbrace{Ax_k + Bu_k}_{=f} + Gw_k \tag{32.17}$$

and

$$y_k = \underbrace{Cx_k + Du_k}_{=g} + v_k \tag{32.18}$$

The models above contains the following variables and functions:

- x is the state vector of n state variables:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{32.19}$$

- u is the input vector of m input variables:

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \tag{32.20}$$

  It is assumed that the value of u is known. u includes control variables and known disturbances.

- f is the system vector function:

$$f = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} \tag{32.21}$$

  where f is any nonlinear or linear function.

- w is random (white) disturbance (or process noise) vector:

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_q \end{bmatrix} \tag{32.22}$$

  with auto-covariance matrix Q which is typically assumed diagonal:

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q_{nn} \end{bmatrix} \tag{32.23}$$

Hence, the number q of process disturbances is assumed to be equal to the number n of state variables.

- G is the process noise gain matrix relating the process noise to the state variables. It is common to assume that $q = n$, making G square:

$$G = \begin{bmatrix} G_{11} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & G_{nn} \end{bmatrix} \tag{32.24}$$

It is actually common to set the elements of G equal to one:

$$G_{ii} = 1 \tag{32.25}$$

making G an identity matrix:

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 1 \end{bmatrix} = I_n \tag{32.26}$$

- y is the measurement vector of r measurement variables:

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_r \end{bmatrix} \tag{32.27}$$

- $g(\cdot)$ is the measurement vector function:

$$g = \begin{bmatrix} g_1 \\ \vdots \\ g_r \end{bmatrix} \tag{32.28}$$

where g_i is any nonlinear or linear function. Typically, g is a linear function on the form

$$g = Cx \tag{32.29}$$

where C is the measurement gain matrix. (32.29) implies that the D matrix in (32.18) is a matrix of zeros.

- v is a random (white) measurement noise vector:

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_r \end{bmatrix} \tag{32.30}$$

with auto-covariance R which is typically assumed diagonal:

$$R = \begin{bmatrix} R_{11} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & R_{rr} \end{bmatrix} \tag{32.31}$$

32.3.2 The result of Kalman Filtering: an optimal state estimate

We define the state estimation error vector:

$$e_{x,k} = x_{\text{est},k} - x_k \tag{32.32}$$

where: x_k is the assumed true state vector, and $x_{\text{est},k}$ is the state estimate. The Kalman Filter produces an optimal estimate in the sense that the expectation value of the sum (actually of any linear combination) of the estimation errors gets a minimal value. In other words, The Kalman Filter minimizes the following sum of squared errors:

$$E[{e_{x,k}}^T e_k] = E\left[{e_{x_{1,k}}}^2 + \cdots + {e_{x_{n,k}}}^2\right] \tag{32.33}$$

The Kalman Filter estimate is therefore sometimes denoted the "least mean-square estimate". This result actually assumes that the model is linear, so for nonlinear models, this result is approximate.

32.3.3 The Kalman Filter algorithm – step by step

I present here the *Kalman Filter algorithm*, in the same order as it may be implemented in a program. The mathematical block diagram shown in Figure 32.3 gives a graphical representation of the Kalman Filter, except that the formulas for the Kalman Filter Gain, K_k, are not shown in the block diagram.

Initialization

This step is the initial step*, and the operations here are executed only once, before the estimation loop starts.*

The initial value $x_{p,0}$ of the predicted state estimate x_p is set equal to this initial (guessed) value of the state estimate:

$$x_{p,0} = x_{\text{init}} \tag{32.34}$$

Also the auto-covariance matrix of predicted state estimate error must be given an initial (a guessed) value. This auto-covariance is defined as:

$$P_{p,k} = E\left[\left(x_k - m_{x_{p,k}}\right)\left(x - m_{x_{p,k}}\right)^T\right] \tag{32.35}$$

Its initial value is set to:

$$P_{p,k} = P_{p,\text{init}} \tag{32.36}$$

A typical initial value is a matrix of ones (the identity matrix):

$$P_{p,\text{init}} = \text{diag}(1, \cdots, 1) = I_n \tag{32.37}$$

The subsequent actions are implemented in the estimation loop in the given order.

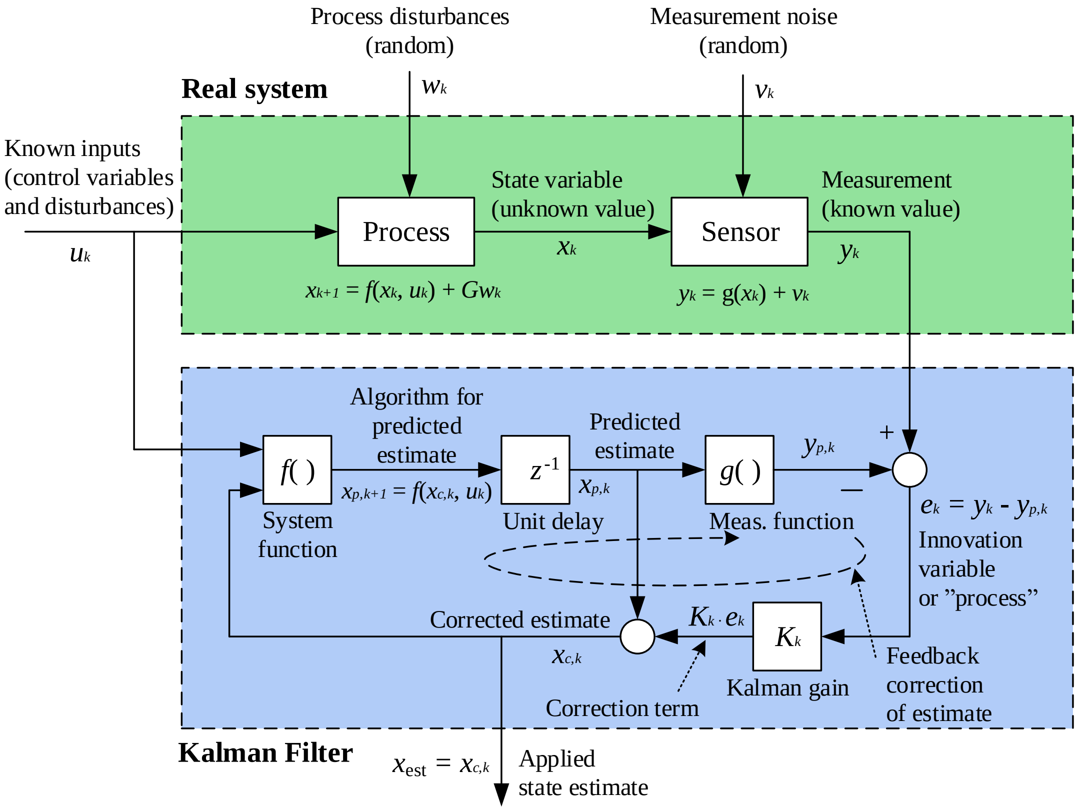

Process disturbances (random)
Measurement noise (random)
Real system
w_k
v_k
Known inputs (control variables and disturbances)
u_k
Process
State variable (unknown value)
x_k
Sensor
Measurement (known value)
y_k
$x_{k+1} = f(x_k, u_k) + Gw_k$
$y_k = g(x_k) + v_k$
Algorithm for predicted estimate
$f(\,)$
System function
$x_{p,k+1} = f(x_{c,k}, u_k)$
z^{-1}
Unit delay
Predicted estimate
$x_{p,k}$
$g(\,)$
Meas. function
$y_{p,k}$
+
−
$e_k = y_k - y_{p,k}$
Innovation variable or "process"
Corrected estimate
$x_{c,k}$
$K_k \cdot e_k$
K_k
Kalman gain
Correction term
Feedback correction of estimate
Kalman Filter
$x_{\text{est}} = x_{c,k}$
Applied state estimate

Figure 32.3: The Kalman Filter algorithm (32.43) – (32.44) represented by a block diagram.

Real measurement

The real measurement, $y_{m,k}$, is read from the sensor.

Predicted measurement

The predicted measurement is calculated from the predicted state according to the sensor model:

$$y_{p,k} = g\left(x_{p,k}\right) \tag{32.38}$$

In most cases, y_p is just one of the (predicted) state variables. If it is, say equal to x_i, we have: $y_{p,k} = x_{i,p,k}$.

Note: In the sensor model (32.18), the measurement noise w_k is an additive term. However, we can not include w_k in (32.38) since w_k is not known or not predictable (since it is assumed being white noise).

Innovation variable

The innovation variable is the difference between the real measurement and the predicted measurement:

$$e_k = y_{m,k} - y_{\mathrm{p},k} \tag{32.39}$$

Kalman Filter gain

The Kalman Filter gain is calculated as:

$$K_k = P_{p,k}C^T[CP_{p,k}C^T + R]^{-1} \tag{32.40}$$

Here, C is the measurement gain matrix of a linearized model of the original nonlinear model (32.13) calculated at the most recent operating point, which is $(x_{p,k},\ u_k)$:

$$C = \left.\frac{\partial g}{\partial x}\right|_{\mathrm{op}\,=\,(x_{p,k},\,u_k)} \tag{32.41}$$

It is common that $g() = C$, and if so, no linearization is actually needed.

***Steady-state Kalman Filter gain*:** If the model is linear and time invariant (i.e. system matrices are not varying with time), the Kalman Filter gain will converge towards a steady state value value, K_s, if the number of iterations goes to infinity:

$$K_k \to K_\infty = K_s \tag{32.42}$$

You can pre-calculate K_s by running the algorithm for K_k for a sufficiently large number of iterations. In this book, I stick to the time-varying Kalman Filter gain since it is relatively easy to calculate with matrix-vector operations.

Corrected state estimate

Calculate the corrected state estimate $x_{c,k}$ by adding the corrective term $K_k e_k$ to the predicted state estimate $x_{p,k}$:

$$x_{c,k} = x_{p,k} + K_k e_k \tag{32.43}$$

Note 1: $x_{c,k}$, and not $x_{p,k}$, is used as the state estimate in applications.

Note 2: Formula (32.43) is an *error-driven* correction term of the state estimates. You can expect the errors of the state estimates to be smaller than if there were no such correction term. This correction can also be regarded as a feedback correction of the estimates – the feedback is from the predicted measurement back to the rate of changes of the state estimates. This feedback is indicated in Figure 32.3. It is well known from control systems theory that error-driven or feedback correction (or control) generally reduces errors.

Note 3: The corrected estimate $x_{c,k}$ is also denoted the a posteriori estimate because it is calculated after the present measurement is taken. It is also denoted the measurement-updated estimate.

Note 4: The state estimates should be prevented from getting unrealistic values. For example, the estimate of a liquid level should not be negative. The following Python code implements such limitation:

```
⋮
x_c_k = ···
if (x_c_k > x_max):
x_c_k = x_max
elif (x_c_k < x_min):
x_c_k = x_min
⋮
```

Such a limitation may also be implemented with some built-in limitation functions, e.g. with numpy.clip() in Python:

```
⋮
x_c_k = ···
x_c_k = np.clip(x_c_k, x_min, x_max)
⋮
```

Predicted state estimate for the next time step

The prediction, $x_{p,k+1}$, is calculated using the present state estimate $x_{c,k}$ and the known input u_k in process model:

$$x_{p,k+1} = f\left(x_{c,k},\, u_k\right) \tag{32.44}$$

Note 1: In the process model (32.12), the process disturbance v_k is an additive term. However, we can not include v_k in (32.44) since v_k is not known or not predictable (since we have assumed it is "white noise").

Note 2: The predicted estimate is also denoted the a priori estimate because it is calculated before the present measurement is taken. It is also denoted the time-updated estimate.

Note 3: The state estimates should be prevented from getting unrealistic values. For example, the estimate of a liquid level should not be negative. The following Python code implements such limitation:

```
⋮
x_p_kp1 = ···
if (x_p_kp1 > x_max):
x_p_kp1 = x_max
elif (x_p_kp1 < x_min):
x_p_kp1 = x_min
⋮
```

Such a limitation may also be implemented with some built-in limitation functions, e.g. with numpy.clip() in Python:

```
⋮
x_p_kp1 = ···
x_p_kp1 = np.clip(x_p_kp1, x_min, x_max)
⋮
```

Auto-covariance of corrected state estimate error

The auto-covariance matrix of the estimation error of the corrected estimate is:

$$P_{c,k} = E\left[\left(x_k - m_{x_{c,k}}\right)\left(x_k - m_{x_{c,k}}\right)^T\right] \tag{32.45}$$

where $m_{x_{c,k}}$ is mean value.

A formula for $P_{c,k}$ is:

$$P_{c,k} = \left[I - K_k C\right] P_{p,k} \tag{32.46}$$

where: K_k is the Kalman Gain given by (32.40). C is given by (32.41). In the first iteration of the estimation loop, $P_{p,k}$ is known from the initialization, (32.36). In subsequent iterations, $P_{p,k}$, is known from its predicted value, see below.

Auto-covariance of predicted state estimate error in the next iteration

The auto-covariance of the state estimate error is defined by (32.35). Its value in the next iteraton is:

$$P_{p,k+1} = AP_{c,k}A^T + GQG^T \tag{32.47}$$

Here, A is the transition matrix of a linearized model of the original discrete-time nonlinear model (32.12) calculated at the most recent operating point, which is $(x_{c,k},\ u_k)$. Although we can obtain A by linearizing $f(\cdot)$ in that model, I suggest that A is obtained by first linearizing the continuous-time model (32.14) and then discretizing that model[1]:

$$A = I + T_s A_{\text{cont}} = I + t_s \left. \frac{\partial f_{\text{cont}}(\cdot)}{\partial x} \right|_{\text{op} = (x_{c,k},\, u_k)} \tag{32.48}$$

Index-shift (or time-shift) to prepare for the next iteration

$$x_{p,k} = x_{p,k+1}$$

$$P_{p,k} = P_{p,k+1}$$

Example 32.2 shows the design and implementation of a Kalman Filter for estimation of the inflow (and the level) of a liquid tank using the level measurement.

32.3.4 Features of the Kalman Filter

32.3.4.1 The error-model

Assuming that the system model is linear and that the model is correct (giving a correct representation of the real system), it can be shown that the behaviour of the error of the corrected state estimation, $e_{x_{c,k}}$, cf. (32.32), is given by the following *error-model*:[2]

$$e_{x_{c,k+1}} = (I - K_k C)\, A e_{x_{c,k}} + (I - K_k C)\, G v_k - K_k w_{k+1} \tag{32.49}$$

This model can be used to analyse the Kalman Filter in terms of dynamics, stability and steady state behaviour.

Note: (32.32) is not identical to the auto-covariance of the estimation error which is

$$P_{c,k} = E\left\{ \left[e_{x_{c,k}} - m_{x_{c,k}} \right] \left[e_{x_{c,k}} - m_{x_{c,k}} \right]^T \right\} \tag{32.50}$$

But (32.32) is the *trace* of P_c (the trace is the sum of the diagonal elements):

$$e_{x_c} = \text{trace}\left[P_{c,k} \right] \tag{32.51}$$

32.3.4.2 The dynamics of the Kalman Filter

The error-model (32.49) of the Kalman Filter represents a dynamic system. The dynamics of the Kalman Filter can be analyzed by calculating the eigenvalues of the system matrix of (32.49). These eigenvalues are:

$$\{\lambda_1, \lambda_2, \ldots, \lambda_n\} = \text{eig}\left[(I - K_k C) A \right] \tag{32.52}$$

[1] because then you can exploit a pertinent function in e.g. Python or Matlab for discretization

[2] You can derive this model by subtracting the model describing the corrected state estimate from the model that describes the real state (the latter is simply the process model).

32.3.4.3 The stability of the Kalman Filter

It can be shown that the Kalman Filter always is an asymptotically stable dynamic system (otherwise it could not give an optimal estimate). In other words, the eigenvalues defined by (32.52) are always inside the unity circle.

32.4 Tuning the Kalman Filter

The main tuning factor of the Kalman Filter is the matrix Q – the process disturbance auto-covariance. Remember that the process disturbance influences the state variables, cf. (32.13). Therefore, a large Q tells the Kalman Filter that the variations in the real state variables are relatively large. Hence, the larger Q, the larger the Kalman Gain K_k will be, to provide a stronger updating of the estimates. However, this strong estimate update causes more measurement noise to be added to the estimates because the measurement noise is a term in the innovation process e which is multiplied by K_k:

$$\begin{aligned} x_{c,k} &= x_{p,k} + K_k e_k \qquad (32.53) \\ &= x_{p,k} + K_k \left[g\left(x_k\right) + v_k - g\left(x_{p,k}\right) \right] \qquad (32.54) \end{aligned}$$

where v is real measurement noise.

Consequently, we can state the main rule for tuning the Kalman Filter as follows:

Select elements of Q as large as possible without the state estimates becoming too noisy.

But Q is a matrix! How to select it “large” or “small”? Since each of the process disturbances typically are assumed to act on their respective state independently, Q can be set as a diagonal matrix:

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q_{nn} \end{bmatrix} = \text{diag}(Q_{11}, \cdots, Q_{nn}) \qquad (32.55)$$

where each of the diagonal elements can be adjusted independently. If you do not have any idea about numerical values, you can start by setting all the diagonal elements to one, and hence Q is

$$Q = Q_0 \begin{bmatrix} 1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (32.56)$$

where Q_0 is the only tuning parameter. If you do not have any idea about a proper value of Q_0 you may initially try

$$Q_0 = 0.01 \qquad (32.57)$$

Then you may adjust Q_0 or try to fine tune each of the diagonal elements individually.

Example 32.2 demonstrates the effect of tuning the Kalman Filter with the Q matrix.

32.5 Estimating parameters and disturbances with Kalman Filter

32.5.1 Introduction

Note that you can try to estimate model errors with a Kalman Filter by defining such errors as *augmentation state variables* and estimating them as ordinary states, in the same way as you can estimate disturbances by defining them as augmentation state variables, cf. Section 32.5.

In many applications the Kalman Filter is used to estimate parameters and/or disturbances in addition to the “ordinary” state variables. Examples:

- Parameter: The fraction of biodegradable organic material in organic feed to a biogas reactor.
- Disturbance: Environmental forces acting on a ship. (The estimate can then be used for feedforward control in the position controller of the ship.

These parameters and/or disturbances must be represented as state variables to be estimated with a Kalman Filter. They represent additional state variables. The original state vector is augmented with these new state variables which we may denote the augmentative states. The Kalman Filter is used to estimate the augmented state vector which consists of both the original state variables and the augmentative state variables.

How can you model these augmentative state variables? The augmentative model must be in the form of a difference equation which in the form of a state space model so that it can be augmented to the original state space model. To set up such an augmentative model you must make an assumption about the behaviour of the augmentative state. Let us look at some augmentative models.

32.5.2 The augmentative state (x_a) is constant

The most common augmentative model is based on the assumption that the augmentative state variable x_a is constant. The corresponding differential equation is:

$$(x_a)' = 0 \tag{32.58}$$

Discretizing this differential equation with the Euler Forward method gives:

$$x_{a,k+1} = x_{a,k} \tag{32.59}$$

It is common to assume that the state is driven by some random disturbance, hence the augmentative model becomes:

$$x_{a,k+1} = x_{a,k} + w_{a,k} \tag{32.60}$$

where w_a is random process disturbance with assumed auto-covariance matrix Q_a. As pointed out in Section 32.3, the variance Q_a can be used as a tuning parameter of the estimate of x_a.

32.5.3 The augmentative state (x_a) has constant rate

The corresponding differential equation is:

$$(x_a)'' = 0 \tag{32.61}$$

or, in state space form, with $x_{a_1} \equiv x_a$:

$$(x_{a_1})' = x_{a_2} \tag{32.62}$$

$$(x_{a_2})' = 0 \tag{32.63}$$

where x_{a_2} is another augmentative state variable. Applying Euler Forward discretization with sampling interval t_c [sec] to (32.62) – (32.63) and including white process noise to the resulting difference equations gives

$$x_{a1,k+1} = x_{a1,k} + t_s x_{a2,k} + w_{a1,k} \tag{32.64}$$

$$x_{a2,k} = x_{a2,k} + w_{a2,k} \tag{32.65}$$

Now follows a comprehensive example which covers:

- Kalman Filter algorithm development
- Kalman Filter tuning
- Kalman Filter implementation in a Python program

Example 32.2 *Kalman Filter for estimation of inflow of a simulated buffer tank*

The system

A buffer tank is presented in Ch. 39.3. In this example, we will design and implement (in Python) a Kalman Filter for a simulated tank which has a level control system. The Kalman Filter will estimate F_{in} and h, hence, it will be a soft sensor for F_{in} and h. We may here regard F_{in}as a disturbance that we want to estimate.

The Kalman Filter needs at least one measurement. In this example, the measurement is the simulated level, h.

The time-step of the simulator and of the Kalman Filter is:

$$t_s = 10 \text{ s} \tag{32.66}$$

Figure 32.4 shows the tank with a level control system.

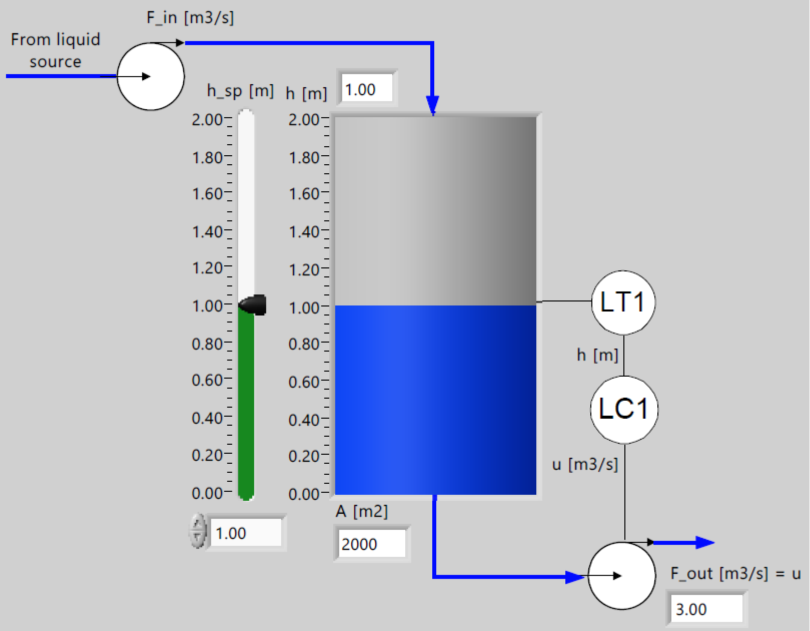


Figure 32.4: Buffer tank with level control system.

The level is controlled with a PI controller tuned with the SIMC method, cf. Ch. 16.4. The level PI controller is tuned with the SIMC method with $T_c = 1000$ s, as in Example 16.5, giving:

$$K_c = -2 \tag{32.67}$$

$$T_i = 2000 \text{ s} \tag{32.68}$$

The process model

Process model parameters:

$$A_{\text{area}} = 2000 \text{ m}^2 \tag{32.69}$$

The mathematical model of the process is based on mass balance of the liquid in the tank. The model is:

$$(Ah)' = F_{\text{in}} - F_{\text{out}} \tag{32.70}$$

Mass balance of the liquid in the tank is (mass is $\rho A h$):

$$(\rho A_{\text{area}} h)' = \rho F_{\text{in}} - \rho F_{\text{out}} \tag{32.71}$$

F_{out} is the outflow demanded by the level controller. Therefore,

$$F_{\text{out}} = u \tag{32.72}$$

where u is the control signal. By inserting u for F_{out} and cancelling out density ρ, (32.71) becomes:

$$h' = \frac{1}{A_{\text{area}}}(F_{\text{in}} - u) \equiv f_1(\cdot) \tag{32.73}$$

We assume that F_{in} is unknown, but slowly changing – almost constant. Therefore, we define the following augmentative model for F_{in}:

$$(F_{\text{in}})' = 0 \equiv f_2(\cdot) \tag{32.74}$$

h is measured. Hence,

$$h_m = h \equiv g(\cdot) \tag{32.75}$$

To summarize: The continuous-time model of the system is given by the differential equations (32.73) – (32.74) with (32.75) as the output or measurement equation.

In the context of linearization, it can be convenient to collect f_1 and f_2 in one vector function:

$$f = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{32.76}$$

In some contexts, it is convenient to introduce standard names of the state variables:

$$x_1 = h \tag{32.77}$$

$$x_2 = F_{\text{in}} \tag{32.78}$$

However, we will in this example continue using the original symbols h and F_{in}.

For the Kalman Filter we need a discrete-time state space model. Applying Euler Forward discretization with time step t_s and including white disturbance noises w_1 and w_2 in the resulting difference equations, give:

$$h_{k+1} = h_k + \frac{t_s}{A_{\text{area}}}\left(F_{\text{in},k} - u_k\right) + w_{1,k} \tag{32.79}$$

$$F_{\text{in},k\,+\,1} = F_{\text{in},k} + w_{2,k} \tag{32.80}$$

w_1 and w_2 are independent (uncorrelated) white process disturbances with assumed variances Q_{11} and Q_{22}, respectively. The process disturbance covariance matrix is then:

$$Q = \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \tag{32.81}$$

As mentioned above, h is measured. The measurement is here denoted h_{meas}. Assuming white measurement noise v_k with variance R_{11} (a scalar), the measurement equation is:

$$h_{m,k} = h_k + v_k \tag{32.82}$$

The measurement noise covariance matrix is:

$$R = [R_{11}] \tag{32.83}$$

where:

$$R_{11} = \left(\sigma_{\text{meas,noise}}\right)^2$$

where:

$$\sigma_{\text{meas,noise}} = \frac{a_{\text{meas,noise}}}{\sqrt{3}} \tag{32.84}$$

where:

$$a_{\text{meas,noise}} = 0.01 \text{ m} \tag{32.85}$$

$a_{\text{meas,noise}}$ is the amplitude of an assumed uniformly distributed random noise signal of amplitide 0.01 m (i.e. the noise is between ±0.01 m. The relation between the standard deviation $\sigma_{\text{meas,noise}}$ and the amplitude $a_{\text{meas,noise}}$ is dervied in Solution 31.6.

The PI level controller

The tank level, h, is controlled with a PI controller which is tuned with the SIMC method. The PI controller is in the Python program presented below in this example, but I do not show details about the controller here.

The Kalman Filter algorithm

The Kalman Filter algorithm is as follows (numerical values are given later in this example).

Initialization:

$$h_{p,\text{init}} = h_{m,\text{init}} = 2.0 \text{ m}$$

$$F_{\text{in},p,k} = F_{\text{in},p,\text{init}} = 3.0 \text{ m}^3/\text{s}$$

$$P_{p,k} = P_{p,\text{init}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

In the simulation/estimation loop:

- The innovation variable:

$$e_k = h_{m,k} - h_{p,k} \tag{32.86}$$

  where: $h_{m,k}$ is the simulated measurement. $h_{p,k}$ is the predicted measurement.

- The Kalman Gain:

$$K_k = P_{p,k}C[CP_{p,k}C^T + R]^{-1} \tag{32.87}$$

  where:

$$C = C_{\text{cont}} = \left.\frac{\partial g}{\partial x}\right|_{\text{op}} = \left[\ 1, \quad 0\ \right] \tag{32.88}$$

- Measurement-corrected estimates, *which are used as the applied estimates*:

$$h_{c,k} = h_{p,k} + K_{0,0,k} \cdot e_k \tag{32.89}$$

$$F_{\text{in},c,k} = F_{\text{in},p,k} + K_{1,0,k} \cdot e_k \tag{32.90}$$

- Model-predicted estimates as one Euler Forward step:

$$h_{p,k+1} = h_{c,k} + \frac{t_s}{A_{\text{area}}} \left(F_{\text{in},c,k} - u_k \right) \tag{32.91}$$

$$F_{\text{in},p,k+1} = F_{\text{in},c,k} \tag{32.92}$$

- Auto-covariance of meas-corrected state estimate error, (32.47):

$$P_{c,k} = \left[I - K_k C \right] P_{p,k} \tag{32.93}$$

- Auto-covariance of predicted state estimate error in the next iteration:

$$P_{p,k+1} = A P_{c,k} A^T + G Q G^T \tag{32.94}$$

  where:

  - Disturbance gain:

$$G = I_2 = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right]$$

  - Discrete-time system transition matrix:

$$\begin{aligned} A =& I + T_s A_{\text{cont}} && (32.95) \\ =& I + t_s \left. \frac{\partial f}{\partial x} \right|_{x_{p,k},\, u_k} && (32.96) \\ =& I + t_s \left. \left[\begin{array}{cc} \frac{\partial f_1}{\partial x_1} = 0 & \frac{\partial f_1}{\partial x_2} = \frac{1}{A_{\text{area}}} \\ \frac{\partial f_2}{\partial x_1} = 0 & \frac{\partial f_2}{\partial x_2} = 0 \end{array} \right] \right|_{x_{p,k},\, u_k} && (32.97) \\ =& \left[\begin{array}{cc} 1 & \frac{t_s}{A_{\text{area}}} \\ 0 & 1 \end{array} \right] && (32.98) \end{aligned}$$

  - Disturbance co-variance:

$$Q = \left[\begin{array}{cc} 1 & 0 \\ 0 & Q_{22} \end{array} \right] \tag{32.99}$$

    where:

$$Q_{22} = 10$$

    which is set by "trial-and-error" to give reasonably fast and smooth estimate of F_{in}. Q_{22} is used as the main tuning factor of the Kalman Filter.

- Index-shift (or time-shift):

$$h_{p,k} = h_{p,k+1} \tag{32.100}$$

$$F_{\text{in},p,k} = F_{\text{in},p,k+1} \tag{32.101}$$

$$P_{p,k} = P_{p,k+1} \tag{32.102}$$

Python program

The Python program 'kalman_pi_buffertank_sim.py' available via the link below implements the tank simulator with the Kalman Filter and the level PI controller.

http://techteach.no/control/python/kalman_pi_buffertank_sim.py

Results

Figure 32.5 shows the responses from the following simulation scenario:

- $h_{sp} = 2.0$ m (constant).
- F_{in} is changed from 3.0 to 4.0 m^3/s at $t = 1000$ s.
- The level measurement noise is uniformly distributed random noise, cf. (32.85), is included in the level measurement.

Figure 32.5 shows simulated responses. The Kalman Filter estimates F_{in} with zero error in steady state, but there is a transient estimation error.

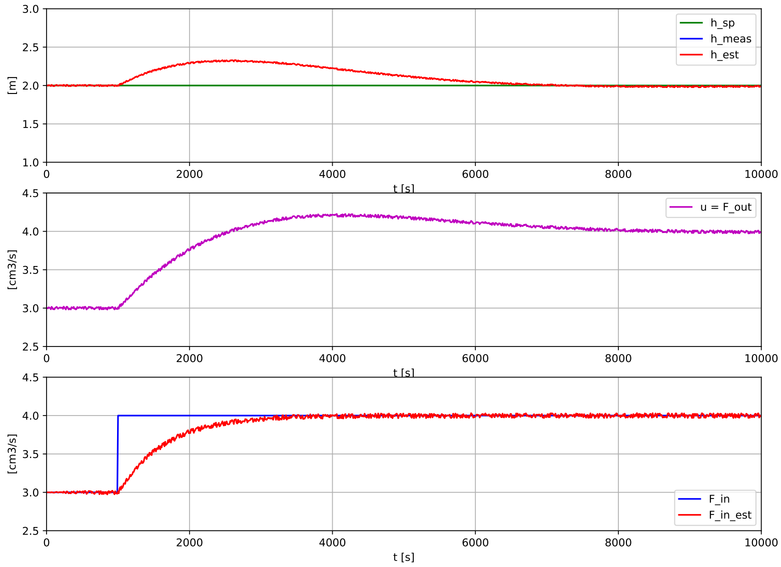


Figure 32.5: Simulated tank with PI level control and Kalman Filter with $Q_{22} = 10$.

It is interesting to see how the estimate of F_{in} is influenced by the tuning factor Q_{22}, while keeping the disturbance variance unchanged (namely keeping it zero) in the simulations:

- Figure 32.6 shows the estimates with $Q_{22} = 1$, which is set with the code Q_22 = 1 in the Python program. As expected, the estimation of F_{in} is less noisy and slower.
- Figure 32.7 shows the estimates with $Q_{22} = 100$. As expected, the estimation of F_{in} is more noisy and faster.

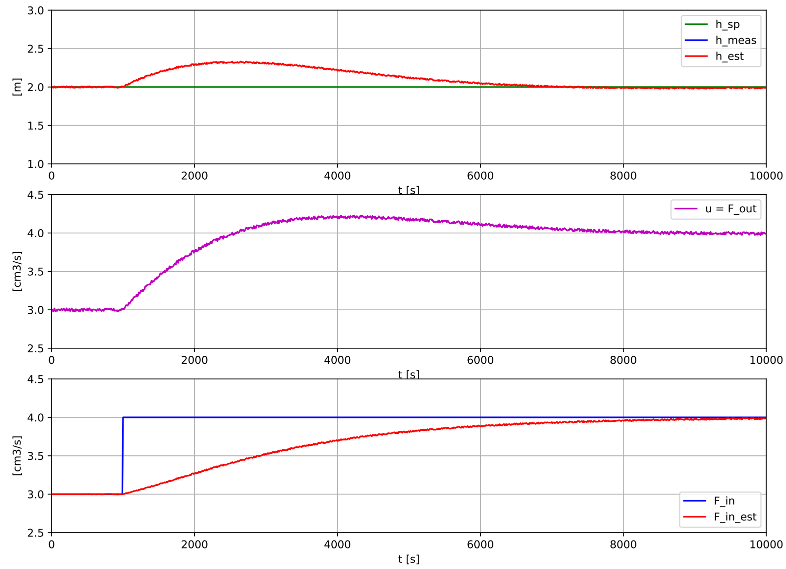


Figure 32.6: Kalman Filter estimates with $Q_{22} = 1$.

[End of Example 32.2]

32.6 Kalman Filtering when process measurement is absent

It may happen that the process measurement, y_m, is absent for some time while the Kalman Filter runs. The reason for the absence may be:

- Sensor failure
- Communication failure between the sensor and the Kalman Filter algorithm
- Low-rate sampling (large time step) of the measurement comparing with a high-rate (small time step) Kalman Filter algorithm

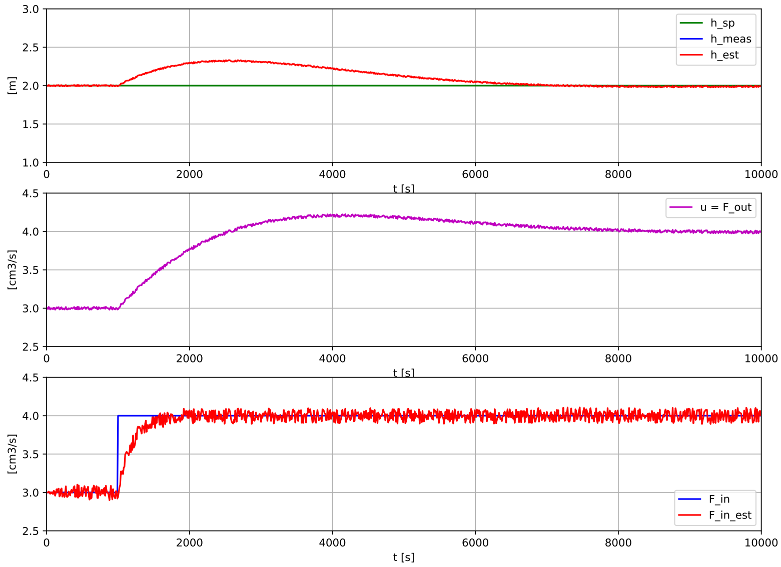


Figure 32.7: Kalman Filter estimates with $Q_{22} = 100$.

We assume that the occurance of the absence is known. How can the Kalman Filter cope with an absent measurement? An absent measurement corresponds to the measurement being irrelevant. An irrelevant measurement can be modelled as a measurement with an infinitely large measurement noise, i.e.:

$$R = [\infty] \tag{32.103}$$

Let us use (32.103) in the formula for the Kalman Gain, (32.40):

$$K_k = P_{p,k}C^T[CP_{p,k}C^T + R]^{-1} = [0] \tag{32.104}$$

So, the Kalman Filter gain becomes a matrix of zeros. This implies that the corrected estimate, (32.43), becomes:

$$x_{c,k} = x_{p,k} + \underbrace{K_k e_k}_{0} = x_{p,k} \tag{32.105}$$

In other words: The corrected estimate is equal to the predicted estimate (which was calculated in the previous iteration of the Kalman Filter algorithm). Or: There is no measurement-based update of the state estimate, which makes sense: The measurement should be disregarded.

Another consequence of (32.104) is that the auto-covariance of corrected state estimate error, (32.46), becomes equal to the auto-covariance of predicted state estimate error:

$$P_{c,k} = [I - K_k C]\, P_{p,k} = P_{p,k} \tag{32.106}$$

Such a Kalman Filter, i.e. a Kalman Filter which runs without the measurement-based, corrected update of the state estimate, only with the model-based, predicted estimated, is sometimes denoted a ballistic Kalman Filter[3], or an open loop-Kalman Filter. In other words: The Kalman Filter is just a *process simulator*.

In a Kalman Filter running in open loop, as described above, the estimated states will certainly eventually drift from the real states since there is no correction of the estimates. Still, the estimates, and in particular the estimate of the process measurement, may be useful, for example for control purposes where the controller can continue operating despite the absent real measurement. The alternative is turning the controller off since there is no real measurement. In other words: A Kalman Filter (when running in absence of real measurements) can increase the robustness of a control system.

[3] "Ballistic" because the estimated state develops uncorrected or unmanipulated, like a ball thrown into space.

32.7 *Problems for Chapter 32*

Problem 32.1 *Kalman Filter for a DC motor*

Ch. 39.8 describes a DC motor. A mathematical model is given by (39.26).

1. The load torque L is to be estimated with a Kalman Filter. The speed S is measured. L is assumed constant. Write a mathematical model for the Kalman Filter.

2. In the Kalman Filter algorithm, the measurement covariance R is needed. How can you in practice obtain an estimate of R

3. Here is a Python program implementing a Kalman Filter for the DC motor:

http://techteach.no/control/python/kalman_filter_sim_dc_motor.py

L is changed from 0 to -1 at $t = 5$ s. Run the simulator. Do you think that the Kalman Filter is working well?

4. With the Python program available above: Play with the parameter std_w_L (increase, decrease) in the program. What is the impact of this parameter?

32.8 *Solutions to problems for Chapter 32*

Solution to Problem 32.1

1. Model for Kalman Filter:

$$S' = \left[K(u+L) - S\right]/t_c + w_1 \tag{32.107}$$

$$L' = 0 + w_2 \tag{32.108}$$

$$S_m = S + v_1 \tag{32.109}$$

w_1 and w_2 are random process disturbances. v_1 is random measurement noise.

Alternatively, using standard symbols:

$$x_1{}' = \left[K(u+x_2) - x_1\right]/t_c + w_1 \tag{32.110}$$

$$x_2{}' = 0 + w_2 \tag{32.111}$$

$$y = x_1 + v_1 \tag{32.112}$$

where:

$$x_1 = S \tag{32.113}$$

$$x_2 = L \tag{32.114}$$

$$y = S_m \tag{32.115}$$

2. R can be calculated (estimated) as the variance of a data series of real speed measurements.

3. Figure 32.8 shows the simulated responses. It seems that the Kalman Filter is working well. The load torque L is estimated well in the sense that the constant L is estimated correctly.

4. The parameter std_w_L is the standard deviation of the assumed process random disturbance acting on L. The variance Q_{22} used in the Q matrix of the Kalman Filter is therefore (std_w_L)ˆ2.

If you *increase* std_w_L (i.e., Q_{22} is increased), you tell the Kalman Filter that L is varying much, and the Kalman Filter will increase the speed of the estimate (good), while the estimate is more noisy (bad). This is confirmed in Figure 32.9 where std_w_L has been increased from 0.05 to 0.5.

If you *decrease* std_w_L (i.e., Q_{22} is decreased), you tell the Kalman Filter that L is varying little, and the Kalman Filter will decrease the speed of the estimate (bad), while the estimate is less noisy (good). This is confirmed in Figure 32.10 where std_w_L has been decreased from 0.05 to 0.005.

So, you should tune std_w_L (or Q_{22}) as a compromise between fast & noisy estimation (std_w_L large) and slow & smooth estimation (std_w_L small).

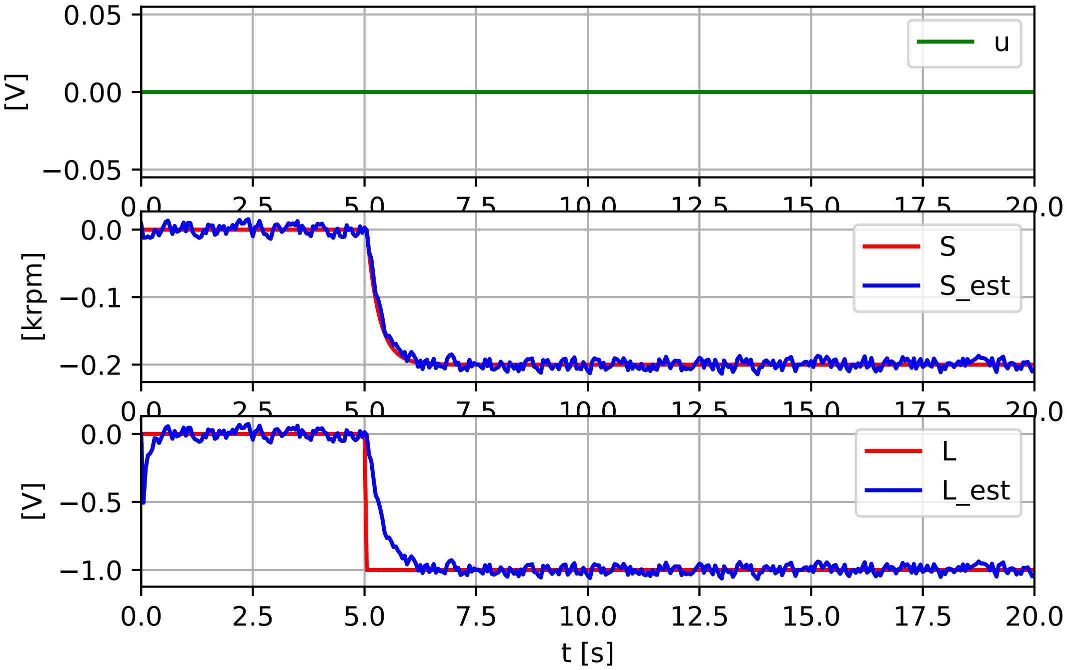


Figure 32.8: Problem 32.1: Kalman Filter with $Q_{22} = 0.05$.

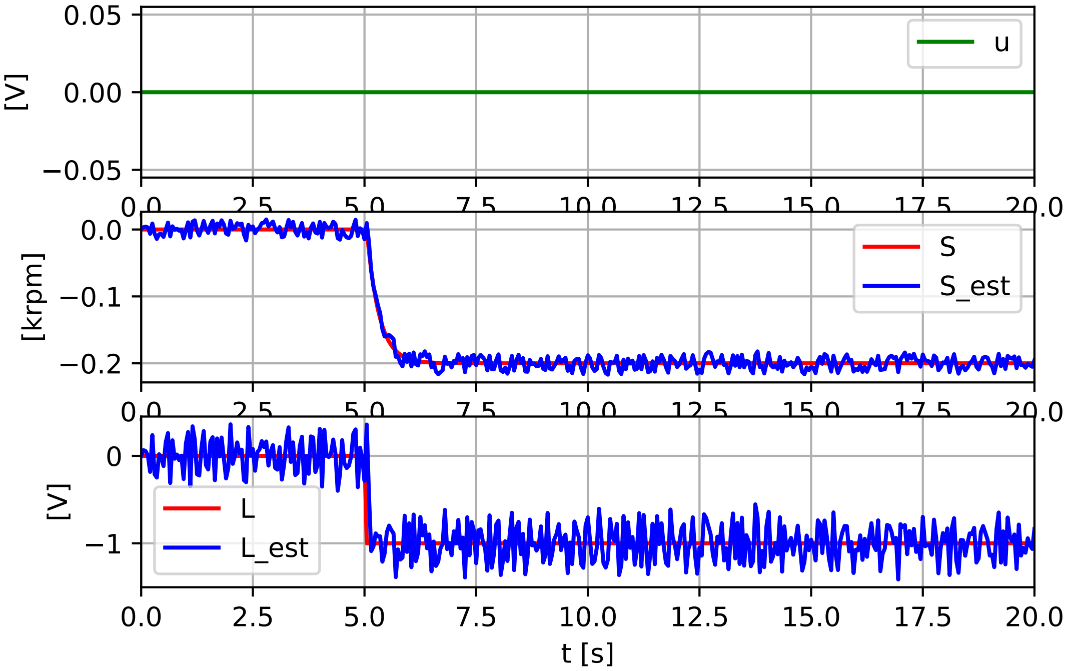


Figure 32.9: Problem 32.1: Kalman Filter with $Q_{22} = 0.5$.

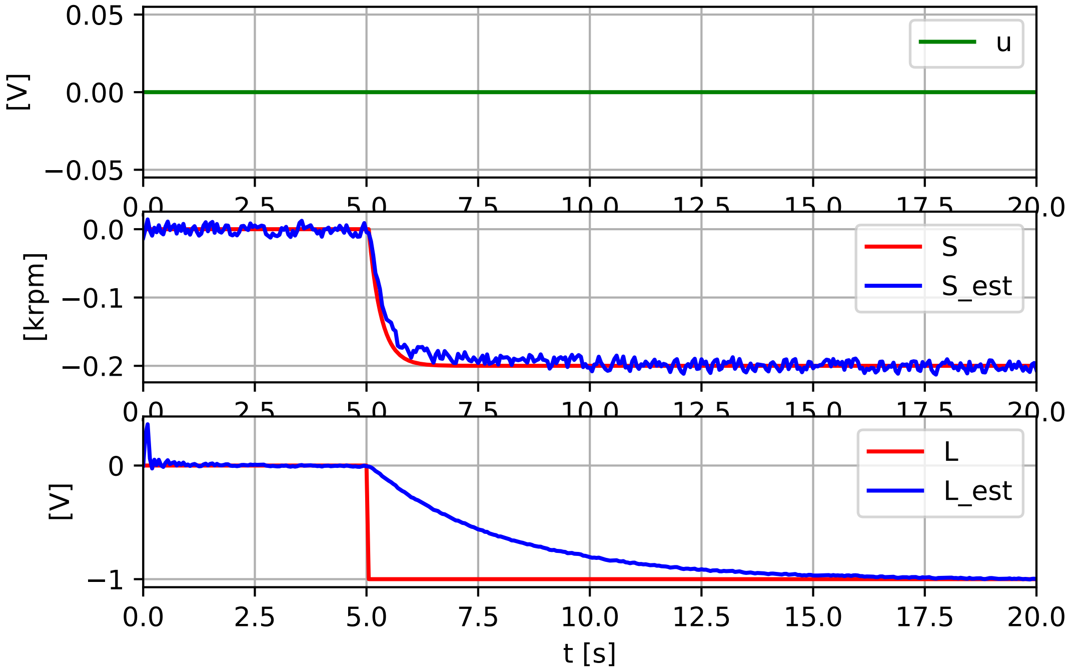


Figure 32.10: Problem 32.1: Kalman Filter with $Q_{22} = 0.005$.

Part VIII

MODEL-BASED CONTROL

Chapter 33

How to test robustness with simulations

This part of the book describes several *model-based* controllers. A model-based controller contains the mathematical model of the process to be controlled. Of course, a process model can never give a perfect description of a physical process. Hence, there are *model errors.* The model errors are in the form of

- erroneous model structure, and/or
- erroneous parameter values.

So, the controller is based on more or less erroneous information about the process. If the model errors are large, the real control system may behave quite different from what is specified during the design. The control system may even become unstable.

A control system that is supposed to work in real life must be sufficiently *robust.* How can you check if the system is robust (before implementation)? You can *simulate the control system.* In the simulation you include reasonable model errors. How do you include model errors in a simulator? By using *different* models in the control function and in the process in the simulator. This is illustrated in Figure 33.1.

You may use the intial model, M_0, in the control function, while you use a changed model, M_1, for the process. You must thoroughly plan which model errors (changes) that you will make, and whether the changes are *additive* or *multiplicative.* A parameter, say K, is changed additively if it is changed as

$$K_1 = K_0 + \Delta K \tag{33.1}$$

A multiplicative change is implemented with

$$K_1 = FK_0 \tag{33.2}$$

where the factor F may be set to e.g. 1.2 (a 20% increase) or 0.8 (a 20% decrease).

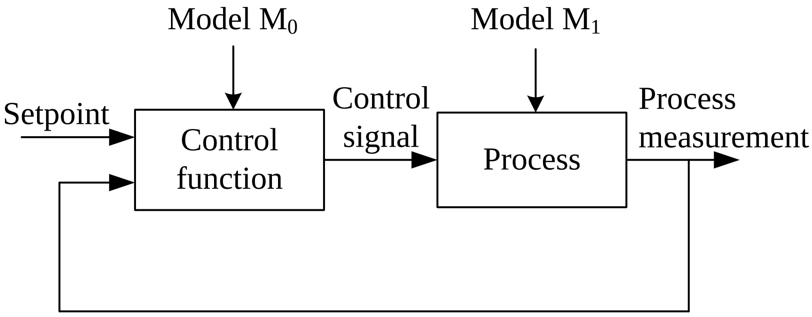


Figure 33.1: Testing the control system with model errors. Models M_0 and M_1 are made different by purpose.

Even if the process model is accurate, the behaviour of the controller can be largely infuenced by measurement noise. Therefore, you get a more real picture if you also include measurement noise in the simulator. Typically the measurement noise is a random signal[1].

Finally, I will mention that there are control system design methods which ensures roubustness of the control system. In the design phase you specify assumed maximum model errors, together with performance specifications. The control function is typically a non-standard controller, which have to be simplified before implementation. It is however beyond the scope of this compendium to describe these design methods. More information can be found in e.g. **?**.

[1] Simulation tools as LabVIEW and Simulink contains signal generators for random signals.

33.1 *Problems for Chapter 33*

Problem 33.1 *Model-based control of DC motor*

Figure 33.2 shows a model-based speed control system of an electric motor.

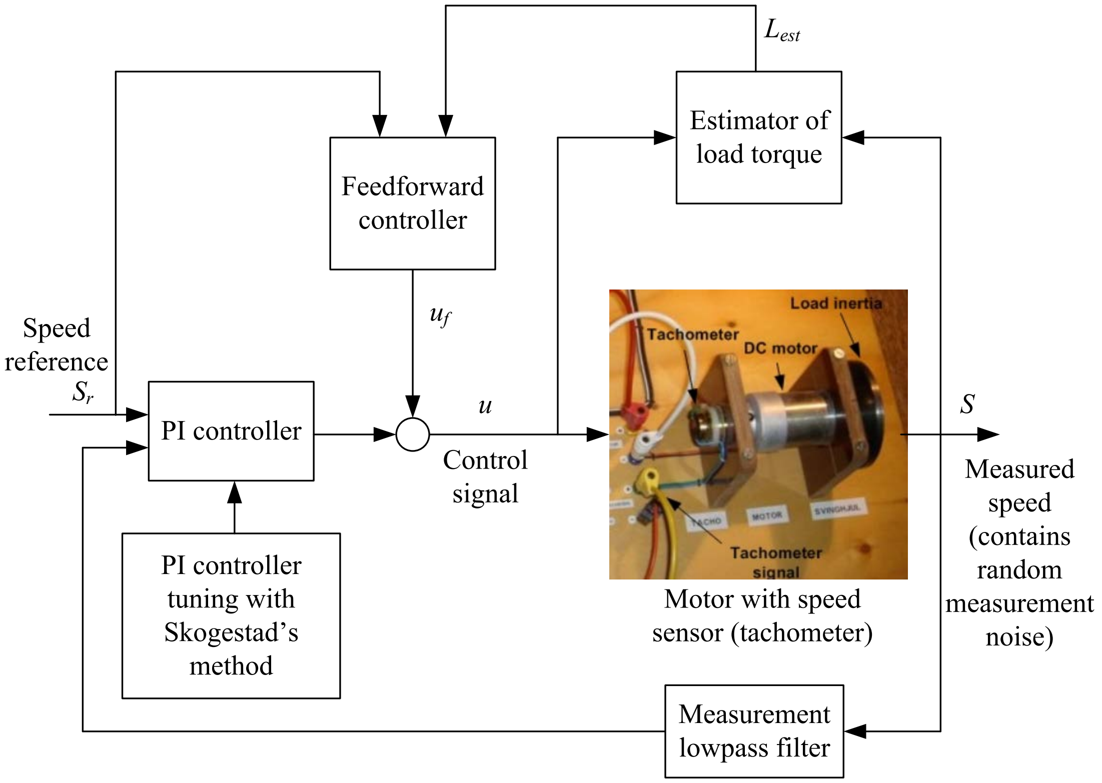


Figure 33.2: A model-based speed control system of an electric motor

The motor is controlled with an input voltage signal, u, and the rotational speed, S, is measured with a tachogenerator which produces a voltage being proportional to the speed.

A proper mathematical model of the motor is the following differential equation:

$$S' = \left[-S + K\left(u + L\right)\right]/t_c \tag{33.3}$$

L is equivalent load torque (represented in the same unit as the control variable, namely voltage). L can be regarded as a process disturbance. K is gain. t_c is time constant. (Parameter values are given in Appendix 39.8, but these values are not needed in the present problem.)

The control system is based on feedback control and feedforward control. The feedback controller is a PI controller which is tuned using the SIMC formulas for a "time constant process", cf. Ch. 14.8.4:

$$K_c = \frac{t_c}{K t_{cc}} \tag{33.4}$$

$$T_i = \min\left[t_c, \, 4t_{cc}\right] \tag{33.5}$$

where t_{cc} is the specified closed-loop time constant. Assume (for simplicity) that it decided to use

$$T_i = t_c \tag{33.6}$$

The feedforward controller is based on the motor model (33.3) from estimated load torque L_{est} and speed reference r:

$$u_{\text{ff}} = \frac{1}{K} t_c r' + r - L_{est} \tag{33.7}$$

L_{est} is estimated with an estimator (an observer or a Kalman Filter – which one of these does not matter here), which uses the mathematical model of the motor to calculate the estimate.

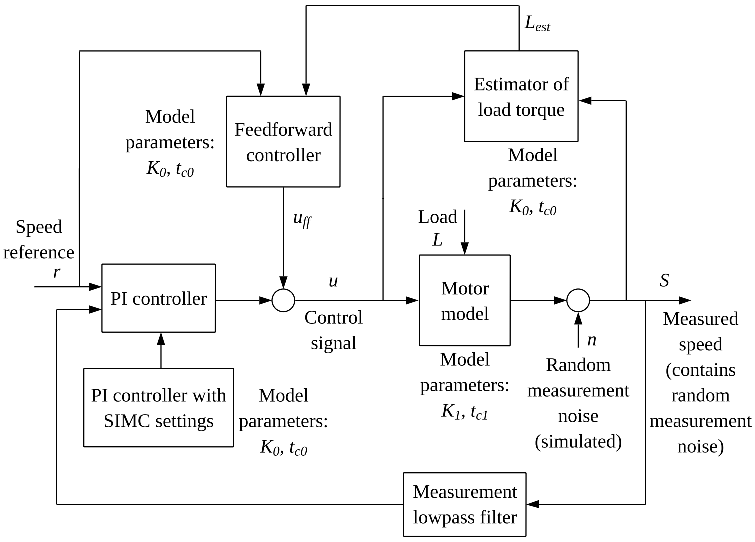
The feedback controller, the feedforward controller, and the estimator are model-based. Hence, the whole control system is *model-based*.

Assume that you intend to test the robustness of the model-based control system against model errors, and also that you want to see how the measurement noise is influencing the behaviour of the control system. Unfortunately, you can not perform experiments on the real motor. Instead you must use a simulated motor. Explain how you can do this simulated experiments. Draw a block diagram similar to the one shown in Figure 33.2, where you indicate which model parameters to use in the individual blocks. You can assume that it is interesting to see if the control system is robust against model parameter variations of ± 20 %.

33.2 *Solutions to problems for Chapter 33*

Solution to Prob

Figure 33.3 shows th ɔr LabVIEW).



L_{est}
Estimator of load torque
Model parameters: K_0, t_{c0}
Feedforward controller
Model parameters: K_0, t_{c0}
u_{ff}
Speed reference r
PI controller
u
Control signal
Load L
Motor model
Model parameters: K_1, t_{c1}
n
Random measurement noise (simulated)
S
Measured speed (contains random measurement noise)
PI controller with SIMC settings
Model parameters: K_0, t_{c0}
Measurement lowpass filter

Figure 33.3: Control system.

You can run the simulations with model parameters (indexed with 0 in the figure) in the simulated motor that are *different* from the model parameters (indexed with 1 in the figure) used in the feedback controller, the feedback controller and the estimator. Measurement noise is included in the simulator using a proper random signal generator.

Parameters K_0 and t_{c0} are the parameters you assume are accurately known and therefore use in the feedback controller, the feedback controller and the estimator. Parameters K_1 and t_{c1} are parameters you can use in the simulated motor when you want to introduce model errors. You can for example systematically vary K_1 between

$$0.8K_0 \le K_1 \le 1.2K_0 \tag{33.8}$$

and vary t_{c1} between

$$0.8t_{c0} \le t_{c1} \le 1.2t_{c0} \tag{33.9}$$

and run simulations with each of the different parameter sets.

Chapter 34

Linear Quadratic (LQ) optimal control

34.1 Introduction

Optimal control of a process means that the control function is designed so that a given *optimization criterion* or *performance index* gets a minimal value. It is assumed that the process can be described by a *linear* model, and that the criterion contains a *quadratic* function of the state variables and the control variables.[1] This type of optimal control is therefore denoted *Linear Quadratic* control – or *LQ control*. The reference is assumed to be zero in the basic LQ control problem, and hence the term *LQ regulation* or LQR is also used. We will however consider a non-zero reference in this chapter, cf. Section 34.3.

A particular feature of the LQ controller is that it will stabilize any linear process! However, you do not have a guarantee that is will stabilize any nonlinear process, even if this process is linearizable. A simulation study should be made to check if the control system works well under varying conditions, including model errors.

LQ control can be applied to both monovariable and multivariable processes. It turns out that the control function is based on feedback from all the states of the process. If not all the states can be measured, an observer or a Kalman Filter can be used to estimate the states, and the controller then uses the estimated states as if they were measured. This principle is denoted the *certainty equivalence principle.* It turns out that the control function and the state estimator can be designed independently as long as the process is linear. The principle of separate design of the controller and the estimator is denoted the *separation principle.*

LQ controllers can be designed for continuous-time and for discrete-time systems, and for stochastic systems (systems excited by random disturbances) and determinstic systems (random noise is not taken into account in the controller design). I have chosen to describe LQ control for deterministic continuous-time systems. Of course, in a practical

[1] The main reason why a quadratic criterion is used is that the control function is relatively easy to derive and easy to implement :-)

implementation you will (probably) need a discrete-time implementation of the controller, and this will be described in this chapter.

LQ control is quite similar to Model-based Predictive Control (MPC), which has become an important control function the last decades. Also MPC is based on a quadratic criterion. However, MPC takes into account limitations in the control variables and the state variables, hence making it somewhat more useful than LQ controller, but also much more computational demanding to implement. MPC is described in Chapter 35.

34.2 The basic LQ controller

In basic LQ control it is assumed that the process to be controlled is given by the following linear state space model

$$x'(t) = Ax(t) + Bu(t) \tag{34.1}$$

The LQ controller brings the state x from any initial state $x(0)$ to zero in an optimal way. What is "optimal"? It is defined by the optimization criterion:

$$J = \int_{t=0}^{t=\infty} \left[x^T(t)Qx(t) + u^T(t)Ru(t) + 2x^T(t)Nu(t) \right] dt \tag{34.2}$$

It is very common that the weight matrix N is a zero matrix (of proper dimension), and in these cases the criterion is

$$J = \int_{t=0}^{t=\infty} \left[x^T(t)Qx(t) + u^T(t)Ru(t) \right] dt \tag{34.3}$$

N is assumed to be zero in the following.

Q and R are *weight or cost matrices* of the states and the control signal, respectively. Q and R are selected by the user, and they are the tuning parameters of the LQ controller. Q is a symmetric positive semidefinite matrix, and R is a symmetric positive definite matrix.

The criterion (34.2) gives you (the user) the possibility to punish large variations in the states (by selecting a large Q) or to punish large variations in the control variable u (by selecting a large R). It is fair to say that the LQ controller is a user-friendly controller because the tuning parameters (Q and R) are meaningful, at least in the qualitative sense.

As an example, assume that the system has two state variables, x_1 and x_2, hence

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{34.4}$$

and one (scalar) control variable, u, and that the weight matrices are:

$$Q = \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \tag{34.5}$$

$$R = [R_{11}] \tag{34.6}$$

The criterion J becomes

$$J = \int_{t=0}^{t=\infty} \left[x^T Q x + u^T R u\right] dt \tag{34.7}$$

$$= \int_{t=0}^{t=\infty} \left\{ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + uRu \right\} dt \tag{34.8}$$

$$= \int_{t=0}^{t=\infty} \left\{ Q_{11} x_1^2 + Q_{22} x_2^2 + R_{11} u^2 \right\} dt \tag{34.9}$$

Thus, J is a sum of quadratic terms of the state variables and the control variable.

It can be shown that the control function that gives J a minimum value is as follows

LQ controller:

$$u(t) = -G(t)x(t) \tag{34.10}$$

In other words, *the control signal is based on feedback from a linear combination of the state variables*. The controller gain $G(t)$ (a matrix) is given by the Riccati equation which will not be shown here. Figure 34.1 shows a block diagram of the control system.

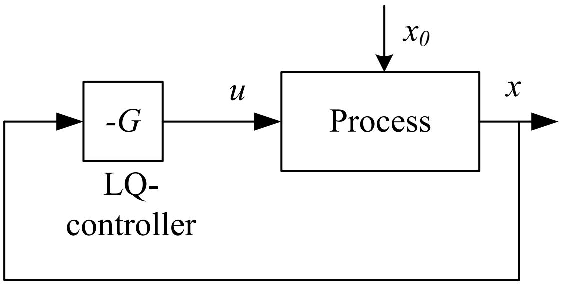


Figure 34.1: Control system with optimal LQ controller.

In the above example, the controller becomes

$$u(t) = -G(t)x(t) = -\begin{bmatrix} G_{11}(t) & G_{12}(t) \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{34.11}$$

$$= -\left[G_{11}(t)x_1(t) + G_{12}(t)x_2(t)\right] \tag{34.12}$$

It is common to implement the *steady state value* $G = G(t = \infty)$ of the controller gain:

Steady-state LQ controller:

$$u(t) = -Gx(t) \tag{34.13}$$

G can be calculated offline, and in advance (before the control system is started).

G can be calculated from the following formulas:

$$G = R^{-1}\left(B^T P + N^T\right) \tag{34.14}$$

where P is the solution of the steady state Riccati equation:

$$A^T P + PA - (PB + N) R^{-1} \left(B^T P + N^T\right) + Q = 0 \tag{34.15}$$

The Python Control Package (a guide to the package is in Appendix 41) has the function lqr to calculate G:

(G, S, E) = control.lqr(A, B, Q, R, [N])

where:

- A and B are the model matrices in (34.1) in the form of 2D-arrays.
- Q, R, and N are the weight matrices in (34.2) in the form of 2D-arrays. It is common that N is zero, and in that case, N can be omitted from the input argument list of the lqr() function.
- G is the array of steady state LQ controller gain.
- S is the array of steady state solution of the Riccati equation.
- E is the array of eigenvalues of the closed loop (feedback) control system.

Figure 34.2 illustrates the information what information is needed to compute the steady state LQ controller gain G.

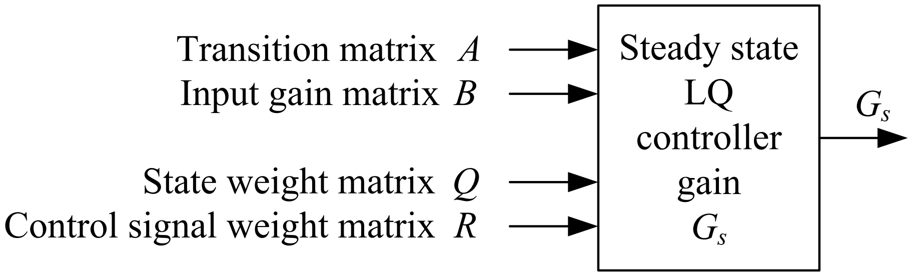


Figure 34.2: Information needed to compute the steady state LQ controller gain, G.

Here are comments about the LQ controller:

- **The reference is zero, and there are no process disturbances**: These assumptions seem somewhat unrealistic, because in real control systems the reference is typically non-zero, and the disturbances are non-zero. Therefore, for the controller to represent realistic control problems, the variables in (34.1) should actually be regarded as deviation variables about some operating point. Then, how do you bring the states to the operating point, so that the mean value control error is zero? By enforcing *integrators* into the controller. This is described in detail in Section 34.3.

- **Controllability:** The process to be controlled has to be *controllable*. If it is not controllable, there exists no finite steady state value of the gain G. Controllability means that there exists a control signal $u(t)$ so that any state can be reached from any initial state in finite time. It can be shown that a system is controllable if the rank of the controllability matrix

$$M_{\text{control}} = \left[B \vdots AB \vdots A^2B \vdots \cdots \vdots A^{n-1}B \right] \tag{34.16}$$

  is n (the order of A, which is the number of state variables).

- **Non-measured states**: If not all the states are measured, you can use estimated states instead of measured states in the controller:

$$u(t) = -Gx_{\text{est}}(t) \tag{34.17}$$

  where x_{est} is the estimated state vector from a state estimator (observer or Kalman Filter). Figure 34.3 shows a block diagram of the control system with state estimator.

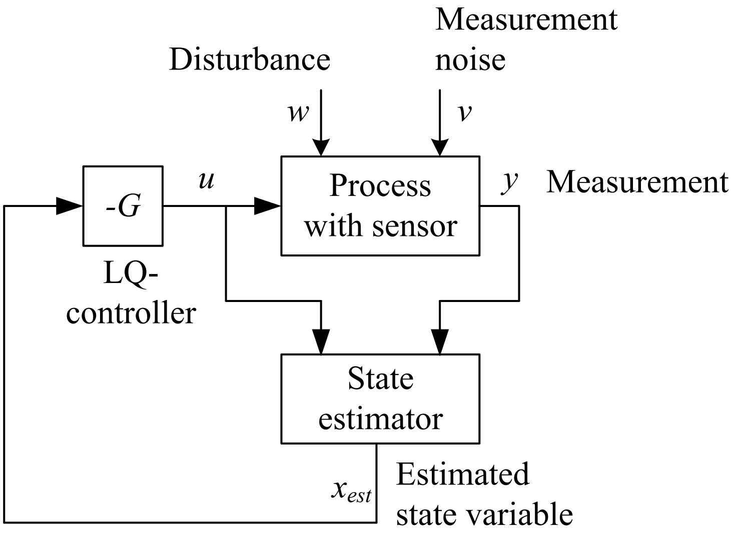


Figure 34.3: Control system with state estimator

- **The eigenvalues of the control system**: Assume that the controller is (34.17). By combining this controller with the process model (34.1) we get the following model of the *control system* (the closed-loop system):

$$\dot{x} = Ax + B(-Gx) \tag{34.18}$$

$$= (A - BG)x \tag{34.19}$$

  The eigenvalues $\{s_1, s_2,...s_n\}$ of the control system are the eigenvalues of the transition matrix $(A - BG)$ in (34.19):

$$0 = \det[sI - (A - BG)] \tag{34.20}$$

$$= (s - s_1)(s - s_2)\cdots(s - s_n) \tag{34.21}$$

$$= s^n + a_{n-1}s^{n-1} + \cdots + a_1s + a_0 \tag{34.22}$$

- **Stability of the control system**: It can be shown that a LQ control system is asymptotically stable.[2] In other words, the eigenvalues of the control system are in the left half of the complex plane.

- **Tuning of the LQ controller**: From the criterion (34.2) we can conclude that *a larger value of the weigth of one particular state variable* causes the time response of that state variable to become smaller, and hence *the control error (deviation from zero) is smaller*. But what should be the initial values of the weight matrices, before they are tuned? One possibility is

$$Q = \operatorname{diag}\left\{\frac{1}{|x_{i_{\max}}|^2}\right\} \tag{34.23}$$

where $x_{i_{\max}}$ is the assumed maximum value of state variable x_i, and

$$R = \operatorname{diag}\left\{\frac{1}{|u_{j_{\max}}|^2}\right\} \tag{34.24}$$

where $u_{j_{\max}}$ is the assumed maximum value of control variable u_j.

- **Pole placement design of the control system:** Above it was stated that the eigenvalues of the control system are the roots of the characteristic equation

$$0 = \det\left[sI - (A - BG)\right] \tag{34.25}$$

For most systems, the poles, $\{s_i\}$, of the system are the same as the eigenvalues. In the following the term poles are used instead of eigenvalues. In pole placement design of control systems the poles are specified by the user, and, assuming the controller has the linear feedback structure as given by (34.13), it is usually possible to solve (34.25) for the controller gain matrix G.

Example 34.1 *LQ control of pendulum on cart*

This example is about stabilization of a pendulum on a cart using LQ optimal control. A mathematical model of the pendulum, and linearized models for the standing position and for the hanging position, are given in Ch. 39.10.

Controller

Assume that the reference of the cart is position 0 m, and the reference of the pendulum is vertically up, i.e. at angle 0. A steady state LQ controller designed at this operating point is:

$$u = -Gx = -\left[G_{11}x_1 + G_{12}x_2 + G_{13}x_3 + G_{14}x_4\right] \tag{34.26}$$

The controller output, u, is applied as the force F acting on the cart. Hence, the force is calculated as a linear combination of the states of the system. The states are assumed to be

[2] Not a big surprise, since the controller minimizes the criterion J.

available via measurements. (Thus, there is a feedback from the measured states to the process via the controller.)

If the reference of the cart is non-zero, r_{x_1}, and the reference of the pendulum angle is non-zero, r_{x_3}(=180° if vertically down), we can modify the controller (34.26) to become:

$$u = -Gx - [G_{11}(x_1 - r_{x_1}) + G_{12}x_2 + G_{13}(x_3 - r_{x_3}) + G_{14}x_4] \tag{34.27}$$

The controller gain,

$$G = [G_{11}, G_{12}, G_{13}, G_{14}]$$

will be calculated using the lqr() function of the Python Control Package described earlier:

(G, S, E) = control.lqr(A, B, Q, R, [N])

where A and B are the matrices in (39.51) corresponding to the pertinent operating point (pendulum up; pendulum down), and Q is the state weight matrix and R is the control weight matrix of the LQ(R) optimization criterion (34.3).

Q has the following form:

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 & 0 \\ 0 & Q_{22} & 0 & 0 \\ 0 & 0 & Q_{33} & 0 \\ 0 & 0 & 0 & Q_{44} \end{bmatrix} = \text{diag}(Q_{11}, Q_{12}, Q_{13}, Q_{14}) \tag{34.28}$$

Q_{ii} can be used as controller tuning parameters. R is

$$R = [R_{11}] \tag{34.29}$$

and is also used as tuning parameter. However, in this example, the scalar R_{11} can be set to 1 (because (34.3) can be scaled by dividing it by R_{11}), leaving Q_{ii} as the only tuning parameters.

Simulations

The following Python program implements the controller tuning, i.e. calculating the G matrix in (34.27), and simulates the cart with pendulum including the controller.

http://techteach.no/control/python/sim_pendulum_on_cart_lqr.py

Note: In the simulations, the non-linear model (39.36)-(39.39) is used to represent the real cart with pendulum. The linear model (39.51) is used only for controller tuning.

In the program, the code cell entitled *Selection of mode* contains the variable op_mode which can be set to select between standing up and hanging down position of the pendulum.

Pendulum up:

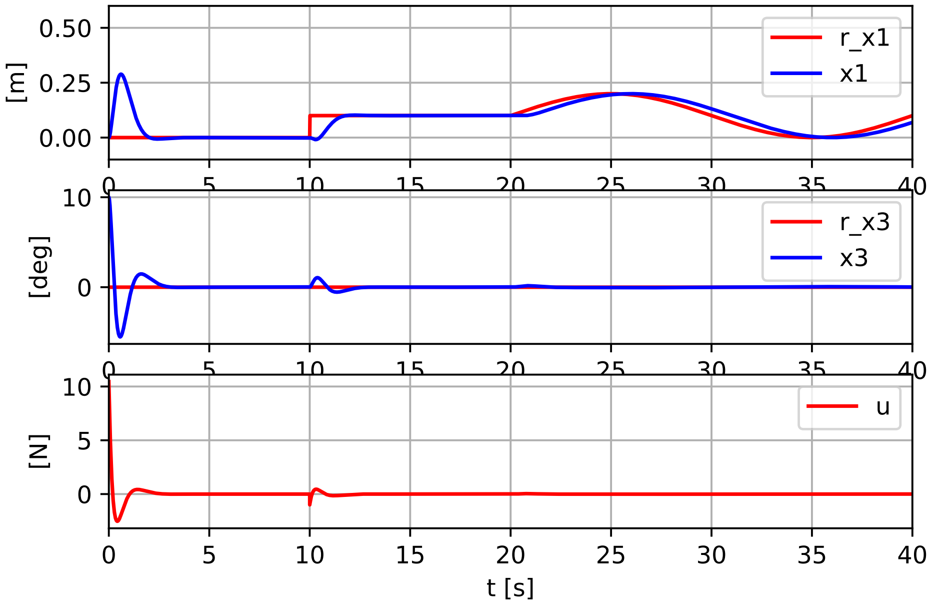


Figure 34.4: Simulated responses for the cart with pendulum with LQ control. The pendulum is controlled to a standing position, while the cart position is varied.

Figure 34.4 shows simulated responses where the pendulum angle reference is $r_{x_3} = 0$ deg = 0 rad, and the cart position reference r_{x_1} is varying. The initial pendulum angle is 10 deg. The weight matrix Q is set to $Q = \text{diag}(100, 0, 100, 0)$, and $R_{11} = 1$. The control system works well: The cart follows well the varying position reference, and the pendulum is stabilized upwards.

Pendulum down:

Figure 34.5 shows simulated responses where the pendulum angle reference is $r_{x_3} = 180$ deg $= \pi$ rad, and the cart position reference r_{x_1} is varying.

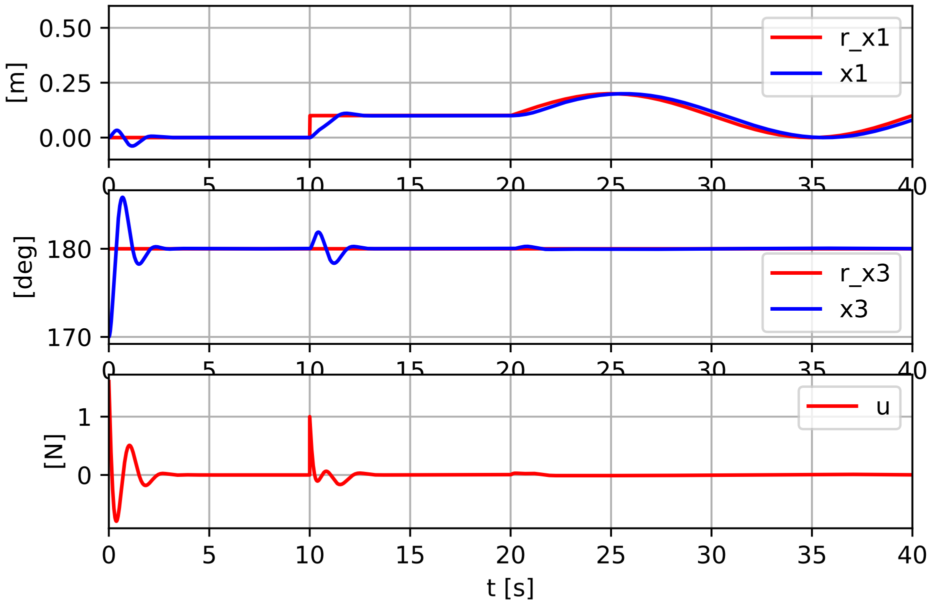


Figure 34.5: Simulated responses for the cart with pendulum with LQ control. The pendulum is controlled to a hanging position, while the cart position is varied.

The initial pendulum angle is 170 deg. The weight matrix Q is set to $Q = \text{diag}(100, 0, 100, 0)$, and $R_{11} = 1$. The control system works well: The cart follows well the varying position reference, and the pendulum is stabilized upwards.

[End of Example 34.1]

34.3 LQ controller with integral action

34.3.1 Introduction

The basic LQ controller described in Section 34.2 does not have integral action. If process disturbances are prevalent, the process variables may not reach their references, due to the lacking integral action of the controller. Let us include integral action in the controller!

34.3.2 Including integrators in the controller

Figure 34.6 shows a block diagram of the control system with integrators included in the controller.

The integrator block actually represents a number of single integrators, as many as there are reference variables.

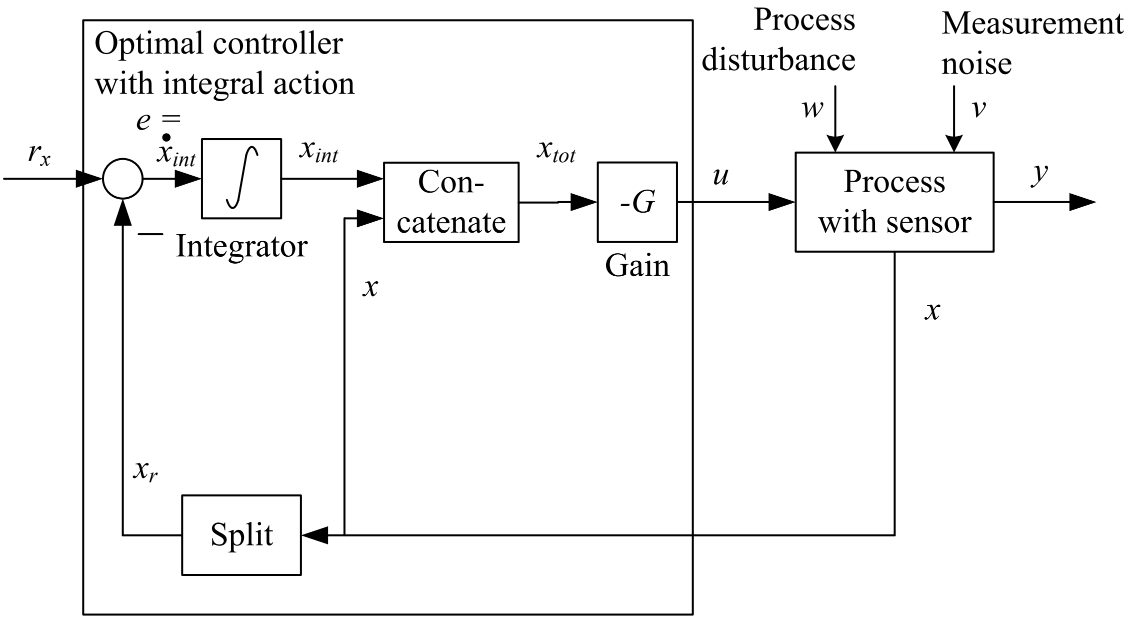


Figure 34.6: Optimal control system with integrators in the controller. e is the contol error.

Example 34.2 ***LQ controller with integrator***

Figure 34.7 shows the detailed structure of a process with two state variables being controlled by a LQ controller with integrator.

[End of Example 34.2]

The output of the integrators are regarded as augmentative state variables, x_{int}. These state variables are given by the following differential equation(s):

$$x'_{\text{int}} = r_x - x_r \tag{34.30}$$

which corresponds to this integral equation:

$$x_{\text{int}}(t) = \int_0^t (r_x - x_r)\, d\tau \tag{34.31}$$

Here, r_x is the reference vector for the state vector x_r which consists of those state variables among the process state vector x that are to track a reference. (In the above equations it is assumed that x_r is directly available from measurements. If x_r is taken from a state estimator, $x_{r,\text{est}}$ is used instead of x_r, of course.)

The total state vector that is used to design the LQ controller is the state vector consisting of the original process state vector augmented with the integrator state vector:

$$x_{\text{tot}} = \begin{bmatrix} x \\ \cdots \\ x_{\text{int}} \end{bmatrix} \tag{34.32}$$

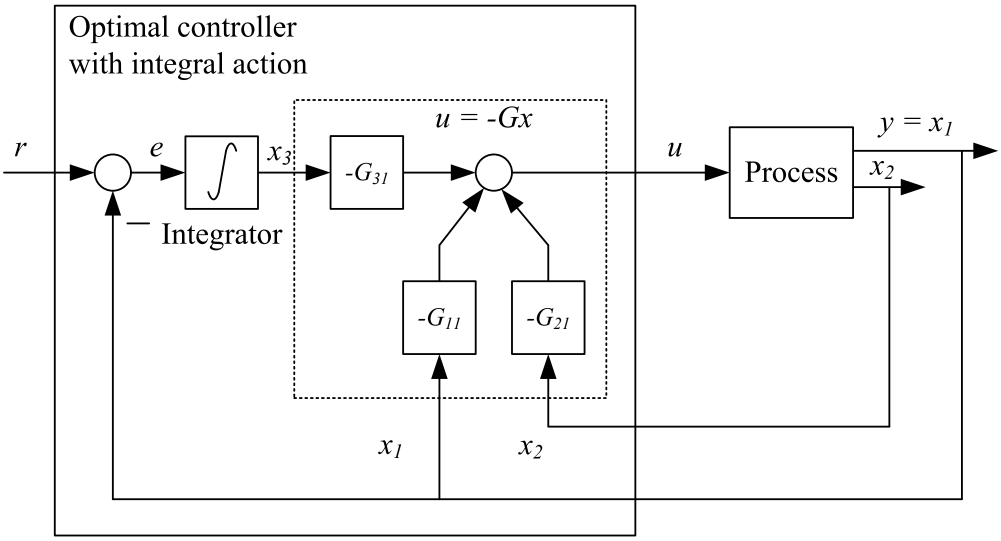


Figure 34.7: Example 34.2: A process with two state variables being controlled by a LQ controller with integrator.

The control variable u is given by the control function

$$u = -Gx_{\text{tot}} = -G \begin{bmatrix} x \\ \cdots \\ x_{int} \end{bmatrix} \tag{34.33}$$

Note: When writing up the state space model that is used for designing the LQ controller, you can disregard the reference r_x, i.e. you set it to zero because it is not taken into account when calculating the controller gain G. But of course it must be included in the implemented controller which is given by (34.33), with x_{int} given by (34.35).

34.3.3 Discrete-time implementation of the LQ controller

In a computer-based implementation of a LQ controller you will probably need to discretize the continuous-time integrator (34.30). This can be done using Backward or Forward discretization. The Backward method is the best with respect to numerical accuracy, and it can be applied to (34.30) without any problems because it is a linear differential equation. Applying Backward discretization on (34.30) gives

$$x'_{\text{int},k} \approx \frac{x_{\text{int},k} - x_{\text{int},k-1}}{t_s} = r_{x,k} - x_{r,k} \tag{34.34}$$

Solving for $x_{\text{int},k}$ gives the final integrator algorithm ready for being programmed:

$$x_{\text{int},k} = x_{\text{int},k-1} + t_s \left[r_{x,k} - x_{r,k} \right] \tag{34.35}$$

A practical issue of any controller having integral action is *anti wind-up*, which is a feature to prevent the integrator to “wind up” – or increasing its output continually – while the total control signal is at its saturation limit, either the maximum or the minimum limit. If anti wind-up is not implemented, the control error may become unnecessarily large for an unnecessarily long time.

34.4 *Problems for Chapter 34*

Problem 34.1 *LQ control of ship*

See Figure 4.19 which shows a ship. In this problem we concentrate on the so-called surge (forward-backward) direction, i.e., the movements in the other directions are disregarded. The wind acts on the ship with the force F_w which is a function of the wind attack angle ϕ and the wind speed V_w. This function is assumed to be known for a given ship. The hydrodynamic damping force F_h (damping from the water) is proportional to the square of the difference between the ship speed, $\dot{y}$, and the water current speed u_c. The proportionality constant is D.

Applying Newtons's Law of Motion we obtain the following mathematical model of the surge motion:

$$my'' = \underbrace{-D|y' - u_c|\left(y' - u_c\right)}_{F_h} + F_w(\phi, V_w) + F_t \tag{34.36}$$

(Model parameter values are given in Appendix 39.9, but these values are not needed in the present problem.)

This problem is about ship position control using LQ optimal control. Assume that the water current u_c has a known value at any instant of time (in a practical application it may have been estimated with a state estimator, e.g. a Kalman Filter). Also, assume that the wind force F_w has a known value at any instant of time (with a mathematical wind model the wind force can be calculate from information about the wind attack angle ϕ and the wind speed V_w provided by the sensor). We also assume that the ship position y and speed $\dot{y}$ are known at any instant of time.

1. Write the ship model as a (nonlinear) state space model using $x_1 = y$ and $x_2 = y'$ as state variables. F_t is control variable.

2. The ship position will be controlled with LQ control with integral action. Figure 34.8 shows a block diagram of the ship.

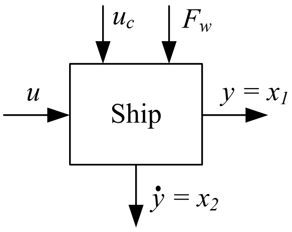


Figure 34.8: Block diagram of ship

Enhance this block diagram so that it shows the control system in detail, including the feedbacks and the integrator of the controller.[3] You can assume that the controller gains have known values (calculation of these gains is the focus of a following task).

[3] Feedforward from wind force F_w and water current u_c is an enhancement of the control system, but we will not include feedforward in this problem.

3. Augment the state space model found in Problem 1 above with the state variable of the integral part of the controller. (This augmented state space model is needed in the following problem.)

4. Figure 34.2 shows the matrices needed to compute the steady state LQ controller gain G_s (using a proper function in e.g. Matlab or LabVIEW). Find the matrices A and B (by linearizing the augmented state space model found in the problem above). Also, write in detail the weight matrices Q and R (the matrix elements) as functions of the allowable maximum values of the proper variable in the model (you do not have to specify these maximum values).

5. Suppose you want to reduce the fuel consumption (less aggressive control). How can you adjust some of the weights of the LQ criterion to obtain this?

6. Suppose you want to increase the damping in the control system. In other words: You want to limit (reduce) the speed of the ship. How can you adjust some of the weights of the LQ criterion to obtain this?

34.5 *Solutions to problems for Chapter 34*

Solution to Problem 34.1

1.

$$x_1' = \overbrace{x_2}^{f_1} \tag{34.37}$$

$$x_2' = \overbrace{\frac{1}{m}\left[-D|x_2 - u_c|\,(x_2 - u_c) + F_w(\phi, V_w) + F_t\right]}^{f_2} \tag{34.38}$$

2. See Figure 34.9.

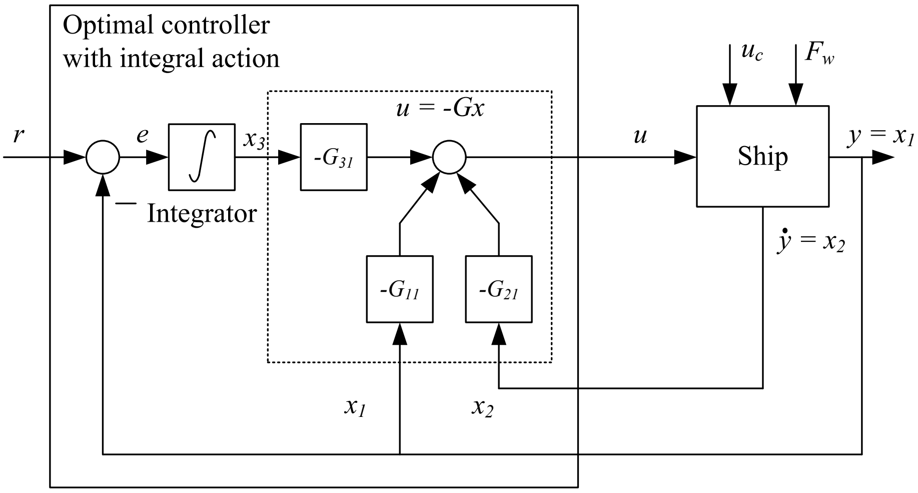


Figure 34.9: State feedback control system.

3. See Figure 34.9. The state-variable of the integrator is defined by

$$x_3' = r - x_1 \tag{34.39}$$

The augmentet (total) state space model becomes

$$x_1' = \overbrace{x_2}^{f_1} \tag{34.40}$$

$$x_2' = \overbrace{\frac{1}{m}\left[-D|x_2 - u_c|\,(x_2 - u_c) + F_w(\phi, V_w) + F_t\right]}^{f_2} \tag{34.41}$$

$$x_3' = \overbrace{r - x_1}^{f_3} \tag{34.42}$$

4. Matrices A and B are found by linearization:

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{2D}{m}|x_2 - u_c| & 0 \\ -1 & 0 & 0 \end{bmatrix} \tag{34.43}$$

$$B = \begin{bmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \\ \frac{\partial f_3}{\partial u} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{m} \\ 0 \end{bmatrix} \tag{34.44}$$

(Element $A(2,2)$ can be found by resolving the absolute value by first assuming $(x_2 - u_c)$ is positive and then assuming $(x_2 - u_c)$ is negative, then taking the partial derivative, and finally expressing the results of the partial differentiations compactly.)

State weight matrix:

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 \\ 0 & Q_{22} & 0 \\ 0 & 0 & Q_{33} \end{bmatrix} = \begin{bmatrix} \frac{1}{|x_{1\max}|^2} & 0 & 0 \\ 0 & \frac{1}{|x_{2\max}|^2} & 0 \\ 0 & 0 & \frac{1}{|x_{3\max}|^2} \end{bmatrix} \tag{34.45}$$

State weight matrix:

$$R = \begin{bmatrix} R_{11} \end{bmatrix} = \begin{bmatrix} \frac{1}{|u_{\max}|^2} \end{bmatrix} \tag{34.46}$$

5. To reduce the fuel consumption (less aggressive control) you can increase the cost (weight) of the control signal, i.e. increase R_{11}.

6. To reduce the speed of the ship you can increase the cost (weight) of the speed, i.e. increase Q_{22}.

Chapter 35

Model Predictive Control (MPC)

35.1 Introduction

Model predictive control (MPC) is the dominant model-based control method. In year 2002, Maciejowski stated that "MPC is the only advanced control technique that is more advanced than standard PID to have a significant and widespread impact on industrial process control" Maciejowski (2002). It is fair to say that this statement still holds. However, it will be exciting to see what will be the role of artifical intelligence (AI) in control of technical and industrial systems. Will it replace MPC and PID control?

The history of MPC may be traced back to Dynamic Matrix Control (DMC) method implemented by Cutler and Ramaker at Shell Oil in 1973 Cutler & Ramaker (1980). A standard overview over MPC technology is given in Qin & Badgwell (2003). A more recent overview is given in Lee (2011).

MPC is available in various professional and industrial software tools, e.g. DeltaV Predict (Emerson Process), 800xA APC (ABB), PCS7 (Siemens), MPC Toolbox of Matlab and Simulink (Mathworks), and Control Design Toolkit of LabVIEW (National Instruments).

MPC exists in different versions. Here, nonlinear MPC is presented. The term "nonlinear" is used because the underlying mathematical model of the process to be controlled, is a *nonlinear* state space model. The model may be multivariable and may contain time delays. Nonlinear MPC can of course be applied to linear models, too, since linear state space models are just a special case of nonlinear state space models.

35.2 The MPC method

35.2.1 The principle of MPC

The principle of MPC is continuously calculation of the optimal ("best") control signal sequence over a future or prediction time horizon using the following information:

- A process model. The model is used by the optimizer to simulate the process over the prediction horizon.
- The current process state as obtained from measurements and/or state estimates from a state estimator which typically is in the form of a Kalman Filter.
- Reference values and process disturbance values known over the prediction horizon.
- Constraints (maximum and minimum values) on the control signal, the process variable, and state variables.

It can be claimed that MPC resembles closely how a human controls a process, like driving a car: The driver looks ahead to take into account future disturbances like other cars, pedestrians and other obstacles, and future speed references as shown on the signs ahead, while manipulating the various actuators (throttle, break, steering wheel, gear).

The predictions/simulations in the MPC can be based on any model that is representative of the process to be controlled. In the MPC presented in this chapter, the following discrete time nonlinear state space model is used.

$$x_{k+1} = f(x_k, u_{k,}, d_k, \cdot) \tag{35.1}$$

$$y_k = g(x_k, \cdot) \tag{35.2}$$

x is the state vector, y is the process output variable vector, u is the control signal vector, and d is the process disturbance vector. f and g are nonlinear (or linear) vectorial functions.

In the MPC presented here, no linearization of the model is needed. So, it is a "nonlinear MPC". There are MPC algorithms assuming a linear model – "linear MPC", for example the MPC available in LabVIEW, which requires linearization of the process model used by the MPC. The linear model assumed is on the the following form:

$$x_{k+1} = Ax_k + B_1 u_k + B_2 d_k \tag{35.3}$$

$$y_k = Cx_k + Du_k \tag{35.4}$$

Nonlinear MPC can be used with linear models too, since a linear model (35.3)-(35.4) is just a special case of a nonlinear model (35.1)-(35.2).

Figure 35.1 illustrates the principle of MPC. The predicted values are found by successive simulations over the prediction horizon performed by the optimizer until the optimal solution (optimal control signal sequence) has been found. The optimal control sequence or array (or matrix in the multivariable case), u_{opt} is calculated as the solution of an optimization problem where typically the future (predicted) control errors and control signal changes are minimized in a least squares sense. And from this optimal future control sequence, the first element is picked out and applied as control signal to the process, i.e.

$$u_k = u_{\text{opt}}(1) \tag{35.5}$$

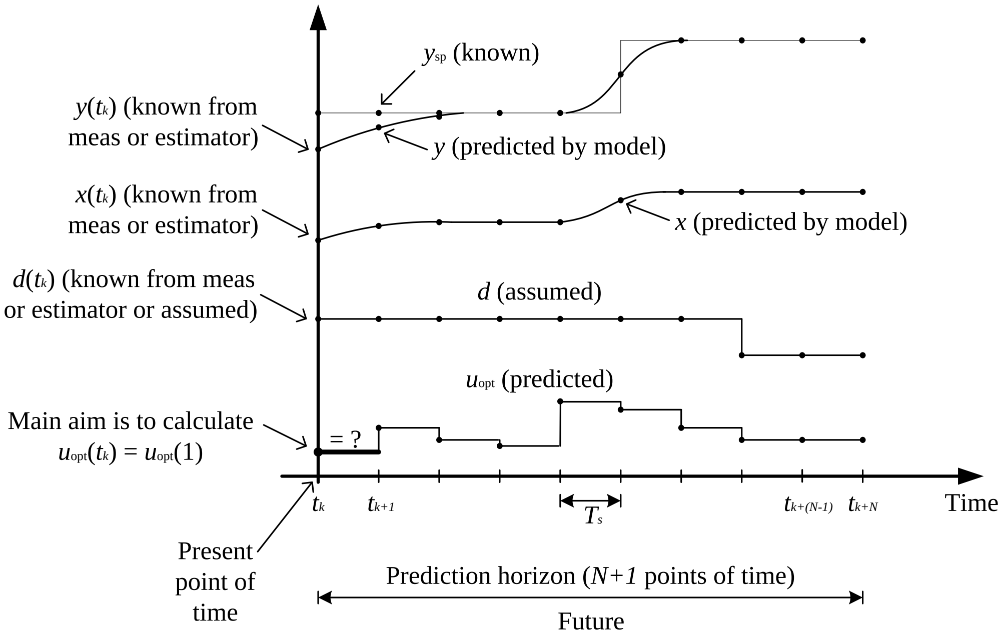

y_{sp} (known)
$y(t_k)$ (known from meas or estimator)
y (predicted by model)
$x(t_k)$ (known from meas or estimator)
x (predicted by model)
$d(t_k)$ (known from meas or estimator or assumed)
d (assumed)
u_{opt} (predicted)
Main aim is to calculate $u_{opt}(t_k) = u_{opt}(1)$
= ?
t_k
t_{k+1}
T_s
$t_{k+(N-1)}$
t_{k+N}
Time
Present point of time
Prediction horizon ($N+1$ points of time)
Future

Figure 35.1: The principle of MPC.

35.2.2 The objective function of MPC

The objective function, or optimization criterion, to be minimized in MPC may be stated as follows:

$$\min_U J \tag{35.6}$$

J is the objective function. It is defined below. U is a matrix containing the r control signals at each point of time of the prediction horizon:

$$\begin{aligned} U &= \left[u_k, u_{k+1}, \cdots, u_{k+(N-1)}, u_N\right] \\ &= \begin{bmatrix} u(1)_k & u(1)_{k+1} & \cdots & u(1)_{k+(N-1)} & u(1)_{k+N} \\ u(2)_k & u(2)_{k+1} & \cdots & u(2)_{k+(N-1)} & u(2)_{k+N} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ u(r)_k & u(r)_{k+1} & \cdots & u(r)_{k+(N-1)} & u(r)_{k+N} \end{bmatrix} \end{aligned} \tag{35.7}$$

The number of optimization variables is the number of elements of U. The number is $r(N+1)$.

U can be denoted the total control signal matrix. U will be the solution of the MPC optimization problem. From this U,

$$u_k = \begin{bmatrix} u(1)_k \\ u(2)_k \\ \vdots \\ u(r)_k \end{bmatrix}$$

is used as the control signal applied to the process actuator.

The objective function to be minimized, cf. Eq. (35.6), is

$$J = \sum_{i=k}^{k+N} \left(\|e\|_{C_e}^2 + \|u'\|_{C_{du}}^2 \right) \tag{35.8}$$

where expressions like $\|\cdot\|_M$ means M-quadratic norm. In more detail, Eq. (35.8) is

$$J = \sum_{i=k}^{k+N} \left[{e_i}^T C_e e_i + {u_i'}^T C_{du} {u'}_i \right] \tag{35.9}$$

The symbols in Eq. (35.9) are described below.

The control error vector:

$$e_i = \begin{bmatrix} e(1)_i \\ \vdots \\ e(m)_i \end{bmatrix} \tag{35.10}$$

where $e(j)_i$ is the control error related to process output no. j at time-index i:

$$e(j)_i = r(j)_i - y(j)_i \tag{35.11}$$

The control signal change vector:

$$du_i = \begin{bmatrix} du(1)_i \\ \vdots \\ du(r)_i \end{bmatrix} \tag{35.12}$$

where $u'(j)_i$ is the rate of change of the control signal which can be implemented as a backward difference:

$$u'(j)_i = \frac{u(j)_i - u(j)_{i-1}}{t_s} \tag{35.13}$$

The matrixes C_e and $C_{u'}$ in Eq. (35.8) are cost (or weight) matrixes which typically are set as constant matrixes:

$$C_e = \begin{bmatrix} C_e(1,1) & & 0 \\ & \ddots & \\ 0 & & C_e(m,m) \end{bmatrix} \tag{35.14}$$

$$C_{u'} = \begin{bmatrix} C_{u'}(1,1) & & 0 \\ & \ddots & \\ 0 & & C_{u'}(r,r) \end{bmatrix} \tag{35.15}$$

C_e and C_{du} are tuning factors in MPC.

Now, Eq. (35.9) can be written in detail as

$$J = \sum_{i=k}^{k+N} \left\{ \left[C_e(1,1)\, {e(1)_i}^2 + \cdots + C_e(m,m)\, {e(m)_i}^2 \right] \right. \tag{35.16}$$

$$\left. + \left[C_{u'}(1,1)\, {u'(1)_i}^2 + \cdots + C_{u'}(r,r)\, {u'(r)_i}^2 \right] \right\} \tag{35.17}$$

Roughly said, MPC produces the control signal that gives the optimal comprimise between control errors and control signal changes. It is not possible to obtain both very small control errors and very small control signal changes. Hence, a comprimise will always exist.

Constraints

You may include constraints in the MPC optimization problem: Typically, upper and lower bounds are set for the control signal. Furthermore, you can set limits on the process output variable and on certain state variables. For example, if the liquid level in a tank is one state variable, it is natural to define a maximum level limit and a minimum level limit.

Guessed value of U

When solving the optimization problem, it is necessary that the optimizer is supplied with a good guess of the optimization variable, U. As a good value of U_{guess} at time index k, here denoted $U_{\text{guess}_\text{k}}$, we can use the optimal solution found at time index $k-1$ (the previous point of time):

$$U_{\text{guess}_k} = U_{\text{opt}_{k-1}} \tag{35.18}$$

Computing tools

The MPC optimization criterion may be solved in Python and Matlab with the following optimization solvers:

- Python: scipy.minimize with solver slsqp.
- Matlab: fmincon in the Optimization Toolbox, or with relevant functions in the MPC Toolbox.

35.2.3 Control signal blocking

Control blocking, or control grouping, can be used to reduce the number of optimization variables. Control blocking is to fix the control signal in time-blocks in the predition horizon, see Figure 35.2. My experience from is that using as small number as 3 intervals may not detoriate the performance of the MPC. I have even tried using only one interval (i.e. constant u throughout the horizon), with acceptable performance. Control blocking, as other settings, should be tested in simulations before being applied to a real process.

35.2.4 Tuning factors of MPC

The main tuning factors of MPC are:

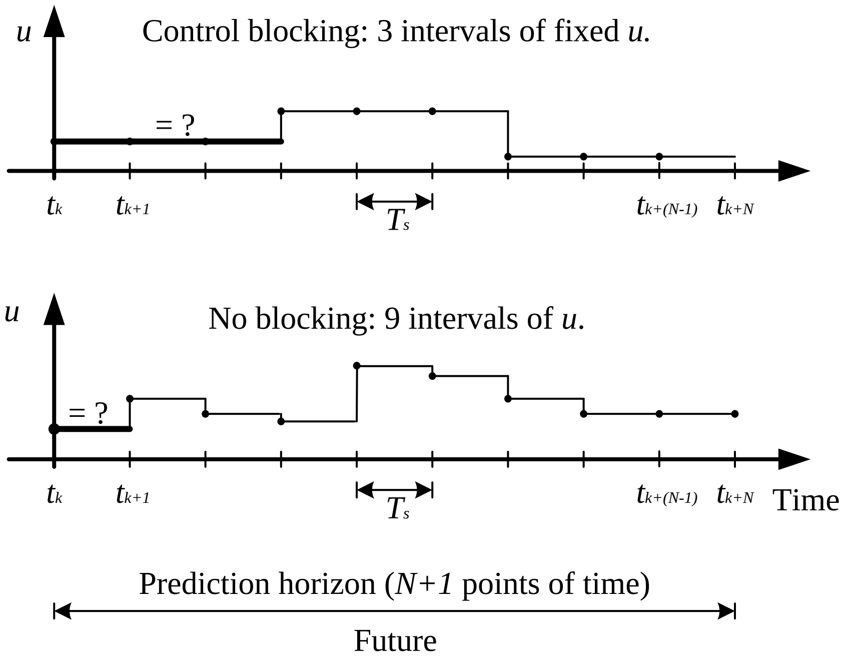

u
Control blocking: 3 intervals of fixed u.
= ?
tk
tk+1
Ts
tk+(N-1)
tk+N
u
No blocking: 9 intervals of u.
= ?
tk
tk+1
Ts
tk+(N-1)
tk+N
Time
Prediction horizon (N+1 points of time)
Future

Figure 35.2: Control blocking

- *The prediction horizon length, N.* The larger N, the better ability to take into account future references and disturbances. A drawback of selecting a large N, is the increase of the computational demand which is due to the more challenging optimization problem (more optimization variables to be optimized) and longer simulations. A typical value of N for simple applications seems to be between 5 and 50, assuming an appropriate time step length (which may be e.g. 1/5 of smallest time constant-like dynamics represented by the model). To reduce the compututional burden, control signal blocking can be considered, as explained earlier in this section.

- *The control error cost matrix, C_e.* Increasing the value of $C_e(j,j)$, forces the pertinent control error, $e(j)$, to become smaller[1], but at the expense of larger variation in the control signal. Initially, you may try setting $C_e(j,j)$ equal to the square of inverse of the maximum expected absolute value of the control error:

$$C_e(j,j) = \frac{1}{[|e(j)|_{\max}]^2} \tag{35.19}$$

This implies a normalization of the error terms in the objective function. For example, the first error term in Eq. (35.16) becomes

$$C_e(1,1)e(1)_i^2 = \frac{e(1)_i^2}{[|e(1)|_{\max}]^2}$$

Then, you may try using only $C_{u'}$ as a tuning parameter (cf. next item). In the scalar case, i.e. $m = 1$ and $r = 1$, you may simply set $C_e = 1$ since it is only the ratio between C_e and $C_{u'}$ that counts for the tuning.

[1] The higher cost of something, the less of it is bought/used.

- *The control signal rate of change cost matrix, $C_{u'}$.* Also the terms in $C_{u'}$ may be normalized before the tuning. Initially, you may try setting $C_{u'}(j,j)$ equal to the square of inverse of the maximum rate of change of the control signal:

  $$C_{u'}(j,j) = \frac{1}{\left[\left|u'(j)\right|_{\max}\right]^2} \tag{35.20}$$

  Increasing the cost gives smoother control signal (less change). Decreasing gives more abrupt changes in the control signal.

35.2.5 The need for a state estimator

The optimizer in MPC uses successive simulations for the prediction. For the simulations to become accurate, it is necessary that the initial state of the simulations are close to the present state, x_k, of the process to be controlled. Typically, not all the states are measured, and if so, a state estimator – most often a Kalman Filter – is used to provide an estimate of x_k. Even if all the states are estimated, state estimator can be useful for several reasons:

- The estimates are typically less noisy than the (raw) measurements.
- If a process sensor fails, and this failure is detected, the state estimator may be configures to continue providing a representative state estimate despite the lack of measurement-based update or correction of the estimate. This enhances the robustness of the MPC.
- A state estimator may be used to estimate disturbances and/or model parameters. This may increases the robustness of the MPC as the process model underlying the MPC becomes more accurate. Disturbances and model parameters can be estimated as augmented state variables modelled as constants, i.e. as state variables having time-derivatives equal to zero but with an additive random disturbance/noise. This augmentation is explained in detail on Page 743.

Example 35.1 *MPC of a simulated air heater*

This example is about MPC of a simulated air heater. The air heater and a mathematical model of it are presented in Appendix 39.5. The dynamics of the air heater is "time constant with time delay".

In this example, the MPC is implemented with Python with the minimize function of the Scipy package of Python with slsqp as solver. The MPC has the following features:

- Time step 0.5 s.
- Prediction horizon: 8.0 s.
- Number of blocks (groups) of the control signal (u): 3.

- Constraint on the temperature (state): $27 \leq T \leq 31$ °C, implemented in the constraints function which is an argument of slsqp(). In general terms, T is the process output variable, y.
- Constraints on the control signal u: $0 \leq u \leq 5$ V, defined as bounds on the optimization variable (u). These bounds are arguments of slsqp().
- Constraint on the rate of change of the control signal: $-0.25 \leq u' \leq 0.25$ V/s, implemented in the constraints function which is an argument of slsqp().
- Cost: $C_e = 1$, and $C_{u'} = 20$.
- An Extended Kalman Filter which estimates an input disturbance d, which is used by the MPC.
- Simulation start time = 0 s. Stop time = 300 s.

The simulation scenario is:

- The reference r is initially constant, then a step, then a ramp, then a sinusoid, and finally constant. During a simulation time interval, the reference is given a value *above* the constraint of T, to demonstrate the ability of the MPC to take the constraints of T into account.
- d is changed as a step during the simulation, to demonstrate the ability of the Kalman Filter to estimate a changing disturbance. The estimate, d_{est}, is used as a "soft measurement" of d by the MPC.

The Python program below implements the MPC.

http://techteach.no/control/python/mpc_airheater.py

In the program, the user can set the value of the variable cont_plot to select between continuous (online) plot (cont_plot = 1) and batch (offline) plot (cont_plot = 0).

Figure 35.3 shows the results with the MPC applied to the simulated air heater.

Comments to the results shown in Figure 35.3:

- *Diagram upper left*: The tracking of r is excellent where there are no change in d and no active constraints on T. The constraint on T is respected by the MPC, except a small disrespect after the step change of d, which is due to the fact that the MPC uses d_{est} and not the true value d when calculating the optimal u.
- *Diagram upper right*: The upper bound on u is respected by the MPC, see around $t = 60$ s. Note that the MPC starts increasing the control signal *before* the point of time of the reference change, see around $t = 100$ s. This demonstrates the predictive nature of the MPC.

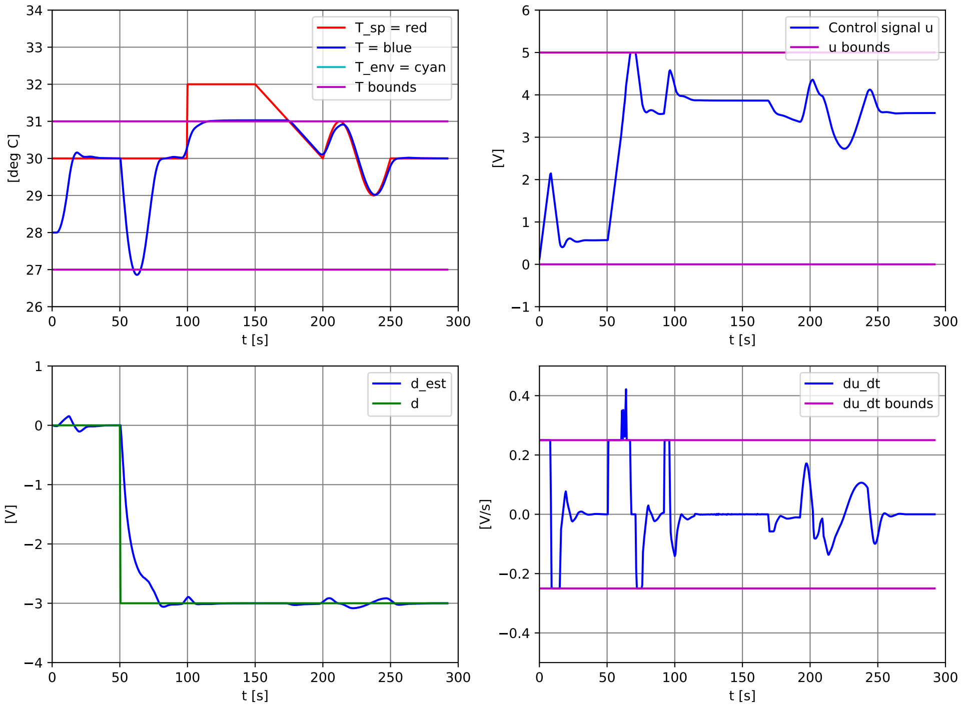


Figure 35.3: Example 35.1: Control of the simulated air heater temperature with MPC.

- *Diagram lower left*: The Kalman Filter estimates d correctly in steady state, but there is (of course) an estimation error in the transient phase.
- *Diagram lower right*: The constraints on u' are respected by the MPC, except around $t = 60$ s which may be due to the fact that the MPC uses d_{est} and not the true value d when calculating the optimal u.

[End of Example 35.1]

35.3 *Problems for Chapter 35*

Problem 35.1 *MPC for ship position control*

See Problem 34.1 which is about positional control of a ship. In that problem the position is controlled with LQ control. Now, assume MPC instead of LQ control. The state variables are $x_1 = y$ and $x_2 = y'$. The control variable is F_t.

1. Define the optimization criterion (or objective function) of the MPC.
2. Assume that you want to make the control system more sluggish so that the positional control error is allowed to become larger. (One possible motivation may be that tight position control is not required for a period, for example if the ship is parked, waiting for a certain operation to be initiated.) Which parameter of the criterion is proper for adjustment, and should you increase of decrease that parameter?
3. The thruster force F_t has of course a positive limit and a negative limit (assuming that the thruster can rotate both directions). Can the MPC algorithm take these limits into account when it calculates the optimal thruster force to be applied to the ship?

Problem 35.2 *Hierarchical control with MPC and PID*

MPC may be used for direct manipulation of the control variables. MPC can also be used in a hierarchic control system. Figure 35.4 shows using MPC in outer control loop while PID controllers are used in inner control loops. This control structure is similar to a conventional control structure frequently used in industry – which?

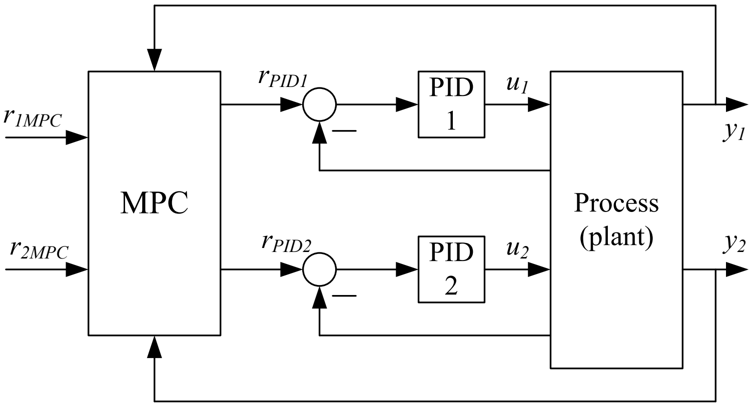


Figure 35.4: MPC in outer control loop while PID controllers are used in inner control loops.

Suggest concrete examples of $r_{1,\text{MPC}}$, $r_{\text{PID},1}$, and u_1 (select any application you want).

35.4 *Solutions to problems for Chapter 35*

Solution to Problem 35.1

1. The optimization criterion can be defined as

$$
\begin{aligned}
J = &\sum_{k=0}^{N_p} \left\{ Q_1 \left[e_{1,k}\right]^2 + Q_2 \left[e_{2,k}\right]^2 + ... + Q_n \left[e_{n,k}\right]^2 \right\} \\
&+ \sum_{k=1}^{N_c} R_1 \left\{ \left[{u_1}'_k\right]^2 + R_2 \left[{u_2}'_k\right]^2 + ... + R_r \left[{u_r}'_k\right]^2 \right\}
\end{aligned}
\tag{35.21}
$$

2. The state variables are $x_1 = y$ and $x_2 = y'$. Thererfore, the error e_1 is the position control error. To allow for a larger e_1 the cost or weight of e_1 – which is Q_1 – should be decreased.

3. Yes, MPC takes into account specific limits of control variable(s) when calculating the optimal control signal.

Solution to Problem 35.2

Cascade control!.

$r_{1,\text{MPC}}$ may be level of a tank. $r_{\text{PID},1}$ may be flow reference to a flow PI(D) controller. u_1 may be control signal to a control valve.

Chapter 36

Inverse dynamics control

36.1 Introduction

Inverse dynamics control is a model-based control method which can be applied to nonlinear multivariable prosesses. [1] It is assumed that the model is a (nonlinear) state space model. The value of each of the state variables and process disturbances must be available at any instance of time – either from measurements or from an estimator (as a Kalman Filter).

Inverse dynamics control has some alternative names, namely *feedback linearization*, *nonlinear decoupling*, and *computed torque control*. The latter is used when the control method is used for position control of robotic systems, cf. Section 36.3.

Note: The version of the inverse dynamics control presented in this chapter assumes that the process has no time delay.

The control function consists of three main parts, cf. Figure 36.1.

- *A decoupler and linearizer* which is based on the process model and the instantaneous values of the states and the disturbances.
- *A multiloop PID controller* which is designed for the decoupled linear process.
- A *feedforward controller* which is designed for the decoupled linear process.

The following sections present inverse dynamics control for two different kinds of prosesses:

- First order processes, Section 36.2.
- Second order processes, Section 36.3.

[1] The method can, of course, be applied also to linear processes and monovariable processes.

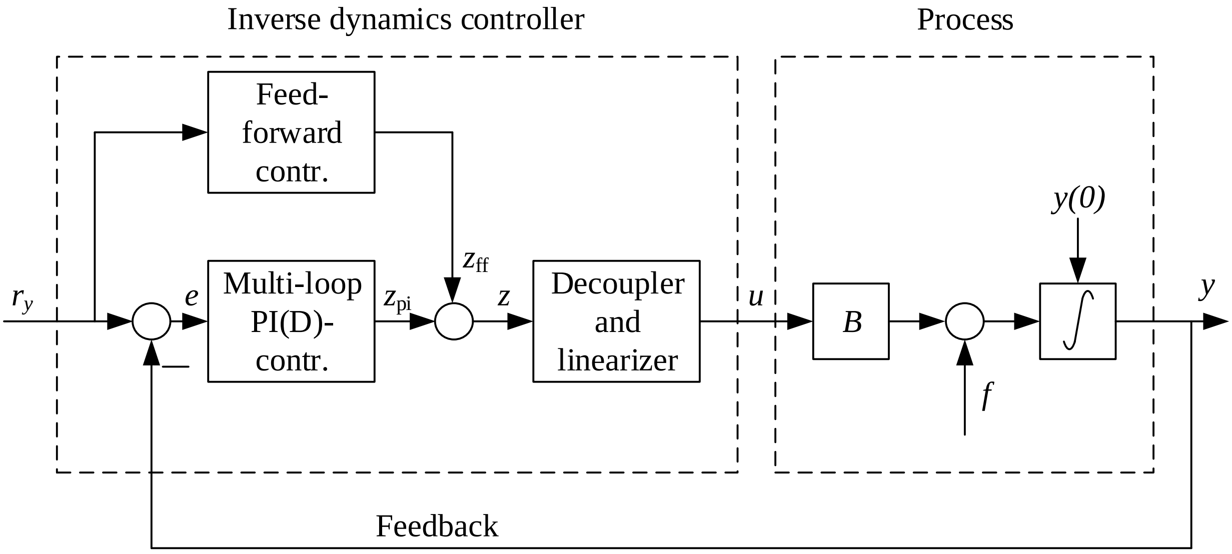


Figure 36.1: An overall block diagram of the inverse dynamics controller.

36.2 Inverse dynamics control of first order processes

36.2.1 The process model

It is assumed that the process model is a state space model on the following form:

$$y' = f(y,d) + B(y,d) \cdot u \tag{36.1}$$

or, simpler:

$$y' = f + Bu \tag{36.2}$$

x is the state vector, d is the disturbance vector, and u is the control vector. f is a vector of scalar functions, and B is a matrix of scalar functions. Note that the control vector u is assumed to appear *linearly* in the model.

Figure 36.2 shows a block diagram representation of the process model (36.2).

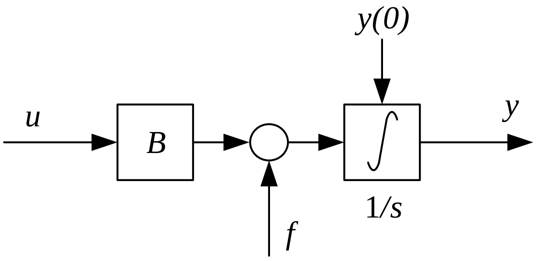


Figure 36.2: A block diagram representation of the process model (36.2).

The reference of y is denoted r_y.

We can derive the control function "graphically" as shown in Figure 36.3. The decoupler

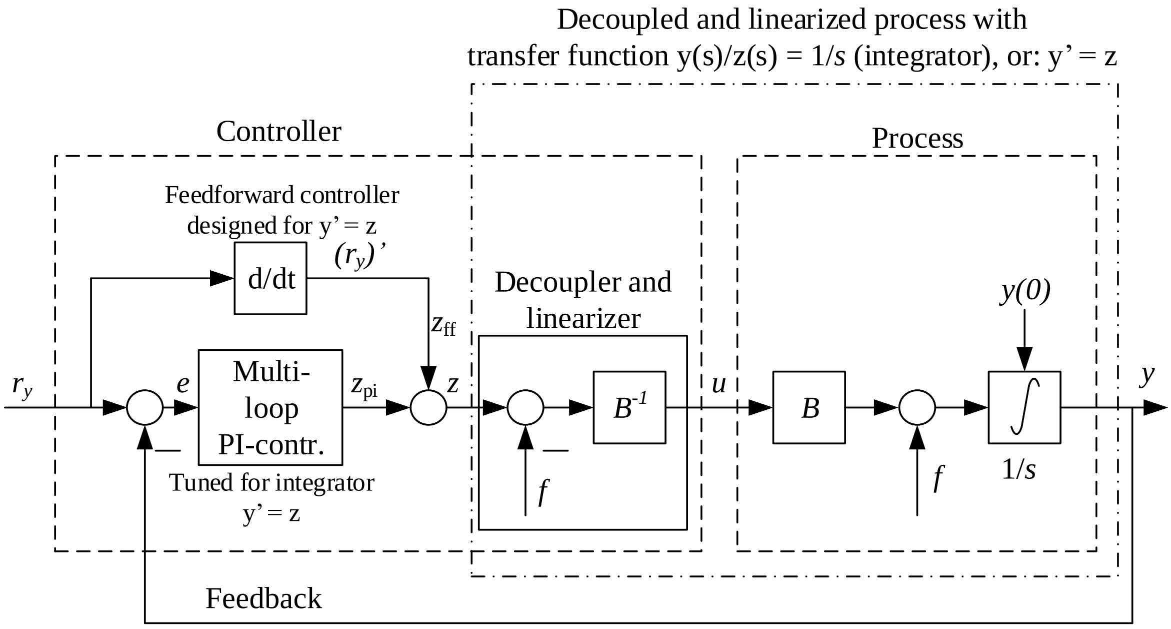


Figure 36.3: Block diagram of the control system based on feedback linearization.

and linearizer creates the following model relating the transformed control signal z and the process output y:

$$y' = z \tag{36.3}$$

which are actually n decoupled or independent *integrators* (n is the number of state variables), because $y(t) = \int_0^t z\, d\tau$. The transfer function from z to y is

$$\frac{Y(s)}{Z(s)} = \frac{1}{s} \tag{36.4}$$

We can denote the integrator (36.3) as the *transformed process.*

We will now derive the control function for this transformed process, and thereafter derive the final control function. How can you control an integrator? With PI feedback and feedforward!

36.2.2 PI tuning

The (multiloop) feedback PI controller is

$$z_{\text{fb}} = K_c e + \frac{K_c}{T_i} \int_0^t e d\tau \tag{36.5}$$

where e is the control error:

$$e \overset{\text{def}}{=} r_y - y \tag{36.6}$$

In (36.5), the coefficients are are actually matrices:

$$K_c = \begin{bmatrix} K_{c_1} & 0 & \cdots & 0 \\ 0 & K_{c_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_{c_n} \end{bmatrix} \tag{36.7}$$

$$\frac{K_c}{T_i} = \begin{bmatrix} \frac{K_{c_1}}{T_{i_1}} & 0 & \cdots & 0 \\ 0 & \frac{K_{c_2}}{T_{i_2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{K_{c_n}}{T_{i_n}} \end{bmatrix} \tag{36.8}$$

We can use the SIMC method to tune K_{c_j} and T_{i_j}. The process to tune for is an integrator process with integrator gain $K_i = 1$ and zero time delay, cf. Ch. 14.8.2.2, giving the following PI settings:

$$K_{c_j} = \frac{1}{t_{cc_j}} \tag{36.9}$$

$$T_{i_j} = 2t_{cc_j} \tag{36.10}$$

where t_{cc_j} is the specified time constant of feedback loop no. j.

36.2.3 Feedforward controller

In addition to the PI feedback action the controller should contain *feedforward* from the reference r_y to get fast reference tracking when needed (assuming the reference is varying). The feedforward control function can be derived by substituting the process output y in the process model (36.3) by r_y and then solving for y, giving:

$$z_{\text{ff}} = {r_y}' \tag{36.11}$$

If the reference is varied abruptly (but you may vary it smoothly), a lowpass filtered can be used before the time derivative is calculated.

36.2.4 The resulting control signal

From Figure 36.3 we see that the controller function for the transformed process (36.3) is the sum of the feedback control function and the feedforward control function derived above:

$$z = z_{\text{pi}} + z_{\text{ff}} \tag{36.12}$$

$$= \underbrace{K_c e + K_i \int_0^t e d\tau}_{z_{\text{pi}}} + \underbrace{{r_y}'}_{z_{\text{ff}}} \tag{36.13}$$

Also from Figure 36.3, we see that the resulting controller function, that is, the formula for the control vector u, is:

$$u = B^{-1}(z - f) = B^{-1}(z_{\text{pi}} + z_{\text{ff}} - f) \tag{36.14}$$

z_{pi} and z_{ff} are shown in (36.13).

36.2.5 About the resulting control system

Here are some characteristics of the control system:

- The controller is *model based* since it contains f and B from the process model.
- Since the process disturbance is an argument of f and/or B the controller implements *feedforward* from the disturbance. (It also implements feedforward from the reference, due to the term $(r_y)'$ in the controller.)
- The control system consists of n *decoupled single-loop* control systems. This is illustrated in Figure 36.4.

Example 36.1 *Inverse dynamics control applied to level control*

In this example, inverse dynamics control will be applied to a level control system. Figure 36.5 shows the control system.

It assumed that the outflow is proportional to the control signal u and to the square root of the pressure drop along the control valve. The process model based on mass balance is (ρ is density)

$$\rho A h' = \rho q_{in} - \rho K_v u \sqrt{dP} \tag{36.15}$$

or

$$h' = \underbrace{\frac{q_{in}}{A}}_{f} + \underbrace{\left(-\frac{K_v\sqrt{dP}}{A}\right)u}_{B} \tag{36.16}$$

The control function becomes, cf. (36.14),

$$\begin{aligned}
u &= B^{-1}\left[K_c e + K_i \int_0^t e d\tau + r_y{}' - f\right] && (36.17)\\
&= \left(-\frac{K_v\sqrt{dP}}{A}\right)^{-1}\left[K_c e + K_i \int_0^t e d\tau + r_y{}' - \frac{q_{in}}{A}\right] && (36.18)\\
&= -\frac{A}{K_v\sqrt{dP}}\left[K_c e + K_i \int_0^t e d\tau + r_y{}' - \frac{q_{in}}{A}\right] && (36.19)
\end{aligned}$$

This control function requires that the differential pressure dP the inflow q_{in} are measured.

[End of Example 36.1]

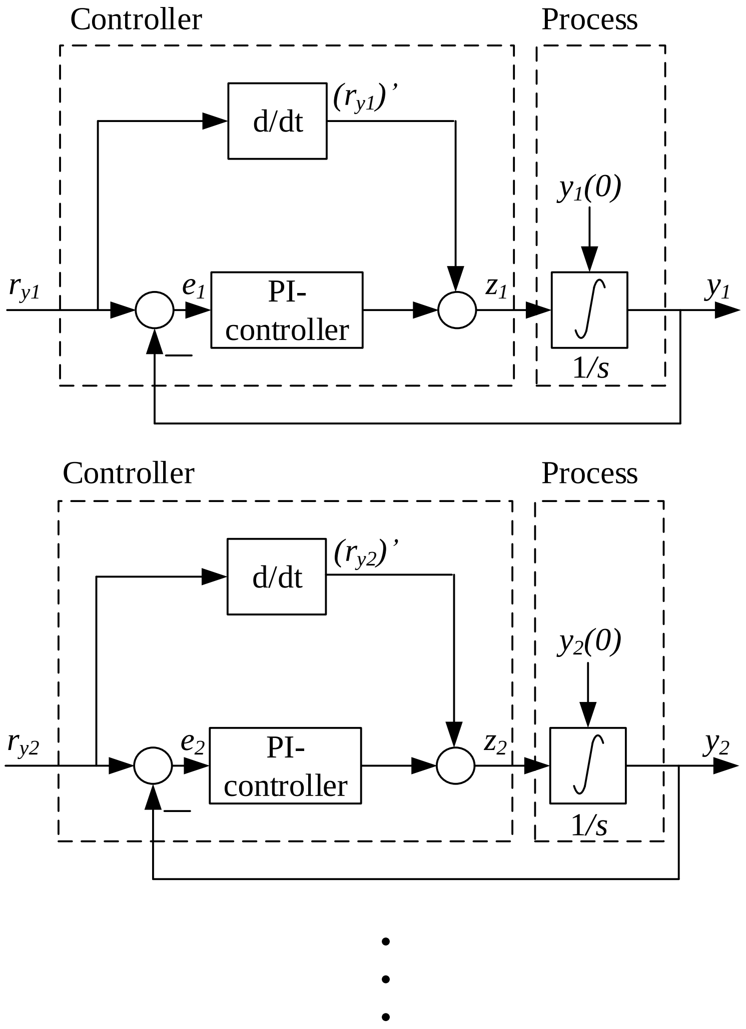


Figure 36.4: The inverse dynamics control system consists of n decoupled single-loop control systems.

36.3 Inverse dynamics control of second order prosesses

36.3.1 The process model

Mechanical processes with position as the process output variable (to be controlled) typically have a model on the form of a set of second order differential equations (due to Newton's Second Law):

$$y'' = f(y, y', d) + B(y, y', d) \cdot u \tag{36.20}$$

or simply:

$$y'' = f + Bu \tag{36.21}$$

where y is the position, y' is the speed, d is a disturbance (e.g. a load force or torque), and u is the control variable.

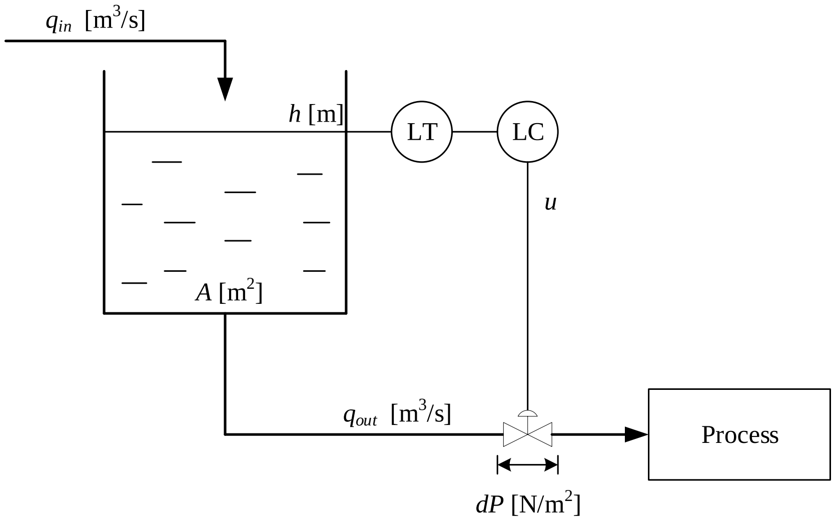


Figure 36.5: Level control system.

We assume that r_y is the reference of y (position).

We can derive the control function “graphically” as shown in Figure 36.6. The decoupler

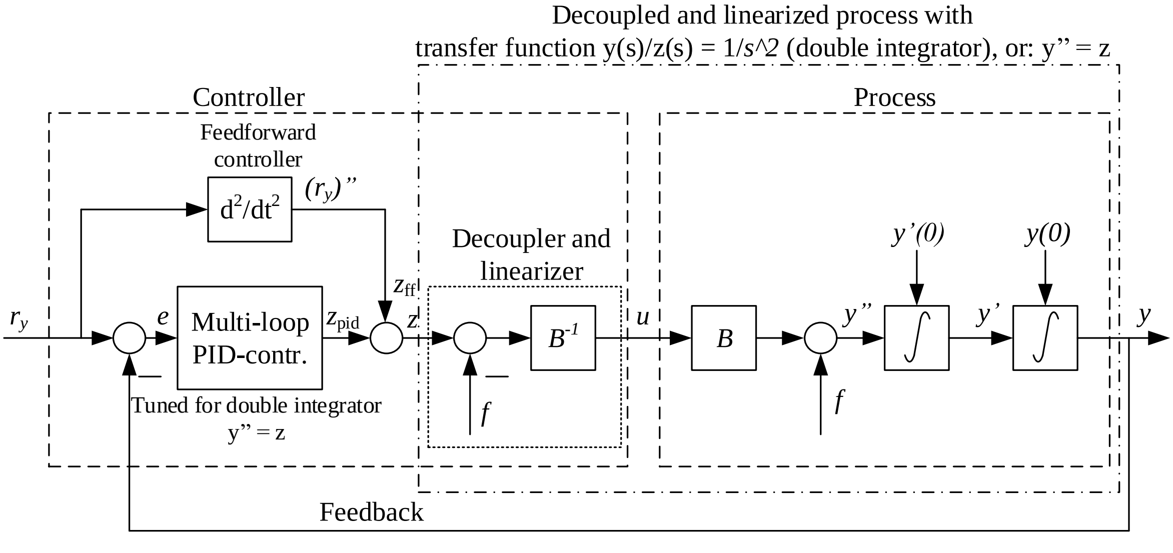


Figure 36.6: Block diagram of the control system based on feedback linearization.

and linearizer creates the following model relating the transformed control signal z and the process output y:

$$y'' = z \tag{36.22}$$

which are n decoupled (independent) *double integrators*. The transfer function from z to y is

$$\frac{Y(s)}{Z(s)} = \frac{1}{s^2} \tag{36.23}$$

These double integrators can be controlled with feedback with PID controllers plus feedforward, as described in the following.

36.3.2 PID tuning

The PID double integrators (36.23) can be controlled with feedback with PID controller:

$$z_{\text{pid}} = K_c e + \frac{K_c}{T_i} \int_0^t e d\tau + K_c T_d e' \tag{36.24}$$

where e is the control error:

$$e \overset{\text{def}}{=} r_y - y \tag{36.25}$$

In (36.24), K_c, K_c/T_i and $K_c T_d$ are actually diagonal matrices:

$$K_c = \begin{bmatrix} K_{c_1} & 0 & \cdots & 0 \\ 0 & K_{c_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_{c_n} \end{bmatrix} \tag{36.26}$$

$$\frac{K_c}{T_i} = \begin{bmatrix} \frac{K_{c_1}}{T_{i_1}} & 0 & \cdots & 0 \\ 0 & \frac{K_{c_2}}{T_{i_2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{K_{c_n}}{T_{i_n}} \end{bmatrix} \tag{36.27}$$

$$K_c T_d = \begin{bmatrix} K_{c_1} T_{d_1} & 0 & \cdots & 0 \\ 0 & K_{c_2} T_{d_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_{c_n} T_{d_n} \end{bmatrix} \tag{36.28}$$

K_{c_j}, T_{i_j} and T_{d_j} can be calculated with the SIMC method for a double integrator process having double integrator gain $K_{ii} = 1$, cf. Ch. 14.8.5. Assuming that the PID controller has the parallel form, the PID settings are (14.108)-(14.110), which are repeated here:

$$K_{c_j} = \frac{2}{\left(T_{C_j}\right)^2} \tag{36.29}$$

$$T_{i_j} = 4t_{cc_j} \tag{36.30}$$

$$T_{d_j} = t_{cc_j} \tag{36.31}$$

where T_{C_j} is the specified time constant of feedback loop no. j.

36.3.3 Feedforward controller

In addition to the PID feedback control, the controller should contain *feedforward* from the reference y_r to get fast reference tracking (assuming the reference is varying). The feedforward controller can be derived by substituting the process output y in the process model (36.3) by y_r and then solving for y, giving:

$$z_{\text{ff}} = r_y{}'' \tag{36.32}$$

36.3.4 The resulting control signal

Now, we have the following control function for the process (36.22) consisting of the sum of the feedback control function and the feedforward control function:

$$z = z_{\mathrm{pid}} + z_{\mathrm{ff}} \tag{36.33}$$

$$= \underbrace{K_c e + \frac{K_c}{T_i}\int_0^t e d\tau + K_c T_d \, {e_f}'}_{z_{\mathrm{pid}}} + \underbrace{{r_y}''}_{z_{\mathrm{ff}}} \tag{36.34}$$

Also from Figure 36.6, we see that the resulting controller function, that is, the formula for the control vector u, is:

$$u = B^{-1}(z - f) = B^{-1}(z_{\mathrm{pid}} + z_{\mathrm{ff}} - f) \tag{36.35}$$

z_{pid} and z_{ff} are as in (36.34).

36.3.5 About the resulting control system

The control system consists of n *decoupled single-loop* control systems. This is illustrated in Figure 36.7.

Example 36.2 *Inverse dynamics control applied to motion control*

Given the following mathematical model of a body:

$$my'' = F_c + F_d \tag{36.36}$$

where y [m] is position, $m = 10$ kg is mass, F_c [F] is the force demanded by the controller, and F_d [N] is net disturbance force (sum of damping, friction, gravitation, etc).

Assume that the position reference of y is r_y, and that the specified time constant of the control system is 1 s. Also, assume that the PID controller is a parallel PID controller.

To derive the control function we first write the process model on the standard form (36.21):

$$y'' = \underbrace{\frac{1}{m}F_d}_{f} + \underbrace{\frac{1}{m}}_{B}\underbrace{F_c}_{u} \tag{36.37}$$

The control function becomes

$$u = B^{-1}\left[z_{\mathrm{pid}} + z_{\mathrm{ff}} - f\right] \tag{36.38}$$

$$= \left(\frac{1}{m}\right)^{-1}\left[z_{\mathrm{pid}} + z_{\mathrm{ff}} - \left(\frac{1}{m}F_d\right)\right] \tag{36.39}$$

where z_{pid} is given by (36.24), and z_{ff} is given by (36.32).

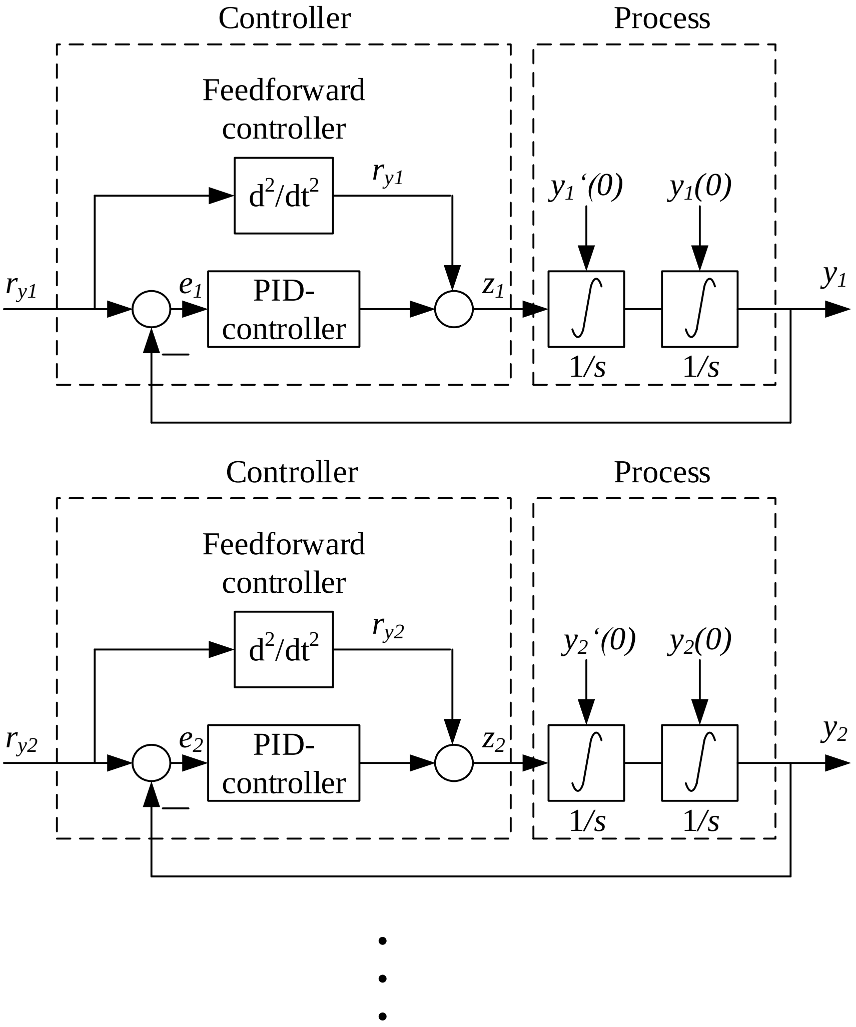


Figure 36.7: The control system consists of n decoupled single-loop control systems.

Let us, as an example, specify the closed-loop time constant t_{cc} as:

$$t_{cc} = 1 \text{ s} \tag{36.40}$$

The PID settings become:

$$K_c = \frac{2}{(t_{cc})^2} = 2 \tag{36.41}$$

$$T_i = 4t_{cc} = 4 \text{ s} \tag{36.42}$$

$$T_d = t_{cc} = 1 \text{ s} \tag{36.43}$$

[End of Example 36.2]

36.3.6 Computed torque control

In robotics, the so-called computed torque controller is well-known. This controller is actually the same as the inverse dynamics controller derived in the present section. I will now derive the computed torque controller from the inverse dynamics controller (36.35).

The computed torque controller assumes a process model of the mechanical system (robot) on the following form:

$$M(q)q'' + C(q,q') + F(q') + G(q) + L = Q \tag{36.44}$$

or simply:

$$Mq'' + C + F + G + L = Q$$

where q is generalized position, i.e. a set of translational position and/or rotational positions of the robot arms, q' is generalized speed, M is the inertia matrix, C is the coriolis and centripetal force matrix, F is the friction force matrix, G is the gravity loading matrix, and L is the generalized load (force and/or torque) matrix, and Q is the generalized applied force (“translation“ force and/or “rotational” force (torque)), assumed being the control (manipulating) variable. Often, $C(q,q')$ is written as $C_1(q,q') \cdot q'$ where C_1 is a matrix, but here I use the simpler form $C(q,q')$.

(36.44) written on the form (36.21) is:

$$q'' = M^{-1}(-C - F - G - L) + M^{-1}Q \tag{36.45}$$

which is on the form

$$y'' = f + Bu \tag{36.46}$$

with

$$y = q \tag{36.47}$$

$$u = Q \tag{36.48}$$

$$f = M^{-1}(-C - F - G - L) \tag{36.49}$$

$$B = M^{-1}$$

The inverse dynamics controller given by (36.35) becomes

$$\begin{aligned} u &= B^{-1}(z_{\text{pid}} + z_{\text{ff}} - f) && (36.50)\\ &= M\left[z_{\text{pid}} + {r_q}'' - M^{-1}(-C - F - G - L)\right] && (36.51)\\ &= M\left(z_{\text{pid}} + {r_q}''\right) + (C + F + G + L) && (36.52) \end{aligned}$$

where z_{pid} is the PID controller (36.34). r_q is the reference or reference of q. (36.52) is the computed torque controller, derived as an inverse dynamics controller.

In some literature, a PD controller is used instead of a PID controller in (36.52). With a PD controller (i.e. lacking integral action), non-modelled forces may give non-zero steady state control errors.

36.4 *Problems for Chapter 36*

Problem 36.1 *Temperature and level control of a tank*

Figure 36.8 shows a tank where continuous flows of cold liquid and hot liquid are mixed in a tank.[2] The liquid in the tank is assumed being homogeneous.

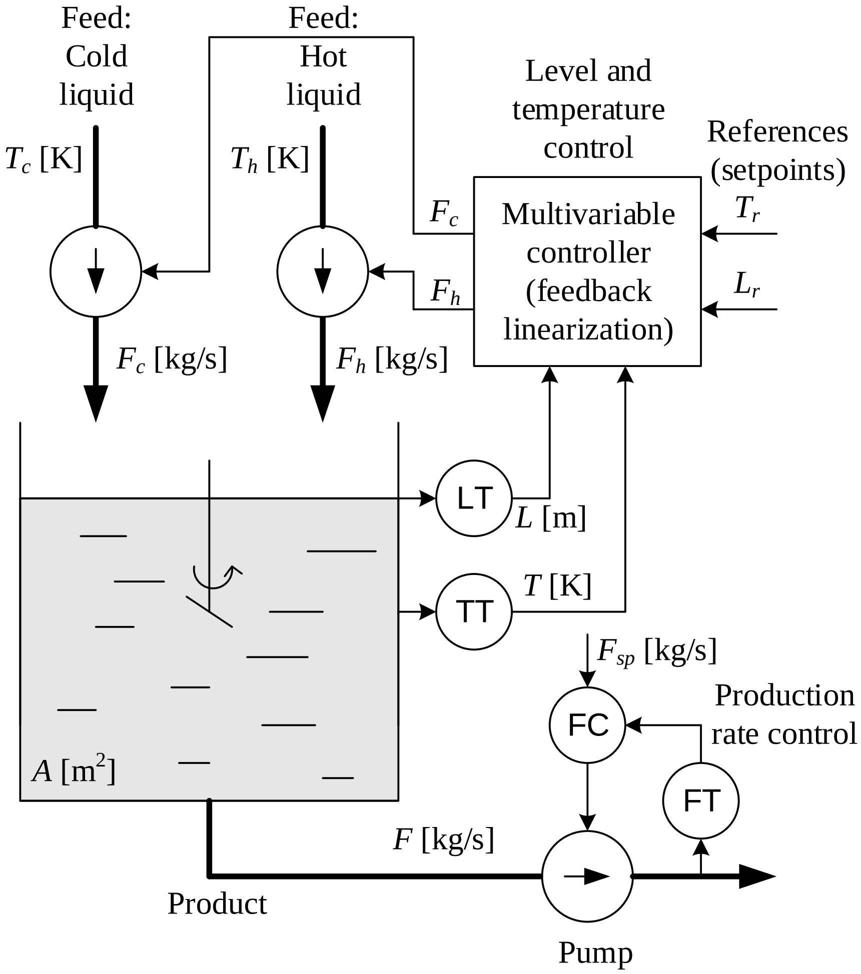


Figure 36.8: A tank.

The product flow rate out of the tank is controlled with a ordinary flow control loop.

The level and the temperature of the tank shall be controlled to follow (track) their reference values. The cold liquid and the hot liquid flows can be manipulated, so they are control variables. (To make the actual flows become equal to the demanded flow as calculated by the multivariable controller, local flow control loops around the pumps may be needed, but these control loops are not shown in the figure.)

It is not obvious how to control the cold flow and the hot flow to obtain the reference level

[2] This example is not very realistic, but it is assumed to be relatively simple and easy to understand.

and temperature because both flows affect both the level and the temperature. The control problem may be solved with a "traditional" control structure where e.g. the level controller adjusts the cold flow, and the temperature controller adjusts the ratio between hot and cold flow. But, instead of such a traditional control structure, you will design a model-based multivariable controller based on inverse dynamics control.

The mathematical model of the process is as follows. The model is based on the following assumptions:

- The density ρ and the specific heat capacity c are the same in all flows and in the tank.
- The temperature is homogeneous in the liquid in the tank.
- There is no heat transfer through the walls of the tank.
- Energy dependent on pressure and kinetics is disregarded.

Mass balance of the mixed liquid of the tank is

$$\rho A h' = F_c + F_h - F \tag{36.53}$$

An energy balance of the liquid in the tank is (it is assumed that both the level and the temperature can vary):

$$c\rho A\,(LT)' = cF_cT_c + cF_hT_h - cFT \tag{36.54}$$

In (36.54),

$$c\rho A\,(LT)' = c\rho A\left(L'\right)T + c\rho AL\left(T'\right) \tag{36.55}$$

With (36.55) inserted into (36.54) and then cancelling c, (36.54) becomes

$$\rho ALT' = F_c\,(T_c - T) + F_h\,(T_h - T) \tag{36.56}$$

The process model is now (36.53) and (36.56).

1. Write the process model (36.53) and (36.56) on the standard form to be used in an inverse dynamics controller where F_c and F_h are control variables, and L and T are state variables to be controlled (to track their respective references).
2. Design the inverse dynamics controller, but you do not have to give the formulas for tuning the gains and the integral times of the internal PI controllers, as this is the topic of the following task. The references are r_L and r_T.
3. Calculate the gains, K_{c_L} and K_L, and the integral times, T_{i_L} and T_{i_T}, of the internal PI controllers. It is specified that the control loops of the decoupled processes (which are just integrators) shall have response times (time constants) t_{cc_L} and t_{cc_T}, respectively.
4. Which parameters and variables must be known at any instant of time to make the control function implementable?

5. Assume that there is a change in the level reference. Will this change cause any change in the temperature?

   Will a change in the temperature reference cause any change in the level?

6. Assume for example that any change in the temperature of the cold inflow, T_c, is regarded as a disturbance to the control system. Does the control function implement *feedforward* from this disturbance?

Problem 36.2 *Position control of a ship*

Figure 36.9 shows a ship.

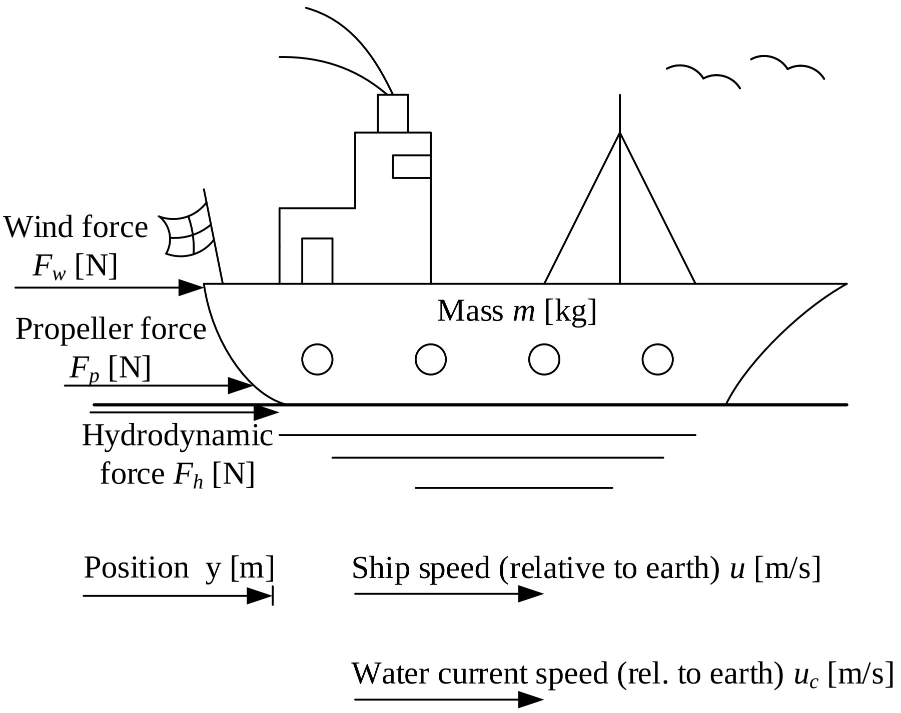


Figure 36.9: Ship.

In this problem we concentrate on the so-called surge (forward-backward) direction, i.e., the motions in the other directions are disregarded. The wind acts on the ship with the force F_w which is a function of the wind attack angle ϕ and the wind speed V_w. This function is assumed to be known for a given ship. The hydrodynamic damping force F_h (damping from the water) is proportional to the square of the difference between the ship speed, y', and the water current speed u_c. The proportionality constant is D.

Applying Newtons's Law of Motion we obtain the following mathematical model of the surge motion:

$$my'' = \underbrace{-D\left|y' - u_c\right|\left(y' - u_c\right)}_{F_h} + F_w(\phi, V_w) + F_t \tag{36.57}$$

(Model parameter values are given in Appendix 39.9, but these values are not needed in the present exercise.)

1. Design the position control function as an inverse dynamics controller. The position reference is r [m]. The closed loop time constant is specified as t_{cc}. Express the PID parameters as a function of t_{cc}.

2. Which variables and parameters must have known values (from measurements or estimators) to make the controller function implementable.

3. Let's define the wind force F_w as a disturbance. Explain how the inverse dynamics controller implements feedforward from this disturbance. Assuming that the ship model is correct and that F_w is perfectly known, what is then the impact that F_w will have on the ship position y?

36.5 *Solutions to problems for Chapter 36*

Solution to Problem 36.1

1. The standard process model form is

$$x' = f + Bu \tag{36.58}$$

We have

$$x = \begin{bmatrix} L \\ T \end{bmatrix} \tag{36.59}$$

and

$$u = \begin{bmatrix} F_c \\ F_h \end{bmatrix} \tag{36.60}$$

We get

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} \frac{1}{\rho A} & \frac{1}{\rho A} \\ \\ \frac{T_c - T}{\rho AL} & \frac{T_h - T}{\rho AL} \end{bmatrix} \tag{36.61}$$

and

$$f = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} -\frac{F}{\rho A} \\ 0 \end{bmatrix} \tag{36.62}$$

2. The control function is generally

$$u = B^{-1} \left[K_p e + K_i \int_0^t e d\tau + r' - f \right] \tag{36.63}$$

where (I use indices L and T instead of 1 and 2 as in the standard form)

$$u = \begin{bmatrix} F_c \\ F_h \end{bmatrix} \tag{36.64}$$

$$B = \begin{bmatrix} \frac{1}{\rho A} & \frac{1}{\rho A} \\ \frac{T_c - T}{\rho AL} & \frac{T_h - T}{\rho AL} \end{bmatrix} \tag{36.65}$$

$$e = \begin{bmatrix} L_r - L \\ T_r - T \end{bmatrix} \tag{36.66}$$

$$K_p = \begin{bmatrix} K_{p_L} & 0 \\ 0 & K_{p_T} \end{bmatrix} \tag{36.67}$$

$$K_i = \begin{bmatrix} K_{i_L} & 0 \\ 0 & K_{i_T} \end{bmatrix} = \begin{bmatrix} \frac{K_{p_L}}{T_{i_L}} & 0 \\ 0 & \frac{K_{p_T}}{T_{i_T}} \end{bmatrix} \tag{36.68}$$

$$r' = \begin{bmatrix} {r_L}' \\ {r_T}' \end{bmatrix} \tag{36.69}$$

$$f = \begin{bmatrix} -\frac{F}{\rho A} \\ 0 \end{bmatrix} \tag{36.70}$$

3.
$$K_{c_L} = \frac{1}{t_{cc_L}} \tag{36.71}$$
$$T_{i_L} = 2t_{cc_L} \tag{36.72}$$
$$K_{c_T} = \frac{1}{t_{cc_T}} \tag{36.73}$$
$$T_{i_T} = 2t_{cc_T} \tag{36.74}$$

4. The following parameters and variables must be known: ρ, A, L (using a level sensor), T (temperature sensor),T_c (temperature sensor), T_h (temperature sensor), F (flow sensor).

5. No.

6. Yes.

Solution to Problem 36.2

1. To derive the control function we first write the process model on the standard form:

$$y'' = \underbrace{\frac{1}{m}\left[-D\left|y'-u_c\right|\left(y'-u_c\right)+F_w(\phi,V_w)\right]}_{f} + \underbrace{\frac{1}{m}}_{B}\underbrace{F_t}_{u} \tag{36.75}$$

The control function becomes

$$\underline{\underline{F_t}} = B^{-1}\left[K_p e + \frac{K_p}{T_i}\int_0^t e d\tau + K_p T_d \frac{de_f}{dt} + r'' - f\right] \tag{36.76}$$

$$= \underline{\underline{\left(\frac{1}{m}\right)^{-1}\left[\begin{array}{c} K_p e + \frac{K_p}{T_i}\int_0^t e d\tau + K_p T_d \frac{de_f}{dt} + r'' \\ \\ +\frac{1}{m}\left[-D\left|y'-u_c\right|\left(y'-u_c\right)+F_w(\phi,V_w)\right] \end{array}\right]}} \tag{36.77}$$

where e is the control error:

$$e = r - y \tag{36.78}$$

The SIMC PID settings become:

$$K_c = \frac{2}{{t_{cc}}^2} \tag{36.79}$$
$$T_i = 4t_{cc} \tag{36.80}$$
$$T_d = t_{cc} \tag{36.81}$$

2. Of course, all variables and parameters of the control function (36.77) must be known to make the controller implementable. In more details:

   - The parameters m and D must be known.
   - The ship position y must be measured usinig e.g. GPS measurement.

- The ship speed y' can be calculated as the time-derivative of y, or it can be estimated with a Kalman Filter or an observer.
- The water speed u_c must either be measured or estimated with a Kalman Filter or an observer.[3]
- The wind angle ϕ and the wind speed V_w must be measured with a wind sensor (mounted on the ship). The wind force function F_w is assumed to be known for a given ship.[4]

3. In (36.77) the control variable F_t is a function of the disturbance $F_w(\phi, V_w)$. This dependency implements feedforward. The controller will compensate for the disturbance, so that the ship position y will not be influenced by F_w, whatever value of F_w.

[3] Kongsberg Maritime (Norway) use a Kalman Filter in their ship positioning systems – or DP (Dynamic Positioning) systems.

[4] F_w is calculated from the geometry of the ship.

Part IX

APPENDICES

Chapter 37

Style of symbols

Symbol t with an appropriate sub-index is used for all parameters relating to time, e.g. t_f for filter time constant. But there are two exceptions: T_i for the integral time and T_d for the derivative time of a PID controller as it is very common to use those upper-case symbols for those parameters.

Laplace transformed variables are represented with upper-case symbols, for example, $F(s)$ as the Laplace transform of the time function $f(t)$. This rule will face trouble in cases where the time function is symbolized with a upper-case letter. In such cases, a star will be used. For example, the Laplace transform of $X(t)$ will be symbolized with $X^*(s)$.

Symbol $H(s)$ with an appropriate sub-index is used for Laplace transform based transfer functions, and $H(z)$ for z-transform based transfer functions.

Chapter 38

Abbreviations

ADC = Analog Digital Converter

DAC = Digital Analog Converter

DC = Direct Current

GM = Gain Margin

GS = Gain Scheduling

IAE = Integral of Absolute value of control Error

MA = Moving Average

MPC = Model-Predictive Control

PID = Proportional Integral Derivative

PLC = Programmable Logical Controller

PM = Phase Margin

SFC = Sequential Function Chart

SIMC = Simple Internal Model Control

ZN = Ziegler-Nichols

WRRF = Water Resource and Recovery Facility

Chapter 39

Some process models

39.1 Introduction

In the following sections, mathematical models of various physical processes are presented. Some of these models are used in examples and in problems in the book. They may be used in additional problems, and as a basis of dynamic simulators.

The models are in the form of first order differential equations (except for the ship model in Section 39.9 which is a second order differential equation), like

$$x'(t) = f(\cdot) \tag{39.1}$$

where the dot in $f(\cdot)$ represents any variables and parameters. (39.1) is a general state space model. However, as explained in Ch. 4, the differential equation tells only "half the story" about $x(t)$, the state variable. When you are implement a simulator for simulating $x(t)$, you must implement also the following integral:

$$x(t) = x(0) + \int_0^t x'(\theta)\, d\theta \tag{39.2}$$

But, I do not show the integrals in the models presented in this appendix as is a tradition to not present the integrals together with the differential equation. If you have an x', I assume that by reading this book you now what to do with it.

39.2 Wood chips tank

39.2.1 System description

Figure 39.1 shows a wood chips tank with a feed screw with continuous feed of wood chips, conveyor belt, which runs with a fixed speed. There is a continuous outflow of wood chips.[1] The conveyor belt makes up a time delay or transport delay from the screw to the tank.

[1] Typically, there is such a wood chips tank in the beginning of the production line of a paper and pulp factory.

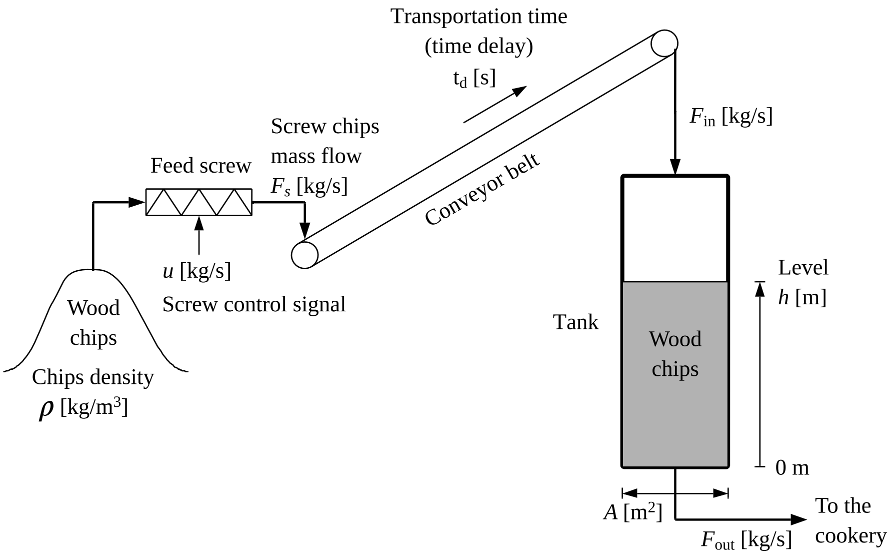


Figure 39.1: Wood chips tank.

39.2.2 Variables and parameters

Variables and parameters of the wood chip tank are defined in Table 39.1.[2] We assume that there is an overflow if the level exceeds the maximum level.

39.2.3 Overall block diagram

Figure 39.2 shows a block diagram of the wood chips tank.

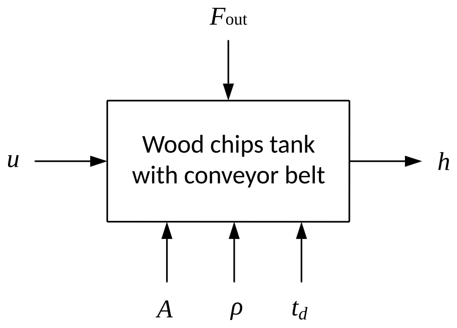


Figure 39.2: Block diagram of the wood chips tank.

[2] Courtesy of earlier Sødra Cell, and even earlier Norske Skog, Tofte, Norway.

Table 39.1: Wood chips tank: Variables and parameters.

Symbol	Value (default)	Unit	Description
h	10	m	Wood chips level
-	[0, 15]	m	Range of level
u	25	kg/s	Control signal to feed screw
F_s	25	kg/s	Feed screw flow (flow into conveyor belt)
F_{in}	25	kg/s	Wood chips flow into tank (from belt)
F_{out}	25	kg/s	Wood chips outflow from tank
ρ	145	kg/m^3	Wood chips density
A	13.4	m^2	Tank cross sectional area
t_d	250 s	s	Transportation time (time delay) on conveyor belt

39.2.4 Mathematical model

The chips flow through the feed screw, F_s, is the flow rate into the conveyor belt. F_s is assumed to have the same value as the control signal (the demanded flow rate), u:

$$F_s(t) = u(t) \tag{39.3}$$

The outflow from the conveyor belt, which is also the inflow to the tank, is the same as the inflow to the belt and the screw flow, but time delayed due to the transportation on the belt:

$$F_{\text{in}}(t) = F_s(t - t_d) = u(t - t_d)$$

The flow rate out of the tank is F_{out}.

A mathematical model of the tank based on material balance of the wood chips in the tank is:

$$\rho A h'(t) \quad = \quad F_{\text{in}}(t) - F_{\text{out}}(t) = u(t - t_d) - F_{\text{out}}(t) \tag{39.4}$$

39.3 Buffer tank

39.3.1 System description

Figure 39.3 shows a water tank. The tank may represent a buffer tank, or an equalization magazine, at the inlet of a plant, e.g. a water resource recovery facility (WWRF). The geometrical design of such a magazine may not have straight walls as in Figure 39.3, see Figure 16.15. However, if relatively small variations in the level are assumed, a tank with straight walls approximates the magazine with non-straight walls. It is assumed that the outlet flowrate, F_{out}, can be manipulated by typically a level controller, while the inlet flowrate, F_{in}, is a disturbance or load which is not manipulated.

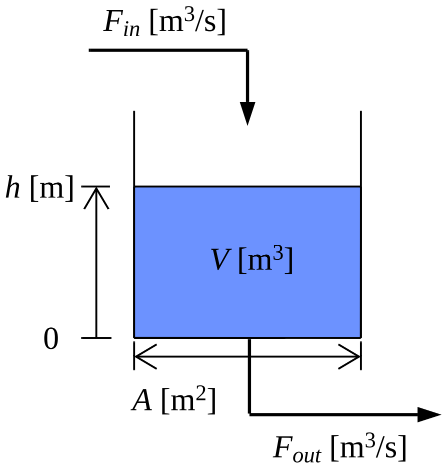


Figure 39.3: Buffer tank.

39.3.2 Variables and parameters

Variables and parameters of the buffer tank are defined in Table 39.2. The numerical values may represent a water resource and recovery facility.

Table 39.2: Buffer tank: Variables and parameters

Symbol	**Value (default)**	**Unit**	**Description**
h	2.0	m	Water level
F_{in}	3.0	m^3/s	Inlet flowrate
F_{out}	3.0	m^3/s	Outlet flowrate
A	2000	m^2	Inner cross sectional area of tank
V	4000	m^3	Volume of water in tank

39.3.3 Overall block diagram

Figure 39.4 shows an overall block diagram of the buffer tank.

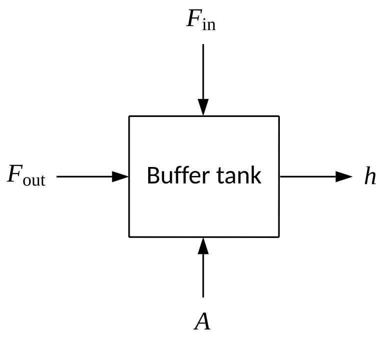


Figure 39.4: Block diagram of the buffer tank.

39.3.4 Mathematical model

A mathematical model expressing the level variations can be derived from the mass balance of the liquid in the tank. The model is:

$$Ah' = F_{\text{in}} - F_{\text{out}} \tag{39.5}$$

39.4 Heated liquid tank

39.4.1 System description

Figure 39.5 shows a heated tank. Liquid (assumed water) flows into and out of the tank. The volume of liquid is constant (this can be realized with overflow or level of regulation). The inflow and outflow are thus equal. There is a heat transfer between the liquid and the air outside the tank. In the tank there are homogeneous conditions thanks to a mixer (there is thus no spatial variations in temperature). It is assumed that the mixer does not add power to the liquid. It is assumed that there is a time delay in the response in the temperature if there is a change in the supplied power.

39.4.2 Variables and parameters

Variables and parameters of the heated tank are defined in Table 39.3.

39.4.3 Overall block diagram

Figure 39.6 shows an overall block diagram of the heated tank.

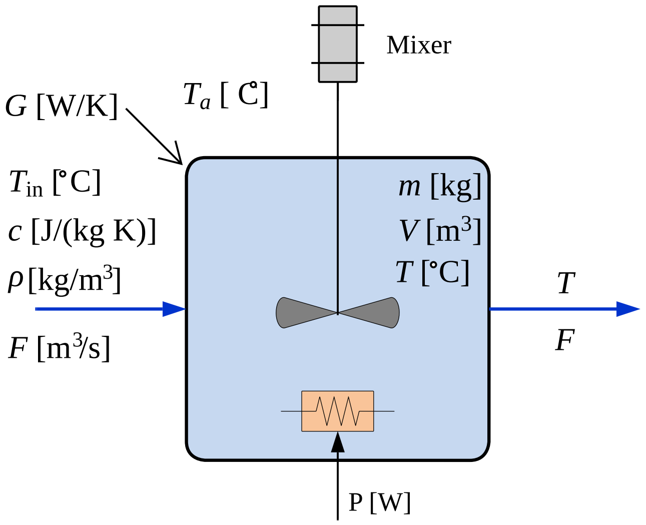


Figure 39.5: Heated tank.

39.4.4 Mathematical model

A mathematical model of the temperature variation, based on energy balance of the liquid in the tank, is

$$cmT'(t) = P(t - t_d) + cF_m\,[T_{\text{in}}(t) - T(t)] + G\,[T_a(t) - T(t)] \tag{39.6}$$

The term $P(t - t_d)$ expresses that the supplied power has time delay t_d. This time delay represents the time delay in the temperature that will be detected by a temperature sensor some place in any practical tank due to the unavoidable imperfect mixing.

The model (39.6) is "mass-based" as mass m and mass flow F are in the model. Alternatively, the model can be made "volume-based" using $m = \rho V$ and $F_m = \rho F_v$:

$$c\rho VT'(t) = P(t - t_d) + c\rho F_v\,[T_{\text{in}}(t) - T(t)] + G\,[T_a(t) - T(t)] \tag{39.7}$$

39.5 Air heater

39.5.1 System description

A laboratory air heater is described on http://techteach.no/air_heater. Figure 39.7 shows the air heater.[3]

[3] University South-Eastern Norway, campus Porsgrunn, has 26 of identical units of this lab station, being used in several control courses in both bachelor and master programmes in technology.

Table 39.3: Heated tank: Variables and parameters.

Symbol	Value (default)	Unit	Description
P	0	W	Supplied power
T	-	°C	Temperature of liquid
T_{init}	20	°C	Initial temperature
T_a	20	°C	Ambient temperature
T_{in}	20	°C	Temperature of liquid inflow
F_v	$0.25{\cdot}10^{-3}$	m^3/s	Liquid volumetric flow
$F_m = \rho F_v$	0.25	kg/s	Liquid mass flow
c	4200	J/(kg·K)	Specific heat capacity of liquid
ρ	1000	kg/m^3	Density of liquid
V	0.2	m^3	Liquid volume in tank
$m = \rho V$	200	kg	Liquid mass in tank
G	1000	W/K	Heat transfer coefficient of tank
t_d	60 s	s	Time delay in the temperature response

39.5.2 Variables and parameters

Variables and parameters and assumed parameter values are defined in Table 39.4.

39.5.3 Overall block diagram

Figure 39.8 shows an overall block diagram of the air heater.

39.5.4 Mathematical model

A mathematical model that has proven to describe quite well the dynamic behaviour of the outlet air temperature is given by the following differential equation representing "time constant with time delay" dynamics from control signal u to outlet temperature T:

$$T'(t) = \{K_h\,[u(t-t_d)] + [T_{\text{env}}(t) - T(t)]\}\,/t_d \tag{39.8}$$

This model may be derived from mechanistic (first-principles) modeling principles, i.e. a simple energy balance of the air, if we make the idealized assumption that the tube is a so-called CSTR (continuous stirred tank reactor) with air inflow and outflow and heat transfer with the environment through the "reactor" (tube) walls, and – in addition – we include the time delay as described above. In reality, the idealized CSTR conditions are not satisfied, but they lead to a useful model structure with parameter values that may be estimated from experimental data.

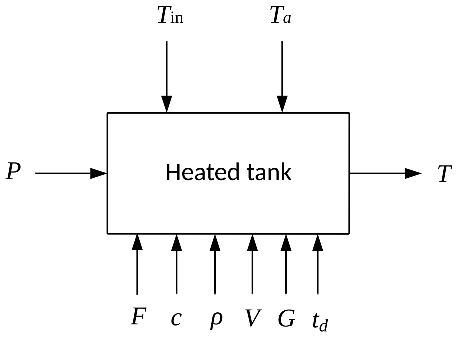


Figure 39.6: Block diagram of the heated tank.

Many other stable physical processes show “time constant with time delay” dynamics, and such processes may be reasonably well represented with models similar to (39.8).

39.5.5 Data file

A datafile from an experiment with the air heater is available on:

http://techteach.no/control/python/airheater_logfile.txt

The time step (sampling time) is 0.1 s.

The datafile containt the following columns (from left in the file):

- Time t [s]
- Control signal u [V]
- Temperature measurement signal T [°C]

39.6 Kettle

39.6.1 System description

Figure 39.9 shows a kettle.

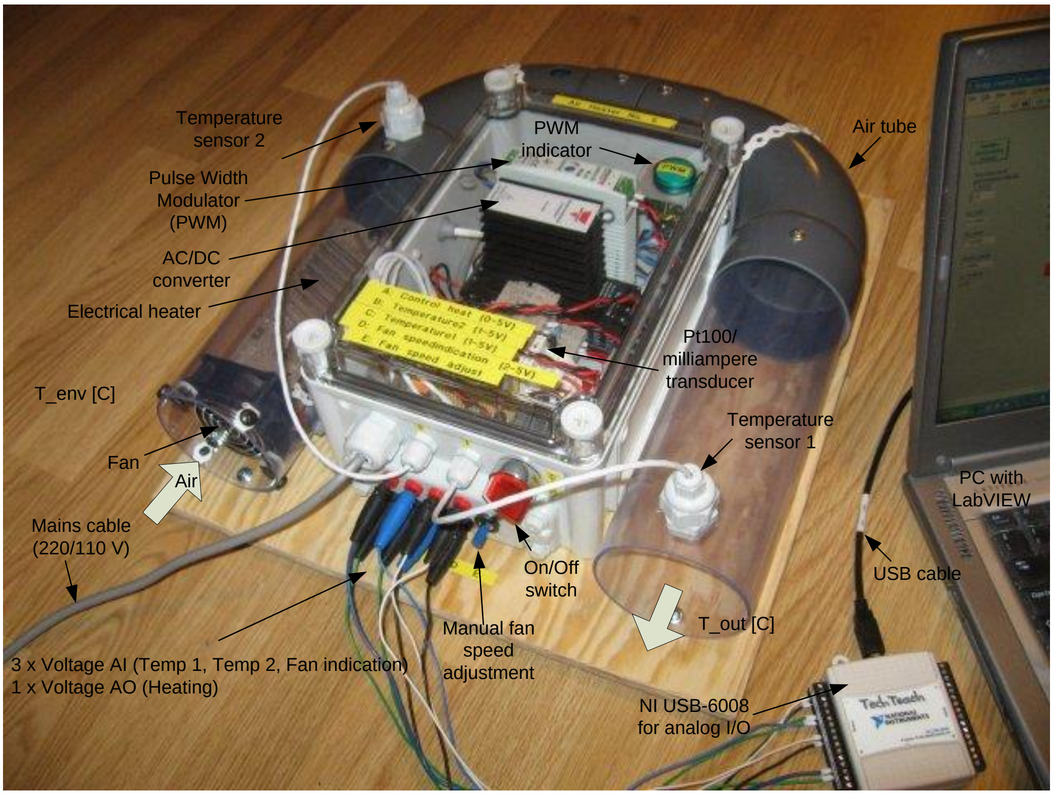


Figure 39.7: Laboratory air heater.

Figure 39.9: Kettle.

Figure 39.10 shows a sketch of the kettle.

Table 39.4: Air heater: Variables and parameters

Symbol	**Value (default)**	**Unit**	**Description**
T	-	°C	Temperature of the air flowing out of tube. Measured by a sensor.
T_{init}	25	°C	Initial temperature
T_{env}	25	°C	The environmental, or ambient, temperature. It is the temperature in the outlet air of the air tube when the control signal to the heater has been set to zero for relatively long time (some minutes).
u	-	V	Control signal to heater
K_h	3.5	°C/V	Heater gain
t_c	23.0	s	Time constant representing sluggishness of heater
t_d	3.0	s	Time delay representing air transportation and sluggishness of heater

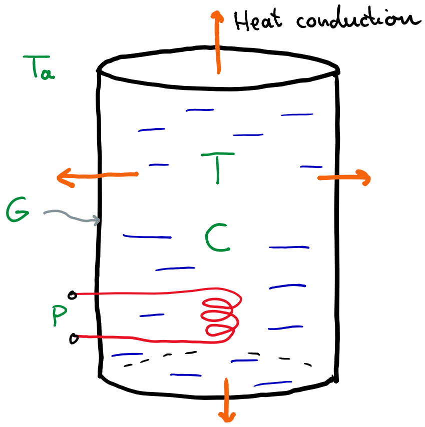


Figure 39.10: Sketch of the kettle.

Figure 39.11 shows a more detailed sketch of the kettle presenting several additional parameters used to calculate the “aggregate” parameters C and G shown in Figure 39.10.

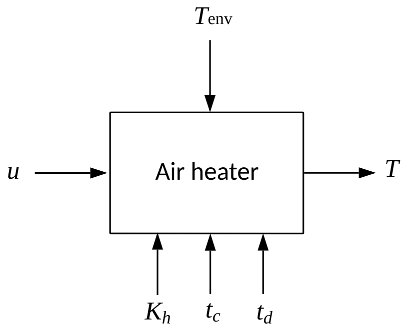


Figure 39.8: Block diagram of the air heater.

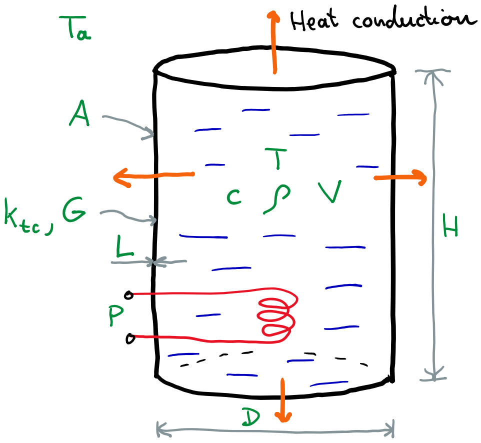


Figure 39.11: Detailed sketch of the kettle.

39.6.2 Parameters and variables

Parameters and variables of the mathematical model presented below are defined in Table 39.5.

About the time delay, t_d, in Table 39.5

It turns out that in practice it takes some time from any change of the supplied power, P, until there is a response in the water temperature, T. This time delay is due to thermal capacity ("sluggishness") in the heater itself and time for heat to "spread" in the water, causeing spatial (nonhomogeneous) temperature variations in the water. In experiments I

Table 39.5: Parameters and variables of the kettle.

Symbol	Value (default)	Unit	Description
T	[0, 100]	°C	Temperature of water in kettle
T_{init}	20	°C	Initial value (state) of T
T_a	20	°C	Ambient (room) temperature
P	–	W	Supplied power to the heater
$P_{\max}$	700	W	Max. power delivered by heater
c	4180	J/(kg·K)	Specific heat capacity of water
ρ	1000	kg/m^3	Density of water
V	0.0005	m^3	Volume of water
C	2101	J/K	Heat capacity of water in kettle
D	0.09	m	Inner diameter of kettle
H	0.079	m	Level of water
L	0.003	m	Thickness of plastic jacket
A	0.0351	m^2	Surface area of plastic jacket between water and air
k_{tc}	0.2	(Wm)/(m^{2}K)	Specific thermal conductivity of plastic
G	2.34	W/K	Thermal conductivity of plastic jacket
t_d	20	s	Time delay

have observed that $t_d \approx 20$ sec.

There is also a time delay from any change of the ambient temperature, T_a, to the water temperature, but since it is reasonable to assume that T_a is constant during experiments, and that the impact of T_a on T is static, I do not include that time delay in the model.

39.6.3 Overall block diagram

Figure 39.12 shows an overall block diagram of the kettle.

39.6.4 Mathematical model

A mathematical model for the water temperature in the kettle is based on the thermal energy balance of the water. We assume homogeneous temperature conditions in the water (the "continuously stirred tank" principle). The energy balance reads:

Change in thermal energy per time unit = net added power

which can be expressed mathematically with the following differential equation of T:

$$CT'(t) = P(t - t_d) + G\,[T_a - T(t)] \tag{39.9}$$

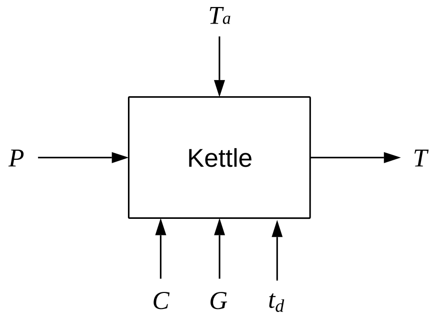


Figure 39.12: Overall block diagram of the kettle.

Written as a state space model:

$$T'(t) = \{[P(t-t_d) + G\,(T_a - T(t))]\}\,/C \tag{39.10}$$

In the model:

- CT [J] is thermal energy in the water. CT' [J/s = W] is then the change in thermal energy per unit of time, or in other words: CT' is the time derivative of energy CT.
- P [W] is supplied electric power to the heating element which heats the water. On the real kettle depicted in Figure 39.9 P can only be 0 W ("off") or 700 W ("on"). However, we shall assume that P is adjustable and can take on any value between 0 and 700 W (this can be obtained in practice with the principle of pulse-width modulation, cf. Section 3.6.3).
- $G\,(T_a - T)$ [W] is the power transfer between the air in the room and the water. The thermal conductivity G is positive. If T_a is greater than T, which may be the situation before the kettle is switched on, the term $G\,(T_a - T)$ is positive, i.e. that it is power supply from the air to the water. If T_a is smaller than T, which eventually becomes the situation after the kettle has been switched on, the term $G\,(T_a - T)$ is negative, i.e. there is a power loss from the water to the air.

The model above contains no limitation of T, and a simulation based on (39.9) could result in T far higher than 100 degrees C. which may mean that the kettle eventually melts and the kitchen is set on fire. Also, we shall assume that the kettle is not able to produce ice. So, we assume that the temperature (the state) is limited to

$$T \in [0,\ 100]^{\circ}\mathrm{C} \tag{39.11}$$

Detailed mathematical modeling

Below is more information about the parameters C, A and G:

- C is
  $$C = c\rho V \tag{39.12}$$
  where
  $$V = H \cdot \pi \cdot \left(\frac{D}{2}\right)^2 \tag{39.13}$$
- A is the area (of plastic) assumed to enclose all the water, both the cylindrical side wall and the circular top and bottom surfaces:
  $$A = \pi D H + 2\pi \left(\frac{D}{2}\right)^2 \tag{39.14}$$
- G is
  $$G = \frac{k_{tc}}{L} A \tag{39.15}$$

39.7 Biogas reactor

39.7.1 System description

A biogas reactor is a vessel or tank in which organic matter, like food waste, slaughter waste, livestock manure, sewage sediments, etc., is converted by various cultures of microorganisms into energy-rich, combustable methane (CH_4) gas, which is the most important product, and carbondioxide (CO_2) gas. The two cultures assumed in the model presented here, are acidogens, which generate volatile fatty acids (VFAs), and methanogens which generate methane. The biological conversion process takes place without oxygen, and is often called an anaerobic digestion (AD) process. The AD process is assumed continuous, not “plug-flow”.

Biogas contains roughly 65 % CH_4 and 35 % CO_2. Combustion of the CH_4 gas results in CO_2 and H_2O. Biogas, raw or upgraded into approx. 98% CH_4, can be used in combustion motors to produce mechanical power to vehichles, or electric power through a generator connected to the motor, or just heat in gas burners. The liquid phase can be used as fertilizer. The AD process is a part of a closed carbon cycle, opposite to processing and utilizing fossil fuel. The climate footprint is favourable as the combustion converts CH_4, which has (gives) a relatively high climate footprint, into CO_2 which has a considerably lower footprint, and H_2O, which has no footprint.

One of the results of my own research in model-based monitoring and control of biogas reactors Haugen et al. (2013), is a mathematical model adapted to a pilot reactor[4] at Foss Biolab at Foss farm in Skien, Norway, using online measurements and laboratory analyses. The model may be referred to as the (modified) Hill model Hill (1983). Figure 39.13 shows the principal construction of the reactor.

[4] Reactor feed is filtered cow manure.

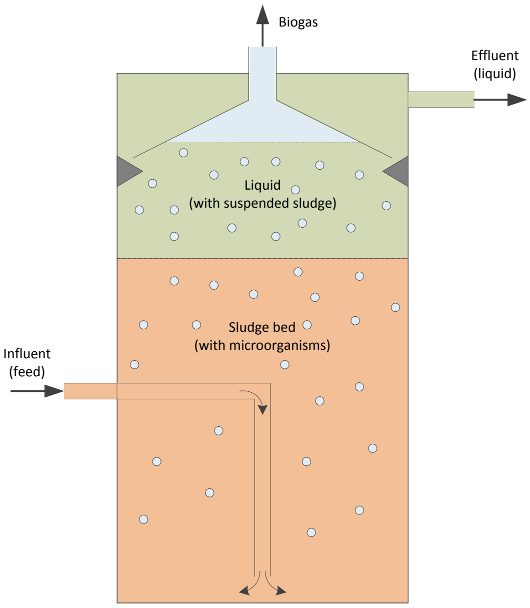

Biogas
Effluent (liquid)
Liquid (with suspended sludge)
Sludge bed (with microorganisms)
Influent (feed)

Figure 39.13: Principal construction of the biogas reactor.

39.7.2 Variables and parameters

Nomenclature of variables are shown in Table 39.6.

Abbreviations are defined in Table 39.8.

39.7.3 Overall block diagram

Figure 39.14 shows an overall block diagram of the mathetmatical model of the biogas reactor.

39.7.4 Mathematical model

The mathematical model is as presented below. Figure 39.14 shows an overall block diagram of the mathetmatical model. Nomenclature of variables is given in Table 39.6. Parameters with assumed values are defined in Table 39.7. Abbreviations are defined in Table 39.8.

- Definition of the portion of the raw waste which can serve as substrate for the conversion done by microorganisms:

$$S_{\text{bvs,in}} = B_0 S_{\text{vs,in}} \tag{39.16}$$

- Definition of the portion of the biodegradable material which is initially in acid form:

$$S_{\text{vfa,in}} = A_f S_{\text{bvs,in}} \tag{39.17}$$

Table 39.6: Biogas reactor: Variables.

Symbol	**Unit**	**Description**
F_{feed}	$\frac{\mathrm{L}}{\mathrm{d}}$	Flow, or load rate
$D = \frac{F_{\mathrm{feed}}}{V}$	d^{-1}	Dilution rate (normalized flow)
F_{meth}	$\frac{\mathrm{L}\ CH_4}{\mathrm{d}}$	Methane gas flow
$q_{\mathrm{meth}} = \frac{F_{\mathrm{meth}}}{V}$	$\left(\frac{\mathrm{L}\ CH_4}{\mathrm{d}}\right)/\mathrm{d}$	Volume-specific (normalized) methane gas flow
μ	d^{-1}	Reaction (growth) rate of acidogens
μ_c	d^{-1}	Reaction (growth) rate of methanogens
μ_m	d^{-1}	Maximum reaction rate for acidogens
μ_{mc}	d^{-1}	Maximum reaction rate for methanogens
$S_{\mathrm{vs,in}}$	$\frac{\mathrm{g\ VS}}{\mathrm{L}}$	Concentration of VS in influent
$S_{\mathrm{bvs,in}}$	$\frac{\mathrm{g\ BVS}}{\mathrm{L}}$	Concentration of BVS in influent
S_{bvs}	$\frac{\mathrm{g\ BVS}}{\mathrm{L}}$	Concentration of BVS in reactor
$S_{\mathrm{vfa,in}}$	$\frac{\mathrm{g\ VFA}}{\mathrm{L}}$	Concentration of VFA in biodegradable part of influent
S_{vfa}	$\frac{\mathrm{g\ VFA}}{\mathrm{L}}$	Concentration of VFA acids in reactor
X_{acid}	$\frac{\mathrm{g\ acidogens}}{\mathrm{L}}$	Concentration of acidogens
X_{meth}	$\frac{\mathrm{g\ methanogens}}{\mathrm{L}}$	Concentration of methanogens
T_{reac}	°C	Reactor temperature

- Material balance of biodegradable volatile solids:

$$S_{\mathrm{bvs}}{}' = \left(S_{\mathrm{bvs,in}} - S_{\mathrm{bvs}}\right) D - \mu k_1 X_{\mathrm{acid}} \tag{39.18}$$

- Material balance of total VFA:

$$S_{\mathrm{vfa}}{}' = \left(S_{\mathrm{vfa,in}} - S_{\mathrm{vfa}}\right) D + \mu k_2 X_{\mathrm{acid}} - \mu_c k_3 X_{\mathrm{meth}} \tag{39.19}$$

- Material balance of acidogens:

$$X_{\mathrm{acid}}{}' = \left(\mu - K_d - \frac{D}{b}\right) X_{\mathrm{acid}} \tag{39.20}$$

- Material balance of methanogens:

$$X_{\mathrm{meth}}{}' = \left(\mu_c - K_{dc} - \frac{D}{b}\right) X_{\mathrm{meth}} \tag{39.21}$$

Table 39.7: Biogas reactor: Parameters.

Symbol	**Value (default)**	**Unit**	**Description**
B_0	0.25	$\frac{\text{g BVS/L}}{\text{g VS/L}}$	Biodegradability constant
A_f	0.69	$\frac{\text{g VFA/L}}{\text{g BVS/L}}$	Acidity constant
b	2.90	d/d	Retention time ratio
k_1	3.89	$\frac{\text{g BVS}}{\text{g acidogens}}$	Yield constant
k_2	1.76	$\frac{\text{g VFA}}{\text{g acidogens}}$	Yield constant
k_3	31.7	$\frac{\text{g VFA}}{\text{g methanogens}}$	Yield constant
k_5	26.3	$\frac{\text{L meth/L reac}}{\text{g methanogens/L reac}}$	Yield constant
K_d	0.02	d^{-1}	Specific death rate of acidogens
K_{dc}	0.02	d^{-1}	Specific death rate of methanogens
K_s	15.5	$\frac{\text{g BVS}}{\text{L}}$	Monod half-velocity constant for acidogens
K_{sc}	3	$\frac{\text{g VFA}}{\text{L}}$	Monod half-velocity constant for methanogens
V	250	L	Reactor volume of Foss pilot reactor

Table 39.8: Abbreviations.

VS	Volatile solids ("organic matter")
BVS	Biodegradable volatile solids
VFA	Volatile fatty acid

- Volume-specific (normalized) methane gas flow (production):

$$q_{\text{meth}} = \mu_c k_5 X_{\text{meth}} \tag{39.22}$$

Reaction rates, assuming Monod kinetics, are:

$$\mu = \mu_m \frac{S_{\text{bvs}}}{K_s + S_{\text{bvs}}} \tag{39.23}$$

$$\mu_c = \mu_{mc} \frac{S_{\text{vfa}}}{K_{sc} + S_{\text{vfa}}} \tag{39.24}$$

where the maximum reaction rates are linear functions of the reactor temperature (the Hashimoto function):

$$\mu_m (T_{\text{reac}}) = \mu_{mc} (T_{\text{reac}}) = 0.013 T_{\text{reac}} - 0.129 \tag{39.25}$$
$$(20\ {}^\circ\text{C} < T_{\text{reac}} < 60\ {}^\circ\text{C})$$

The reactor temperature may be kept at a specified temperature reference, e.g. 35 °C (as for mesophilic conditions), with an automatic temperature control system.

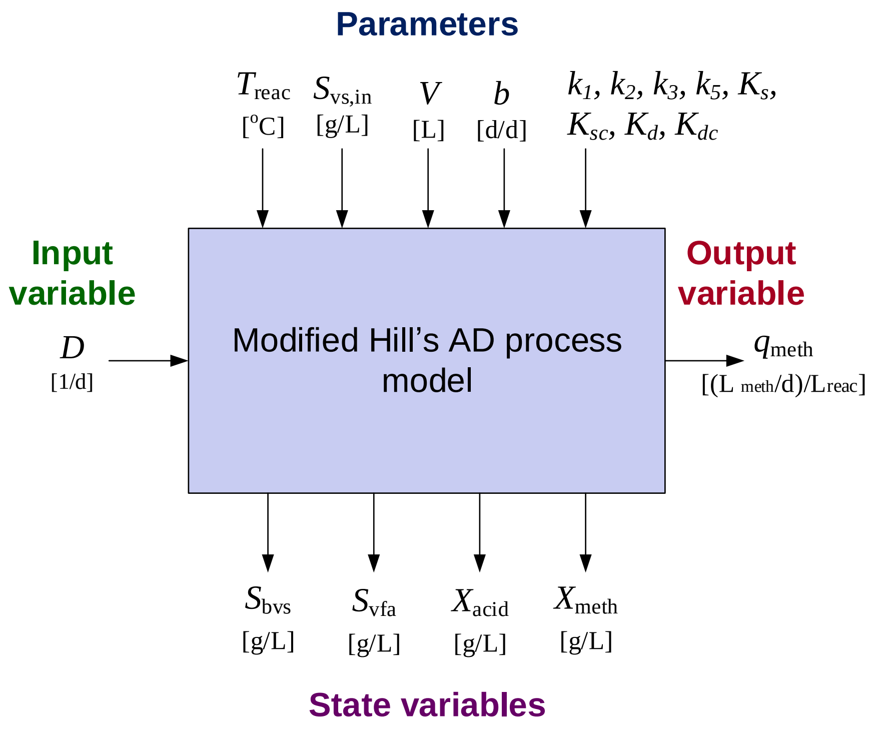


Figure 39.14: Overall block diagram of the mathematical model of the biogas reactor.

Model of temperature

In the above model, the temperature, T_{reac}, is regarded as a model parameter with a set value. However, T_{reac} is actually a dynamic variable, and it can be modeled with an energy balance, cf. Sec. 39.4. The dynamic behaviour of T_{reac} can be simulated using that model.

39.7.5 Operating point

In analysis of reactor dynamics and stability and in design of some types of state estimators and controllers it may be necessary to define a proper steady state operating point. A steady state operating point can be found from e.g. a simulation by reading off the value of the state variables at steady state. One example of a steady state operating point is given in Table 39.9. (On the Foss pilot reactor, $D = 0.18$ d^{-1}corresponds to $F_{\text{feed}} = VD = 250$ L $\cdot$ 0.18 d^{-1} $= 45$ L/d.)

39.8 DC motor

39.8.1 System description

Figure 39.15 shows a DC motor. The motor can be manipulated with an input voltage signal, u.

Table 39.9: Values of inputs and states in one example of a steady state operation point.

$D = 0.18$ d^{-1}
$T_{\text{reac}} = 35$ °C
$S_{\text{vs,in}} = 30.2$ g/L
$S_{\text{bvs}} = 5.2149$ g/L
$S_{\text{vfa}} = 1.0093$ g/L
$X_{\text{acid}} = 1.3166$ g/L
$X_{\text{meth}} = 0.3637$ g/L

The rotational speed is measured with both a tachogenerator which produces a output voltage signal which is proportional to the speed, and an encoder with resolution 360 pulses per revolution.

39.8.2 Overall block diagram

Figure 39.16 shows an overall block diagram of the motor with tachogenerator.

39.8.3 Variables and parameters

Variables and parameters with assumed values are defined in Table 39.10.

Table 39.10: DC motor with tachogenerator: Variables and parameters.

Symbol	**Value (default)**	**Unit**	**Description**
S	-	krpm	Rotational speed of the motor
S_{init}	0	krpm	Initial speed
S_m		V	Speed measurement signal generated by the tachogenerator
u	[−10 V, +10 V]	V	Control signal to the motor
L	0	V	Equivalent load torque, represented in the same unit as the control variable. L can be regarded as environmental variable or process disturbance.
K	0.15	krpm/V	Motor gain
t_c	0.30	s	Time constant of motor
K_{tacho}	0.17	krpm/V	Tachogenerator gain

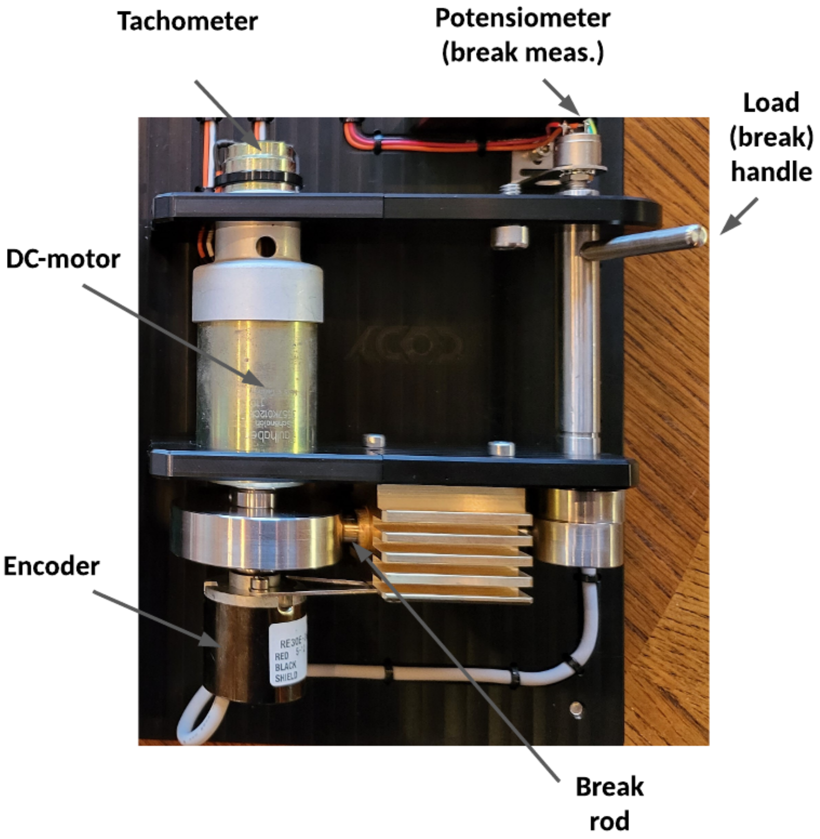


Figure 39.15: DC motor.

39.8.4 Mathematical model

A mathematical model of the motor is (omitting the time argument for simplicity):

$$S' = [K(u + L) - S]/t_c \tag{39.26}$$

When the speed is measured with the tachogenerator, the speed can be calculated from the tachogenerator voltage with

$$S = K_{\text{tacho}} S_{\text{m}} \tag{39.27}$$

39.8.5 Datafile

A datafile from an experiment is available on:

http://techteach.no/control/python/data_dc_motor.txt

The time step (sampling time) is 0.02 s.

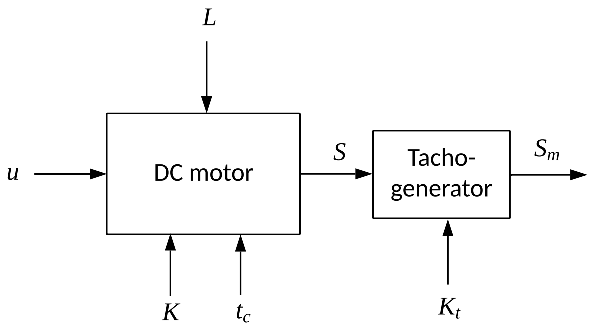


Figure 39.16: Block diagram of DC motor with tachogenerator.

The datafile containt the following columns (from left in the file):

- Time t [s]
- Control signal u [V]
- Tachogenerator measurement signal S_m [V]

39.9 Ship

39.9.1 System description

Figure 39.17 shows a ship. In this example we will only take the longitudinal motion relative to the ship, also denoted the surge motion, into account. [5]

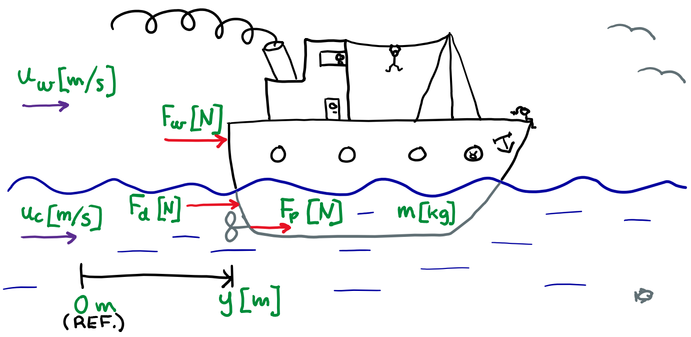


Figure 39.17: Ship.

[5] Courtesy of Kongsberg Maritime AS, Norway for providing realistic parameter values.

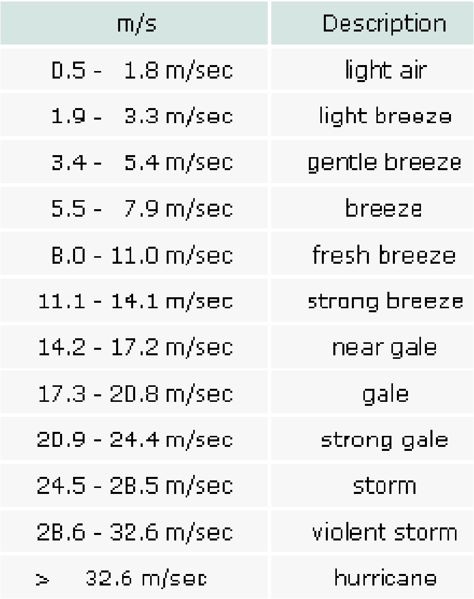
39.9.2 Variables and parameters

Variables and parameters are defined in Table 39.11. The parameter values are realistic.[6]

Table 39.11: Ship: Variables and parameters

Symbol	Value (default)	Unit	Description
y	-	m	Longitudinal or surge position of the ship
y'	-	m/s	Ship speed
F_p	$[-467 \cdot 10^3, +552 \cdot 10^3]$	N	Propeller force applied to move the ship
F_h	-	N	Hydrodynamic force on the ship
F_w	-	N	Wind force on the ship
u_c	$[-3\ ,\ +3]$	m/s	Water current speed
V_w	Cf. windscale in Fig. 39.18	m/s	Wind speed
m	$71164 \cdot 10^3$	kg	Mass of ship
D_h	$8.4 \cdot 10^3$	$\mathrm{N/(m/s)^2}$	Hydrodynamic force constant
D_w	$0.177 \cdot 10^3$	$\mathrm{N/(m/s)^2}$	Wind force constant

Figure 39.18 shows the wind scale which characterizes various ranges of wind speed V_w.



m/s	Description
0.5 - 1.8 m/sec	light air
1.9 - 3.3 m/sec	light breeze
3.4 - 5.4 m/sec	gentle breeze
5.5 - 7.9 m/sec	breeze
8.0 - 11.0 m/sec	fresh breeze
11.1 - 14.1 m/sec	strong breeze
14.2 - 17.2 m/sec	near gale
17.3 - 20.8 m/sec	gale
20.9 - 24.4 m/sec	strong gale
24.5 - 28.5 m/sec	storm
28.6 - 32.6 m/sec	violent storm
> 32.6 m/sec	hurricane

Figure 39.18: Wind scale

[6] Courtesy of Kongsberg Maritime AS, Norway.

39.9.3 Overall block diagram

Figure 39.19 shows a block diagram of the ship.

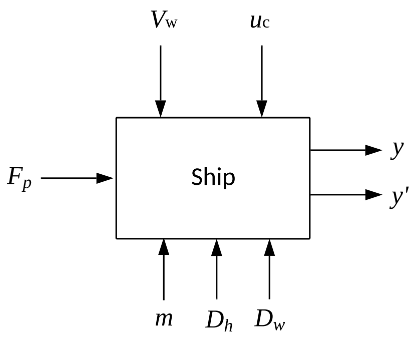


Figure 39.19: Block diagram of the ship.

39.9.4 Mathematical model

We use the Newton's Second Law to model the motion of the ship:

Mass times acceleration = Sum of forces

In mathematical terms:

$$my'' = F_p + F_h + F_w \tag{39.28}$$

where:

- F_h is proportional to the difference between the water speed and the ship speed:

  $$F_h = D_h \left(u_c - y'\right) \left|u_c - y'\right| \tag{39.29}$$

  The absolute value in the last term ensures that the sign of F_h correct.

- F_w is proportional to the square of the difference between the wind speed and the ship speed:

  $$F_w = D_w \left(V_w - y'\right) \left|V_w - y'\right| \tag{39.30}$$

  The absolute value in the last term ensures that the sign of F_w is correct.

When the ship moves, there is also a motion of an amount of water. The mass of the water is denoted the added mass. However, we disregard the added mass here.

39.10 Pendulum on cart

39.10.1 System description

Figure 39.20 shows the cart with the pendulum.

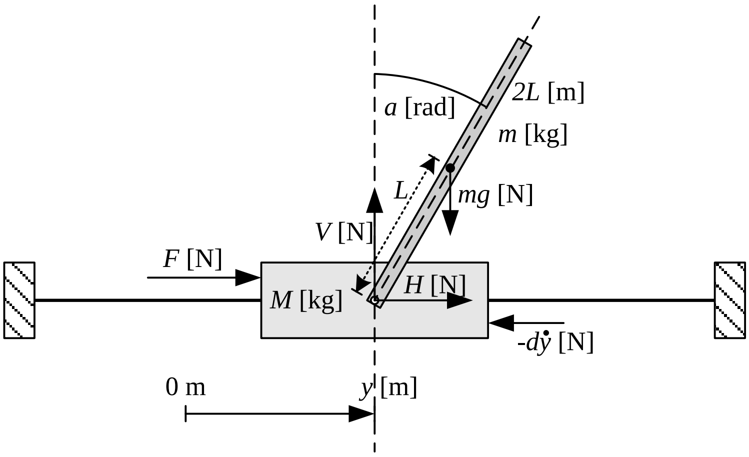


Figure 39.20: Cart with pendulum.

A motor attached to the cart acts on the cart with a force F. The mass of the motor is assumed included in the mass of the cart. This force is manipulated by the controller to stabilize the pendulum in an standing position (similar to a rocket) or in a hanging position (similar to a crane) at a specified position of the cart.

39.10.2 Variables and parameters

Variables and parameters and assumed parameter values are defined in Table 39.12.

Table 39.12: Pendulum on cart: Variables and parameters.

Symbol	**Value (default)**	**Unit**	**Description**
y	0	m	Cart position.
a	0 (standing) or π (hanging)	rad	Pendulum angle
L	0.5	m	Half length of pendulum (from fixed point on cart to center of gravity)
M	1	kg	Mass of cart
m	0.1	kg	Mass of pendulum
d	0	N/(m/s)	Damping coefficient
g	9.81	m/s^2	Gravity
F	0	N	Force applied to cart

39.10.3 Overall block diagram

Figure 39.21 shows an overall block diagram of the pendulum.

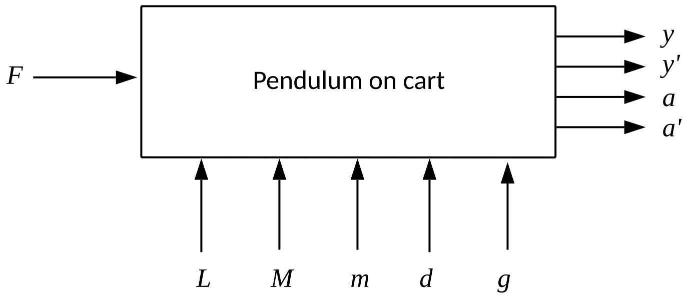


Figure 39.21: Block diagram of the pendulum.

39.10.4 Mathematical model

The mathematical model is based on the following principles:

1. Force balance applied to the cart:
$$My'' = F - H - dy' \tag{39.31}$$

2. Force balance (Newton's Second Law) applied to the horizontal movement of the center of gravity of the pendulum:
$$m\,(y + L\sin a)'' = H \tag{39.32}$$
The second order differentiation must be carried out, but the result of it is not shown here.

3. Force balance applied to the vertical movement of the center of gravity of the pendulum:
$$m\,(L\cos a)'' = V - mg \tag{39.33}$$
The second order differentiation must be carried out, but the result of the differentiation is not shown here.

4. Torque balance (the rotational version of Newton's Second Law) applied to the center of gravity of the pendulum:
$$Ia'' = VL\sin a - HL\cos a \tag{39.34}$$

In the above equations,

- I is the moment of inertia of the pendulum about it's center of gravity. For the pendulum shown in Figure 1,
$$I = \frac{mL^2}{3} \tag{39.35}$$
- V and H are vertical and horizontal forces, respectively, in the pivot.
- d is a damping coefficient.

From Eq. (39.32)-(39.31), the internal forces V and H can be eliminated, resulting in two second order differential equations containing y'' and a''. These differential equations are, however, not shown here.

Let us define the following state variables:

- $x_1 = y$ (cart horizontal position)
- $x_2 = y'$ (cart horizontal speed)
- $x_3 = a$ (pendulum angular position)
- $x_4 = a'$ (pendulum angular speed)

The two differential equations mentioned above can then be written as the following non-linear state space model:

$$x_1{}' = f_1 \tag{39.36}$$
$$x_2{}' = f_2 \tag{39.37}$$
$$x_3{}' = f_3 \tag{39.38}$$
$$x_4{}' = f_4 \tag{39.39}$$

where:

$$f_1 = x_2 \tag{39.40}$$
$$f_2 = \frac{-m^2L^2g\cos x_3 \sin x_3 + \left(I + mL^2\right)\left[mL\left(x_4\right)^2 \sin x_3 - dx_2\right]}{D_1} + \frac{\left(I + mL^2\right)}{D_1}u \tag{39.41}$$
$$f_3 = x_4 \tag{39.42}$$
$$f_4 = \frac{\left(m + M\right)\left(mgL\sin x_3\right) - mL\cos x_3\left[mL\left(x_4\right)^2 \sin x_3 - dx_2\right]}{D_1} + \frac{-mL\cos x_3}{D_1}u \tag{39.43}$$

where:

$$D_1 = \left(I + mL^2\right)\left(m + M\right) - m^2L^2\cos^2 x_3 \tag{39.44}$$

Linearized model

A control system can be used to stabilize the pendulum at two alternative operating points, namely vertically up or vertically down. The matrices A and B of a linear model are derived for each of these operating points. Instead of linearizing using the general linearization formulas presented in Ch. 6.5, we will use a simpler approach where we assume small variations around the pertinent operating point.

Pendulum up:

Linearization of (39.36)-(39.39) about $x_3 = a = 0$ is based on the following assumptions:

$$\cos x_3 \approx 1 \tag{39.45}$$

$$\sin x_3 \approx x_3 \text{ [rad]} \tag{39.46}$$

$$(x_4)^2 = (\dot{a})^2 \approx 0 \tag{39.47}$$

Pendulum down:

Linearization of (39.36)-(39.39) about $x_3 = a = \pi$ is based on the following assumptions:

$$\cos x_3 \approx -1 \tag{39.48}$$

$$\sin x_3 \approx \pi - x_3 \text{ [rad]} \tag{39.49}$$

$$(x_4)^2 = (\dot{a})^2 \approx 0 \tag{39.50}$$

Using these assumptions, the linearized model becomes:

$$\begin{bmatrix} \Delta x_1{}' \\ \Delta x_2{}' \\ \Delta x_3{}' \\ \Delta x_4{}' \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{\left(I+mL^2\right)d}{D_2} & -\frac{m^2L^2g}{D_2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{K_{\cos}mLd}{D_2} & \frac{K_{\sin}(m+M)mgL}{D_2} & 0 \end{bmatrix}}_{A} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{I+mL^2}{D_2} \\ 0 \\ -\frac{K_{\cos}mL}{D_2} \end{bmatrix}}_{B} [\Delta u] \tag{39.51}$$

where:

$$D_2 = \left(I + mL^2\right)(m + M) - m^2L^2 \tag{39.52}$$

The values of $K_{\cos}$ and $K_{\sin}$ in (39.51) are:

- Pendulum up:

$$K_{\cos} = 1 \tag{39.53}$$

$$K_{\sin} = 1 \tag{39.54}$$

- Pendulum down:

$$K_{\cos} = -1 \tag{39.55}$$

$$K_{\sin} = -1 \tag{39.56}$$

Chapter 40

Python

40.1 About Python

Python was developed by Guido van Rossum and lauched in its first version in 1991. It is a free, open source programming tool which has become one of the most popular programming tools in the world. A large number of packages for special applications have been developed, e.g. Numpy for numerical calculations, Matplotlib for plotting, Scipy for scientific computing, etc.

The official “home” of Python is maintained by the Python Software Foundation:

https://www.python.org/

40.2 Installing Python

A popular way of getting Python with the most commonly used packages installed on your computer is to simply install the Anaconda distribution of Python tools from

https://www.anaconda.com/

In this book, the Python Control package is used. This package is not included in the Anaconda distribution, so it must be installed separately. The Python Control package is presented in Appendix 41.

40.3 Learning Python

There are lots of online resources and books for learning Python. Below are some resources which I have developed:

A presentation file (pdf) which gives a flash course in Python programming:

http://techteach.no/control/python_course/python_flash_course_2021_08_30.pdf

A sequence of three video lectures which covers most of the topics of the above presentation:

https://www.dropbox.com/s/r6o5swordk90qxb/python_flash_course_part_1_2021_08_30.mp4?dl=0

https://www.dropbox.com/s/d2u45acf5qnjzog/python_flash_course_part_2_2021_08_30.mp4?dl=0

https://www.dropbox.com/s/91egnyu6zc0m4au/python_flash_course_part_3_2021_08_30.mp4?dl=0

This Python book in Norwegian may be of interest (but it is not free):

http://techteach.no/python/

Chapter 41

Python Control package

41.1 Introduction

41.1.1 What is the Python Control package?

The Python Control Package is for analysis and design of dynamic systems in general and feedback control systems in particular. The package resembles the Control System Toolbox in MATLAB.

The package is developed at California Institute of Technology (Caltech), USA, by prof. Richard M. Murray and coworkers.

The package requires Numpy, Scipy, and Matplotlib (these packages are installed with the Anaconda distribution of Python tools).

The home page of the Python Control package is

https://pypi.org/project/control/

A complete list of the functions in the package is available via the link named Homepage on the above home page.

41.1.2 About this guide

The guide covers only some of the functions in the Python Control Package. However, these function are basic, and if you master these functions, you should be well prepared for using other functions in the package.

Most of the tutorial is about continuous-time models, i.e. transfer functions based on the

Laplace transform and state space models based on differential equations. Discrete-time models are briefly covered in one chapter at the end of the tutorial. That coverage is brief because the basic functions for continuous-time models can be used also for discrete-time models, i.e. with the same syntax, however with the sampling time (period) as an extra input argument in the functions.

41.1.3 Installing the Python Control package

You can install the package with the command

```
pip install control
```

executed e.g. at the Anaconda prompt (in the Anaconda command window)[1].

Some functions in the Python Control package, for example the lqr function for calculating the stationary controller gain G in LQ control, requires that the package slycot is installed. You can install it with the following command at the Anaconda prompt:

```
conda install -c conda-forge slycot
```

(The straightforward “pip install slycot” may not work.)

41.1.4 Importing the Python Control package into Python

The following command (in Python) imports the Python Control package into Python:

```
import control
```

41.1.5 Using arrays for numerical data

In Python, tuples, lists, dictionaries, and arrays can be used to store numerical data. However, only arrays are practical for mathematical operations on the data, like addition and multiplication. Therefore, I use arrays as the numerical data type consistently in this book.

To use arrays, you must import the numpy package. It has become a tradition to rename the numpy package as np. Thus, to import numpy, include the following command in the beginning of your program:

```
import numpy as np
```

[1] In Windows: Start menu / Anaconda / Anaconda prompt.

41.2 Transfer functions

This section is about Laplace transform based transfer functions, which may be referred to as s-transfer functions (s is the “Laplace variable”). Discrete time transfer functions, or z-transfer functions, are covered by Section 41.5.1.2.

41.2.1 How to create transfer functions

There are two ways to create transfer functions in the Python Control Package:

- By defining a variable named ’s’ representing the Laplace variable, and creating transfer functions in terms of s, see Section 41.2.1.1.
- By creating arrays (or lists) to represent the numerator and denominator of the transfer function, see Section 41.2.1.2.

Usually, I prefer the first of these ways since it is closer to how I define transfer functions with hand-writing.

41.2.1.1 Creating transfer functions using the Laplace variable

For illustration, assume that the transfer function is

$$H(s) = \frac{b_1 s + b_0}{a_1 s + a_0} \tag{41.1}$$

In this case, we can create $H(s)$ in Python Control Package simply with the following two lines of Python code:

```
s = control.tf('s')
```

which defines s as the Laplace variable, and

```
H = (b1*s + b0)/(a1*s + a0)
```

where H is the resulting transfer function (an object). Above it is assumed that b1, b0, a1 and a0 have already been defined as Python variables with some values.

Example 41.1 *Creating a transfer function using the Laplace variable*

Let us create the following transfer function:

$$H(s) = \frac{2}{5s + 1} \tag{41.2}$$

The Python program 41.1 creates this transfer function. The code print(’H(s) = ’, H) is used to present the transfer function in the console (of Spyder).

http://techteach.no/control/python/create_tf_using_s.py

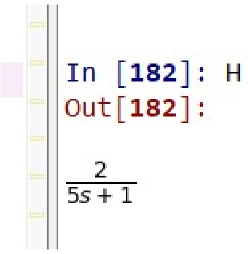
Listing 41.1: create_tf_using_s.py

```
import control

# %% Creating the transfer function:

s = control.tf(’s’)
H = 2/(5*s + 1)

# %% Displaying the transfer function:

print(’H(s) =’, H)
```

The result as shown in the console is:

```
H(s) =
  2
--------
5 s + 1
```

If you execute “H” (+ enter) in the Spyder console, the transfer function is more nicely displayed, see Figure 41.1.



```
In [182]: H
Out[182]:
```

$$\frac{2}{5s + 1}$$

Figure 41.1: The transfer function nicely displayed with H (+ enter) executed in the console.

[End of Example 41.1]

41.2.1.2 Creating transfer functions using coefficient arrays of numerator and denominator

As an alternative to defining transfer functions with the Laplace variable, they can be defined using coefficient arrays of numerator and denominator. The syntax is as follows:

$$\text{H} = \text{control.tf(num, den)}$$

where H is the resulting transfer function. num (representing the numerator) and den (representing the denominator) are arrays where the elements are the coefficients of the s-polynomials in descending order from left to right. Of course, you can use any other names than H, num, and den in your own programs.

To illustrate the syntax, assume that the transfer function is

$$H(s) = \frac{b_1 s + b_0}{a_1 s + a_0} \tag{41.3}$$

The Python code:

num = np.array([b1, b0])

den = np.array([a1, a0])

where, of course, the values of b1, b0, a1 and a0 have already been defined and assigned values.

Example 41.2 *Creating a transfer function using coefficient arrays*

We will create the following transfer function:

$$H(s) = \frac{2}{5s + 1} \tag{41.4}$$

The Python program 41.2 creates this transfer function. The code print('H(s) = ', H) is used to present the transfer function in the console (of Spyder).

http://techteach.no/control/python/create_tf.py

Listing 41.2: create_tf.py

```
import numpy as np
import control

# %% Creating the transfer function:

num = np.array([2])
den = np.array([5, 1])
H = control.tf(num, den)

# %% Displaying the transfer function:

print('H(s) =', H)
```

The result of the code above is shown as follows in the console:

```
H(s) =
  2
--------
5 s + 1
```

[End of Example 41.2]

41.2.2 Combinations of transfer functions

The following sections shows how we can combine transfer functions in

- series combination
- parallel combination
- feedback combination

41.2.2.1 Series combination

Figure 41.2 illustrates a series combination of two transfer functions.

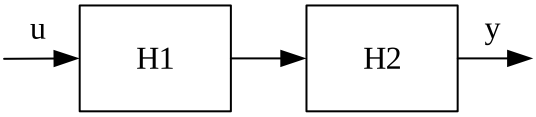


Figure 41.2: A series combination of two transfer functions, $H_1(s)$ and $H_2(s)$.

The resulting transfer function is

$$\frac{y(s)}{u(s)} = H(s) = H_2(s)H_1(s) \tag{41.5}$$

If you are to calculate the combined transfer function manually using (41.5), the order of the factors in (41.5) is of no importance for SISO[2] transfer functions. But for MIMO[3] transfer functions, the order in (41.5) is crucial.

Whether SISO or MIMO, you can create a series combination with the multiplication operator, *, in Python:

H = H1*H2

Example 41.3 *Series combination of transfer functions*

Assume a series combination,

$$H(s) = H_1(s)H_2(s)$$

of the following two transfer functions:

$$H_1(s) = \frac{K_1}{s} \tag{41.6}$$

[2] SISO = Single Input Single Output

[3] MIMO = Multiple Input Multiple Output

$$H_2(s) = \frac{K_2}{T_1 s + 1} \tag{41.7}$$

where $K_1 = 2$, $K_2 = 3$, and $T = 4$.

Manual calculation gives:

$$H(s) = \frac{K_1}{s} \cdot \frac{K_2}{Ts + 1} = \frac{K_1 K_2}{Ts^2 + s} = \frac{6}{4s^2 + s}$$

Program 41.3 shows how the calculations can be done with the * operator.

http://techteach.no/control/python/series_tf.py

Listing 41.3: series_tf.py

```python
import control

s = control.tf('s')

K1 = 2
K2 = 3
T = 4

H1 = K1/s
H2 = K2/(T*s + 1)

H = H1*H2

print('H =', H)
```

The result of the code above as shown in the console is:

```
H =
  6
------------
4 s^2 + s
```

[End of Example 41.3]

Alternative: control.series()

As an alternative to using the * operator, you can use the series() function of the Python Control package:

H = control.series(H1, H2)

41.2.2.2 Parallel combination

Figure 41.3 illustrates a parallel combination of two transfer functions.

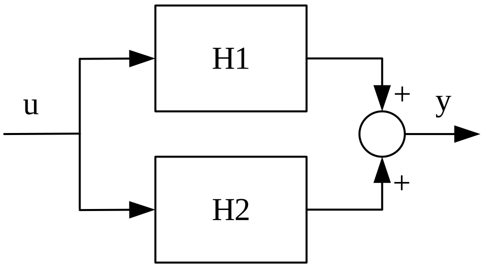


Figure 41.3: A parallel combination of two transfer functions, $H_1(s)$ and $H_2(s)$.

The resulting transfer function is

$$\frac{y(s)}{u(s)} = H(s) = H_2(s) + H_1(s) \tag{41.8}$$

You can create a parallel combination with the sum operator, +, in Python:

H = H1 + H2

Example 41.4 *Parallel combination of transfer functions*

Given the transfer functions, $H_1(s)$ and $H_2(s)$, as in Example 41.3.

Manual calculation of their parallel combination gives[4]:

$$H(s) = \frac{2}{s} + \frac{3}{4s+1} = \frac{2(4s+1)+3s}{s(4s+1)} = \frac{11s+2}{4s^2+s}$$

Program 41.4 shows how the calculations can be done with the control.parallel() function.

http://techteach.no/control/python/parallel_tf.py

Listing 41.4: parallel_tf.py

```
import control

s = control.tf('s')

K1 = 2
K2 = 3
T = 4

H1 = K1/s
H2 = K2/(T*s + 1)
```

[4] For simplicity, I insert here the numbers directly instead of the symbolic parameters, but in general I recommend using symbolic parameters.

```
H = H1 + H2

print('H =', H)
```

The result of the code above as shown in the console is:

```
H =
  11 s + 2
------------
4 s^2 + s
```

[End of Example 41.4]

Alternative: control.parallell()

As an alternative to using the + operator, you can use the control.parallell() function:

H = control.parallel(H1, H2)

41.2.2.3 Feedback combination

Figure 41.4 illustrates a feedback combination of two transfer functions.

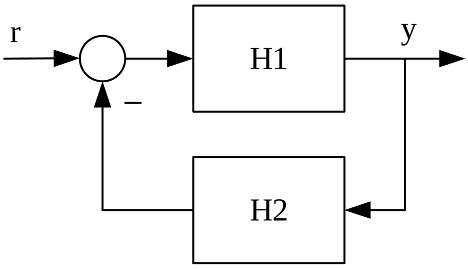


Figure 41.4: A feedback combination of two transfer functions, $H_1(s)$ and $H_2(s)$.

The resulting transfer function, from r (reference) to y, which can be denoted the closed loop transfer function, can be calculated from the following expression defining y (for simplicity, I drop the argument s here):

$$y = H_1 \cdot (r - H_2 y) = H_1 r - H_1 H_2 y$$

which gives

$$y = \frac{H_1}{1 + H_1 H_2} r$$

Thus, the resulting transfer function is

$$\frac{y(s)}{r(s)} = T(s) = \frac{H_1(s)}{1 + H_1(s) H_2(s)} \tag{41.9}$$

The transfer function (41.9) can be derived with following Python code with ordinary arithmetic operators:

T = H1/(1 + H1*H2)

where it is assumed that the transfer functions H1 and H2 has already been defined.

A note about (non)minimal transfer functions

Creating transfer functions using arithmetic operators in Python may produce a non-minimal transfer function, which means that there are one or more common factors in the numerator and denominator. To obtain a minimal transfer function, i.e. to remove common factors, you can use the control.minreal() function:

T_min = control.minreal(T_nonmin)

Example 41.5 *Transfer function of negative feedback combination*

Given a negative feedback loop with the following open loop transfer function:

$$L(s) = \frac{2}{s} \tag{41.10}$$

Manual calculation of the closed loop transfer function $T(s)$ based on (41.12) gives

$$T(s) = \frac{L(s)}{1 + L(s)} = \frac{\frac{2}{s}}{1 + \frac{2}{s}} = \frac{2}{s + 2} \tag{41.11}$$

Program 41.5 shows how the calculations can be implemented with Python code with arithmetic operators. The program include code for obtaining a minimal transfer function. Both the minimal transfer function (T_min) and the non-minimal transfer function (T_nonmin) are presented.

http://techteach.no/control/python/feedback_tf.py

Listing 41.5: feedback_tf.py

```
import control

s = control.tf('s')

L = 2/s
T_nonmin = L/(1 + L)
T_min = control.minreal(T_nonmin)

print('T_min =', T_min)
print('T_nonmin =', T_nonmin)
```

The result:

```
T_min =
  2
---------
s + 2
T_nonmin =
    2 s
---------
s^2 + 2 s
```

[End of Example 41.5]

Alternative: control.feedback()

As an alternative to using the ordinary arithmetic operators to derive the transfer function of a feedback combination, you can use the control.feedback() function:

```
H = control.feedback(H1, H2, sign=−1)
```

where negative feedback is assumed. You may drop the argument sign = −1 if there is negative feedback since negative feedback is the default setting.

You *must* use sign = 1 if there is a positive feedback instead of a negative feedback in Figure 41.4.

In most cases – at least in feedback control systems – a negative feedback with $H_2(s) = 1$ in the feedback path is assumed. Then, $H_1()$ is the open loop transfer function, $L(s)$, and (41.9) becomes

$$\frac{y(s)}{r(s)} = H(s) = \frac{L(s)}{1 + L(s)} \tag{41.12}$$

In such cases, you can use this code:

```
H = control.feedback(L, 1)
```

$L(s)$ may be the series combination (i.e. the product) of the controller, the process, the sensor, and the measurement filter:

$$L(s) = C(s) \cdot P(s) \cdot S(s) \cdot F(s) \tag{41.13}$$

Series combination of transfer functions is described in Section 41.2.2.1.

41.2.3 How to get the numerator and denominator of a transfer function

You can get (read) the numerator coefficients and denominator coefficients of a transfer function, say H, with the control.tfdata() function:

(num_list, den_list) = control.tfdata(H)

where num_list and den_list are *lists* (not arrays) containing the coefficients.

To convert the lists to arrays, you can use the np.array() function:

num_array = np.array(num_list)

and

den_array = np.array(den_list)

Example 41.6 *Getting the numerator and denominator of a transfer function*

See Program 41.6.

http://techteach.no/control/python/get_tf_num_den.py

Listing 41.6: get_tf_num_den.py

```
import numpy as np
import control

# %% Creating a transfer function:

num = np.array([2])
den = np.array([5, 1])
H = control.tf(num, den)

# Alternatively, using the Laplace variable, s:
# s = control.tf('s')
# H = 2/(5*s + 1)

# %% Getting the num and den coeffs as lists and then as arrays:

(num_list, den_list) = control.tfdata(H)
num_array = np.array(num_list)
den_array = np.array(den_list)

# %% Displaying the num and den arrays:

print('num_array =', num_array)
print('den_array =', den_array)
```

The result:

```
num_array = [[[2]]]
den_array = [[[5 1]]]
```

To “get rid of” the two inner pairs of square brackets, i.e. to reduce the dimensions of the arrays:

```
num_array = num_array[0,0,:]
den_array = den_array[0,0,:]
```

producing:

```
[2]
[5 1]
```

[End of Example 41.6]

41.2.4 Simulation with transfer functions

The function control.forced_response() is a function for simulation with transfer function and state space models. Here, we focus on simulation with transfer functions.

control.forced_response() can simulated with any user-defined input signal. Some alternative simulation functions assuming special input signals are:

- control.step_response()
- control.impulse_response()
- control.initial_response()

control.forced_response() may be used in any of these cases. Therefore, I limit the presentation in this document to the control.forced_response() function.

The syntax of control.forced_response() is:

(t, y) = control.forced_response(sys, t, u)

where:

- Input arguments:
  - sys is the transfer function to be used in the simulation.
  - t is the user-defined array of points of simulation time.
  - u is the user-defined array of values of the input signal of same length at the simulation time array.
- Output (return) arguments:
  - t is the returned array of time – the same as the input argument.
  - y is the returned array of output values.

To plot the simulated output (y above), and maybe the input (u above), you can use the plotting function in the matplotlib.pyplot module which requires import of this module. The common way to import the module is:

import matplotlib.pyplot as plt

Example 41.7 *Simulation with a transfer function*

We will simulate the response of the transfer function

$$\frac{Y(s)}{U(s)} = \frac{2}{5s+1}$$

with the following conditions:

- Input u is a step of amplitude 4, with step time $t = 0$.
- Simulation start time is t0 = 0 sec.
- Simulation stop time is t1 = 20 sec.
- Simulation time step, or sampling time, is dt = 0.01 s.
- Initial state is 0.

Program 41.7 implements this simulation.

http://techteach.no/control/python/sim_tf.py

Listing 41.7: sim_tf.py

```
"""
Sim of time constant system with forced_response() of Python Control package
Finn Aakre Haugen, TechTeach. finn@techteach.no
2022 12 22
"""

# %% Import:
import numpy as np
import control
import matplotlib.pyplot as plt

# %% Creating model:

s = control.tf(’s’)

H = 2/(5*s + 1)

# %% Defining signals:
t0 = 0
t1 = 20
```

```
dt = 0.01
nt = int(t1/dt) + 1 # Number of points of sim time
t = np.linspace(t0, t1, nt)
u = 2*np.ones(nt)

# %% Simulation:
(t, y) = control.forced_response(H, t, u)

# %% Plotting:
plt.close('all')
plt.figure(1)

plt.subplot(2, 1, 1)
plt.plot(t, y, 'blue', label='y')
#plt.xlabel('t [s]')
plt.grid()
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(t, u, 'green', label='u')
plt.xlabel('t [s]')
plt.grid()
plt.legend()

# plt.savefig('sim_tf.pdf')
plt.show()
```

Figure 41.5 shows plots of the output y and the input u.

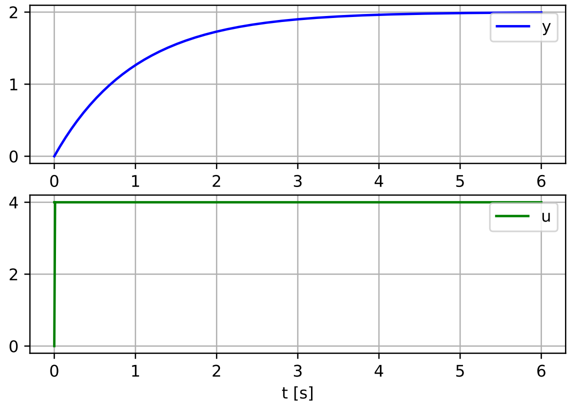


Figure 41.5: Plots of the output y and the input u.

[End of Example 41.7]

41.2.5 Poles and zeros of transfer functions

Poles and zeros of a transfer function, H, can be calculated and plotted in a cartesian diagram with

(p, z) = control.pzmap(H)

Example 41.8 *Poles and zeros of a transfer function*

Given the following transfer function:

$$H(s) = \frac{s+2}{s^2+4}$$

Manual calculations gives:

- Poles:

$$p_{1,2} = \pm 2j$$

- Zero:

$$z = -2$$

Program 41.8 calculates the poles and the zero and plots them with the control.pzmap() function. The plt.savefig() function is used to generate a pdf file of the diagram.

http://techteach.no/control/python/poles_tf.py

Listing 41.8: poles_tf.py

```
import control
import matplotlib.pyplot as plt

s = control.tf(’s’)
H = (s + 2)/(s**2 + 4)

(p, z) = control.pzmap(H)
plt.grid()

print(’poles =’, p)
print(’zeros =’, z)

plt.savefig(’poles_zeros.pdf’)
```

The result:

```
poles = [-0.+2.j 0.-2.j]
zeros = [-2.]
```

Figure 41.6 shows the pole-zero plot.

[End of Example 41.8]

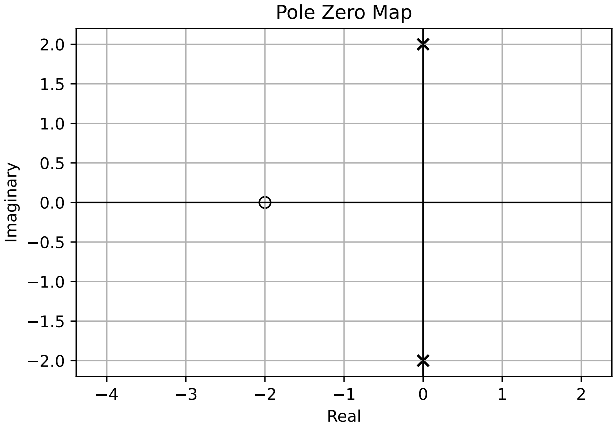


Figure 41.6: Pole-zero plot.

41.2.6 The Padé-approximation of a time delay

The transfer function of a time delay is

$$e^{-T_d s} \tag{41.14}$$

where T_d is the time delay. In the Python Control Package, there is no function to define this s-transfer function (while this is straightforward for z-transfer functions, cf. Ch. 41.5.1.2). However, you can use the control.pade() function to generate a Padé-approximation of the time delay (41.14).

Once you have a Padé-approximation of the time delay, you may use the control.series() function to combine it with the transfer function having no time delay:

$$H_{\text{with\_delay}}(s) = H_{\text{without\_delay}}(s) \cdot H_{\text{pade}}(s) \tag{41.15}$$

Example 41.9 *Padé-approximation*

Given the following transfer function with time constant of 10 s and *no* time delay:

$$H_{\text{without\_delay}}(s) = \frac{1}{10s+1} \tag{41.16}$$

Assume that this transfer function is combined in series with a transfer function, $H_{\text{pade}}(s)$, of a 10th order Padé-apprioximation representing a time delay of 5 s. The resulting transfer function is:

$$H_{\text{with\_delay}}(s) = H_{\text{without\_delay}}(s) \cdot H_{\text{pade}}(s) = \frac{1}{10s+1} \cdot H_{\text{pade}}(s) \tag{41.17}$$

Program 41.9 generates these transfer functions and simulated a step response of $H_{\text{with\_delay}}(s)$.

http://techteach.no/control/python/pade_approx.py

Listing 41.9: pade_approx.py

```
import numpy as np
import control
import matplotlib.pyplot as plt

# %% Generating transfer function of Pade approx:

T_delay = 5
n_pade = 10
(num_pade, den_pade) = control.pade(T_delay, n_pade)
H_pade = control.tf(num_pade, den_pade)

# %% Generating transfer function without time delay:

s = control.tf('s')
K = 1
T = 10
H_without_delay = K/(T*s + 1)

# %% Generating transfer function with time delay:

H_with_delay = H_without_delay*H_pade

# %% Simulation of step response:
t = np.linspace(0, 40, 100)
(t, y) = control.step_response(H_with_delay, t)

# %% Plotting

plt.plot(t, y, label='y')
plt.legend()
plt.title('Step response of time delay with Pade-approx.')
plt.xlabel('t [s]')
plt.grid()

# plt.savefig('pade_approx.pdf')
plt.show()
```

Figure 41.7 shows the step response of $H_{\text{with\_delay}}(s)$.

[End of Example 41.9]

41.3 Frequency response

41.3.1 Frequency response of transfer functions

The function control.bode_plot() generates frequency response data in terms of magnitude and phase. The function may also plot the data in a Bode diagram. However, in the

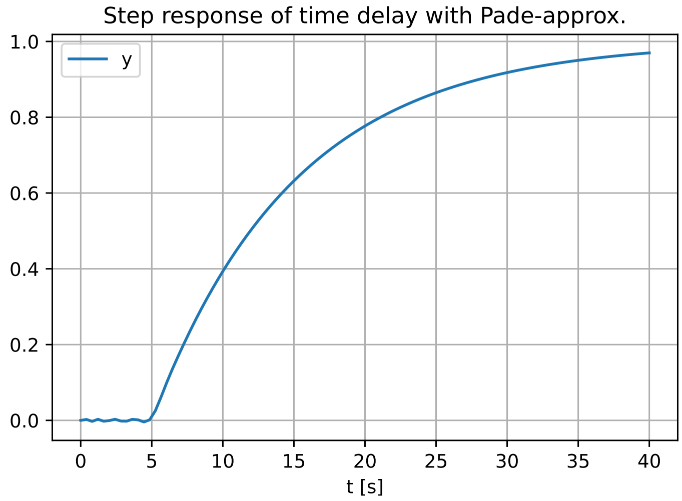


Figure 41.7: Step response of $H_{\text{with\_delay}}(s)$ where the time delay is approximated with a Padé-approximation.

following example, I have instead used the plt.plot() function to plot the data as this gives more freedom to configure the plot.

Example 41.10 *Frequency response*

A first order lowpass filter has the following transfer function:

$$H(s) = \frac{1}{\frac{s}{\omega_b} + 1} \tag{41.18}$$

where $\omega_b = 1$ rad/s, which is the bandwidth.

Program 41.10 generates and plots frequency response of $H(s)$ in terms of magnitude and phase.

http://techteach.no/control/python/bode_plot_lowpass_filter.py

Listing 41.10: bode_plot_lowpass_filter.py

```
import numpy as np
import control
import matplotlib.pyplot as plt

# %% Creating transfer function:

s = control.tf(’s’)
wb = 1  # Bandwidth [rad/s]
H = 1/((1/wb)*s + 1)

# %% Generating Bode plot:
```

```
w0 = 0.1
w1 = 10
dw = 0.001
nw = int((w1-w0)/dw) + 1 # Number of points of freq
w = np.linspace(w0, w1, nw)

(mag, phase_rad, w) = control.bode_plot(H, w)

# %% Plotting:

plt.close(’all’)
plt.figure(1, figsize=(12, 9))

plt.subplot(2, 1, 1)
plt.plot(np.log10(w), mag, ’blue’)
#plt.xlabel(’w [rad/s]’)
plt.grid()
plt.legend(labels=(’Amplitude gain’,))

plt.subplot(2, 1, 2)
plt.plot(np.log10(w), phase_rad*180/np.pi, ’green’)
plt.xlabel(’w [rad/s]’)
plt.grid()
plt.legend(labels=(’Phase shift [deg]’,))

# plt.savefig(’bode_plot_filter.pdf’)
plt.show()
```

Figure 41.8 shows the Bode plot. In the plot we can that bandwidth is indeed 1 rad/s (which is at 0 = log10(1) rad/s in the figure).

[End of Example 41.10]

41.3.2 Frequency response and stability analysis of feedback loops

Figure 41.9 shows a feedback loop with its loop transfer function, $L(s)$.

control.bode_plot()

We can use the function control.bode_plot() to calculate the magnitude and phase of L, and to plot the Bode plot of L.

The syntax of control.bode_plot() is:

$$\text{(mag, phase\_rad, w)} = \text{control.bode\_plot()}$$

Several input arguments can be set, cf. Example 41.11.

In addition to calculating the three return arguments above, control.bode_plot() can show the following analysis values in the plot:

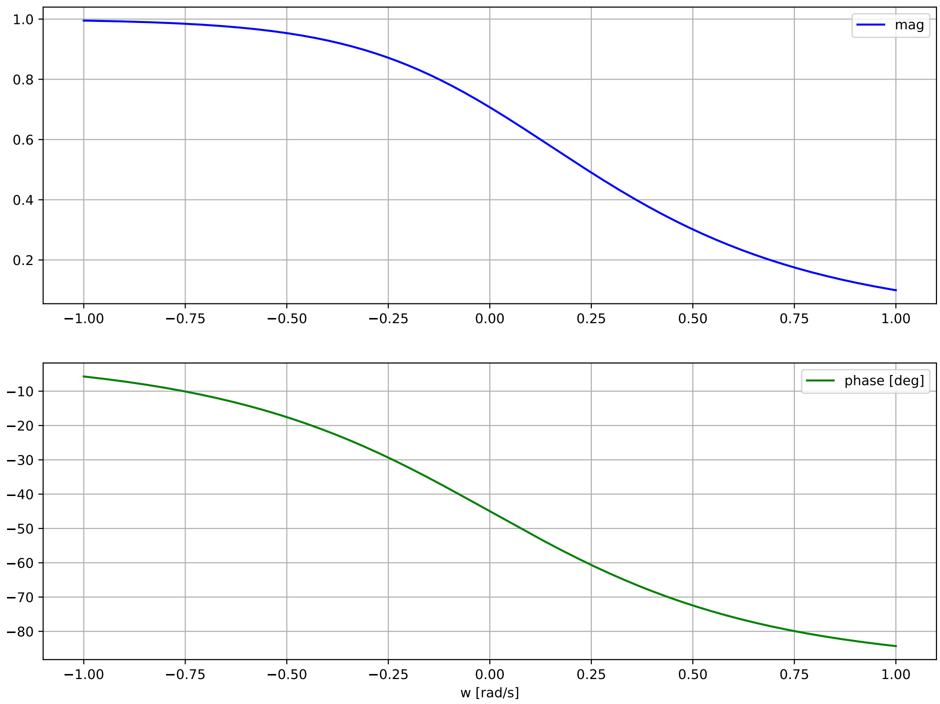


Figure 41.8: Bode plot.

- The amplitude cross-over frequency, ω_b [rad/s], which is also often regarded as the bandwidth of the feedback system.
- The phase cross-over frequency, ω_{180} [rad/s].
- The gain margin, GM, which is found at $\omega_{180} \equiv \omega_g$ [rad/s] (g for gain margin).
- The phase margin, PM, which is found at $\omega_b \equiv \omega_p$ [rad/s] (p for phase margin).

control.margin()

The control.bode_plot() does *not* return the above four analysis values to the workspace (although it shows them in the Bode plot). Fortunately, we can use the control.margin() function to calculate these analysis values. control.margin() can be used as follows:

$$(\text{GM, PM, wg, wp}) = \text{control.margin(L)}$$

where L is the loop transfer function, and the four return arguments are as in the list above. Note that GM has unit one; *not* dB, and that PM is in degrees.

Example 41.11 demonstrates the use of control.bode_plot() and control.margin().

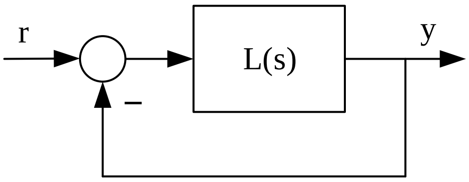


Figure 41.9: A feedback loop with its loop transfer function, $L(s)$

Example 41.11 *Frequency response*

Given a control loop where the process to be controlled has the following transfer function (an integrator and two time constants in series):

$$P(s) = \frac{1}{(s+1)^2 s}$$

The controller is a P controller:

$$C(s) = K_c$$

where $K_c = 2$ is the controller gain.

The loop transfer function becomes:

$$L(s) = P(s) \cdot C(s) = \frac{K_c}{(s+1)^2 s} = \frac{K_c}{s^3 + 2s + s} \tag{41.19}$$

Program 41.11 generates and plots frequency response of $H(s)$, and shows the stability margins and the cross-over frequencies. The control.minreal() function is used to ensure $L(s)$ is a minimum transfer function, cf. Section 41.2.2.3.

http://techteach.no/control/python/bode_plot_with_stab_margins.py

Listing 41.11: bode_plot_with_stab_margins.py

```
import numpy as np
import control
import matplotlib.pyplot as plt

# %% Creating the loop transfer function:

s = control.tf('s')

Kp = 1
C = Kp
P = 1/(s**3 + 2*s**2 + s)
L = C*P
L = control.minreal(L)  # To obtain minimum transf func

# %% Frequencies:

w0 = 0.1
```

```
w1 = 10
dw = 0.001
nw = int((w1-w0)/dw) + 1 # Number of points of freq
w = np.linspace(w0, w1, nw)

# %% Plotting:

plt.close('all')
plt.figure(1, figsize=(12, 9))
(mag, phase_rad, w) = control.bode_plot(L,
                                        w,
                                        dB=True,
                                        deg=True,
                                        margins=True)
plt.grid()

# %% Calculating stability margins and crossover frequencies:

(GM, PM, wg, wp) = control.margin(L)

# %% Printing:

print(f'GM [1 (not dB)] = {GM:.2f}')
print(f'PM [deg] = {PM:.2f}')
print(f'wg [rad/s] = {wg:.2f}')
print(f'wp [rad/s] = {wp:.2f}')

# %% Generating pdf file of the plotting figure:

plt.savefig('bode_with_stab_margins.pdf')
```

Below are the results of control.margin() as shown in the console. The values are the same as shown in the Bode plot in Figure 41.10 (2 dB $\approx$ 6).

```
GM [1 (not dB)] = 2.00
PM [deg] = 21.39
wg [rad/s] = 1.00
wp [rad/s] = 0.68
```

[End of Example 41.11]

41.4 State space models

41.4.1 How to create state space models

The function control.ss() creates a *linear* state space model with the following form:

$$\dot{x} = Ax + Bu \tag{41.20}$$

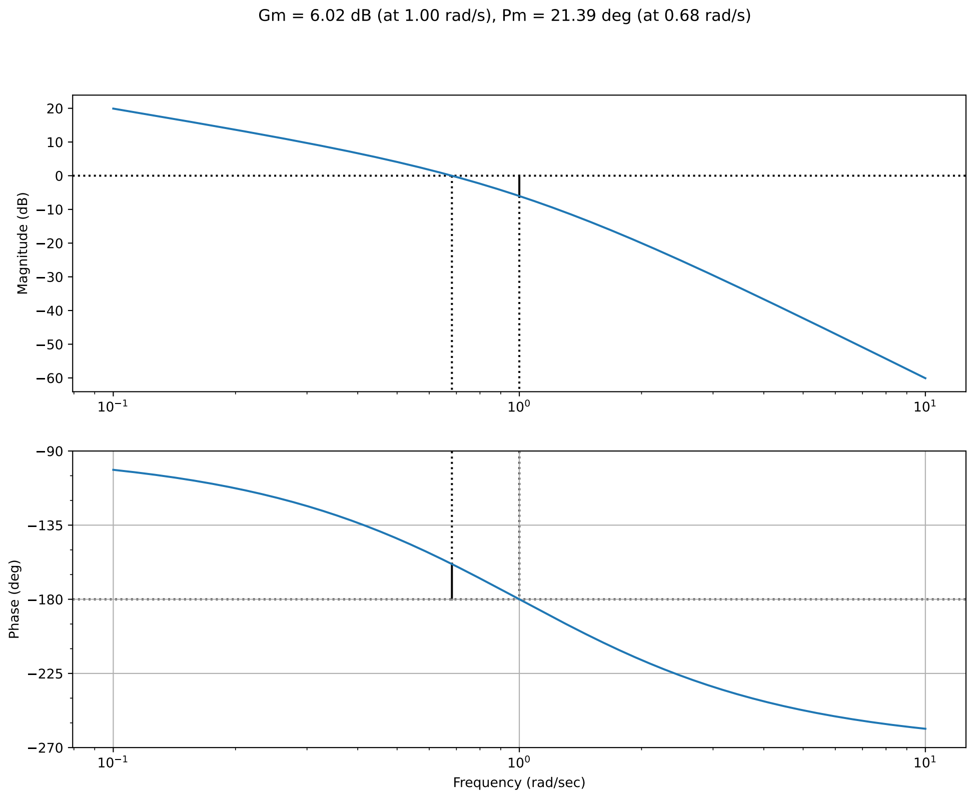


Figure 41.10: Bode plot including the stability margins and the crossover frequencies.

$$y = Cx + Bu \tag{41.21}$$

where A, B, C, D are the model matrices.

The syntax of control.ss() is:

S = control.ss(A, B, C, D)

where S is the resulting state space model, and the matrices A, B, C, D are in the form of 2D arrays in Python. (Actually, they may be of the list data type, but I recommend using arrays, cf. Section 41.1.5.)

Example 41.12 *Creating a state space model*

Figure 41.11 shows a mass-spring-damper-system.

z is position. F is applied force. D is damping constant. K is spring constant. Newton's 2. Law gives the following mathematical model:

$$m\ddot{z}(t) = F(t) - D\dot{z}(t) - Kz(t) \tag{41.22}$$

Let us define the following state variables:

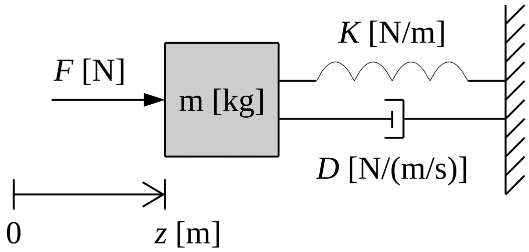


Figure 41.11: Mass-spring-damper system.

- Position:

$$x_1 = z$$

- Speed:

$$x_2 = \dot{z} = \dot{x}_1$$

Let us define the position x_1 as the output variable:

$$y = x_1$$

Eq. (41.22) can now be expressed with the following equivalent state space model:

$$\underbrace{\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\frac{K}{m} & -\frac{D}{m} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}}_{B} F \tag{41.23}$$

$$y = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \end{bmatrix}}_{D_1} F \tag{41.24}$$

Assume following parameter values:

$$m = 10 \text{ kg}$$

$$k = 4 \text{ N/m}$$

$$d = 2 \text{ N/(m/s)}$$

Program 41.12 creates the above state space model with the control.ss() function.

http://techteach.no/control/python/create_ss.py

Listing 41.12: create_ss.py

```
import numpy as np
import control

# %% Model parameters:

m = 10  # [kg]
k = 4  # [N/m]
d = 2  # [N/(m/s)]

# %% System matrices as 2D arrays:

A = np.array([[0, 1], [-k/m, -d/m]])
B = np.array([[0], [1/m]])
C = np.array([[1, 0]])
D = np.array([[0]])

# %% Creating and printing the state space model:

S = control.ss(A, B, C, D)

print('S =', S)
```

The results as shown in the console of Spyder:

```
A = [[ 0.  1.]
 [-0.4 -0.2]]

B = [[0. ]
 [0.1]]

C = [[1. 0.]]

D = [[0.]]
```

41.4.2 How to get the model matrices of a state space model

You can get (read) the model matrices of a given state space model, say S, with the control.ssdata() function:

(A_list, B_list, C_list, D_list) = control.ssdata(S)

where the matrices are in the form of *lists* (not arrays).

To convert the lists to arrays, you can use the np.array() function, e.g.

A_array = np.array(A_list)

Example 41.13 *Getting the model matrices of a given state space model*

Program 41.13 creates a state space model and gets its matrices with the control.ssdata() function.

http://techteach.no/control/python/get_ss_matrices.py

Listing 41.13: get_ss_matrices.py

```
import numpy as np
import control

# %% Creating a state space model:
A = np.array([[0, 1], [2, 3]])
B = np.array([[4], [5]])
C = np.array([[6, 7]])
D = np.array([[8]])

S = control.ss(A, B, C, D)

# %% Getting the model matrices as lists and then as arrays:
(A_list, B_list, C_list, D_list) = control.ssdata(S)
A_array = np.array(A_list)
B_array = np.array(B_list)
C_array = np.array(C_list)
D_array = np.array(D_list)

# %% Displaying the matrices as arrays:
print(’A_array =’, A_array)
print(’B_array =’, B_array)
print(’C_array =’, C_array)
print(’D_array =’, D_array)
```

The results as shown in the console:

```
A_array = [[0. 1.] [2. 3.]]
B_array = [[4.] [5.]]
C_array = [[6. 7.]]
D_array = [[8.]]
```

[End of Example 41.13]

41.4.3 Simulation with state space models

Simulation with state space models can be done with the control.forced_response() function:

(t, y, x) = control.forced_response(sys, t, u, x0, return_x=True)

where sys is the state space model, cf. Section 41.4.1.

Example 41.14 *Simulation with a state space model*

The program shown below runs a simulation with the state space model presented in Example 41.12 with the following conditions:

- Force (input signal) F is a step of amplitude 10 N, with step time $t = 0$.
- Simulation start time: t0 = 0 s.
- Simulation stop time: t1 = 50 s.
- Simulation time step, or sampling time: dt = 0.01 s.
- Initial states: $x_{1,0}$= 1 m, $x_{2,0}$ = 0 m/s.

Program 41.14 implements the simulation.

http://techteach.no/control/python/sim_ss.py

Listing 41.14: sim_ss.py

```
# %% Import:

import numpy as np
import control
import matplotlib.pyplot as plt

# %% Model parameters:

m = 10  # [kg]
k = 4  # [N/m]
d = 2  # [N/(m/s)]

# %% System matrices as 2D arrays:

A = np.array([[0, 1], [-k/m, -d/m]])
B = np.array([[0], [1/m]])
C = np.array([[1, 0]])
D = np.array([[0]])

# %% Creating the state space model:

S = control.ss(A, B, C, D)

# %% Defining signals:

t0 = 0  # [s]
t1 = 50  # [s]
dt = 0.01  # [s]
nt = int(t1/dt) + 1 # Number of points of sim time
t = np.linspace(t0, t1, nt)
F = 10*np.ones(nt)  # [N]
```

```
# %% Initial state:

x1_0 = 1  # [m]
x2_0 = 0  # [m/s]
x0 = np.array([x1_0, x2_0])

# %% Simulation:

(t, y, x) = control.forced_response(S, t, F, x0,
                                    return_x=True)

# %% Extracting individual states:

x1 = x[0,:]
x2 = x[1,:]

# %% Plotting:

plt.close(’all’)
plt.figure(1, figsize=(12, 9))

plt.subplot(3, 1, 1)
plt.plot(t, x1, ’blue’)
plt.grid()
plt.legend(labels=(’x1 [m]’,))

plt.subplot(3, 1, 2)
plt.plot(t, x2, ’green’)
plt.grid()
plt.legend(labels=(’x2 [m/s]’,))

plt.subplot(3, 1, 3)
plt.plot(t, F, ’red’)
plt.grid()
plt.legend(labels=(’F [N]’,))
plt.xlabel(’t [s]’)

# plt.savefig(’sim_ss.pdf’)
plt.show()
```

Figure 41.12 shows the simulated signals.

[End of Example 41.14]

41.4.4 From state space model to transfer function

The function control.ss2tf() derives a transfer function from a given state space model. The syntax is:

$$\mathrm{H} = \mathrm{control.ss2tf(S)}$$

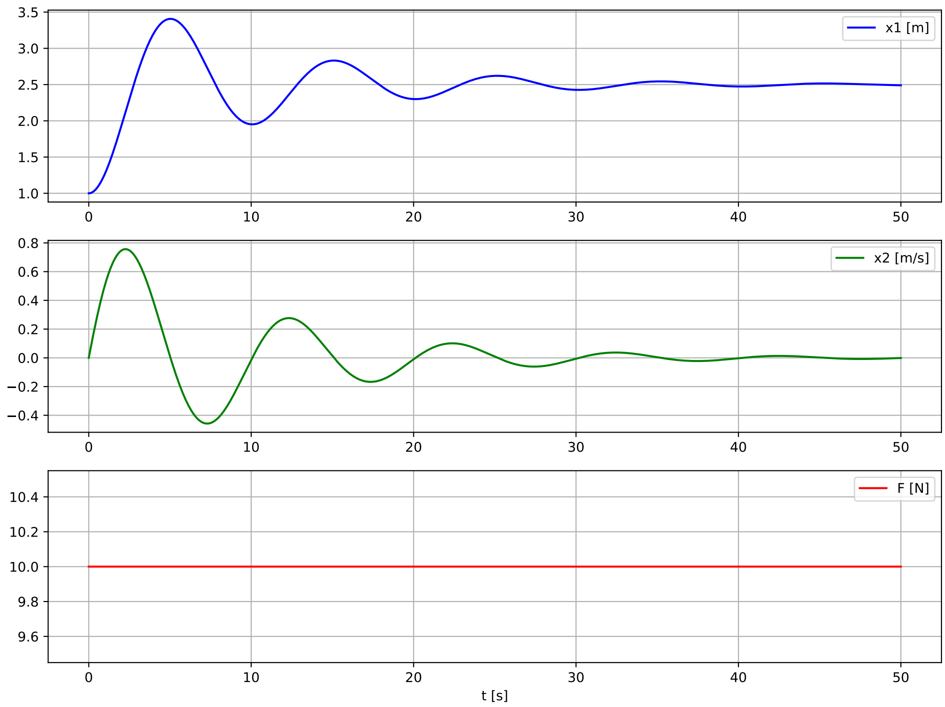


Figure 41.12: Plots of the simulated signals of the mass-spring-damper system.

where H is the transfer function and S is the state space model.

Example 41.15 *From state space model to transfer function*

In Example 41.12 a state space model of a mass-spring-damper system is created with the control.ss() function. The program shown below derives the following two transfer functions from this model:

- The transfer function, H_1, from force F to position x_1. To obtain H_1, the output matrix use is set as
$$C = [1,\ 0]$$
- The transfer function, H_2, from force F to position x_2. To obtain H_2, the output matrix is set as
$$C = [0,\ 1]$$

Program 41.15 derives the two transfer functions from a state space model.

http://techteach.no/control/python/from_ss_to_tf.py

Listing 41.15: from_ss_to_tf.py

```
import numpy as np
import control

# %% Model params:

m = 10  # [kg]
k = 4  # [N/m]
d = 2  # [N/(m/s)]

# %% System matrices as 2D arrays:

A = np.array([[0, 1], [-k/m, -d/m]])
B = np.array([[0], [1/m]])
D = np.array([[0]])

# %% Creating the state space model with x1 as output:

C1 = np.array([[1, 0]])
S1 = control.ss(A, B, C1, D)

# %% Deriving transfer function H1 from S1:

H1 = control.ss2tf(S1)

# %% Displaying H1:

print(’H1 =’, H1)

# %% Creating the state space model with x2 as output:

C2 = np.array([[0, 1]])
S2 = control.ss(A, B, C2, D)

# %% Deriving transfer function H2 from S2:

H2 = control.ss2tf(S2)

# %% Displaying H1:

print(’H2 =’, H2)
```

The result of the code above, as shown in the console of Spyder, is shown below. The very small numbers – virtually zeros – in the numerators of H1 and H2 are due to numerical inaccuracies in the control.ss2tf() function.

```
H1 =
    0.1
---------------------
s^2 + 0.2 s + 0.4

H2 =
0.1 s + 1.665e-16
---------------------
s^2 + 0.2 s + 0.4
```

[End of Example 41.15]

41.5 Discrete-time models

41.5.1 Transfer functions

41.5.1.1 Introduction

Many functions in the Python Control Package are used in the same way for discrete-time transfer functions, or z-transfer functions, as for continuous-time transfer function, or s-transfer function, except that for z-transfer functions, you must include the sampling time Ts as an additional parameter. For example, to create a z-transfer function, the control.tf() is used in this way:

H_d = control.tf(num_d, den_d, Ts)

where Ts is the sampling time. H_d is the resulting z-transfer function.

Thus, the descriptions in Ch. 41.2 gives you a basis for using these functions for z-transfer functions as well. Therefore, the descriptions are not repeated here. Still there are some specialities related to z-transfer function, and they are presented in the subsequent sections.

41.5.1.2 How to create transfer functions

The control.tf() function is used to create z-transfer functions with the following syntax:

H = control.tf(num, den, Ts)

where H is the resulting transfer function (object). num (representing the numerator) and den (representing the denominator) are arrays where the elements are the coefficients of the z-polynomials of the numerator and denominator, respectively, in descending order from left to right, with positive exponentials of z. Ts is the sampling time (time step).

Note that control.tf() assumes *positive* exponents of z. Here is one example of such a transfer function:

$$H(z) = \frac{0.1z}{z-1} \tag{41.25}$$

(which is used in Example 41.16). However, in e.g. signal processing, we may see negative exponents in transfer functions. $H(z)$ given by (41.25) and written in terms of negative exponents of z, are:

$$H(z) = \frac{0.1}{1-z^{-1}} \tag{41.26}$$

(41.25) and () are equivalent. But, in the Python Control Package, we must use only positive exponents of z in transfer functions.

Example 41.16 *Creating a z-transfer function*

Given the following transfer function[5]:

$$H(z) = \frac{0.1z}{z-1} \tag{41.27}$$

Program 41.16 creates $H(z)$. The code print('H(z) = ', H) is used to present the transfer function in the console (of Spyder).

http://techteach.no/control/python/create_tf_z.py

Listing 41.16: create_tf_z.py

```
import numpy as np
import control

# %% Creating the z-transfer function:

Ts = 0.1
num = np.array([0.1, 0])
den = np.array([1, -1])
H = control.tf(num, den, Ts)

print('H(z) =', H)
```

The result as shown in the console:

```
H(z) =
  0.1 z
--------
z - 1

dt = 0.1
```

[End of Example 41.16]

[5] This is the transfer function of an integrator based on the Euler Backward method of discretization: $y_k = y_{k-1} + Ts \cdot u_k$with sampling time $Ts = 0.1$ s.

41.5.1.3 Discretizing an s-transfer function

The control.sample_system() function can be used to discretize given continuous-time models, including s-transfer functions:

sys_disc = control.sample_system(sys_cont, Ts, method='zoh')

where:

- sys_cont is the continuous-time model – a transfer function, or a state space model.
- Ts is the sampling time.
- The discretization method is 'zoh' (zero order hold) by default, but you can alternatively use 'matched' or 'tustin'. (No other methods are supported.)
- sys_disc is the resulting discrete-time model – a transfer function, or a state space model.

Example 41.17 *Discretizing an s-transfer function*

Given the following s-transfer function:

$$H_c(s) = \frac{3}{2s+1} \tag{41.28}$$

Program 41.17 discretizes this transfer function using the zoh method with sampling time 0.1 s.

http://techteach.no/control/python/discretize_tf.py

Listing 41.17: discretize_tf.py

```
import control

# %% Creating the s-transfer function:

s = control.tf('s')
H_cont = 3/(2*s + 1)

# %% Discretizing:

Ts = 0.1  # Time step [s]
H_disc = control.sample_system(H_cont, Ts, method='zoh')

print('H_disc(z) =', H_disc)
```

The result as shown in the console:

```
H_disc(z) =
  0.1463
-----------
z - 0.9512

dt = 0.1
```

[End of Example 41.17]

41.5.1.4 Exact representation of a time delay with a z-transfer function

In Section 41.2.6 we saw how to use the control.pade() function to generate a transfer function which is an Padé-approximation of the true transfer function of the time delay, $e^{-T_d s}$. As alternative to the Padé-approximation, you can generate an exact representation of the time delay in terms of a z-transfer function.

The z-transfer function of a time delay is:

$$H_d(z) = \frac{1}{z^{n_d}} \tag{41.29}$$

where

$$n_d = \frac{T_d}{T_s} \tag{41.30}$$

Example 41.18 *Creating a z-transfer function of a time delay*

Assume the time delay is

$$T_d = 5\,\mathrm{s}$$

and the sampling time is

$$T_s = 0.1\,\mathrm{s}$$

So, the transfer function of the time delay becomes

$$H_{\text{delay}}(z) = \frac{1}{z^{n_d}}$$

with

$$n_d = \frac{T_d}{T_s} = \frac{5}{0.1} = 50$$

Python program 41.18 creates $H_{\text{delay}}(z)$, which represents this time delay exactly. The program also simulates the step response of $H_{\text{delay}}(z)$.[6]

http://techteach.no/control/python/time_delay_hz.py

[6] For some reason, the returned simulation array, y, becomes a 2D array. I turn it into a 1D array with y = y[0,:] for the plotting.

Listing 41.18: time_delay_hz.py

```
import numpy as np
import control
import matplotlib.pyplot as plt

# %% Generating a z-transfer function of a time delay:

Ts = 0.1
Td = 5
nd = int(Td/Ts)
denom_tf = np.append([1], np.zeros(nd))
H_delay = control.tf([1], denom_tf, Ts)

# %% Displaying the z-transfer function:

print(’H_delay(z) =’, H_delay)

# %% Sim of step response of time delay transfer function:

t = np.arange(0, 10+Ts, Ts)
(t, y) = control.step_response(H_delay, t)
plt.plot(t, y)
plt.xlabel(’t [s]’)
plt.grid()

plt.savefig(’step_response_hz_time_delay.pdf’)
```

The result as shown in the console:

```
H_delay(z) =
  1
--------
z^50

dt = 0.1
```

Figure 41.13 shows the step response of $H_{\text{delay}}(z)$.

[End of Example 41.18]

41.5.2 Frequency response

Frequency response analysis of z-transfer functions is accomplished with the same functions as for s-transfer function. Therefore, I assume it is sufficient that I refer you to Ch. 41.3.

However, note the following comment in the manual of the Python Control Package: “If a discrete time model is given, the frequency response is plotted along the upper branch of the unit circle, using the mapping z = exp(j omega dt) where omega ranges from 0 to pi/dt and dt is the discrete timebase. If not timebase is specified (dt = True), dt is set to 1.”

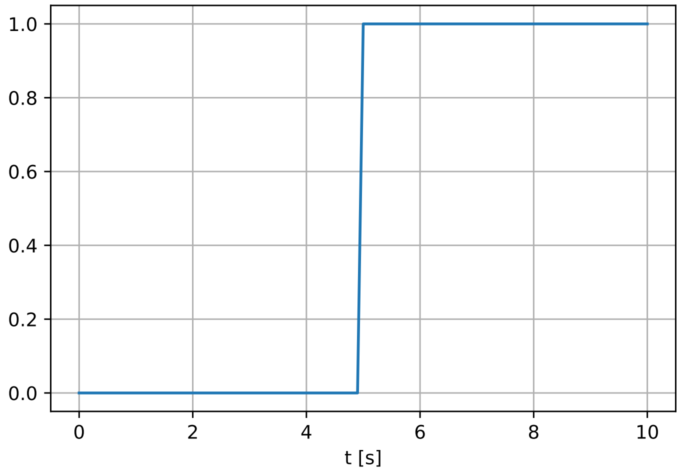


Figure 41.13: The step response of $H_{\mathrm{delay}}(z)$

41.5.3 State space models

In the Python Control Package, discrete-time linear state space models have the following form:

$$x_{k+1} = A_d x_k + B_d u_k \tag{41.31}$$

$$y_k = C_d x_k + B u_{dk} \tag{41.32}$$

where A_d, B_d, C_d, D_d are the model matrices.

Many functions in the package are used in the same way for both discrete-time linear state space models and for continuous-time state space models, except that for discrete-time state space models, you must include the sampling time Ts as an additional parameter. For example, to create a discrete-time state space model, the control.ss() is used in this way:

S_d = control.ss(A_d, B_d, C_d, D_d, Ts)

where Ts is the sampling time. S_d is the resulting discrete time state space model.

Thus, the descriptions in Ch. 41.4 gives you a basis for using these functions for continuous-time state space models as well. Therefore, the descriptions are not repeated here.

Chapter 42

OpenModelica

The free tool OpenModelica can be used to simulate block diagram models.

Figure 42.1 shows a picture of the OpenModelica homepage on:

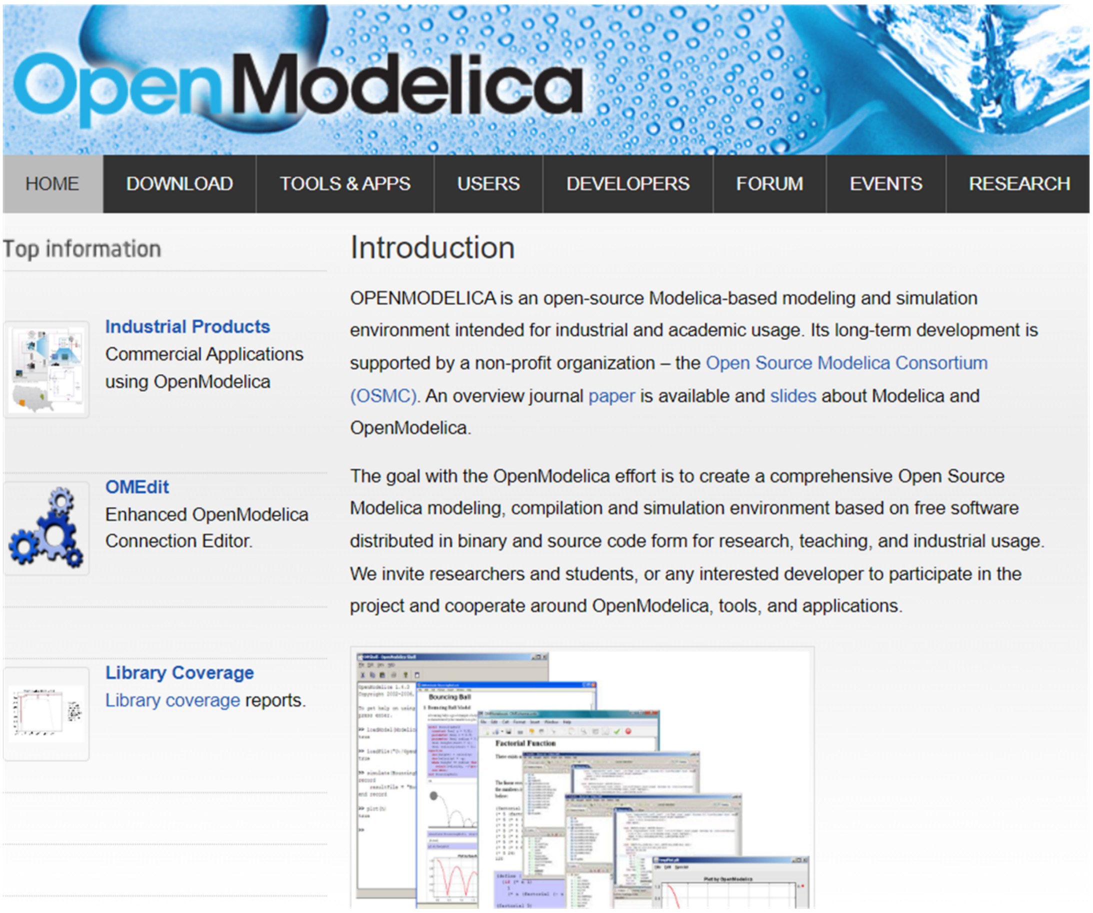
https://openmodelica.org

A video-based tutorial to simulation with OpenModelica is available via the link below. The video shows how to simulate the response in the level due to a step in the feed screw flow of the wood chips tank presented in Ch. 39.2.

https://www.dropbox.com/s/le29x24bkihkwa4/sim_wood_chips_tank.mp4?dl=0

The resulting OpenModelica simulator is available on:

http://techteach.no/control/openmodelica/wood_chips_tank_2023_01_02.mo



Top information

Industrial Products
Commercial Applications using OpenModelica

OMEdit
Enhanced OpenModelica Connection Editor.

Library Coverage
Library coverage reports.

Introduction

OPENMODELICA is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. Its long-term development is supported by a non-profit organization – the Open Source Modelica Consortium (OSMC). An overview journal paper is available and slides about Modelica and OpenModelica.

The goal with the OpenModelica effort is to create a comprehensive Open Source Modelica modeling, compilation and simulation environment based on free software distributed in binary and source code form for research, teaching, and industrial usage. We invite researchers and students, or any interested developer to participate in the project and cooperate around OpenModelica, tools, and applications.

Figure 42.1: The OpenModelica homepage (https://openmodelica.org).

Chapter 43

SimView

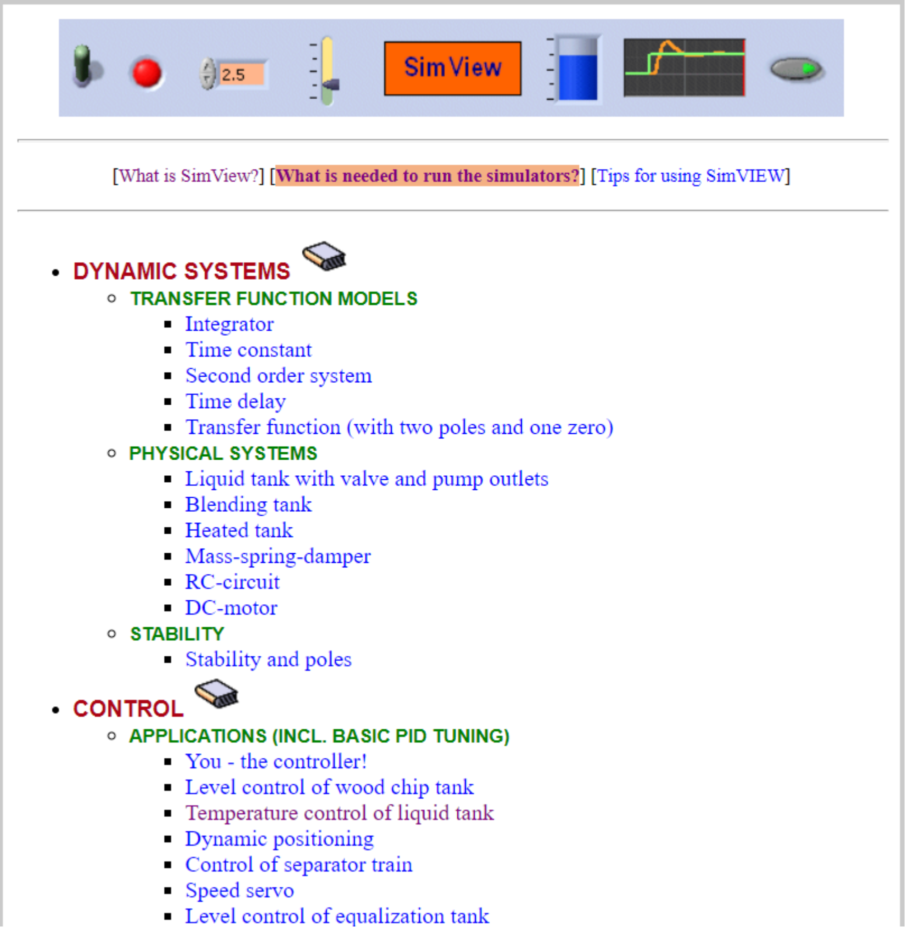
SimView is a set of about 45 freely available simulators in the area of dynamic systems and control systems which I have developed. The address of SimView is:

http://techteach.no/simview

Each simulator is in the form of an executable (an exe file). SimView is developed with LabVIEW, but only the LabVIEW Run-time Engine (version 2017), which is freely available from ni.com, is needed to run the simulators.

The simulators are interactive, and run in real time, or scaled real time, which gives an experience which is quite realistic.

Figure 43.1 shows a part of the home page of SimVIEW.



[What is SimView?] [What is needed to run the simulators?] [Tips for using SimVIEW]

- DYNAMIC SYSTEMS
  - TRANSFER FUNCTION MODELS
    - Integrator
    - Time constant
    - Second order system
    - Time delay
    - Transfer function (with two poles and one zero)
  - PHYSICAL SYSTEMS
    - Liquid tank with valve and pump outlets
    - Blending tank
    - Heated tank
    - Mass-spring-damper
    - RC-circuit
    - DC-motor
  - STABILITY
    - Stability and poles
- CONTROL
  - APPLICATIONS (INCL. BASIC PID TUNING)
    - You - the controller!
    - Level control of wood chip tank
    - Temperature control of liquid tank
    - Dynamic positioning
    - Control of separator train
    - Speed servo
    - Level control of equalization tank

Figure 43.1: SimView home page (http://techteach.no/simview).

Chapter 44

The Laplace transform

44.1 Introduction

The Laplace transform is a mathematical tool which is useful in systems theory. It is the foundation of transfer functions which is a standard model form of dynamic systems. Transfer functions are described in Chapter 8. Furthermore, with the Laplace transform you relatively easily calculate responses in dynamic systems by hand.[1]

In this chapter, I present the Laplace transform at a minimum level. You can find much more information in a mathematics text-book.

Information about the mathematical notation of Laplace transformed variables used in this book is in Appendix 37.

44.2 Definition of the Laplace transform

Given a time-evaluated function $f(t)$ – that is, $f(t)$ is a function of time t. It can be a sinusoid, a ramp, an impulse, a step, a sum of such functions, or any other function of time. The Laplace transform of $f(t)$ can be denoted $F(s)$, and is given by the following integral:

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^\infty e^{-st} f(t)\, dt \tag{44.1}$$

Expressed with words, $f(t)$ is multiplied by the weight function e^{-st}, and the resulting product $e^{-st} f(t)$ is integrated from start time $t = 0$ to end time $t = \infty$. The Laplace transform does not care about any value that $f(t)$ might have at negative values of time t. In other words, you can think of $f(t)$ as being "switched on" at $t = 0$. (The so-called

[1] However, we rarely need to perform manual calculations of the responses. When we need to know the responses, it is in most situations more conventient to obtain them by simulating the system. With the Laplace transform you can calculate responses only for *linear systems*, that is, systems having a model which can be expressed as a linear differential equation.

two-sided Laplace transform is defined also for negative time, but it is not relevant for our applications.)

s is the Laplace variable.[2] $F(s)$ is a function of s. The time t is not a variable in $F(s)$ – it disappeared through the time integration. $F(s)$ will look completely different from $f(t)$, cf. the following example.

Example 44.1 *Laplace transform of a step*

Given the function

$$f(t) = 1 \text{ (for } t \geq 0) \tag{44.2}$$

which is a step of amplitude 1 at time $t = 0$. Using (44.1), its Laplace transform becomes

$$F(s) = \int_0^\infty e^{-st} \cdot 1 \cdot dt = \left[-\frac{1}{s} e^{-st} \right]_{t=0}^{t=\infty} = \frac{1}{s} \tag{44.3}$$

[End of Example 44.1]

Calculating the time function $f(t)$ from its Laplace transform $F(s)$, or in other words: going from the Laplace doman to the time domain, is denoted *inverse Laplace transform.* This can be expressed as

$$f(t) = \mathcal{L}^{-1} \{F(s)\} \tag{44.4}$$

Figure 44.1 illustrates the time domain and the Laplace domain, and the Laplace transformation and the inverse Laplace transformation.

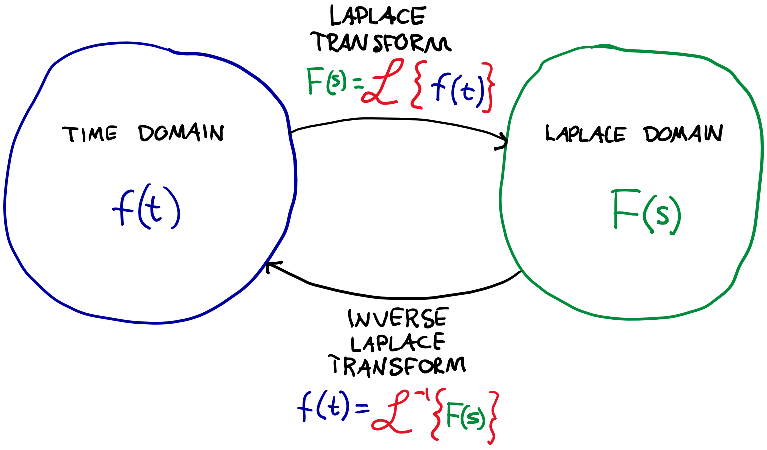


Figure 44.1: The time domain and the Laplace domain.

The inverse Laplace transform is actually defined by a complicated complex integral.[3] If you *really* want to calculate this integral, you should use the Residue Theorem in

[2] You may wonder what is the physical meaning of s. It can be interpreted as a complex frequency, but I think the best answer is that there is no meaningful physical meaning.

[3] $f(t) = \frac{1}{2\pi j} \int_{\sigma - j\infty}^{\sigma + j\infty} F(s) e^{st} ds$

mathematics. However, I suggest you instead try the simplest method, namely to find $f(t)$ from the precalculated Laplace transform pairs, cf. Section 44.3, possibly combined with one or more of the Laplace transform properties, cf. Section 44.4.

44.3 Laplace transform pairs

Laplace transform pairs:

$$f(t) \qquad \text{(time function)} \qquad \Longleftrightarrow \qquad F(s) \qquad \text{(Laplace transform)} \tag{44.5}$$

$$k\delta(t) \qquad \text{(impulse of strength or area } k) \qquad \Longleftrightarrow \qquad k \tag{44.6}$$

$$k \qquad \text{(step of amplitude } k) \qquad \Longleftrightarrow \qquad \frac{k}{s} \tag{44.7}$$

$$kt \qquad \text{(ramp of slope } k) \qquad \Longleftrightarrow \qquad \frac{k}{s^2} \tag{44.8}$$

$$kt^n \qquad (n^{\text{th}} \text{ order exponential of } t) \qquad \Longleftrightarrow \qquad k\frac{n!}{s^{n+1}} \tag{44.9}$$

$$\frac{ke^{-t/T}}{T} \qquad \text{(decaying 1. order exponential of } t) \qquad \Longleftrightarrow \qquad \frac{k}{Ts+1} \tag{44.10}$$

$$k\left(1-e^{-t/T}\right) \qquad \text{(raising 1. order exponential of } t) \qquad \Longleftrightarrow \qquad \frac{k}{(Ts+1)s} \tag{44.11}$$

$$k\left[1+\frac{1}{T_2-T_1}\left(T_1e^{-t/T_1}-T_2e^{-t/T_2}\right)\right] \qquad \Longleftrightarrow \qquad \frac{k}{(T_1s+1)(T_2s+1)s} \tag{44.12}$$

Example 44.2 *Calculation of Laplace transform*

We will calculate the Laplace transform, $F(s)$, of the following time function:

$$f(t) = e^{-t} \tag{44.13}$$

using the definition of the Laplace transform.

We start by setting $f(t) = e^{-t}$ in the integral that defines the Laplace transform (44.1):

$$\begin{aligned}
\mathcal{L}\{e^{-t}\} &= \int_0^\infty e^{-st}e^{-t}dt \\
&= \int_0^\infty e^{-(s+1)t}dt \\
&= \frac{1}{-(s+1)}\left[e^{-(s+1)t}\right]_{t=0}^{t=\infty} \\
&= \frac{1}{-(s+1)}[0-1] \\
&= \frac{1}{s+1}
\end{aligned} \tag{44.14}$$

Is this result in accordance with one of the Laplace transform pairs in Section 44.3? The pertinent Laplace transform pair is

$$\frac{k}{Ts+1} \quad \Longleftrightarrow \quad \frac{ke^{-t/T}}{T} = e^{-t} \tag{44.15}$$

Here, $T = 1$ and $k = 1$, $F(s)$ becomes

$$F(s) = \frac{1}{s+1} = \mathcal{L}\{e^{-t}\} \tag{44.16}$$

which (fortunately) is the same as (44.14) found using the definition of the Laplace transform.

[End of Example 44.2]

44.4 Laplace transform properties

In calculations with the Laplace transform you will probably need one or more of the Laplace transform *properties* presented below.[4] We will definitely use some of them for deriving transfer functions, cf. Chapter 8.

Linear combination:

$$k_1 f_1(t) + k_2 f_2(t) \Longleftrightarrow k_1 F_1(s) + k_2 F_2(s) \tag{44.17}$$

Special case: Multiplication by a constant:

$$kf(t) \Longleftrightarrow kF(s) \tag{44.18}$$

Time delay:

$$f(t-\tau) \quad \Longleftrightarrow \quad F(s)e^{-\tau s} \tag{44.19}$$

Time derivative:

$$\overset{(n)}{f}(t) \quad \Longleftrightarrow \quad s^n F(s) - s^{n-1} f(0) - s^{n-2} f'(0) - \ldots - \overset{(n-1)}{f}(0) \tag{44.20}$$

Special case: Time derivative with zero initial conditions:

$$s^n F(s) \quad \Longleftrightarrow \quad \overset{(n)}{f}(t) \tag{44.21}$$

[4] Additional properties could have been given here, too, but the ones presented are the most useful.

Special case: Time derivative with non-zero initial condition:

$$f'(t) \quad \Longleftrightarrow \quad sF(s) - f_0 \tag{44.22}$$

Special case: First order time derivative with zero initial condition:

$$f'(t) \quad \Longleftrightarrow \quad sF(s) \tag{44.23}$$

(So, differentiation corresponds to multiplication by s.)

Integration:

$$\int_0^t f(\tau)d\tau \quad \Longleftrightarrow \quad \frac{1}{s}F(s) \tag{44.24}$$

(So, integration corresponds to division by s.)

Final Value Theorem:

$$\lim_{t\to\infty} f(t) \quad \Longleftrightarrow \quad \lim_{s\to 0} sF(s) \tag{44.25}$$

Example 44.3 *Calculation of time response (inverse Laplace transform)*

Given the following differential equation:

$$y'(t) = -2y(t) + u(t) \tag{44.26}$$

with initial value $y(0) = 4$. Assume that the input variable $u(t)$ is a step of amplitude 1 at time $t = 0$.

Problem formulation:

Calculate the time response in the output variable, $y(t)$, using the Laplace transform.

Solution:

To calculate $y(t)$ we start by taking the Laplace transform of both sides of the given differential equation:

$$\mathcal{L}\left\{y'(t)\right\} = \mathcal{L}\left\{-2y(t) + u(t)\right\} \tag{44.27}$$

Here, we apply the time derivative property, cf. (44.22), at the left side, and the linear combination property, cf. (44.18), to the right side, to get

$$sY(s) - 4 = -2Y(s) + U(s) \tag{44.28}$$

Here,

$$U(s) = \frac{1}{s} \tag{44.29}$$

since the Laplace transform of a step of amplitude 1 is $\frac{1}{s}$, cf. the transform pair (44.7).

By now we have

$$sY(s) - 4 = -2Y(s) + \frac{1}{s} \tag{44.30}$$

Solving for $Y(s)$ gives

$$Y(s) = \underbrace{\frac{4}{s+2}}_{Y_1(s)} + \underbrace{\frac{1}{(s+2)\,s}}_{Y_2(s)} \tag{44.31}$$

To get the corresponding $y(t)$ from this $Y(s)$ we take the inverse Laplace transform of $Y_1(s)$ and $Y_2(s)$ to get $y_1(t)$ and $y_2(t)$ respectively, and then we calculate $y(t)$ as

$$y(t) = y_1(t) + y_2(t) \tag{44.32}$$

according to the linearity property of the Laplace transform. $y_1(t)$ and $y_2(t)$ are calculated below.

Calculation of $y_1(t)$:

We can use the transform pair (44.10), which is repeated here:

$$\frac{k}{Ts+1} \iff \frac{ke^{-t/T}}{T} \tag{44.33}$$

We have

$$Y_1(s) = \frac{4}{s+2} = \frac{2}{0.5s+1} \tag{44.34}$$

Hence, $k = 2$, and $T = 0.5$. Therefore,

$$y_1(t) = \frac{ke^{-t/T}}{T} = \frac{2e^{-t/0.5}}{0.5} = 4e^{-2t} \tag{44.35}$$

Calculation of $y_2(t)$:

We can use the transform pair (44.11), which is repeated here:

$$\frac{k}{(Ts+1)s} \iff k\left(1 - e^{-t/T}\right) \tag{44.36}$$

We have

$$Y_2(s) = \frac{1}{(s+2)\,s} = \frac{0.5}{(0.5s+1)\,s} \tag{44.37}$$

Hence, $k = 0.5$, and $T = 0.5$. Therefore,

$$y_2(t) = k\left(1 - e^{-t/T}\right) = 0.5\left(1 - e^{-t/0.5}\right) = 0.5\left(1 - e^{-2t}\right) \tag{44.38}$$

The final result becomes

$$\begin{aligned} y(t) &= y_1(t) + y_2(t) && (44.39)\\ &= 4e^{-2t} + 0.5\left(1 - e^{-2t}\right) && (44.40)\\ &= 0.5 + 3.5e^{-2t} && (44.41) \end{aligned}$$

[End of Example 44.3]

Example 44.4 *Final Value Theorem*

See Example 44.3.

Problem formulation:

Calculate the steady state value of $y(t)$ given by (44.41) using the Final Value Theorem. Also calculate the steady state value, y_s, from $y(t)$, and from (44.26) directly. Are all these (three) values of y_s the same?

Solution:

Using the Final Value Theorem on (44.31):

$$\begin{aligned} y_s = \lim_{s\to 0} sY(s) &= \lim_{s\to 0} s\left[\frac{4}{s+2} + \frac{1}{(s+2)\,s}\right] && (44.42)\\ &= \lim_{s\to 0} s\frac{4}{s+2} + \lim_{s\to 0} s\frac{1}{(s+2)\,s} = 0 + \frac{1}{2} = 0.5 && (44.43) \end{aligned}$$

From (44.41) we get

$$y_s = \lim_{t\to\infty} y(t) = 0.5 \qquad (44.44)$$

And from the differential equation we get (because the time-derivative is zero in steady state)

$$0 = -2y_s(t) + u_s(t) \qquad (44.45)$$

which gives

$$y_s = \frac{u_s}{2} = \frac{1}{2} = 0.5 \qquad (44.46)$$

So, the three results are the same.

[End of Example 44.4]

Chapter 45

Selected mathematical formulas

45.1 Differentiation of vector functions

In the formulas below, it is assumed that v is a column vector:

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \tag{45.1}$$

Assuming s is scalar, then

$$\frac{ds}{dv} = \begin{bmatrix} \frac{\partial s}{\partial v_1} \\ \vdots \\ \frac{\partial s}{\partial v_n} \end{bmatrix} \tag{45.2}$$

Assuming v is a vector and c is a *constant* vector:

$$\frac{d}{dv}\left(c^T v\right) = c \tag{45.3}$$

$$\frac{d}{dv}\left(v^T c\right) = c \tag{45.4}$$

Assuming M is a matrix:

$$\frac{d}{dv}\left(v^T M v\right) = 2Mv \tag{45.5}$$

Chapter 46

Some good control questions

Below are some good questions you should pose to yourself if you get involved in control system design. Many terms in the answers below are explained during this book.

- **Is there really a need for control?** There *is* a need for control if there is a chance that the process output variable can drift too far away from its desired value with just constant control. Such a drift can be caused by severe variations of environmental variables (process disturbances). For *unstable* processes, like water tanks and exothermal reactors and motion systems like robots and ships which must be positioned, there will always be a need for control to keep the process output variable (level; temperature; position, respectively) at a reference value. And if you conclude that there is a need for control, I guess you aim at *automatic* control.

- **Which process variable(s) needs to be controlled (to make it become equal to a reference value)**? Liquid level in a given tank? Pressure of vapour in the tank? Temperature? Flow? Composition?

- **How to measure that process variable?** Select an appropriate sensor, and make sure it detects the value of the process variable with as little time delay and sluggishness as possible! In other words, measure as directly as you can.

- **Is the measurement signal noisy?** It probably is. Use a lowpass filter to filter or smooth out the noise, but don't make the filtering too strong – or you will also filter out significant contents of the measurement signal, causing the controller to react on erroneous information.

- **How to manipulate the process variable?** Select an actuator that gives a strong impact on the process variable (to be controlled)! Avoid time delays if possible, because time delays in a control loop will limit the speed of the control, and you may get a sluggish control loop, causing the control error to become large after disturbance variations.

- **Which controller function (in the feedback controller)?** Try the PI controller. Most PID controllers actually operate as PI controllers since the derivative (D) term is deactivated because it amplifies measurement noise through the controller, causing noisy control signal which may cause excessive wear of a mechanical actuator. However, there are certain processes which requires an active D term, e.g. the double integrator which is an approximate model of a kinetic system (a "body") to be position controlled with force or torques as control variable.

- **How to tune the controller?**

  If you don't have mathematical model of the process to be controlled, try Skogestad's SIMC method, which requires only a single experiment. A good alternative is the Åstrøm-Hägglund Relay-tuning method, but be aware that this method makes the control system oscillate during the tuning. The Good Gain method and the Ziegler-Nichols method with relaxed PI settings may also be used, of course, but I would not put them on top if my list.

  If you do have a mathematical process model, consider the SIMC (Skogestad 2003) model-based tuning method to get the controller parameters directly from the process model parameters! Alternatively, with a model, you can create a simulator of your control system in e.g. Python, or LabVIEW or Matlab or Simulink, and then apply any of the methods on the simulator.

  If your controller has an Auto-tune function, you may try it in the first place!

- **Is the stability of the control system good?** The most important requirement for a control system is that it has acceptable stability: The responses due to absupt excitations may oscillate somewhat, but they should be well damped.

  If you don't have a mathematical model of the system, apply a (small) step change of the reference and observe whether the stability of the control system is ok. If you think that the stability is not good enough (too little damping of oscillations), try reducing the controller gain and increasing the integral time somewhat, say by a factor of two. Also, derivative control action can improve stability of the control loop, but remember the drawback of the D-term related to amplification of random measurement noise.

  If you do have a mathematical model of the control system, create a simulator, and simulate responses in the process variable and the control variable due to step changes in the reference and the disturbances (load variables), e.g. environmental temperature, feed composition, external forces, etc. With a simulator, you may also want to make realistic changes of certain parameters of the process model, for example time delays or some other physical parameters, to see if the control system behaves well despite these parameter changes (assuming the controller is not tuned again). In this way you can test the robustness of the control system against process variations. If the control system does not behave well, for example having too poor stability, after the parameter changes, consider Gain scheduling which is based on changing the PID parameters automatically as functions of certain process parameters.

- **Is the control error small enough after disturbance changes, and after reference changes**?

  If you do not have a simulator of the control system, it may be difficult to generate disturbance changes yourself, but reference changes can of course be applied easily.

  If you have a simulator, you can apply both reference changes and disturbance changes.

If – using either experiments or simulator – the control error is too large after the changes of the reference and the disturbance, try to tune the controller again to obtain faster control.

- **Still not happy with control system performance after retuning the controller?** Then, look for other control structures or methods based on exploiting *more process information*:
    - ***Feedforward control***: This requires that you measure one or more of the disturbances (load variables) acting on the process, and using these measurements to directly adjust the control signal to compensate for the disturbance(s).
    - ***Cascade control***: This requires that you measure some internal process variable which is influenced by the disturbance, and construct an inner control loop (inside the main control loop) based on this internal measurement to quickly compensate for the disturbance.
    - ***Model-based control***: Consider for example optimal control with state-variable feedback (LQ (Linear Quadratic) optimal control), or model-based predicitive control (MPC).

Bibliography

Åstrøm, K. J. & Hägglund, T. (1995), *PID Controllers: Theory, Design and Tuning*, ISA.

Åstrøm, K. J. & Wittenmark, B. (1994), *Adaptive Control*, Prentice Hall.

Blickley, G. J. (1990), ‘Modern control started with ziegler-nichols tuning’, *Control Engineering* **2**.

Boegli, M. (2014), *Real-Time Moving Horizon Estimation for Advanced Motion Control Application to Friction State and Parameter Estimation*, PhD thesis. Arenberg Doctoral School, KU Leuven, Belgium.

Cutler, C. R. & Ramaker, B. L. (1980), ‘Dynamic matrix control - a computer control algorithm’, *Proc. Joint Automatic Control Conference, USA-CA* .

Haugen, F., Bakke, R. & Lie, B. (2013), ‘Adapting dynamic mathematical models to a pilot anaerobic digestion reactor’, *Modeling, Identification and Control* **34**(2), 35–54.

Haugen, F. & Lie, B. (2013), ‘Relaxed Ziegler-Nichols Closed Loop Tuning of PI Controllers’, *Modeling, Identification and Control* **34**(2), 83–97.

Hill, D. T. (1983), ‘Simplified monod kinetics of methane fermentation of animal wastes’, *Agric. Wastes* **5**.

Johnson, C. D. (2000), *Process Control Instrumentation Control Technology*, Prentice-Hall.

Kalman, R. E. (1960), ‘A new approach to linear filtering and prediction problems’, *Transactions of the ASME - Journal of Basic Engineering* **82**(Series D), 35–45.

Lee, J. H. (2011), ‘Model predictive control: Review of the three decades of development’, *International Journal of Control, Automation, and Systems* **9**(3), 415–424.

Maciejowski, J. (2002), *Predictive Control with Constraints*, Prentice-Hall.

Mayr, O. (1970), *The Origins of Feedback Control*, M.I.T. Press.

Qin, S. J. & Badgwell, T. A. (2003), ‘A survey of industrial model predictive control technology’, *Control Engineering Practice* **11**, 733–764.

Saelid, S. & Foss, B. A. (1983), ‘Adaptive controllers with a vector variable forgetting factor’, *Proceedings of the 22nd IEEE Conference on Decision and Control* pp. 1488–1494.

Seborg, D. E., Edgar, T. F. & Mellichamp, D. A. (2004), *Process Dynamics and Control*, John Wiley and Sons.

Simon, D. (2006), *Optimal State Estimation*, Wiley.

Skogestad, S. (2003), 'Simple analytic rules for model reduction and PID controller tuning', *Journal of Process Control* **14**.

Skogestad, S. (2023), 'Advanced control using decomposition and simple elements', *Annual Reviews in Control* **56**.

Ziegler, J. & Nichols, N. (1942), 'Optimum settings for automatic controllers', *Trans. ASME* **64**(3), 759–768.

Index