Kapittel 20

Automatisk regulering

20.1 Innledning

Det meste av denne boken handler om hvordan du kan bruke programmering til å løse forskjellige slags matematiske problemer og til å programmere simulatorer. Vi skal nå studere en annen type anvendelse av programmering, nemlig bruk av programmering til å regulere eller styre et system automatisk – med et program i en datamaskin – slik at det oppfører seg slik du vil! Dette fagfeltet har alternative navn: Reguleringsteknikk eller automatiseringsteknikk eller teknisk kybernetikk. Det fins tusenvis av automatiske reguleringssystemer rundt omkring som regulerer eller styrer f.eks. biler, skip, fly, roboter, biogassreaktorer, kjemiske prosesser, vannkraftsystemer, bygningers temperatur osv. Kjernen i reguleringssystemene er en programmert regulatoralgoritme. Vi går igjennom de viktigste regulatoralgoritmene og programmering av dem i Python i dette kapitlet.

20.2 Reguleringssystemers struktur og virkemåte

20.2.1 Manuell regulering

Vi skal lage et reguleringssystem for vannkokeren som vi studerte i kap. 15.4. Prinsippene er de samme også for reguleringssystemer for andre systemer enn vannkokeren.

Tenk deg at du selv (manuelt) skal stå for temperaturreguleringen. Anta at du kan påvirke vanntemperaturen T ved å justere varmeelementets avgitte effekt, P [W] med f.eks. en skruknapp. Hva er din strategi for å oppnå at T blir nær, eller helst: lik, et gitt temperatursettpunkt, T_{sp}, som vi kan anta er – la oss si – 70 grader? Jeg tipper at du gjør slik – kontinuerlig:

- Måler temperaturen T med et termometer.
- Sammenlikner T med T_{sp}.

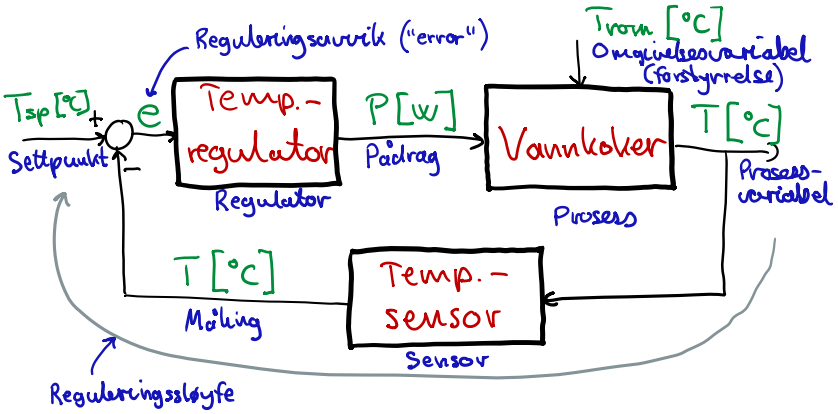
- Så lenge det er et avvik mellom T_{sp} og T, dvs. så lenge reguleringsavviket (e for error)

$$e = T_{sp} - T \tag{20.1}$$

  er forskjellig fra null, justerer du pådraget P for å få T nær, eller helst lik, T_{sp}.

Med andre ord: Du bruker reguleringsavviket som grunnlag for reguleringen. Dette prinsippet kan vi kalle avviksdrevet regulering. Prinsippet kalles også tilbakekoplet regulering (engelsk: feedback control) fordi

Figur 20.1 viser et blokkdiagram av temperaturreguleringssystemet for vannkokeren. Generelle reguleringstekniske begreper er skrevet i blått.



Figur 20.1: Blokkdiagram av temperaturreguleringssystemet for vannkokeren.

Her er noen andre eksempler – blant svært mange eksempler – på reguleringssystemer som fungerer etter samme prinsipp, altså avviksdrevet regulering:

- Regulering av dusjtemperaturen: Du (regulatoren) justerer blandekranen for varmt/kaldt vann kontinuerlig for å oppnå den temperaturen du ønsker deg, altså inntil temperaturavviket er null. Temperaturavviket driver reguleringen.
- Regulering av hastigheten til en bil: Du (regulatoren) justerer gasspedalen (ev. bremsepedalen) kontinuerlig for å oppnå ønsket fart, dvs. null fartsavvik. Fartsavviket driver reguleringen.
- Regulering av posisjonen til en bil: Du (regulatoren) justerer rattet kontinuerlig for å oppnå ønsket plassering i veibanen. Posisjonsavviket driver reguleringen.

20.2.2 Automatisk regulering

I underkap. 20.2.1 sto du selv for temperaturreguleringen. Det er manuell regulering. Tenk deg at du har viktigere ting å gjøre, så du vil sette bort jobben til en datamaskin. Du må da programmere datamaskinen til å utføre reguleringen automatisk. Prinsippet for automatisk regulering er akkurat det samme som for manuell regulering, dvs. at blokkdiagrammet i

figur 20.1 gjelder, bortsett fra at regulatoren ikke er ditt hode, men en reguleringsalgoritme i en datamaskin. Datamaskinen må kontinuerlig registrere temperaturmålingen fra sensoren (f.eks. et såkalt Pt100-element) og kontinuerlig generere et styresignal til varmeelementet.

Vi går ikke nærmere inn på hvilke komponenter du må skaffe deg for å få laget et automatisk temperaturreguleringssystem. Men vi skal se på den «intelligente» delen av reguleringssystemet, som er regulatoralgoritmen som vi skal programmere i Python. Og vi skal teste den programmerte regulatoren mot en simulert vannkoker. Generelt er det smart å teste regulatoren på en simulator før ev. bygging.

Vi skal se på de mest brukte regulatoralgoritmene i kap. 20.3.

20.3 Regulatoralgoritmer

Vi skal i dette kapitlet beskrive tre av de mest vanlige regulatoralgoritmene, nemlig:

- Proporsjonalregulatoren (P-regulatoren), kap. 20.3.1
- Proporsjonal pluss integral-regulatoren (PI-regulatoren), kap. 20.3.2
- Av/på-regulatoren, kap. 20.3.3

20.3.1 P-regulator

Regulatoralgoritmen

Proporsjonal-regulatoren (P-regulatoren) har følgende algoritme:

$$u_k = u_{\text{man}} + \underbrace{K_p e_k}_{u_{p,k}} \tag{20.2}$$

der:

- u_{man} er det manuelt innstilte styresignalet. u_{man} stilles inn slik at det gir et «fornuftig» styresignal.
- e_k er reguleringsavviket, som generelt kan skrives:

$$e_k = K_p \left(y_{sp,k} - y_k \right) \tag{20.3}$$

- K_p er regulatorforsterkningen (eller proporsjonalforsterkningen eller bare forsterkningen), som er en regulatorparameter som skal ha en passende konstant verdi, slik at reguleringssystemet får passende stabilitet og hurtighet. Vi kommer i denne boken ikke til å beskrive hvordan vi kan stille inn K_p – det er en oppgave for spesialister innen reguleringsteknikk, se f.eks. Haugen (2014).

- $u_{p,k}$ er proporsjonalleddet i det totale pådraget u_k.

Vi bør implementere en begrensing av u for å sikre at regulatoren ikke genererer en umulig pådragsverdi, f.eks. at den ber om en negativ effekt dersom pådragsorganet er et varmeelement, eller at den ber om en negativ ventilåpning i en reguleringsventil. Vi kan begrense u slik:

$$u \in [u_{\min}, u_{\max}] \tag{20.4}$$

Egenskaper ved P-regulatoren

Som vi ser av (20.2), gir P-regulatorens proporsjonalledd, $u_p = K_p e$, en justering av styresignalet som er *proporsjonal* med reguleringsavviket – jo større avvik, jo mer vil regulatoren endre styresignalet, i et forsøk på å få avviket til å bli mindre.

Som vi skal se i eksempel 20.1, klarer P-regulatoren dessverre ikke å sikre null avvik stasjonært, hvilket kan være en svakhet i mange anvendelser.[1] Men i noen anvendelser er det ok med et avvik, bare det er lite nok.

En forklaring på at P-regulatoren ikke klarer å gi stasjonært avvik $e_s = 0$, er som følger: Anta at den virkelig klarte å gi $e_s = 0$. Da blir P-leddet 0, altså ingen bidrag til det totale pådraget, men vanligvis trenger vi et bidrag forskjellig fra 0 (med mindre vi er så heldige at u_{man} gir perfekt pådrag – men slik flaks kan vi ikke regne med). Reguleringssystemet vil da stabilisere seg slik at e blir forskjellig fra null, slik at P-leddet gir noe bidrag, men ikke et tilstrekklig stort bidrag til å gi $e_s = 0$.

Eksempel 20.1 *P-regulering av simulert vannkoker*

Vi skal bruke P-regulatoren for temperaturregulering av den simulerte vannkokeren fra kap. 15.4.4. Program 20.1 implementer en simulator for dette reguleringssystemet. Euler-integrasjonen for temperaturen (prosessimulatoren) og P-regulatoren er programmert i hver sine funksjoner.

Figur 20.2 viser en simulering av temperaturreguleringssystemet med P-nivåregulator med følgende innstillinger:

- Settpunkt: $T_{\text{sp}} = 70$ °C.
- Regulatorparametre: $u_{\text{man}} = 0$ W. $K_p = 50$ W/°C.
- Initialverdi av temperaturen: $T_{\text{init}} = 20$ °C.
- Simulatorparametre: $t_{\text{start}} = 0$ s. $t_{\text{stop}} = 600$ s. $dt = 0{,}1$ s.

http://techteach.no/python/files/prog_kettle_p_control.py

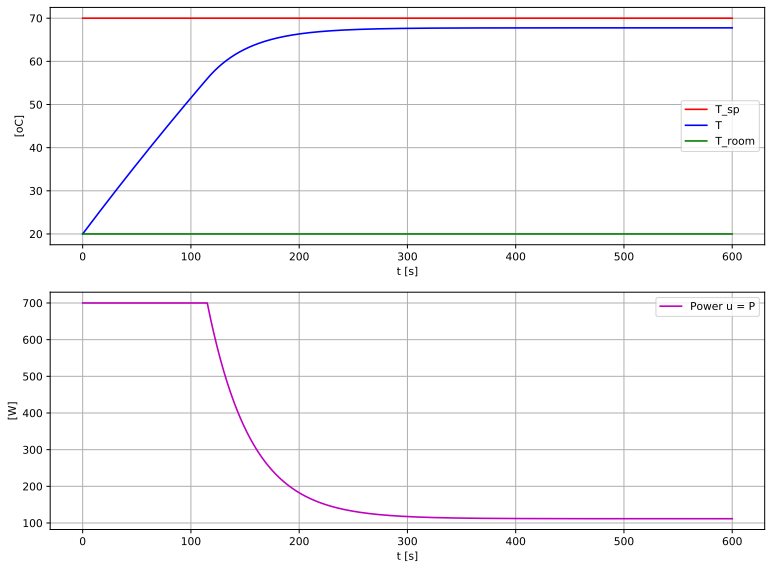
[1] PI-regulatoren er langt bedre på den måten, jf. underkap. 20.3.2. Den gir null avvik!

Listing 20.1: prog_kettle_p_control.py

```
# %% Import of packages:

import matplotlib.pyplot as plt
import numpy as np

# %% Def of PI controller function:

def fun_p(y_sp_k, y_k, p_params, dt):

    (Kp, u_man, u_max, u_min) = controller_params
    e_k = y_sp_k - y_k  # Control error
    u_p_k = Kp*e_k  # P term
    u_k = u_man + u_p_k   # Total control signal
    u_k = np.clip(u_k, u_min, u_max)  # Limit of control

    return u_k

# %% Def of process model simulator

def process_sim(T_k, u_k, process_params, dt):

    (C, G, T_min, T_max) = process_params
    T_k = np.clip(T_k, T_min, T_max)
    dT_dt_k = (1/C)*(u_k + G*(T_room - T_k)) #Time deriv.
    T_kp1 = T_k + dt*dT_dt_k  # Euler integration

    return T_kp1

# %% Model parameters:

H = 0.079  # [m]
D = 0.090  # [m]
c = 4180  # [J/(kg*K)]
rho = 1000  # [kg/m3]
k_tc = 0.2  # [(W*m/(m2*K)) = W/(m*K)]
L = 3e-3  # [m]

# Derived model params:

V = H*np.pi*(D/2)**2  # [m3]
C = c*rho*V  # [J/K]
A = np.pi*D*H + 2*np.pi*(D/2)**2  # [m2]
G = (k_tc/L)*A  # [W/K]

T_min = 0  # [oC] State limit
T_max = 100  # [oC] State limit

process_params = (C, G, T_min, T_max)

# %% Simulation time settings:

dt = 0.1  # [s]
t_start = 0  # [s]
```

```
t_stop = 600  # [s]
N_sim = int((t_stop - t_start)/dt) + 1  # Num time-steps

# %% Params of input signals:

T_room = 20  # [oC]
T_sp = 70  # {oC] Setpoint

# %% P controller settings:

Kp = 50.0  # W/K]

u_man = 0  # [W]
u_max = 700  # [W]
u_min = 0  # [W]

controller_params = (Kp, u_man, u_max, u_min)

# %% Preallocation of arrays for plotting etc:

t_array = np.zeros(N_sim)
T_sp_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
T_room_array = np.zeros(N_sim) + T_room
u_array = np.zeros(N_sim)

# %% Initialization:

T_init = 20.0  # [oC]
T_k = T_init  # [oC]
u_i_km1 = 0  # [W]

# %% Simulation loop:

for k in range(0, N_sim):

    t_k = k*dt

    # Setting input:
    if (t_k >= t_start):
        T_sp_k = T_sp

    # P controller:
    u_k = fun_p(T_sp_k, T_k, controller_params, dt)

    # Process sim (Euler integration):
    T_kp1 = process_sim(T_k, u_k, process_params, dt)

    # Updating arrays for plotting:
    t_array[k] = t_k
    T_array[k] = T_k
    T_sp_array[k] = T_sp_k
    u_array[k] = u_k
```

```
    # Time shift:
    T_k = T_kp1

# %% Plotting:

plt.close('all')
plt.figure(1, figsize=(12, 9))

plt.subplot(2, 1, 1)
plt.plot(t_array, T_sp_array,'r', label='T_sp')
plt.plot(t_array, T_array,'b', label='T')
plt.plot(t_array, T_room_array,'g', label='T_room')
plt.legend()
plt.xlabel('t [s]')
plt.ylabel('[oC]')
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(t_array, u_array, 'm', label='Power u = P')
plt.legend()
plt.grid()
plt.xlabel('t [s]')
plt.ylabel('[W]')

# plt.savefig('prog_kettle_p_control.pdf')
plt.show()
```



Figur 20.2: Simulering av temperaturregulering av vannkoker med P-regulator.

Kommentarer til simuleringen:

- T stabiliserer seg på 67,8 °C. Det stasjonære reguleringsavviket er altså $e_s = 2{,}2$ °C.
- Hvis kaffen blir like god med temperatur 67,8 °C som med 70 °C, så kan vi klare oss med en P-regulator. Hvis vi insisterer på 70 °C, kan vi bytte ut P-regulatoren med en PI-regulator, jf. kap. 20.3.2.

[Slutt på eksempel 20.1]

20.3.2 PI-regulator

Regulatoralgoritmen

I de fleste anvendelser er det ikke gunstig at det statiske reguleringsavviket er forskjellig fra null. Det bør være null. Med utgangspunkt i P-regulatoren, hvordan kan vi forbedre regulatorfunksjonen for å oppnå null statisk avvik? Hvis du selv var regulator, hvordan ville du ha manipulert effekten (vha. pumpa) for å oppnå null nivåavvik? Vi antar at du ville ha økt utstrømmen så lenge nivåavviket er forskjellig fra null, og når avviket ble null, ville du holdt utstrømmen fast. Det er akkurat en slik oppførsel vi trenger i regulatorfunksjonen! Vi kan realisere dette med følgende regulatoralgoritme, slik det er vanlig i automatiseringsutstyr:

$$u_k = u_{\text{man}} + u_{p,k} + u_{i,k} \tag{20.5}$$

der $u_{p,k}$ er pådragets proporsjonalledd (P-ledd), som før:

$$u_{p,k} = K_p e_k \tag{20.6}$$

og $u_{i,k}$ er det såkalte integralleddet (I-leddet) gitt ved

$$u_{i,k} = u_{i,k-1} + \frac{K_p}{T_i} T_s e_k \tag{20.7}$$

der:

- $u_{i,k}$ er I-leddet beregnet for nåværende tidspunkt.
- $u_{i,k-1}$ er I-leddet beregnet for forrige tidspunkt (ett tidsskritt tilbake i tiden).
- $e_k = y_{sp,k} - y_k$ er reguleringsavviket ved nåværende tidspunkt.
- dt [s] er tidsskrittet (samplingstiden).
- K_p er regulatorforsterkningen, som er den samme som i P-leddet.
- T_i [s] er integraltiden, som er en regulatorparameter som må gis en fornuftig verdi, men vi skal ikke gå inn på regulatorinnstilling i denne boken.

Vi bør implementere to begrensinger i PI-regulatoren:

- Begrensing av den totale pådragverdien u, på samme måte som for P-regulatoren, jf. kap. 20.3.1. Altså:

$$u \in [u_{\min},\, u_{\max}] \tag{20.8}$$

- Begrensing av I-leddet u_i for å unngå at u_i bygger seg opp (eller ned) til en uforholdsmessig stor tallverdi, hvilket kan skje dersom e er stor i en tidsperiode. Hvis u_i får en veldig stor tallverdi, kan det ta lang til før u_i igjen får en fornuftig verdi. Vi må tillate at u_i bidrar med både positive og negative verdier. Vi kan begrense u_i slik (der det er forutsatt at $u_{\max}$ er positiv):

$$u_i \in [-u_{\max},\, u_{\max}] \tag{20.9}$$

  Begrensingen av u_i kalles integratorbegrensing (engelsk faguttrykk: anti windup).

Egenskaper ved PI-regulatoren

I (20.7) er e_k er oppdateringsledd eller forbedringsledd, for så lenge e_k er forskjellig fra null, vil u_i få et tillegg, dvs. endres – økes eller minkes; avhengig av fortegnet på e. Og siden u_i forbedres, vil det totale pådraget u_k gitt ved (20.5) endres. Med andre ord: Vi har nå en avviksdrevet pådragsjustering, eller: en avviksdrevet regulering, og pådragsjusteringen vil pågå helt til avviket er blitt null – midt i blinken!

I industrien er PI-regulator svært mye brukt. I et prosessanlegg kan det være titalls, kanskje hundrevis, av PI-regulatorer programmert i automatiseringsutstyr som sikrer at prosessanlegg holdes i drift på de spesifiserte arbeidspunktene (settpunktene) for materialstrøm, temperatur, væskenivå, gasstrykk, produktkvalitet osv.

Eksempel 20.2 *PI-regulering av simulert vannkoker*

Program 20.2 implementer PI-regulering av den simulerte vannkokeren. Euler-integrasjonen for temperaturen (prosessimulatoren) og PI-regulatoren er programmert i hver sine funksjoner.

Figur 20.3 viser en simulering av temperaturreguleringssystemet med PI-nivåregulator med følgende innstillinger:

- Settpunkt: $T_{\text{sp}} = 70$ °C.
- Regulatorparametre: $u_{\text{man}} = 0$ W. $K_p = 50$ W/°C. $T_i = 120$ s.
- Pådragsgrenser: $u_{\min} = 0$ W. $u_{\max} = 700$ W.
- Initialverdi av temperaturen: $T_{\text{init}} = 20$ °C.
- Simulatorparametre: $t_{\text{start}} = 0$ s. $t_{\text{stop}} = 600$ s. $dt = 0{,}1$ s.

http://techteach.no/python/files/prog_kettle_pi_control.py

Listing 20.2: prog_kettle_pi_control.py

```
# %% Import of packages:

import matplotlib.pyplot as plt
import numpy as np

# %% Def of PI controller function:

def fun_pi(y_sp_k, y_k, u_i_km1, controller_params, dt):

    (Kp, Ti, u_man, u_min, u_max) = controller_params
    e_k = y_sp_k - y_k  # Control error
    u_p_k = Kp*(y_sp_k - y_k)  # P term
    u_i_k = u_i_km1 + (Kp*dt/Ti)*e_k  # PI term
    u_i_k = np.clip(u_i_k,
                    -(u_max-1*u_man),
                    (u_max-1*u_man))  # Limit ui
    u_k = u_man + u_p_k + u_i_k  # Total control signal
    u_k = np.clip(u_k, u_min, u_max)  # Limit of control

    return (u_i_k, u_k)

# %% Def of process simulator:

def process_sim(T_k, u_k, process_params, T_room, dt):

    (C, G, T_min, T_max) = process_params
    T_k = np.clip(T_k, T_min, T_max)
    dT_dt_k = (1/C)*(u_k + G*(T_room - T_k)) #Time deriv.
    T_kp1 = T_k + dt*dT_dt_k  # Euler integration

    return T_kp1

# %% Model parameters:

H = 0.079  # [m]
D = 0.090  # [m]
c = 4180  # [J/(kg*K)]
rho = 1000  # [kg/m3]
k_tc = 0.2  # [(W*m/(m2*K)) = W/(m*K)]
L = 3e-3  # [m]

# Derived model params:

V = H*np.pi*(D/2)**2  # [m3]
C = c*rho*V  # [J/K]
A = np.pi*D*H + 2*np.pi*(D/2)**2  # [m2]
G = (k_tc/L)*A  # [W/K]

T_min = 0  # [oC] State limit
T_max = 100  # [oC] State limit

process_params = (C, G, T_min, T_max)
```

```
# %% Simulation time settings:

dt = 0.1  # [s]
t_start = 0  # [s]
t_stop = 600  # [s]
N_sim = int((t_stop - t_start)/dt) + 1  # Num time-steps

# %% Params of input signals:

T_room = 20  # [oC]
T_sp = 70  # {oC] Setpoint

# %% PI controller settings:

Kp = 50.0  # W/K]
Ti = 120  # [s]

u_man = 0  # [W]
u_min = 0  # [W]
u_max = 700  # [W]

controller_params = (Kp, Ti, u_man, u_min, u_max)

# %% Preallocation of arrays for plotting etc:

t_array = np.zeros(N_sim)
T_sp_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
T_room_array = np.zeros(N_sim) + T_room
u_array = np.zeros(N_sim)

# %% Initialization:

T_init = 20.0  # [oC]
T_k = T_init  # [oC]
u_i_km1 = 0  # [W]

# %% Simulation loop:

for k in range(0, N_sim):

    t_k = k*dt

    # Setting input:
    if (t_k >= 0):
        T_sp_k = T_sp

    # PI controller:
    (u_i_k, u_k) = fun_pi(T_sp_k, T_k, u_i_km1,
                          controller_params, dt)

    # Process sim (Euler integration):
    T_kp1 = process_sim(T_k, u_k, process_params, T_room,
                        dt)
```

```
    # Updating arrays for plotting:
    t_array[k] = t_k
    T_array[k] = T_k
    T_sp_array[k] = T_sp_k
    u_array[k] = u_k

    # Time shift:
    T_k = T_kp1
    u_i_km1 = u_i_k

# %% Plotting:

plt.close('all')
plt.figure(1, figsize=(12, 9))

plt.subplot(2, 1, 1)
plt.plot(t_array, T_sp_array,'r', label='T_sp')
plt.plot(t_array, T_array,'b', label='T')
plt.plot(t_array, T_room_array,'g', label='T_room')
plt.legend()
plt.xlabel('t [s]')
plt.ylabel('[oC]')
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(t_array, u_array, 'm', label='Power u = P')
plt.legend()
plt.grid()
plt.xlabel('t [s]')
plt.ylabel('[W]')

# plt.savefig('prog_kettle_pi_control.pdf')
plt.show()
```

Kommentar til simuleringen:

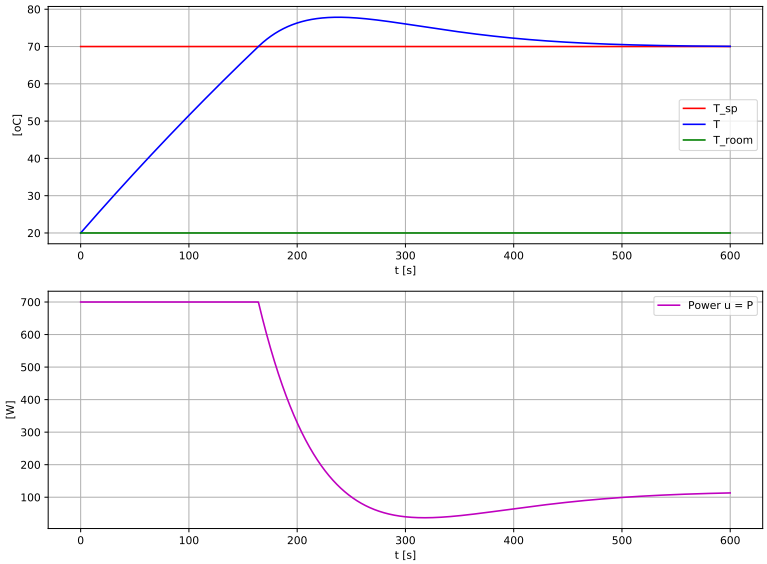
- T stabiliserer seg på $T_{sp} = 70$ °C, etter et oversving. Det stasjonære reguleringsavviket er altså $e_s = 0$ °C. Veldig bra!
- Hvis vi velger en annen verdi for T_{sp} enn 70 °C, får vi også da $e_s = 0$.

[Slutt på eksempel 20.2]

20.3.3 Av/på-regulator

Regulatoralgoritmen

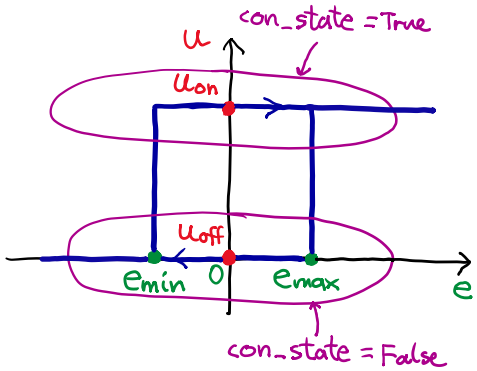
Den såkalte av/på-regulatoren (engelsk: on/off controller) setter styresignalet til en av to mulige verdier: en høy verdi eller lav verdi, avhengig av størrelsen av reguleringsavviket, e.



Figur 20.3: Simulering av temperaturregulering av vannkoker med PI-regulator.

En termostatregulator for regulering av romtemperatur eller kjøleskaptemperatur m.m. er egentlig en av/på-regulator. En kan betrakte av/på-regulatoren som den aller enkleste regulatoralgoritmen for automatisk, avviksbasert regulering siden den setter styresignalet til en av to mulige verdier. Du kan altså klare deg med et «enkelt» pådragsorgan som skal kunne slås enten på eller av.

Av/på-regulatoren for temperaturregulering av vannkokeren kan defineres som vist i figur 20.4. Vi har brukt en såkalt tilstandsmaskin (engelsk: state machine) for å lagre informasjon



Figur 20.4: Av/på-regulator for temperaturregulering av vannkokeren.

om de to mulige tilstandene av/på-regulatoren kan befinne seg i.[2] Vi har valgt navnet con_state på tilstandsvariabelen, og de to mulige tilstandene er True og False.

[2]Tilstandsmaskiner kan brukes for å strukturere kompliserte algoritmer på en oversiktlig og veldefinert måte. Programmet befinner seg til enhver tid i en bestemt tilstand, og til hver tilstand er det knyttet aksjoner,

I hver tilstand utføres tilhørende styringer,[3] her uttrykt i Python-kode:

- if con_state == True: u_k = u_on
- if con_state == False: u_k = u_off

Hvilken av disse to tilstandene (True, False) regulatoren befinner seg i, er bestemt både av verdien av reguleringsavviket e og av nåværende tilstand i av/på-regulatoren, jf. figur 20.4, her uttrykt i Python-kode:

- if e_k <= e_min: con_state = False
- if e_k >= e_max: con_state = True
- if e_min <= e_k <= e_max and con_state == True: con_state = True
- if e_min <= e_k <= e_max and con_state == False: con_state = False

Egenskaper ved av/på-regulatoren

Av/på-regulatoren er relativt enkel og trenger ingen annen innstilling enn dødsonen mellom e_{min} og e_{max}. En ulempe ved denne regulatoren er at prosessverdien vil svinge vedvarende omkring settpunktet. Det gjennomsnittlige avviket blir generelt forskjellig fra null. Vanligvis er PI-regulatoren, ev. P-regulatoren, å foretrekke. Men som nevnt i innledningen til dette kapitlet, finner du av/på-regulatoren i temperaturreguleringssystemer for ovner, kjøleskap m.m.

Eksempel 20.3 *Av/på-regulering av simulert vannkoker*

Program 20.3 implementer av/på-regulering av den simulerte vannkokeren.
Euler-integrasjonen for temperaturen (prosessimulatoren) og av/på-regulatoren er programmert i hver sine funksjoner.

Figur 20.3 viser en simulering av temperaturreguleringssystemet med av/på-regulator med følgende innstillinger:

- Tidsskritt: $dt = 0{,}1$ s
- Settpunkt: $T_{\text{sp}} = 70$ °C
- Initialtilstand: $T_{\text{init}} = 20$ °C
- Regulatorparametre: $u_{\text{on}} = 700$ W, $u_{\text{off}} = 0$ W, $e_{\text{max}} = 5$ °C, $e_{\text{min}} = -0{,}5$ °C

f.eks. sette et styresignal til en bestemt verdi som kan være forskjellig fra tilstand til tilstand.

[3] Kalles også aksjoner (engelsk: actions) i noen sammenhenger.

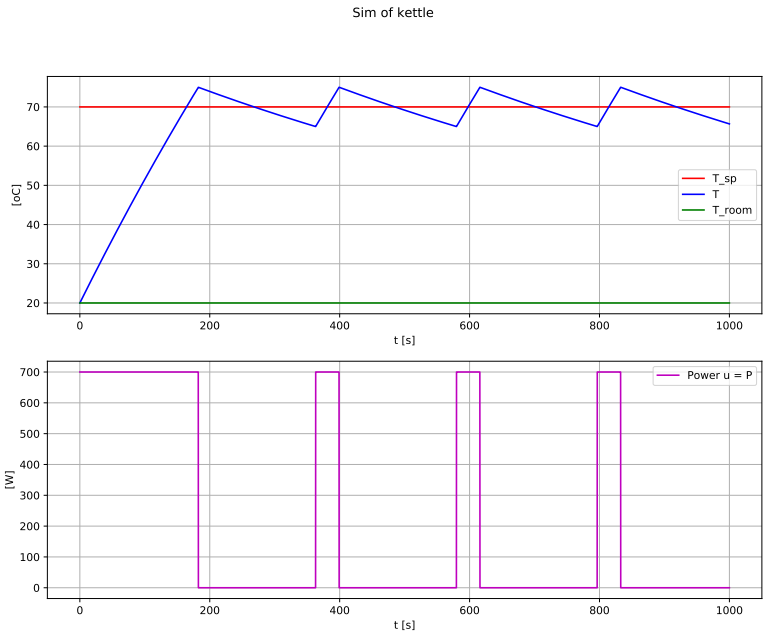
http://techteach.no/python/files/prog_kettle_on_off_control.py

Listing 20.3: prog_kettle_on_off_control.py

```
# %% Import of packages:

import matplotlib.pyplot as plt
import numpy as np

# %% Def of on/off controller:

def fun_on_off(y_sp_k, y_k, controller_params,
               contr_state):

    # Control error:
    e_k = y_sp_k - y_k

    # Setting state of the controller:
    if e_k <= e_min:
        contr_state = False
    if e_k >= e_max:
        contr_state = True
    if (e_min <= e_k <= e_max) and (contr_state == True):
        contr_state = True
    if (e_min <= e_k <= e_max) and (contr_state== False):
        contr_state = False

    # Selecting control signal based on controller state:
    if contr_state == True:
        u_k = u_on
    if contr_state == False:
        u_k = u_off

    return (u_k, contr_state)

# %% Def of process model simulator

def process_sim(T_k, u_k, process_params, dt):

    (C, G, T_min, T_max) = process_params
    T_k = np.clip(T_k, T_min, T_max)
    dT_dt_k = (1/C)*(u_k + G*(T_room - T_k)) #Time deriv.
    T_kp1 = T_k + dt*dT_dt_k  # Euler integration

    return T_kp1

# %% Model parameters:

H = 0.079  # [m]
D = 0.090  # [m]
c = 4180  # [J/(kg*K)]
rho = 1000  # [kg/m3]
k_tc = 0.2  # [(W*m/(m2*K)) = W/(m*K)]
L = 3e-3  # [m]
```

```
# Derived parameters:

V = H*np.pi*(D/2)**2  # [m3]
C = c*rho*V  # [J/K]
A = np.pi*D*H + 2*np.pi*(D/2)**2  # [m2]
G = (k_tc/L)*A  # [W/K]

T_min = 0  # [oC] State limit
T_max = 100  # [oC] State limit

process_params = (C, G, T_min, T_max)

# %% Simulation time settings:

dt = 0.1  # [s]
t_start = 0  # [s]
t_stop = 1000  # [s]
N_sim = int((t_stop - t_start)/dt) + 1  # Num time-steps

# %% Params of input signals:

T_room = 20  # [oC]
T_sp = 70  # [oC] Setpoint

# %% Controller params:

u_on = 700  # [W]
u_off = 0  # [W]
e_max = 5  # [oC]
e_min = -5  # [oC]

controller_params = (u_on, u_off, e_max, e_min)

# %% Preallocation of arrays for plotting etc:

t_array = np.zeros(N_sim)
T_sp_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
T_room_array = np.zeros(N_sim) + T_room
u_array = np.zeros(N_sim)

# %% State limits:

T_min = 0  # [oC]
T_max = 100  # [oC]

# %% Initialization:

T_init = 20.0  # [oC]
T_k = T_init  # [oC]
contr_state = True

# %% Simulation loop:
```

```
for k in range(0, N_sim):

    t_k = k*dt

    # Setting input:
    if (t_k >= t_start):
        T_sp_k = T_sp

    # On/off controller:
    (u_k, contr_state) = fun_on_off(T_sp_k, T_k,
                                    controller_params,
                                    contr_state)

    # Process sim (Euler integration):
    T_kp1 = process_sim(T_k, u_k, process_params, dt)

    # Updating arrays for plotting:
    t_array[k] = t_k
    T_array[k] = T_k
    T_sp_array[k] = T_sp_k
    u_array[k] = u_k

    # Time shift:
    T_k = T_kp1

# %% Plotting:

plt.close('all')
plt.figure(num=1, figsize=(12, 9))
plt.suptitle('Sim of kettle')

plt.close('all')
plt.figure(1, figsize=(12, 9))

plt.subplot(2, 1, 1)
plt.plot(t_array, T_sp_array,'r', label='T_sp')
plt.plot(t_array, T_array,'b', label='T')
plt.plot(t_array, T_room_array,'g', label='T_room')
plt.legend()
plt.xlabel('t [s]')
plt.ylabel('[oC]')
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(t_array, u_array, 'm', label='Power u = P')
plt.legend()
plt.grid()
plt.xlabel('t [s]')
plt.ylabel('[W]')

# plt.savefig('prog_kettle_on_off_control.pdf')
plt.show()
```



Figur 20.5: Simulering av vannkoker med av/på-regulator.

Legg merke til følgende i responsene:

- T svinger omkring T_{sp}. Det er typisk at det svinger med av/på-regulatorer. Dette kan være ok i temperaturregulering, men ville du hatt en av/på-regulator som cruise-kontroller i bilen?
- Styresignal endres sprangvis mellom u_{on} og u_{off}. Disse sprangene plager ikke varmeelementer, men hva hvis styringen utføres av pumper, ventiler, motorer eller andre mekaniske komponenter?

[Slutt på eksempel 20.3]

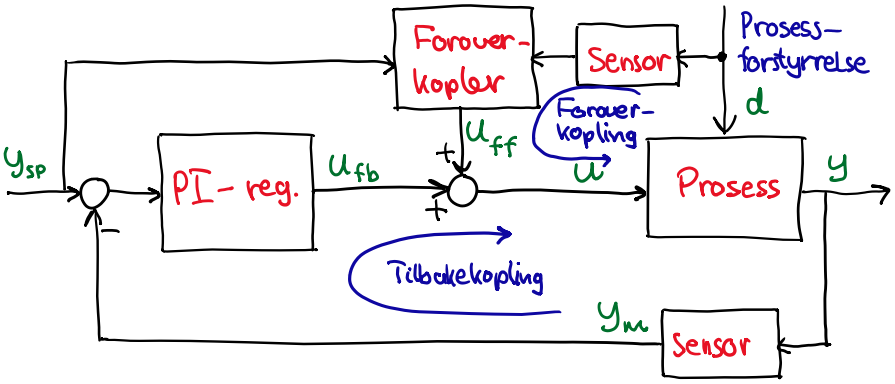
20.3.4 Foroverkopling

Reguleringsmetoden som er beskrevet i underkap. 20.2.1, kalles avviksdrevet regulering eller tilbakekoplet regulering. En annen reguleringsmetode er såkalt foroverkopling. Når foroverkopling brukes, er det vanligvis sammen med tilbakekopling. Foroverkopling kan gi en stor forbedring av reguleringen sammenliknet med bruk av bare tilbakekopling – varierende settpunkt følges mer nøyaktig, og prosessforstyrrelser kompenseres for raskere.

Figur 20.6 viser strukturen av et generelt reguleringssystem med både foroverkopling og tilbakekopling. Foroverkopleren er en matematisk funksjon (i eksempel 20.4 skal vi utlede en foroverkopler for vannkokeren). Det totale pådraget framkommer som summen

$$u = u_{\text{fb}} + u_{\text{ff}} \tag{20.10}$$

der u_{fb} er pådragsbidraget fra PI-regulatoren i tilbakekoplingen[4] og u_{ff} er pådragsbidraget fra foroverkoplingen.



Figur 20.6: Reguleringssystem med både foroverkopling og tilbakekopling.

Pådraget fra foroverkopleren er basert på følgende *informasjon* (eller kunnskap) om prosessen som skal reguleres:

- **Målinger av omgivelsesvariable**r, som gjerne kalles prosessforstyrrelser, som virker på prosessen.
- **En matematisk prosessmodell** i form av differensiallikningene som beskriver prosessens oppførsel. Modellen uttrykker den kunnskapen vi har om prosessen, så det at vi bruker modellen direkte i reguleringen, er modellbasert regulering.
- **Settpunktet**, som jo er den ønskede verdien av prosessutgangen.

Bakgrunnen for begrepet foroverkopling er at det er en kopling fra inngangsvariablene prosessforstyrrelse og settpunkt direkte til inngangsvariabelen pådrag, se figur 20.6. Det å kople fra én inngang til en annen inngang, er ≪foroverkopling≫.

Eksempel 20.4 viser hvordan vi kan lage en foroverkopler for vannkokeren.

Eksempel 20.4 *Fororverkopler for vannkokeren*

Vi tar utgangspunkt i vannkokerens matematiske modell (15.7), som vi gjengir her:

$$T' = \frac{1}{C}\left[u + G\left(T_{\text{room}} - T\right)\right] \tag{20.11}$$

der pådraget u [W] er tilført effekt, altså $u = P$.

Utvikling av foroverkopleren kan foregå i to trinn:

[4]Det kan generelt være en annen regulator enn en PI-regulator, men PI-regulatoren er den vanligste.

1. **Sett inn settpunktet for prosessutgangen i prosessmodellen:** Vi setter T_{sp} inn for T i (20.11) og får:
$$T'_{sp} = \frac{1}{C}\left[u + G\left(T_{\text{room}} - T_{sp}\right)\right] \tag{20.12}$$

2. **Løs prosessmodellen mht. pådragsvariabelen u, dvs. finn u fra prosessmodellen**. Fra (20.11) får vi:
$$u_{\text{ff}} = CT'_{sp} - G\left(T_{\text{room}} - T_{sp}\right) \tag{20.13}$$
som er foroverkopleren!

Foroverkopleren (20.13) er en formel for hvordan pådraget skal justeres slik at T – ideelt sett – blir lik T_{sp}. I (20.13) ser vi at for at u_{ff} skal kunne beregnes (i foroverkopleren), må alle størrelsene på høyre side være kjente: C, og G, T_{room}, T_{sp} og T'_{sp}. For å få verdien av T_{room}, må vi kanskje montere en temperaturmåler. T'_{sp} kan vi beregne med f.eks. Eulers bakovermetode, likn. (13.6).

Program http://techteach.no/python/files/prog_kettle_fb_and_ff.py implementerer en simulator av temperaturreguleringssystemet med foroverkopling og tilbakekopling med PI-regulator. (Vi lister ikke opp programmet her.) I programmet er det mulig å kople ut foroverkoplingen ved å multiplisere u_{ff} med null. PI-regulatoren er som i eksempel 20.2.

Vi har simulert med en sinusformet T_{sp}. Vi innrømmer at dette er et noe underlig scenario for vannkokeren, men vi har valgt et slikt settpunktsforløp for å vise hvordan foroverkopling kan virke, prinsipielt.

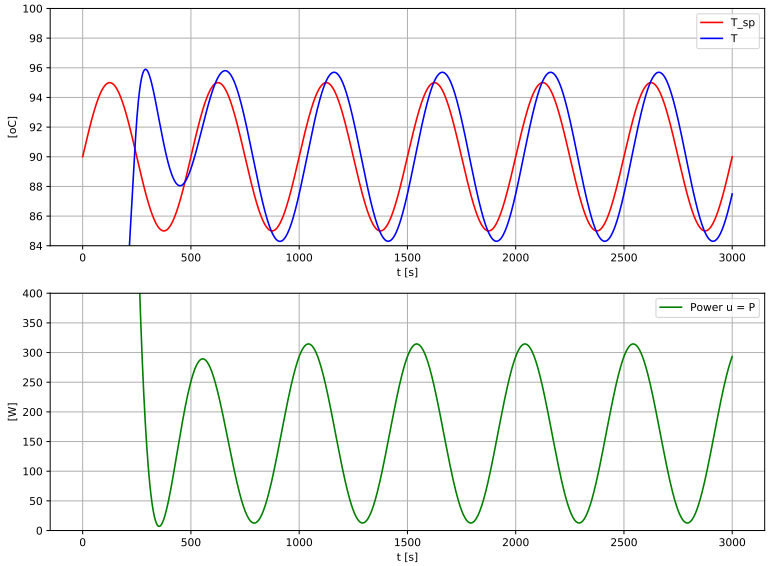
Figur 20.7 viser en simulering med uten foroverkopling – kun med tilbakekopling med PI-regulator.

Figur 20.8 viser en simulering *med* foroverkopling og med tilbakekopling med PI-regulator.

Kommentarer til simuleringene:

- Se figur 20.7 (kun tilbakekopling): Reguleringsavviket er forskjellig fra null når det reguleres kun med tilbakekopling med PI-regulator (ingen foroverkopling). Settpunktsfølgingen er ikke perfekt.
- Se figur 20.8 (tilbakekopling og foroverkopling): Avviket fram til ca. 600 s skyldes at initialtilstanden på 20 grader er forskjellig fra settpunktet, og tilbakekoplingen kompenserer for dette initialavviket. Etter ca. 600 s er avviket så å si null, dvs. at det er perfekt settpunktsfølging – og det skal foroverkoplingen ha all ære for.
- Vi har ikke lagt inn modellfeil i disse simuleringene, dvs. at vi har brukt samme modell i foroverkoplingen som i modellen i prosessimulatoren. Det går an å teste hvor robust foroverkoplingen er overfor modellfeil ved å gjøre de to modellene forskjellige mht. en eller flere parameterverdier.

[Slutt på eksempel 20.4]



Figur 20.7: Simulering av vannkokerregulering *uten* foroverkopling – kun med tilbakekopling med PI-regulator.

20.4 Hvordan kjøre et regulatorprogram i sann tid

Hvis du skal bruke ditt regulatorprogram til å regulere et reelt fysisk system[5], *må* du sørge for at programmets while-løkke (eller for-løkke) *kjører i sann tid*. Dette kan realiseres på akkurat samme måte som for simulatorer som skal kjøre i sann tid, jf. kap. 15.8, der formel (15.72) blir:

$$t_{\text{cycle}} = dt \tag{20.14}$$

Vi har ingenting å tilføye ut over det.

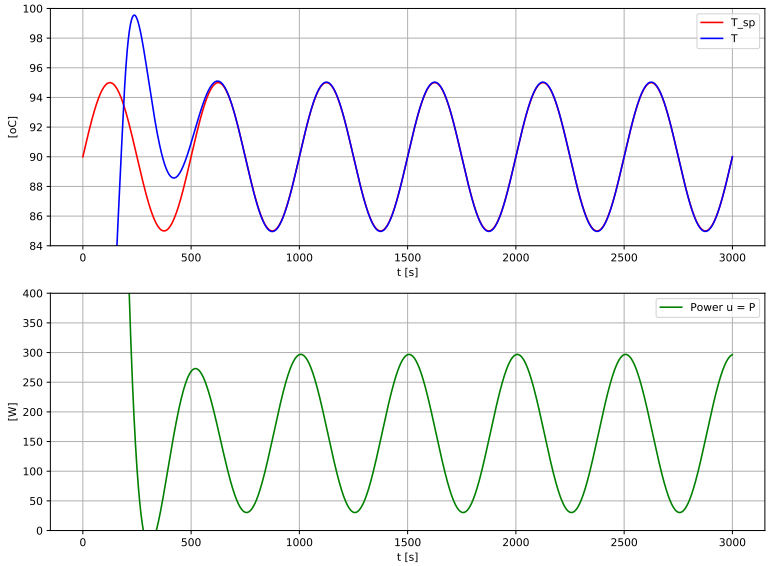
20.5 Oppgaver til kapittel 20

Oppgave 20.1 *Simulator for automatisk nivåregulering av tank*

I eksempel 15.1 i kap. 15.5 programmerte vi en simulator for en tank, nærmere bestemt et (innløps)magasin i et renseanlegg. I denne oppgaven skal du utvide simulatoren med en automatisk nivåregulator. Også det virkelige magasinet er styrt av en automatisk regulator, som er et program som kjører på en spesialdatamaskin for prosesstyring.[6]

[5] eller til å samle inn måledata, altså uten styring.

[6] Hos VEAS ressursgjenvinningsanlegg på Slemmestad er prosesstyringen implementert på et ABB 800xA styringssystem, som omfatter både programvaren og maskinvaren.



Figur 20.8: Simulering av vannkokerregulering *med* foroverkopling og med tilbakekopling.

Bakgrunnsinformasjon

Figur 20.9 viser et såkalt teknisk flytskjema av nivåreguleringssystemet. Figuren inneholder noen generelle reguleringstekniske begreper. I figur 20.9 representerer symbolet LT (Level Transmitter) nivåmåleren, og LC (Level Controller) nivåregulatoren. LT og LC (og f.eks. TT for temperaturmåler og TC for temperaturregulator) er vanlige symboler i tekniske flytskjemaer.[7]

Oppgaven til regulatoren (LC) er å holde nivået, h, i magasinet tilstrekkelig nær det ønskede nivået – nivåsettpunktet[8] h_{sp}, til tross for variasjoner i innstrømmen, som her betraktes som forstyrrelse på magasinnivået. Årsaker til at innstrømmen varierer, er nedbør, snøsmelting og avløp, som jo kommer som de kommer.

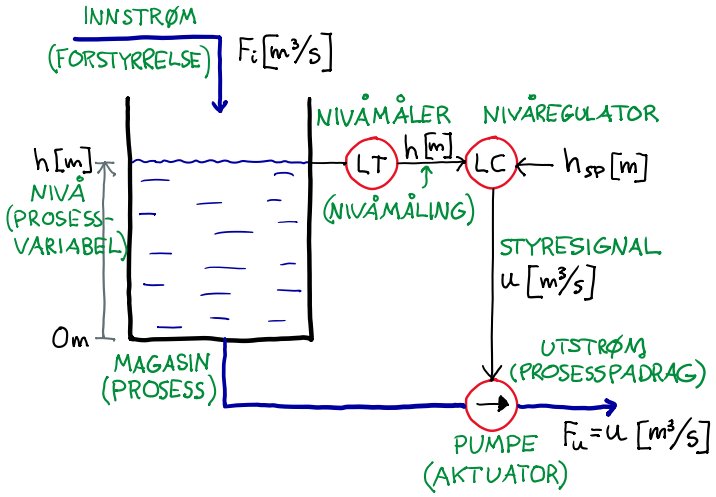
Hvordan virker nivåregulatoren? Den manipulerer pumpestrømmen F_u ut fra magasinet på basis av reguleringsavviket $e = h_{sp} - h$ der h_{sp} er nivåsettpunktet og h er nivåmålingen.

Vi skal anta at regulatoren er en PI-regulator som manipulerer F_u med styresignalet u til pumpa. Vi antar at

$$F_u = u \ [\text{m}^3/\text{s}] \tag{20.15}$$

PI-regulatorfunksjonen er beskrevet i kap. 20.3.2.

[7] Engelsk terminologi: Piping & Instrumation Diagrams (P&I Ds).
[8] Alternative betegnelser på nivåsettpunkt er nivåreferanse og nivå-skalverdi.



Figur 20.9: Nivåreguleringssystem for (avløps)magasinet. LT = Level Transmitter. LC = Level Controller.

Oppgave a

Tegn et blokkdiagram av nivåreguleringssystemet (samme type blokkdiagram som i figur 20.1).

Oppgave b

Ta utgangspunkt i program 15.2, som implementerer en simulator for (den uregulerte) tanken. Modifiser programmet iht. følgende spesifikasjoner:

- Nivåregulator med PI-regulator (jf. kap. 20.3.2).
- Regulatorinnstilling: $u_{\text{man}} = 3$ m^3/s. $K_p = -2$ (m^3/s)/m. $T_i = 2000$ s.
- Pådragsgrenser: $u_{\min} = 0$ m^3/s. $u_{\max} = 8$ m^3/s.
- Simuleringsparametre: Tidsskritt $dt = 1$ s. $t_{\text{start}} = 0$ s. $t_{\text{stop}} = 12000$ s.
- Initialtilstand: $h_{\text{init}} = 2$ m.
- Tilstandbegrensning: $h \in [0 \text{ m}, 4 \text{ m}]$.
- Settpunkt: $h_{\text{sp}} = 2$ m.
- Forstyrrelse (innstrøm) F_i: Fra $t = 0$ til 2000 s: 3 m^3/s. Endring som sprang til 4 m^3/s ved $t = 2000$ s.
- Plotting: h_{sp} og h sammen i et subplott, og F_i og $F_u = u$ sammen i et annet subplott.

Kjør en simulering. Hvor mye (ca) avviker h fra h_{sp} på det meste? Går h mot h_{sp} stasjonært?

Oppgave c

Inn til det virkelige innløpsmagasinet strømmer den såkalte vaskevannsreturen fra renseanlegget, som er vann som brukes til regelmessig vask av lecakule-formede materiale som fungerer som «hjem» for mikroorganismer som utfører biologiske rensing av avløpsvannet. Vaskevannsreturen er en kraftig innstrøm til magasinet, som kan variere med en amplitude på opp til 0,5 m^3/s, altså total variasjon innenfor 1 m^3/s med en periode på ca. 1000 s. Det er viktig for driften av renseanlegget at det nivåregulerte magasinet *demper* denne sinusformede innstrømmen, altså at F_u viser mye mindre variasjon enn F_i.

Anta at F_i er sinusformet slik:

$$F_i(t) = B_0 + A_0 \sin(2\pi f_0 t) \tag{20.16}$$

med: Bias $B_0 = 3$ m^3/s. Amplitude $A_0 = 0{,}5$ m^3/s. Periode $P_0 = 1000$ s, som gir frekvens $f_0 = 1/P_0$ [Hz].

Implementer den sinusformede F_i i programmet, og kjør en simulering. Er F_u tydelig dempet i forhold til F_i? Hvis ja, har du laget/simulert et velfungerende nivåreguleringsssystem for innløpsmagasinet!

Oppgave 20.2 *Foroverkopling i temperaturreguleringssystem for vannkoker*

I eksempel 20.4 utviklet vi en foroverkopler for temperaturreguleringssystemet for vannkokeren. I simuleringene med program http://techteach.no/python/files/prog_kettle_fb_and_ff.py ble temperatursettpunktet T_{sp} variert som et sinussignal. I denne oppgaven skal T_{sp} være konstant, mens romtemperaturen T_{room} skal varieres.

Modifiser program http://techteach.no/python/files/prog_kettle_fb_and_ff.py som følger:

- Simuleringens stopptid: 1500 s.
- $T_{sp} = 90$ grader (konstant).
- $T_{\text{init}} = 90$ grader.
- T_{room} endres som et sprang fra 20 grader til 30 grader ved $t = 1000$.
- T_{room} plottes i et eget subplott.
- Bruk liten nok skala langs y-aksen i subplottet for T_{sp} og T slik at du ser responsen tydelig (du kan egentlig droppe å sette y-skalaen slik at det blir automatisk skalering).

Kjør to simuleringer slik:

- En simulering uten foroverkopling, men med tilbakekopling (med PI-regulator)
- En simulering med både foroverkopling og tilbakekopling

Kommenter simuleringene!

Oppgave 20.3 *Skalert sanntids simulering av temperaturreguleringssystem for vannkoker*

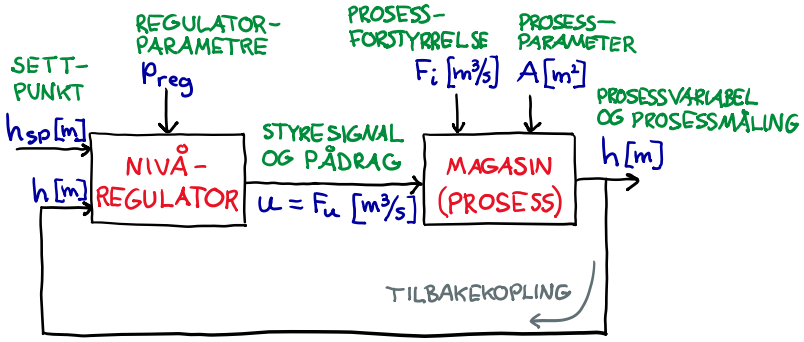
Program 20.2 i eksempel 20.2 implementerer en simulator for et temperaturreguleringssystem med PI-regulator for en vannkoker. Modifiser programmet slik at simulatoren kjører i skalert sann tid og plotter dataene kontinuerlig. Du kan følge oppskriften for simulering i skalert sann tid i kap. 15.8. Mer spesifikt: Se program 15.3. Tips: Du kan øke simulatorens tidsskritt dt betydelig fra 0,1 s, som er brukt i program 20.2. Du kan prøve f.eks. $dt = 10$ sek.

20.6 Løsninger til kapittel 20

Løsning til oppgave 20.1

Løsning til oppgave a

Figur 20.10 viser et blokkdiagram av nivåreguleringssystemet.



Figur 20.10: Blokkdiagram av nivåreguleringssystemet.

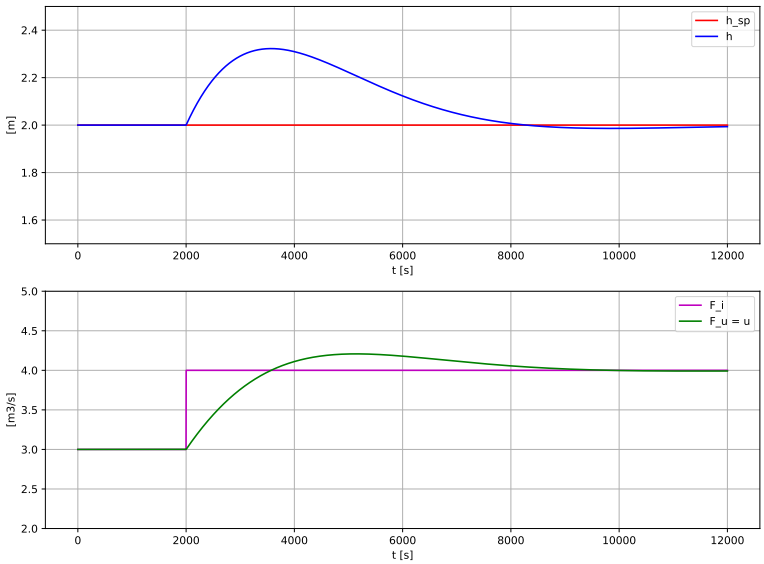
Løsning til oppgave b

Program http://techteach.no/python/files/prog_sim_pi_level_control_tank.py implementerer en simulator av nivåreguleringssystemet.

Figur 20.11 viser resultat av en simulering.

Med koden np.max(h_array) utført f.eks. i konsollen finner vi at h avviker fra h_{sp} med 0,32 m på det meste.

h går mot h_{sp} stasjonært.



Figur 20.11: Simulering av magasin med PI-regulator ved sinusformet innløp.

Løsning til oppgave c

Simulatorprogrammet blir lik dette programmet:
http://techteach.no/python/files/prog_sim_pi_level_control_tank.py, bortsett fra at F_i er sinusformet. F_i kan kodes slik:

```
F_i_k = 3 + 1*np.sin(2*np.pi*(1/1000)*t_k)
```

Simulatorprogrammet er tilgjengelig her:
http://techteach.no/python/files/prog_sim_tank_sinus_inflow.py.

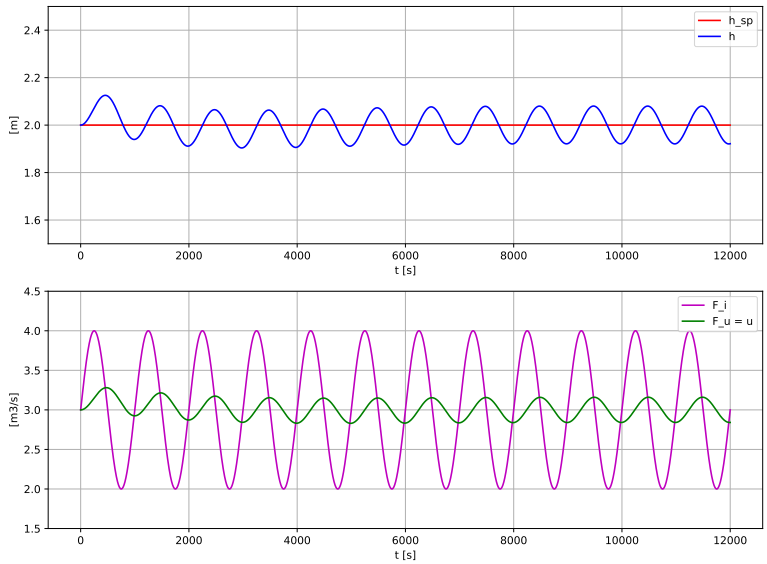
Figur 20.12 viser en simulering. I det nederste plottet ser vi at F_u er tydelig dempet i forhold til F_i. Nivåreguleringssystemet fungerer godt!

Løsning til oppgave 20.2

Program http://techteach.no/python/files/prog_kettle_fb_and_ff_troom.py viser en løsning.

Figur 20.13 viser en simulering uten foroverkopling, men med tilbakekopling med PI-regulator.

Figur 20.14 viser en simulering *med* foroverkopling og med tilbakekopling med PI-regulator.



Figur 20.12: Simulering av nivåregulert magasin med PI-regulator ved sinusformet innløp.

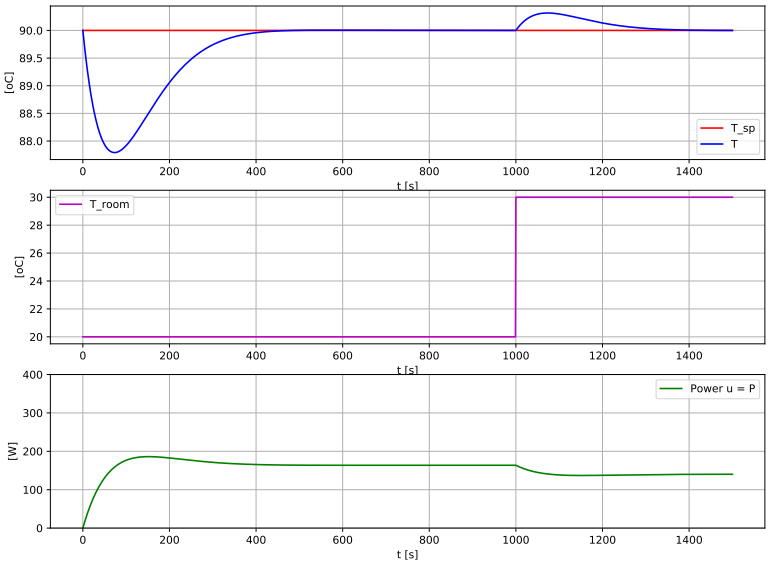
Kommentarer til simuleringene:

- Se figur 20.13 (kun tilbakekopling). Reguleringsavviket er forskjellig fra null rett etter start og etter at T_{room} er endret. Avviket er riktignok ikke så stort, men det er tydelig med den valgte skalaen i plottet.
- Se figur 20.14 (både foroverkopling og tilbakekopling). Avviket er nå null hele tiden, til tross for at prosessforstyrrelsen er endret som et sprang – takket være foroverkoplingen.

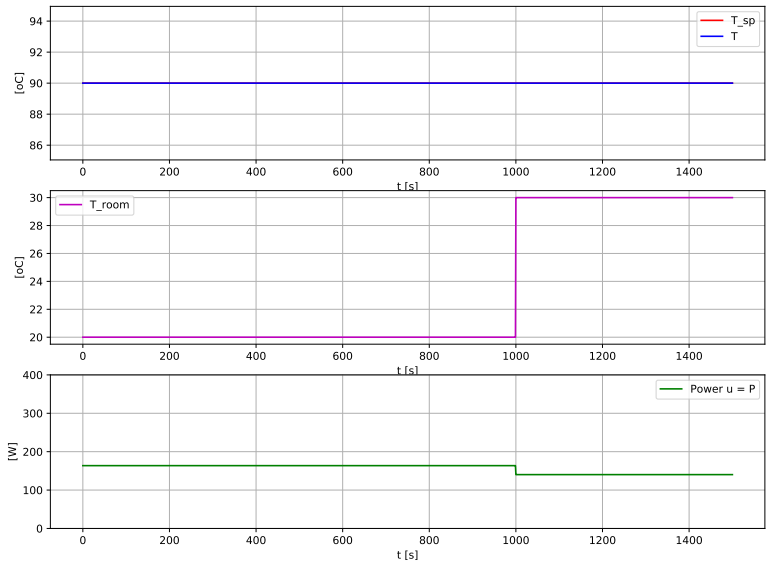
Løsning til oppgave 20.3

Program http://techteach.no/python/files/prog_realtime_pi_control_kettle.py implementerer en skalert sanntids simulator av temperaturreguleringssystemet for vannkokeren.

Den simulerte responsen (etter at simuleringen er ferdig) er identisk med responsen vist i figur 20.3 og vises derfor ikke her.



Figur 20.13: Simulering av vannkokerregulering *uten* foroverkopling, men med tilbakekopling med PI-regulator.



Figur 20.14: Simulering av vannkokerregulering *med* foroverkopling og med tilbakekopling.