

# Compulsory assignment 6

Aleksander Hykkerud

## Exercise 1: Kalman filter

The system to be estimated is a filling tank system as shown below in Figure 1.

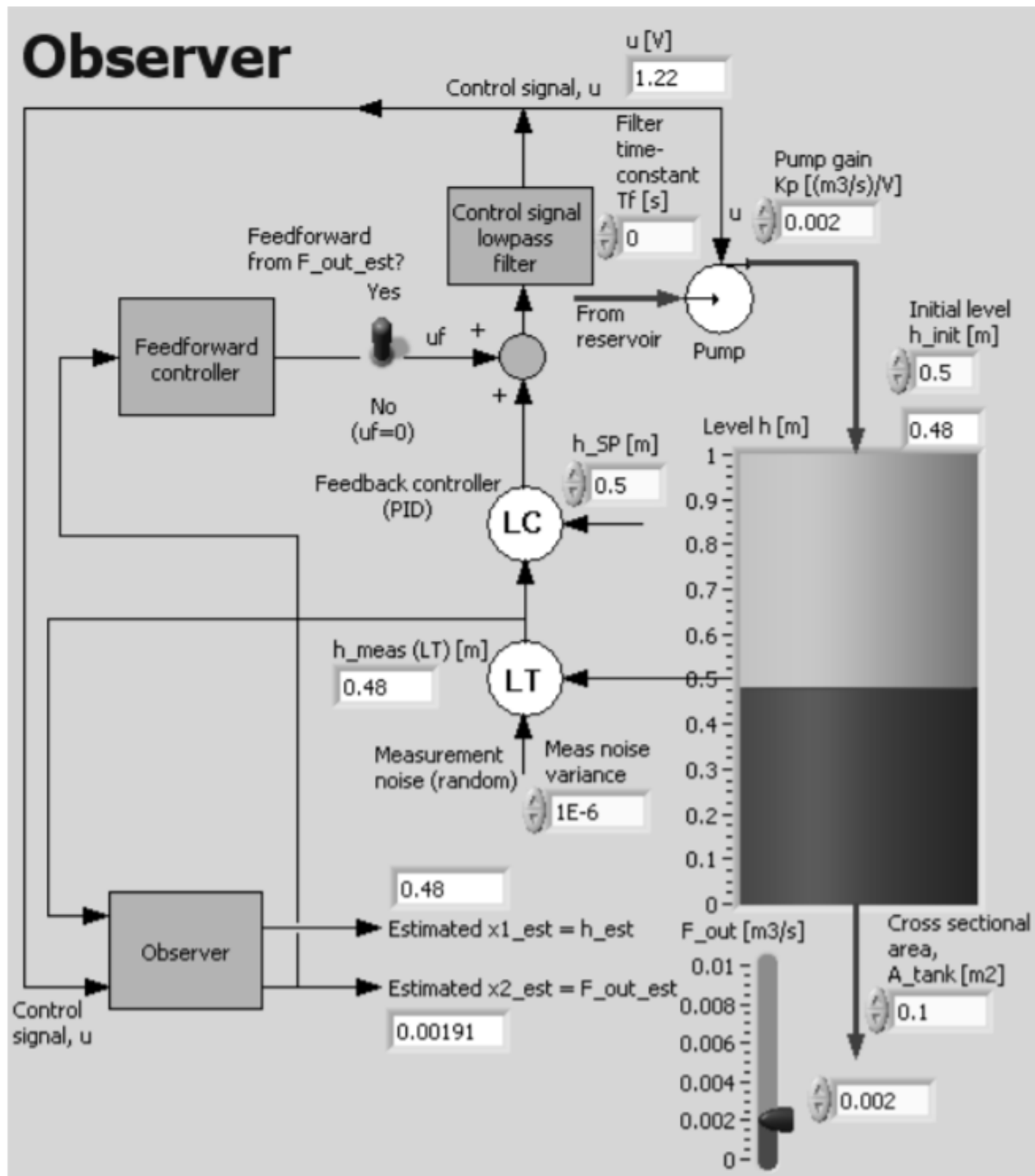


Figure 1: filling tank system (Advanced dynamics and control)

The parameter to be controlled in this system is the water level in the tank  $h$ . This height can be modelled as show below.

$$\frac{\delta h}{\delta t} = \frac{1}{A_{tank}} [K_p u - F_{out}(t)]$$

With  $F_{out}$  given as a constant

$$\frac{\delta F_{out}}{\delta t} = 0$$

The variables can be written in standard form as follows

$$\begin{aligned} x_1 &= h \\ x_2 &= F_{out} \\ \frac{\delta x_1}{\delta t} &= \frac{1}{A_{tank}} [K_p u(t) - x_2(t)] \\ \frac{\delta x_2}{\delta t} &= 0 \end{aligned}$$

A simulator was written in Matlab using an Euler forward implementation of the model as shown below.

$$\begin{aligned} x_1(k+1) &= x_1(k) + \frac{T_s}{A_{tank}} [K_p u(k) - x_2(k)] \\ x_2(k+1) &= x_2(k) \end{aligned}$$

The system was implemented with a PI controller. The P and I gains  $T_i$  and  $K_c$  were choose as 10 and showed good stability. A simple simulation with a step response is shown below in Figure 2.

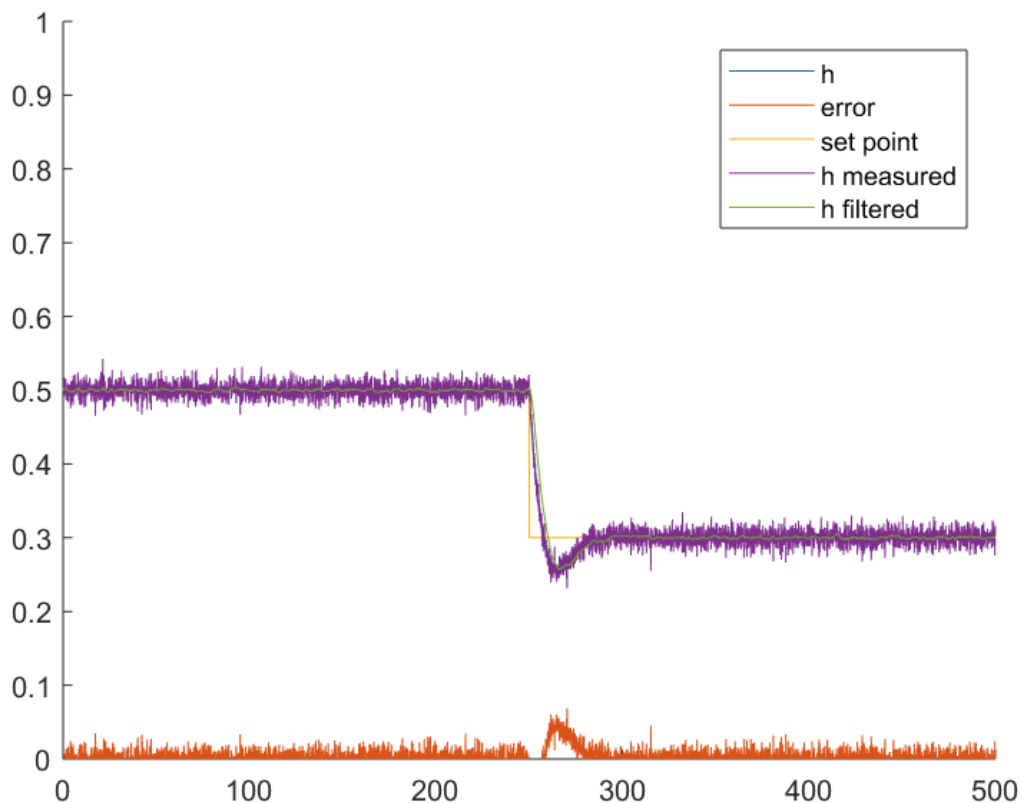


Figure 2: matlab simulator of filling tank system

To get the kalman filter was initiated with an initial guess of  $x_{p_k} = 0.5$ . The observer then calculated the predicted measurement  $y_{p_k}$ , the innovation variable  $e$  and the corrected state estimate  $x_{c_k}$ . Once the corrected state has been calculated the prediction for the next step  $x_{p_{k+1}}$  is calculated. The process is repeated for the entire simulation, but using  $x_{p_{k+1}}$  values in the calculation of  $x_c$  for the next step.

The kalman filter requires the Kalman gain to function, which is calculated using the following information


$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.0001 \end{bmatrix}$$


$$R = 0.0001 \text{ m}^2$$

$$A = \begin{bmatrix} 1 & -\frac{T_s}{A_{tank}} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

$$G = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$C = [1 \ 0]$$

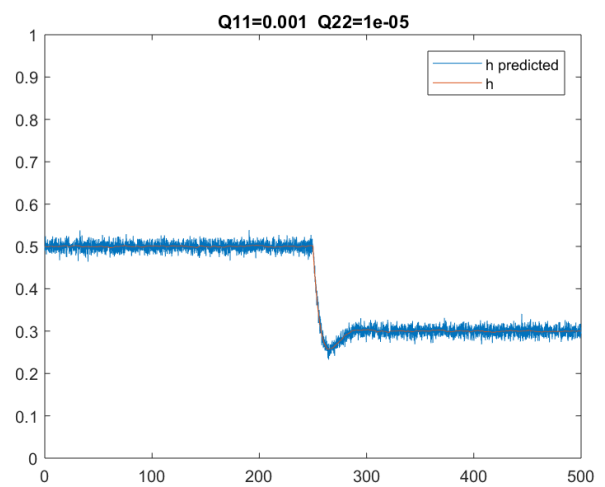
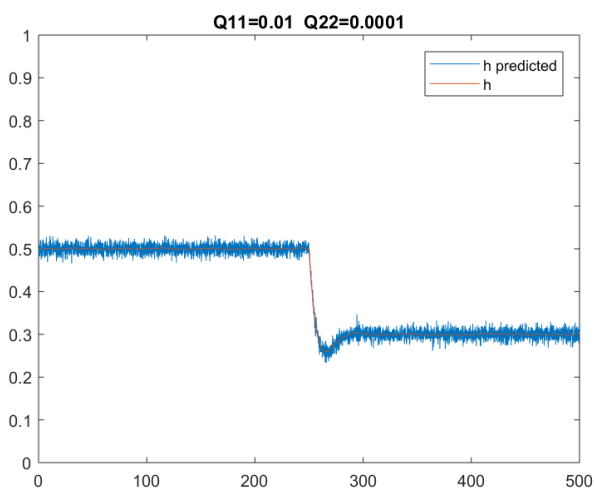
The Kalman gain was calculated using the following iterative process 1000 steps with the initial  $P_p$  set as the 2x2 identity matrix. 

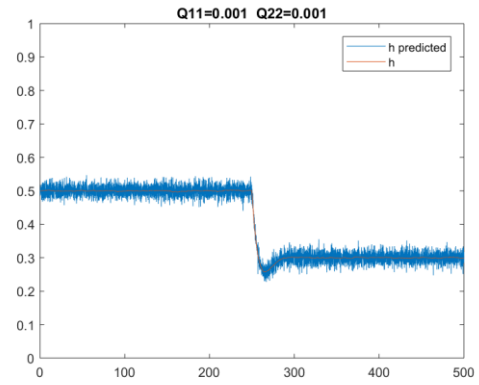
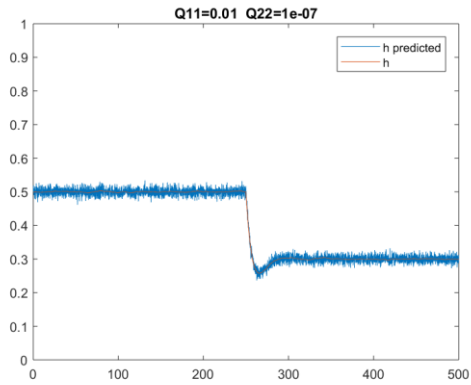
$$K(k) = \frac{P_p(k)C^t}{[CP_p(k)C^t + R]} \quad \text{$$

$$P_c(k) = [I - K(k)C]P_p(k)$$

$$P_p(k + 1) = AP_c(k)A^t + GQG^t$$

The simulator was run for different values of Q





It appears that the higher the Q22 is, the more noise comes through to the correction.

### Solution script

```
% simulation parameters
numsteps = 5000;
dt = 0.1; % seconds
sensor_noise = 0.01;
a = 0.05;

%PI parameters
Kc = 10;
Ti = 10;
Td = 0;
PI_max = 5;
PI_min = -5;

%Process initials
u = 0;
h = 0.5;
h_filtered = 0.5;
h_max = 1;
h_min = 0;
F_out = 0.005; % m^3/s
bias = F_out;
set_point = 0.5;
Kp = 0.002;
A_tank = 0.1; % m^2
error_sum = 0;

% kalman initials
hp0 = h; % predicted h
Fp0 = 0;
hp = hp0;
Fp = Fp0; % kalman starting outflow
% steady state kalman gain K
Q0 = 1;
R = 0.0001;
A = [1 -dt/A_tank
      0 1];
G = [1 0
      0 1];
C = [1 0];
Q = [0.001 0
      0 0.001]*Q0;

Pp0 = eye(2); % initial Pp0
K = Pp0*transpose(C) / (C*Pp0*transpose(C)+R);
Pc = (eye(2)-K*C)*Pp0;
Pp = A*Pc*transpose(A)+G*Q*transpose(G);
K = Pp*transpose(C) / (C*Pp*transpose(C)+R);
for x = 1:1000
    K = Pp*transpose(C) / (C*Pp*transpose(C)+R);
    Pc = (eye(2)-K*C)*Pp;
    Pp = A*Pc*transpose(A)+G*Q*transpose(G);
```

```

end
%K = [0.9 -0.1];
% simulation history
t = ones(1,numsteps);
F_out_arr = zeros(1,numsteps);
F_out_arr = F_out_arr+F_out;
h_arr = zeros(1,numsteps);
h_measured_arr = zeros(1,numsteps);
h_filtered_arr = zeros(1,numsteps);
perror_arr = zeros(1,numsteps);
set_point_arr = zeros(1,numsteps);
set_point_arr(:) = set_point;
hp_arr = zeros(1,numsteps);

% set steps in setpoint
set_point_arr(numsteps/2:end)=0.3;

for step = 1:numsteps
    t(step)=step*dt;

    % sensor fetch
    h_measured = h + normrnd(0,sensor_noise*h_max);
    h_measured_arr(step)=h_measured;
    % filter
    h_filtered = (1-a)*h_filtered + a*h_measured;
    h_filtered_arr(step)=h_filtered;

    % process error
    perror = set_point_arr(step)-h_measured;
    perror_arr(step) = perror;
    % PI(D)
    P = Kc*perror;

    error_sum = error_sum+perror;
    I = Kc/Ti*error_sum*dt;

    u = P+I;

    % kalman observer
    % predicted measurement
    h_measured_predicted = hp; % predicted measured h

    % innovation variable
    innovation = h_measured - h_measured_predicted;

    % corrected state estimate
    hp_arr(step) = hp;
    hc = hp+K(1)*innovation;
    Fc = Fp+K(2)*innovation;

    % predicted state estimate for next
    hp = hc+(dt/A_tank)*(Kp*u-Fc);
    Fp = Fc;

    % change reality
    F_in = bias + u*Kp;
    dh = (1/A_tank)*(F_in-F_out)*dt;
    h = h+dh;
    h_arr(step) = h;
end

figure()
plot(t,hp_arr,t,h_arr)
legend('h predicted','h')
title('Q11='+string(Q(1))+ ' Q22='+string(Q(4)))
ylim([0 1])

```

## Kalman filter b.

The model can start out with the same physical relationship as the the one in a. with dh as a starting point.

$$\frac{\delta h}{\delta t} = \frac{1}{A_{tank}} [K_p u - F_{out}(t)]$$

But in this case the Fout isn't constant.

$$F_{out} = K_v * f(z) * \sqrt{\frac{p}{G}}$$

$$\frac{\delta F_{out}}{\delta h} = K_v * \sqrt{\frac{p(h)}{G}}$$

$$\frac{\delta p}{\delta h} = P_k \frac{\delta h}{dt}$$



Where Pk is a constant related to the hydrostatic pressure of the water above the outlet. The model will not be linear as the parameters become dependent of eachother (dh is fed back from itself due to Fout increasing with increasing h).

## 2 Moving Horizon estimation

Unable to figure out how to modify this script to estimate the K instead of the d.

## 3 Model predictive control

### a. Block diagram

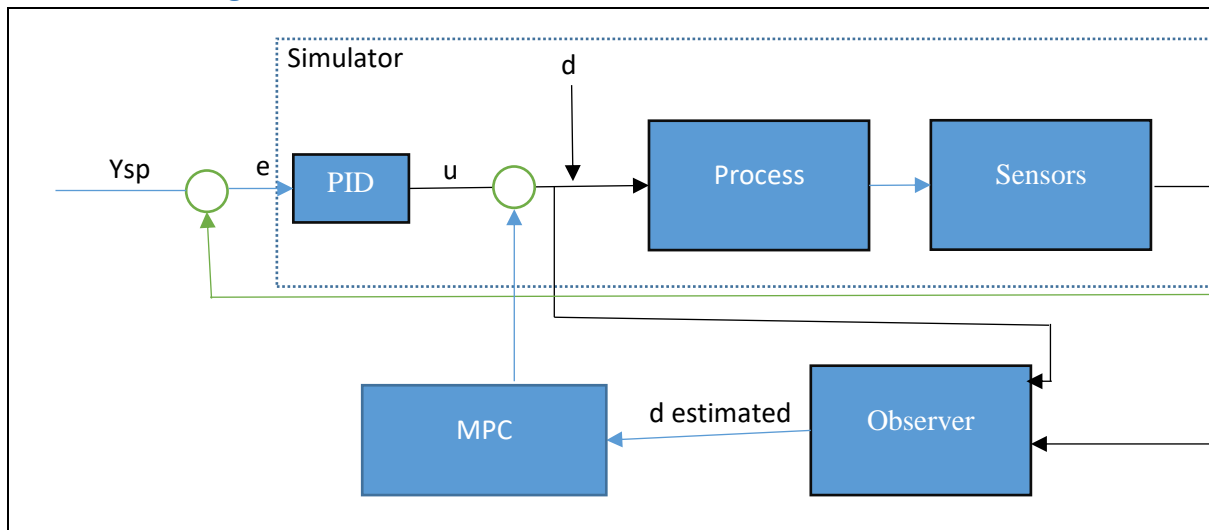


Figure 3: block diagram of heater simulator with MPC

### b. Running the script

The script runs. The initial control is poor, but this is probably due to startup conditions (heater takes some time to heat up) and from the fact that the MPC needs some time to be able to estimate the parameters correctly.

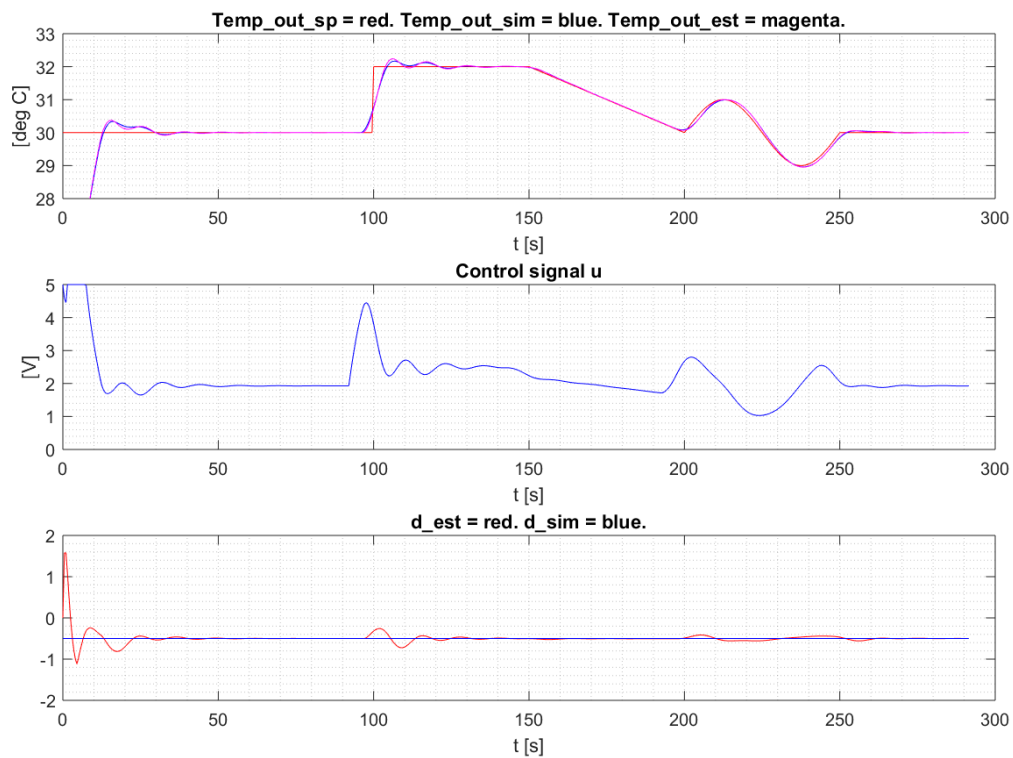


Figure 4: Simulator with MPC



### c. Understanding

The script creates a horizon to look at (16 steps). The horizon is used to predict the behavior of the system and use this to both calculate an optimal control signal and estimate the disturbances. It creates fixed step point changes that are saved in arrays for plotting and simulation purposes.

For each simulation step the script uses `fmincon` to minimize the error using `u` based on simulations of a horizon giving an estimate of the disturbance. To save calculation the previous solution is used as the initial guess for the next step.

### d. Playing around

By reducing the horizon time to 2, the simulation became marginally unstable.

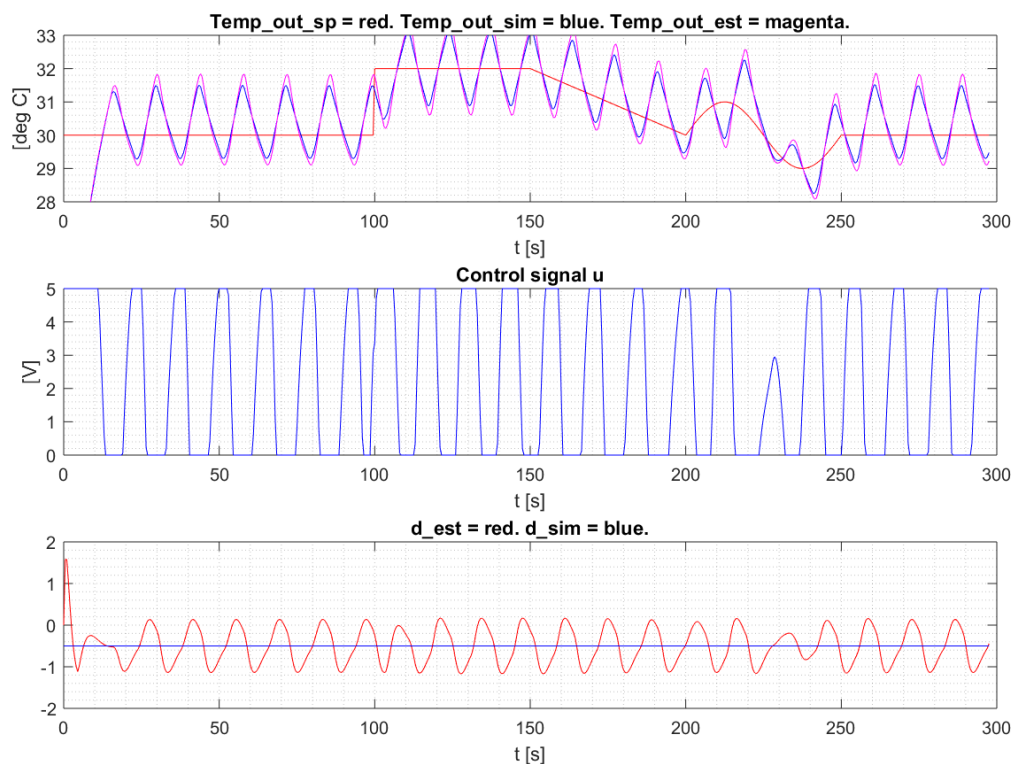


Figure 5: Short horizon experiment

Increasing the horizon yielded stable results and great disturbance estimation, but highly increased the computation time of the simulation. The system also seems to lose some responsiveness as well, taking longer to reach setpoint than under a horizon of 8 seconds.

```
%Time settings:
```

```
Ts = 0.5; %Time-step [s]  
t_pred_horizon = 14;
```

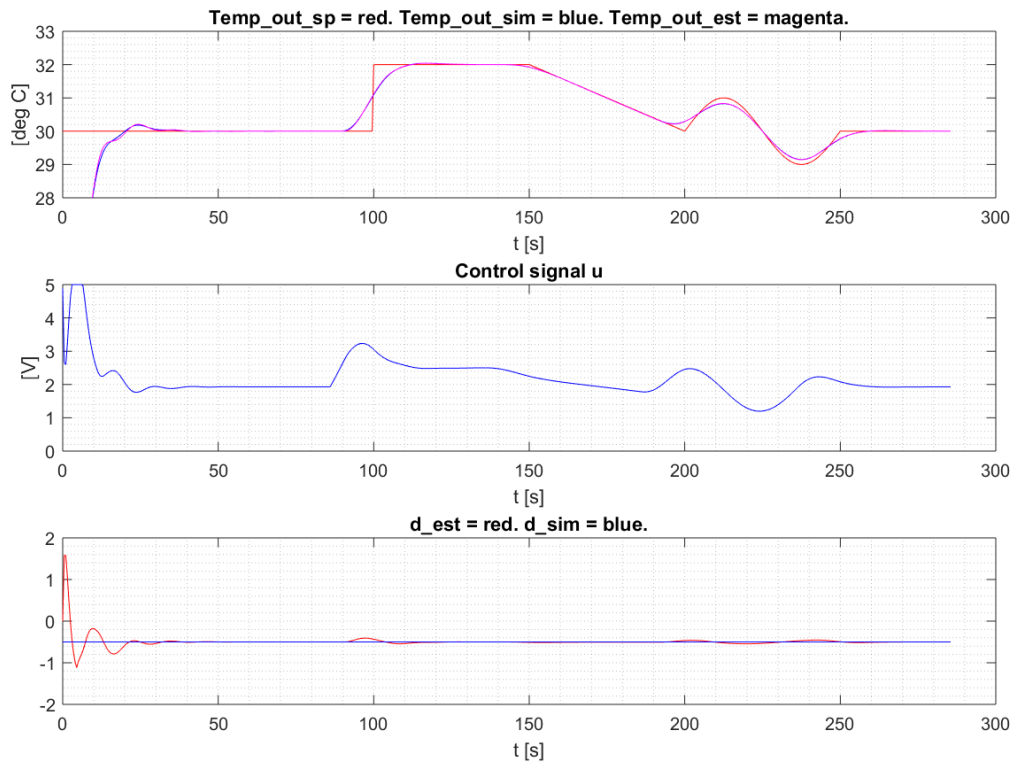


Figure 6: Long horizon experiment

### e. Adding gain

I added a gain of +4.

```
gain = model_params.gain+4;
theta_const = model_params.theta_const;
theta_delay = model_params.theta_delay;
```

By adding gain to the function, the control signal and the setpoint tracking is different than what we get previously, but the program still tracks the disturbances quite well.



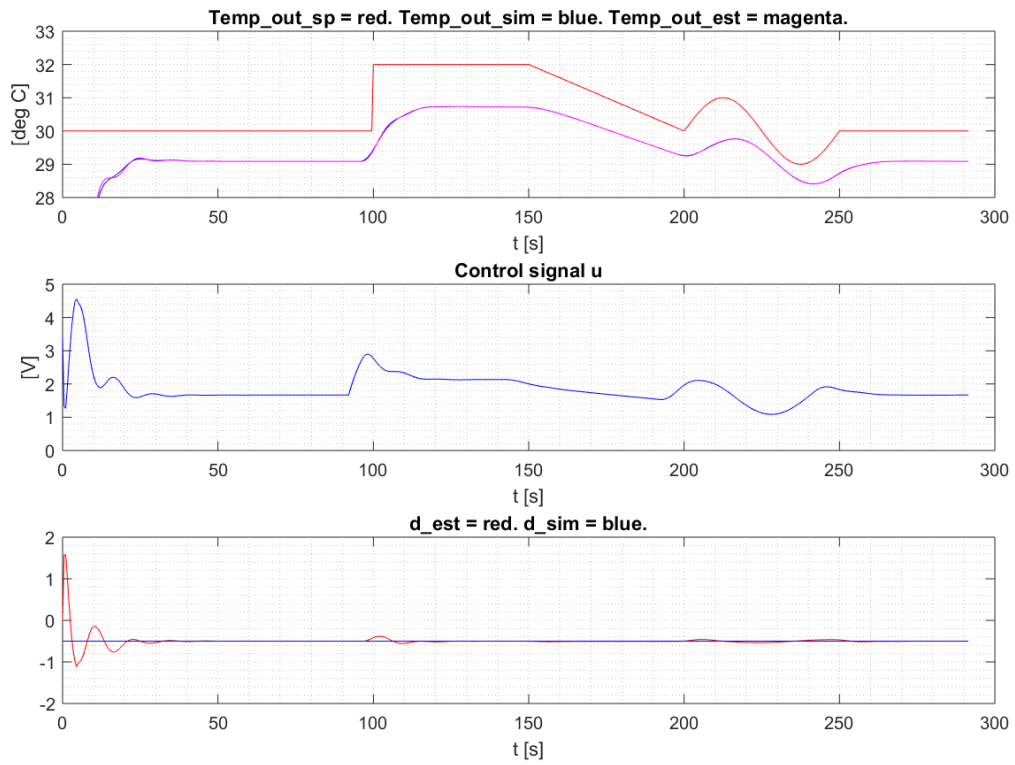


Figure 7: added gain simulation results