Model Predictive Control - Assigment

```
% Clearing variables
clearvars
close all
```

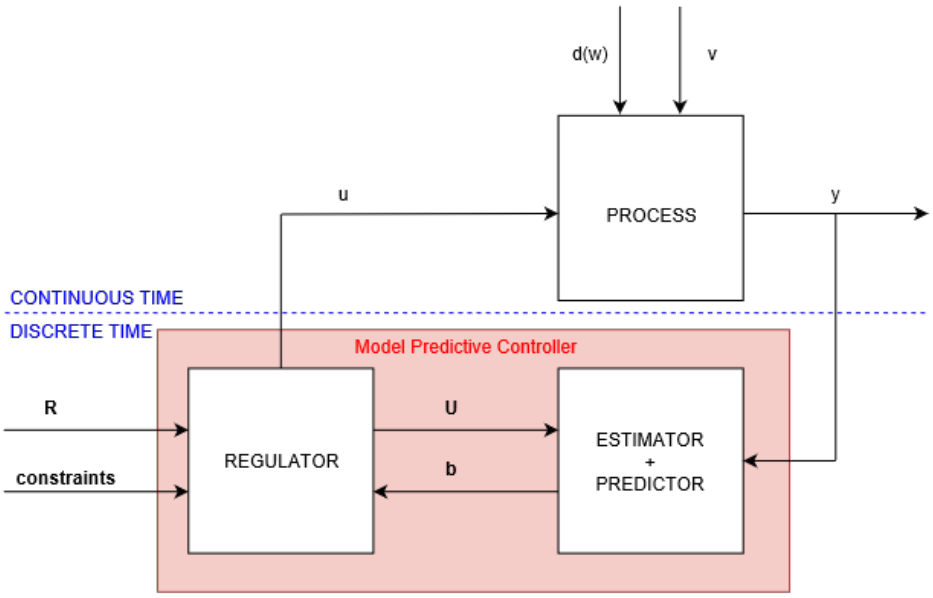
Problem 1 - Control Structure

Question 1.1

The states are the masses, whih are changed by inlet and outlet mass flows. The manipulated variables (inputs) are the mass flows that enter the distribution valve. The disturbance variables (disturbances) are the mass flows that enter the third and the fourth tank. The measured variables (measurements) are the heights of the water levels in the tanks. The controlled variables (outputs) are the height in tank 3 and tank 4.

Question 1.2

The block diagram of the system is below. The bold letters indicate horizonwise stackings. As we can see, everyhting that the computer does is in discrete time. The interactions with the process happen time-instant wise, but the remaining interactions of the regulator and the predictor are stacked: the constraints, the reference R, the input U, and the zero-input transient b. Note that the red part of the system is linear as well, therefore it only interacts correctly with the process, when that is close to the linearization point. The process is nonlinear, and it is measured by signal y, corrupted by measurement noise v. There are inputs that cannot be manipulated by us, they are called disturbances d. These disturbances have deterministic and stochastic parts, the latter is denoted by w.



Problem 2 - Deterministic and Stochastic Nonlinear Modelling

The following steps are done to show two parameter sets as being nonminimal phase and minimal phase. The certain functions will be discussed in a deeper approach along the documentation.

The selected operating point is the one enforced on the system by a steady state input combination of

```
% Choice for valve division
gamma_minphase = [0.65; 0.55];
```

Then an instance of the nonlinear plant object is created.

```
% First we make an instance of the four tank system object.
fts_minphase = FourTankSystem;
% Initial values of states
initials.m = zeros(4,1);
% Options of the solver
ODEoptions = [];
```

Initialization of the instance.

```
% Initialization of the Four Tank System object
fts_minphase.initialize(gamma_minphase,initials,ODEoptions);
```

A nonminimal phase version of the system is created then by choosing new gamma values:

```
% Non minimal phase version
fts_nonminphase = copy(fts_minphase);
gamma_nonminphase = [0.65; 0.1];
fts_nonminphase.gamma = gamma_nonminphase;
```

Then the steady state is calculated (the derivatives are zero).

```
% The steady state input signals
ssInput  = [250; 325; 0; 0];
% The initialization guess for the steady state solution
initialGuess = [5000; 5000; 5000; 5000];
% Calculation of the steady state
fts_minphase.steadyState(ssInput,initialGuess)
```

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>
```

```
fts_nonminphase.steadyState(ssInput,initialGuess)
```

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
```

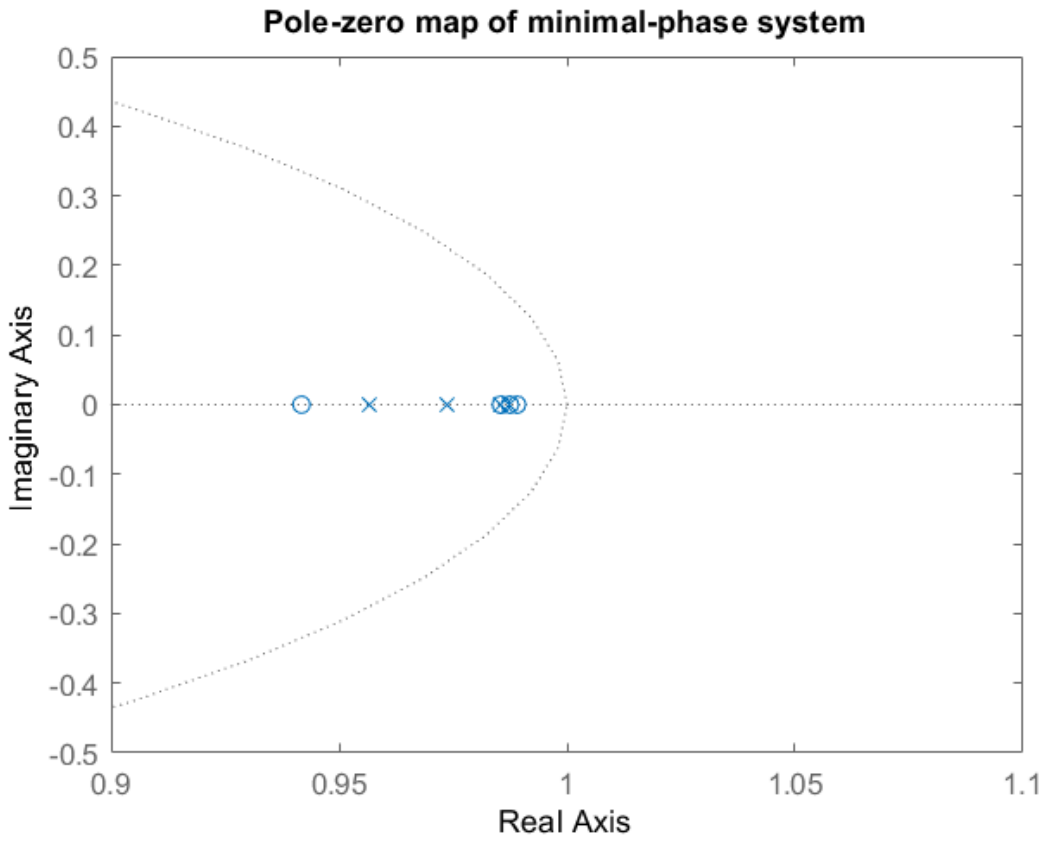
```
<stopping criteria details>
```

Linearizing, then discretizing the system with sample time of 1 second.

```
% Sample time
Ts = 1;
% Linearization and discretization
fts_minphase.linearize;
fts_nonminphase.linearize;
fts_minphase.discretize(Ts);
fts_nonminphase.discretize(Ts);
```

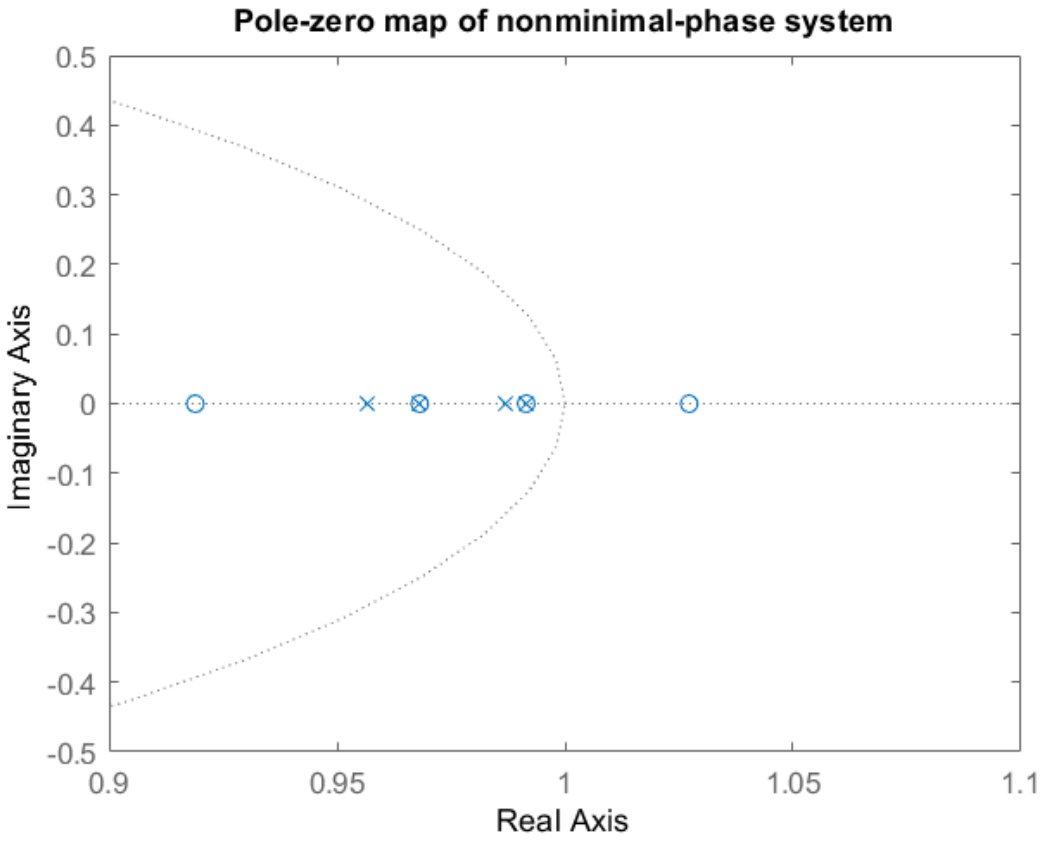
Plotting the poles and the zeros.

```
% Poles and zeros
fighand = figure;
pzmap(fts_minphase.tfd)
xlim([0.9 1.1])
title('Pole-zero map of minimal-phase system')
saveas(fighand,[pwd '\figures\f01_minphase'],'png')
```




```
fighand = figure;
pzmap(fts_nonminphase.tfd)
xlim([0.9 1.1])
```

```
title('Pole-zero map of nonminimal-phase system')
saveas(fighand,[pwd '\figures\f02_nonminphase'],'png')
```



We can see that the second parameter set (where the inverse of the discrete transfer function is unstable, in other words, it is nonminimal phase) has discrete zeroes outside the unit circle. Since the continuous models do not have zeros, this was caused by discretization. This means that if the input is applied as a zero-order-hold, there is a certain frequency, around which the system will produce significant attenuation.

From now on, the minimal phase system is used.

Question 2.1.1

According to the mass balance equation, for a given control volume, the change of the mass in the control volume is going to be the sum of the inflows and the outflows.

$$\dot{m} = \sum_i \rho q_i$$

The convenience is that the volume inflows have positive and the outflows have negative signs.

The valve outflows are given by the division of the flow done by the valve, which is described by a constant. Therefore the two outflows will have the values

$$q_{out,1} = \gamma F$$
$$q_{out,2} = (1-\gamma) F$$

where F is one of the manipulated variables.

Outflows of the tanks are described by Bernoulli law and the inner states.

$$q = a\sqrt{2gh}\,, \text{where} \quad h = \frac{m}{\rho A}$$

Therefore the first order state equations can be described as

$$\frac{dm_1}{dt} = \rho\gamma_1 F_1 + \rho a\sqrt{2g\frac{m_3}{\rho A}} - \rho a\sqrt{2g\frac{m_1}{\rho A}}$$

$$\frac{dm_2}{dt} = \rho\gamma_2 F_2 + \rho a\sqrt{2g\frac{m_4}{\rho A}} - \rho a\sqrt{2g\frac{m_2}{\rho A}}$$

$$\frac{dm_3}{dt} = \rho(1-\gamma_2) F_2 - \rho a\sqrt{2g\frac{m_3}{\rho A}} + \rho F_3$$

$$\frac{dm_4}{dt} = \rho(1-\gamma_1) F_1 - \rho a\sqrt{2g\frac{m_4}{\rho A}} + \rho F_4$$

Question 2.1.2

The measurement equations for the tanks are the following in the deterministic case:

$$h_i = \frac{m_i}{\rho A}$$

Question 2.1.3

The output equations are

$$h_1 = \frac{m_1}{\rho A}$$

$$h_2 = \frac{m_2}{\rho A}$$

Question 2.2.1

This time we have the same state equations as the ones developed in Question 2.1.1, but with more information about the (deterministic) disturbances: they are piece-wise constant.

$$F_3(t) = F_{3,k} \quad \text{for} \quad t_k \le t < t_{k+1}$$
$$F_4(t) = F_{4,k} \quad \text{for} \quad t_k \le t < t_{k+1}$$

Question 2.2.2

There is Gaussian noise on the measurement signals. It can be modelled such that

$$h_i = \frac{m_i}{\rho A} + v(t), \text{where} \quad v(t) \sim N(0, R_{vv})$$

Question 2.2.3

The output equations have not been changed with regard to the deterministic equations.

Question 2.3.1

Having the F_3 and F_4 as Brownian motion, we can get the elements $\rho F_i dt$ in the equation of dt. This means that the differential of the state is going to be proportial to the Brownian motion $F_i \sim N(F_{i,k}, R_{ww})$

Question 2.3.2

The output equations have not been changed compared to Question 2.2.2.

Question 2.3.3

The output equations have not been changed compared to Question 2.2.3.

Problem 3 - Nonlinear Simulation - Step Responses

In the assignment, the use of objects was enhanced to increase readability and reduce the amount of codes.Every object has an initialization and a usage function, the latter usually has different names to fit the functionality. In case of the FourTankSystem object, it is timestep, which integrates the process function for the given time interval. The object can also calculate steady state, linearize, discretize, or add noise to the process or the measurement and then discretize that. In case of reusing the object, reinitialization must be applied. Furthermore, the object can only simulate the nonlinear system.

Question 3.1

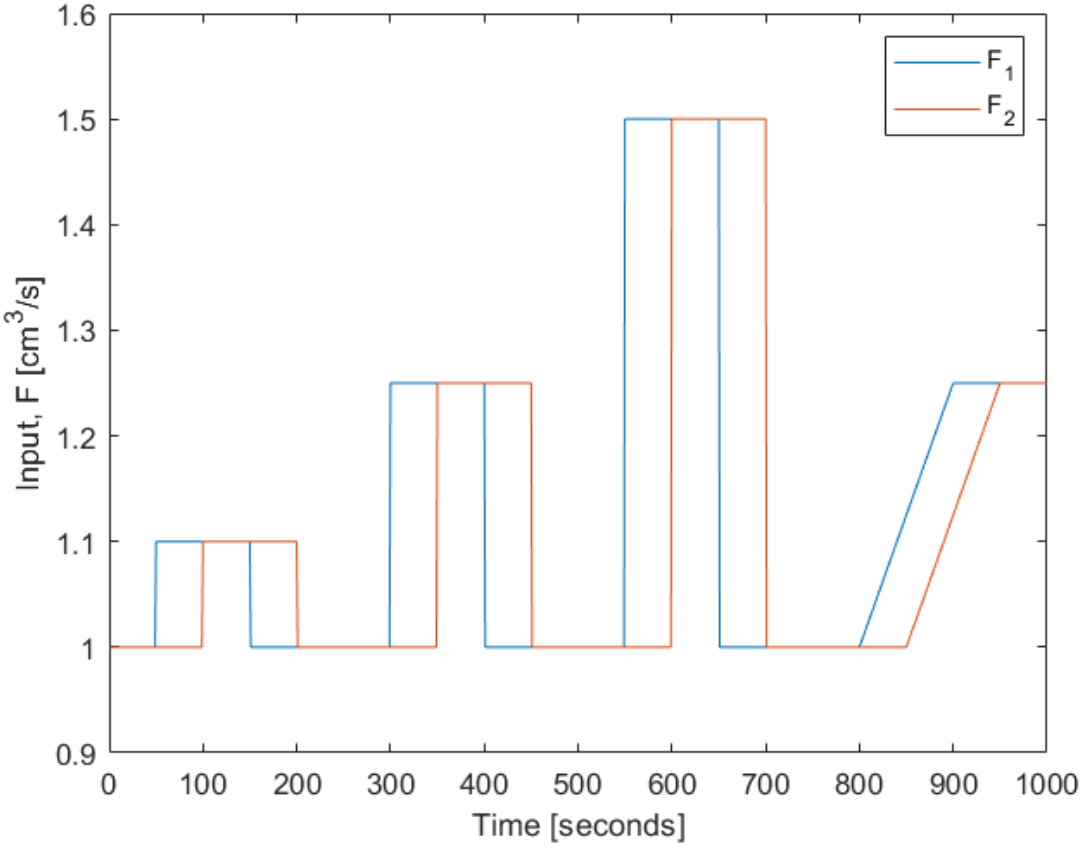
First the records are deleted and initial values are set. A new copy is started

```
% Start from zero again
fts_minphase.reinitialize();
fts_det = copy(fts_minphase);
```

Simulation of the plant for different deviations from the current equilibrium input.

```
% Simulation
Ts = 1;
itmax = 1000;
t = 0:Ts:itmax*Ts;
% Deviation (from input steady state)
deviation = ones(4,length(t));
deviation(1,itmax*0.05:itmax*0.15) = 1.1;
deviation(1,itmax*0.3:itmax*0.4) = 1.25;
deviation(1,itmax*0.55:itmax*0.65) = 1.5;
deviation(1,itmax*0.8:itmax*0.9) = linspace(1,1.25,itmax*0.1+1);
deviation(1,itmax*0.9:end) = 1.25;
deviation(2,itmax*0.1:itmax*0.2) = 1.1;
deviation(2,itmax*0.35:itmax*0.45) = 1.25;
```

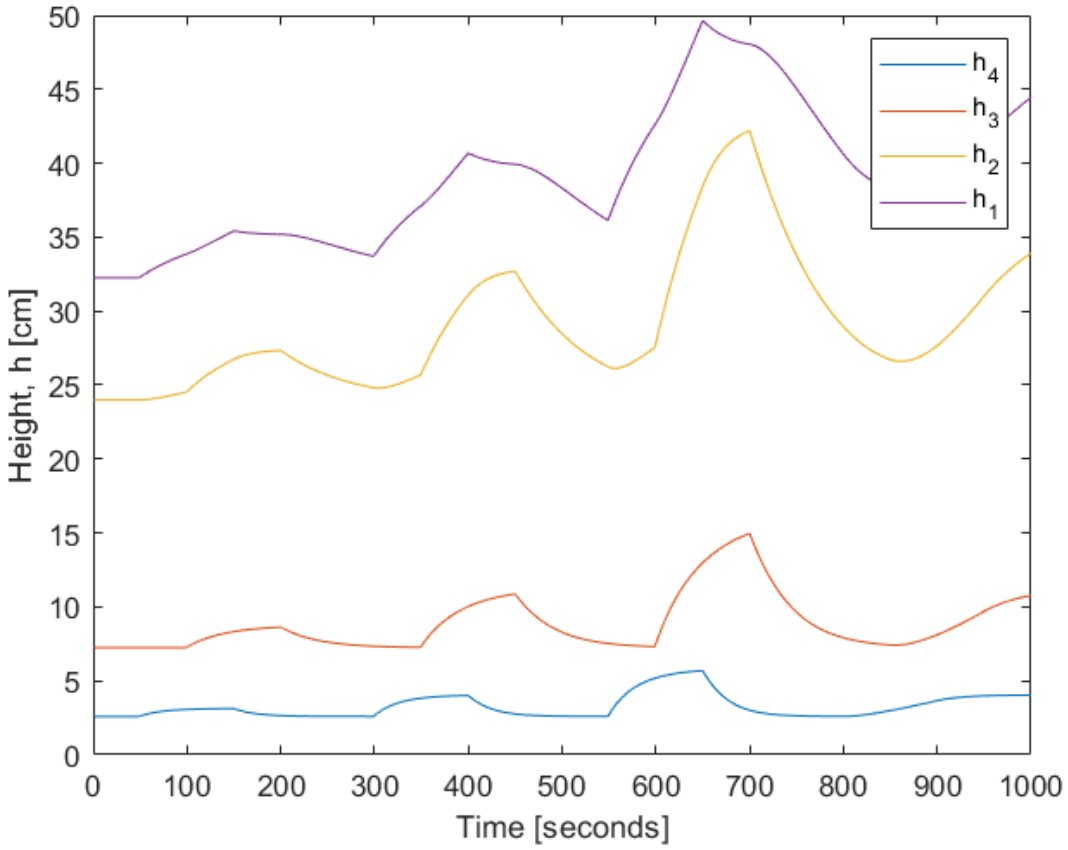
```
deviation(2,itmax*0.6:itmax*0.7) = 1.5;
deviation(2,itmax*0.85:itmax*0.95) = linspace(1,1.25,itmax*0.1+1);
deviation(2,itmax*0.95:end) = 1.25;
fighand = figure;
plot(deviation(1:2,:)')
xlim([0 itmax])
xlabel('Time [seconds]')
ylabel('Input, F [cm^3/s]')
legend('F_1','F_2')
ylim([0.9 1.6])
saveas(fighand,[pwd '\figures\f03_input'],'png')
```



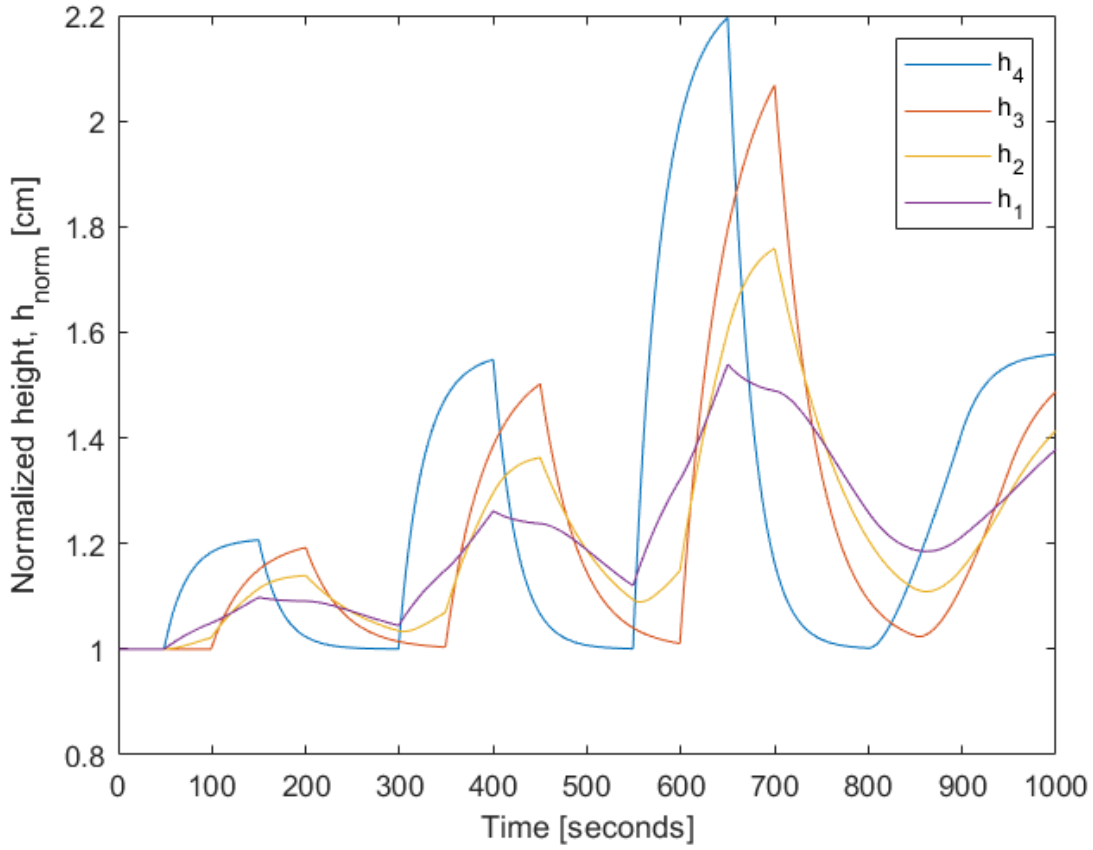

```
for it = 1:length(t)-1
    fts_det.timestep([t(it); t(it+1)],ssInput.*deviation(:,it));
end
```

Now the results are plotted being nonnormalized and normalized:

```
% Plot
fighand = figure;
plot(flip(fts_det.record.meas,2))
xlabel('Time [seconds]')
ylabel('Height, h [cm]')
legend('h_4','h_3','h_2','h_1')
saveas(fighand,[pwd '\figures\f04_output'],'png')
```




```
% Normalized plot
fighand = figure;
plot(flip(fts_det.record.meas./fts_det.hs',2))
xlabel('Time [seconds]')
ylabel('Normalized height, h_{norm} [cm]')
legend('h_4','h_3','h_2','h_1')
saveas(fighand,[pwd '\figures\f05_normoutput'],'png')
```

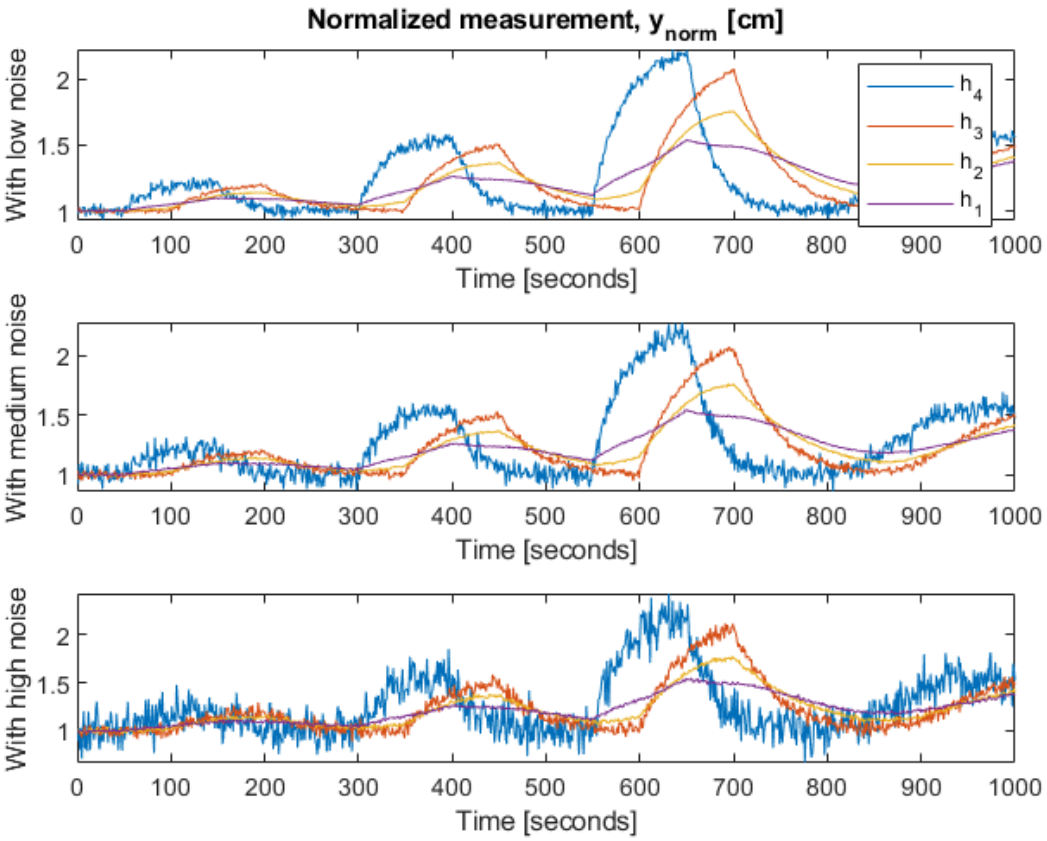


Question 3.2-3

Now we duplicate the original instance to the noisy ones:

```
% Start from zero again
fts_det.reinitialize()
fts_vlow = copy(fts_det);
fts_vmedium = copy(fts_det);
fts_vhigh = copy(fts_det);
% Initializing the noise
noise.Rww = zeros(4);
noise.Rvv = 0.005*eye(4);
fts_vlow.addnoise(noise)
noise.Rvv = 0.02*eye(4);
fts_vmedium.addnoise(noise)
noise.Rvv = 0.1*eye(4);
fts_vhigh.addnoise(noise)
% Simulation including noise
for it = 1:length(t)-1
    fts_vlow.timestep([t(it); t(it+1)],ssInput.*deviation(:,it));
    fts_vmedium.timestep([t(it); t(it+1)],ssInput.*deviation(:,it));
    fts_vhigh.timestep([t(it); t(it+1)],ssInput.*deviation(:,it));
end
% Plots of the noise corrupted measurements:
fighand = figure;
```

```
subplot(311)
plot(flip((fts_vlow.record.meas)./fts_vlow.hs',2))
xlabel('Time [seconds]')
ylabel('With low noise')
legend('h_4','h_3','h_2','h_1')
title('Normalized measurement, y_{norm} [cm]')
subplot(312)
plot(flip((fts_vmedium.record.meas)./fts_vmedium.hs',2))
xlabel('Time [seconds]')
ylabel('With medium noise')
subplot(313)
plot(flip((fts_vhigh.record.meas)./fts_vhigh.hs',2))
xlabel('Time [seconds]')
ylabel('With high noise')
saveas(fighand,[pwd '\figures\f06_measnoise'],'png')
```



We can see that since the noises are absolutely identical, when we normalise the plots, the lower tanks have more corrupted measurements due to the change in signal-to-noise ratio.

Question 3.5

Adding noise to the process, through the disturbance:

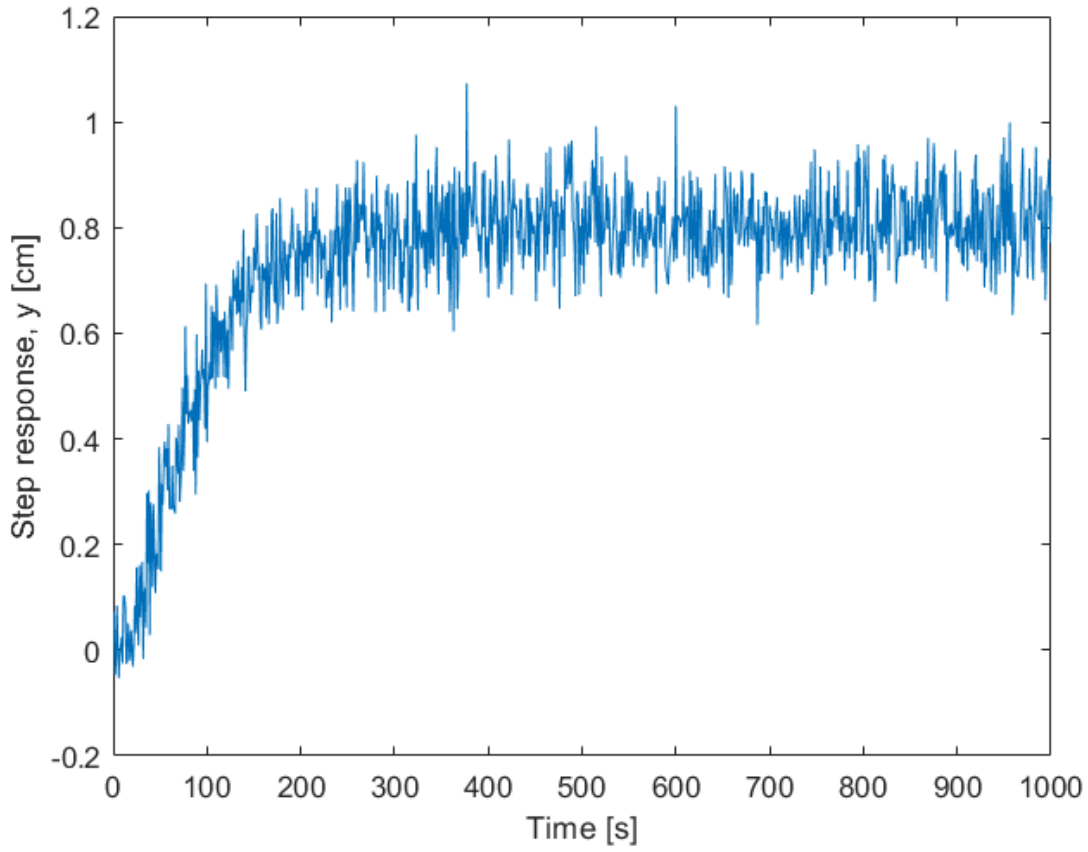
```
% Create the object with noisy disturbance
fts = copy(fts_det);
```

Only the last two states are corrupted by noise.

```
% Noises
noise.Rww = diag([0 0 1e0 1e0]);
noise.Rvv = 0.005*eye(4);
fts.addnoise(noise)
```

We will identify the transfer function from the first input to the second measurement.

```
% Reinitialization of plant object and simulating a step response
fts.reinitialize();
stepInputSize = 1.05;
for it = 1:length(t)-1
    fts.timestep([t(it); t(it+1)],ssInput.*[stepInputSize 1 0 0]');
end
idArguments.t = [0:Ts:(length(fts.record.meas)-Ts)]';
% Identification of first transfer function
idArguments.y = fts.record.meas(:,2) - fts.hs(2);
fighand = figure;
plot(idArguments.y)
xlabel('Time [s]')
ylabel('Step response, y [cm]')
saveas(fighand,[pwd '\figures\f07_stepresponse'],'png')
```



Since the estimation was for a step input, we need to resize the gain.

```
% Resizing the gain
dat = iddata(idArguments.y,(stepInputSize-1)*ssInput(1)*ones(length(idArguments.y),1),Ts);
p = tf;
Parameters = tfest(dat,2,0);
```

```
Warning: For transient data (step or impulse experiment), make sure that the change in input signal does not
happen too early relative to the order of the desired model. You can achieve this by prepending sufficient
number of zeros (equilibrium values) to the input and output signals. For example, a step input must be
represented as [zeros(nx,1); ones(N,1)] rather than ones(N,1), such that nx > model order.
```

```
p(2,1) = tf(Parameters.Numerator,Parameters.Denominator);
```

We can see that the noisiness and the nonlinearity cause the final statistics not to be so decent, the normalized RMSE is only 61%; however, as it will turn out, the parameters will still be close to the linear system's parameters. The 61% statistics can be accepted, since there is a significant amount of noise on the measurements.

Question 3.6

We know that there is process noise affecting the system. However, we assume now that it does not exist, or the system filters it out enough not to be relevant at the level of measurements. Therefore, if the measurements are defined as

$$Y = GU + E,$$

where G is the system and E is the noise of Y, since U is deterministic. This means that E should be the same as the measurement noise. The variance can be calculated for the second half of the time series as it starts with a transient:

```
var(idArguments.y((length(idArguments.y)-1)/2:end))
```

```
Warning: Integer operands are required for colon operator when used as index
ans = 0.0047
```

and the true measurement noise is

```
fts.Lrvv*fts.Lrvv'
```

```
ans = 4×4
    0.0050         0         0         0
         0    0.0050         0         0
         0         0    0.0050         0
         0         0         0    0.0050
```

We can see that the empirical error is within 5%.

Question 3.7

The requirements of a step experiment are that the data should not be too noisy, it should not deviate too much from the linearization point, and it should not be shut off until the measurements get close enough to a steady state.

The identification of the rest of the transfer functions:

```
% From input 2 to output 1
% Reinitialization of plant object and simulating a step response
fts.reinitialize();
for it = 1:length(t)-1
    fts.timestep([t(it); t(it+1)],ssInput.*[1 stepInputSize 0 0]');
end
% Identification of first transfer function
idArguments.y = fts.record.meas(:,1) - fts.hs(1);
% Resizing the gain
dat = iddata(idArguments.y,(stepInputSize-1)*ssInput(2)*ones(length(idArguments.y),1),Ts);
Parameters = tfest(dat,2,0);
```

```
Warning: For transient data (step or impulse experiment), make sure that the change in input signal does not
happen too early relative to the order of the desired model. You can achieve this by prepending sufficient
number of zeros (equilibrium values) to the input and output signals. For example, a step input must be
represented as [zeros(nx,1); ones(N,1)] rather than ones(N,1), such that nx > model order.
```

```
p(1,2) = tf(Parameters.Numerator,Parameters.Denominator);
% From input 1 to output 1
% Reinitialization of plant object and simulating a step response
fts.reinitialize();
for it = 1:length(t)-1
    fts.timestep([t(it); t(it+1)],ssInput.*[stepInputSize 1 0 0]');
end
% Identification of first transfer function
idArguments.y = fts.record.meas(:,1) - fts.hs(1);
% Resizing the gain
dat = iddata(idArguments.y,(stepInputSize-1)*ssInput(1)*ones(length(idArguments.y),1),Ts);
Parameters = tfest(dat,1,0);
```

```
Warning: For transient data (step or impulse experiment), make sure that the change in input signal does not
happen too early relative to the order of the desired model. You can achieve this by prepending sufficient
number of zeros (equilibrium values) to the input and output signals. For example, a step input must be
represented as [zeros(nx,1); ones(N,1)] rather than ones(N,1), such that nx > model order.
```

```
p(1,1) = tf(Parameters.Numerator,Parameters.Denominator);
% From input 2 to output 2
% Reinitialization of plant object and simulating a step response
fts.reinitialize();
for it = 1:length(t)-1
    fts.timestep([t(it); t(it+1)],ssInput.*[1 stepInputSize 0 0]');
end
% Identification of first transfer function
idArguments.y = fts.record.meas(:,2) - fts.hs(2);
% Resizing the gain
dat = iddata(idArguments.y,(stepInputSize-1)*ssInput(2)*ones(length(idArguments.y),1),Ts);
Parameters = tfest(dat,1,0);
```

```
Warning: For transient data (step or impulse experiment), make sure that the change in input signal does not
happen too early relative to the order of the desired model. You can achieve this by prepending sufficient
number of zeros (equilibrium values) to the input and output signals. For example, a step input must be
represented as [zeros(nx,1); ones(N,1)] rather than ones(N,1), such that nx > model order.
```

```
p(2,2) = tf(Parameters.Numerator,Parameters.Denominator);
```

We can see that in these cases we got much better statistics of accuracy, namely normalised RMSEs of 81%, 78% and 75%.

The calculated linear and continuous state space model is:

```
% Linear and continuous state space model
system_id = ss(p)
```

```
system_id =

  A =
           x1        x2        x3        x4        x5
   x1   -0.0127         0         0         0         0
   x2         0  -0.06654   -0.0256         0         0
   x3         0   0.03125         0         0         0
   x4         0         0         0  -0.04583  -0.02532
   x5         0         0         0   0.01563         0
   x6         0         0         0         0         0

           x6
   x1         0
   x2         0
   x3         0
   x4         0
   x5         0
   x6  -0.01407

  B =
          u1       u2
   x1  0.03125        0
   x2  0.03125        0
   x3        0        0
   x4        0   0.0625
   x5        0        0
   x6        0  0.03125

  C =
          x1       x2       x3       x4       x5       x6
   y1  0.05576        0        0        0  0.03871        0
   y2        0        0  0.05277        0        0  0.04537

  D =
       u1  u2
   y1   0   0
   y2   0   0

Continuous-time state-space model.
```

This is a 6th order model due to the non-exact identification, which cannot fit the states exactly. If necessary, model order reduction can be used, for example with the matlab command balred, specifying that we would like to have a state space model with 4 poles (states).

The state space model is not a good basis of comparison with a true model, since the states of a system are not unique.

Question 3.8

Plotting the Markov parameters of the discrete time linear model with 1 second of sampling time. Since the code for calculating the Markov parameters is embedded into a Kalman filter object, we are going to first initialize the Kalman filter.

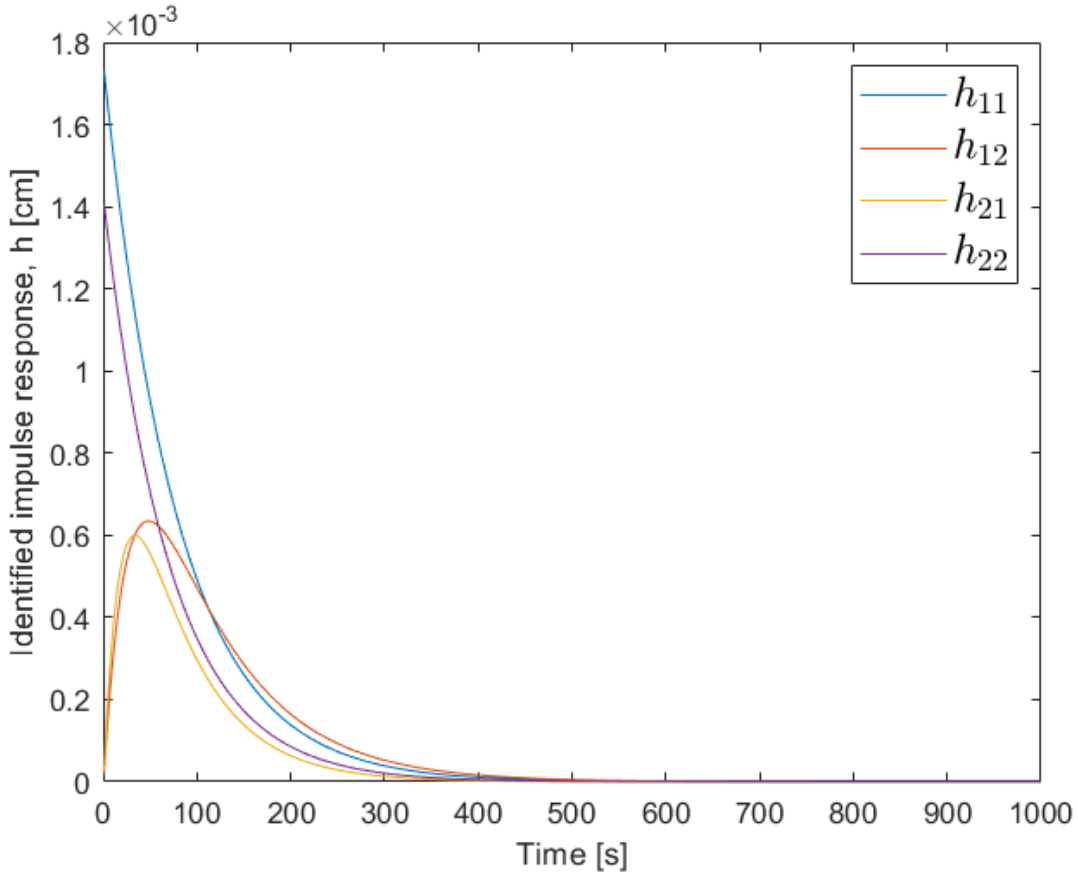
```
% Sharing the linearized state space model
[Ad,Bd] = c2d(system_id.A,system_id.B,Ts);
system.A = Ad;
system.B = Bd;
system.Cz = system_id.C;
system.C = eye(4,6);
system.D = system_id.D;
```

Since noise does not matter now, the noises of the system are not correctly prepared.

```
% Noise will not matter in this particular case
system.G = eye(6);
noise.R = eye(4);
noise.Q = eye(6);
noise.S = zeros(6,4);
% Initials
initial.x = zeros(6,1);
initial.P = eye(6);
% Making an instance the Kalman Filter object
kf_id = KalmanFilter;
kfType = 'stationary';
horizon = 1000;
kf_id.initialize(system,noise,initial,kfType,horizon);
```

The Markov parameters are calculated then:

```
% The Markov parameters
fighand = figure;
plot(kf_id.Markov(1:2:end,1))
hold on
plot(kf_id.Markov(1:2:end,2))
plot(kf_id.Markov(2:2:end,1))
plot(kf_id.Markov(2:2:end,2))
hold off
xlabel('Time [s]')
ylabel('Identified impulse response, h [cm]')
legend({'$h_{11}$','$h_{12}$','$h_{21}$','$h_{22}$'},'Interpreter','latex','FontSize',15)
saveas(fighand,[pwd '\figures\f08_idimpulse'],'png')
```



The Markov parameters could be shown by the matlab command impulse as well.

Problem 4 - Linearization and discretization

Question 4.1

Taylor expansion around the steady state (zero-derivative) gives close values even for low (e.g. first) orders of x. Therefore, we can define first-order linear differential equation systems, by defining the following matrices:

$$\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_s \mathbf{u}_s)$$

$$\mathbf{B} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{x}_s \mathbf{u}_s)$$

$$\mathbf{C} = \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}_s \mathbf{u}_s)$$

$$\mathbf{D} = \frac{\partial \mathbf{g}}{\partial \mathbf{u}}(\mathbf{x}_s \mathbf{u}_s)$$

After the Taylor expansion (for the order of one) we get the following state space matrices around the steady state point:

$$\mathbf{A} = \begin{pmatrix} -\frac{1}{T_1} & 0 & \frac{1}{T_3} & 0 \\ 0 & -\frac{1}{T_2} & 0 & \frac{1}{T_4} \\ 0 & 0 & -\frac{1}{T_3} & 0 \\ 0 & 0 & 0 & -\frac{1}{T_4} \end{pmatrix}$$

$$T_i = \sqrt{\frac{2Am_{s,i}}{\rho g}}\frac{1}{a}$$

$$\mathbf{B} = \begin{pmatrix} \rho\gamma_1 & 0 \\ 0 & \rho\gamma_2 \\ 0 & \rho(1-\gamma_2) \\ \rho(1-\gamma_2) & 0 \end{pmatrix}$$

$$\mathbf{C} = \begin{pmatrix} \frac{1}{\rho A_1} & 0 & 0 & 0 \\ 0 & \frac{1}{\rho A_2} & 0 & 0 \\ 0 & 0 & \frac{1}{\rho A_3} & 0 \\ 0 & 0 & 0 & \frac{1}{\rho A_4} \end{pmatrix}$$

$$\mathbf{C_z} = \begin{pmatrix} \frac{1}{\rho A_1} & 0 & 0 & 0 \\ 0 & \frac{1}{\rho A_2} & 0 & 0 \end{pmatrix}$$

$$\mathbf{D} = \mathbf{0}$$

The disturbances can be both steps and Brownian motion. The measurement noise is also Brownian motion. Therefore the matrix G is made up of the disturbance matrices leading the former, and latter effects into the dynamics.

$$\mathbf{G}_{mean} = \begin{pmatrix} \mathbf{0} \\ \rho\mathbf{I} \end{pmatrix}_{2x4}$$

$$\mathbf{G}_{variance} = \begin{pmatrix} \mathbf{0} \\ \rho\mathbf{I}\sqrt{\mathbf{Q}_L} \end{pmatrix}_{2x4}$$

Where the square root denotes the lower triangular square root matrix. The second matrix is necessary, when the discretization is carried out, since in that case the formulae assume standard Gaussian noise as input. Fortunately, the mean and the variance can be treated independently.

The continuous time linearized state space models for the three models from Problem 2:

```
% Analytically linearize the system
fts.linearize
% Plot the system matrices
fts.ssc
```

```
ans = struct with fields:
      A: [4×4 double]
      B: [4×2 double]
      G: [4×2 double]
      C: [4×4 double]
     Cz: [2×4 double]
    Rww: [4×4 double]
    Rvv: [4×4 double]
    Rwv: [4×4 double]
```

Question 4.2

The zero-pole-gain form transfer functions are the following:

```
% Zero-pole-gain form of the continuous transfer function
zpk(fts.tfc);
```

We can see that certain poles appear in more transfer functions. Since leading the fluid through one extra tank just gives extra dynamics but does not change the earlier, the dynamics leading to the same measurements share poles for different inputs.

Question 4.3

The general (polynomial) form of the continuous transfer functions are the following:

```
% Polynomial forms of the continuous transfer functions
fts.tfc;
```

Question 4.4

As it was described earlier, the results of the analytical linearization of the nonlinear plant, and the identification from measurements on the same nonlinear plant, are closely related, with good normalized RMSE value, even though there is a significant amount of noise on the measurements.

Question 4.5

For an easy interpretation choice, the chosen sampling time is 1 second. In this case, the discrete state space model is:

```
% Chosen sampling time
Ts = 1;
% Discretization
fts.discretize(Ts)
% Converting the state space system to discrete
fts.ssd
```

```
ans = struct with fields:
```

```
     A: [4×4 double]
     B: [4×2 double]
     G: [4×4 double]
     C: [4×4 double]
    Cz: [2×4 double]
   Rww: [4×4 double]
   Rvv: [4×4 double]
   Rwv: [4×4 double]
    Ts: 1
```

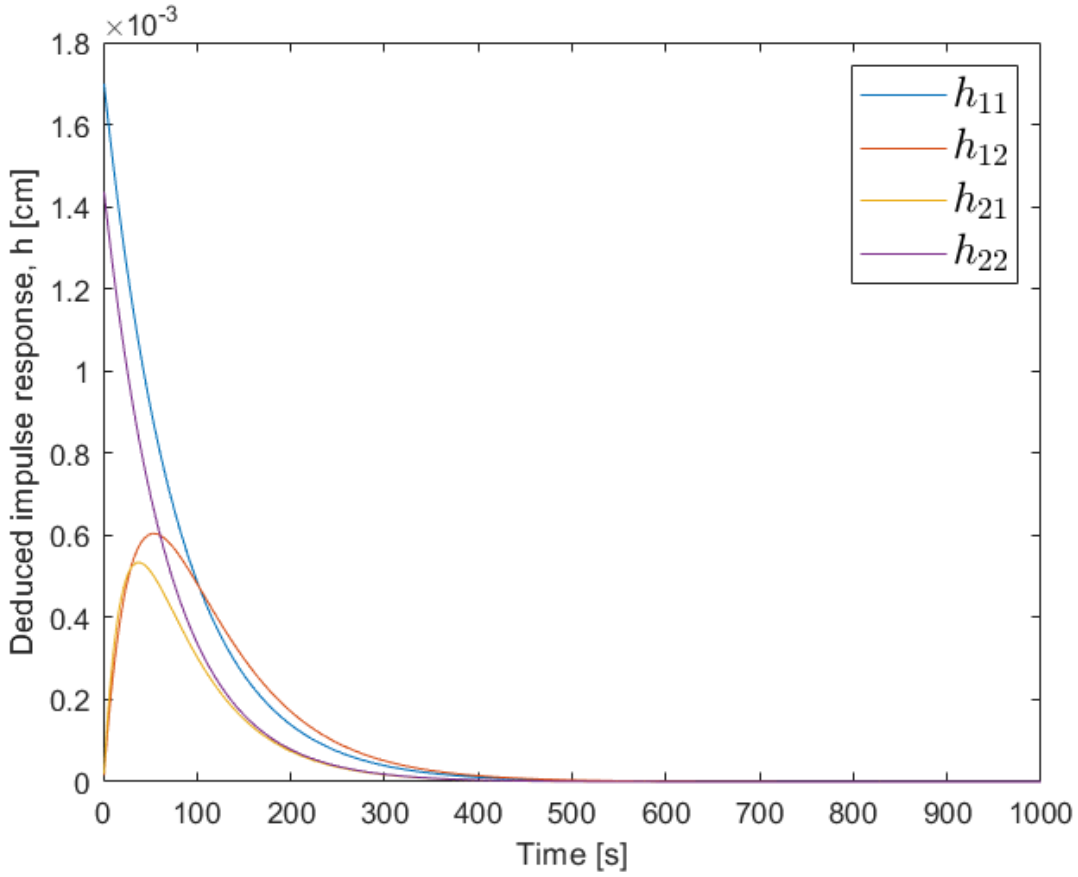
Question 4.6-7

Since the code for calculating the Markov parameters is embedded into a Kalman filter object, we are going to first initialize the Kalman filter.

```
% Sharing the linearized state space model
system = fts.ssd;
noise.R = system.Rvv;
noise.Q = system.Rww;
noise.S = system.Rwv;
% Making an instance the Kalman Filter object
kf_stat = KalmanFilter;
% Initials
initial.x = zeros(6,1);
initial.P = eye(6);
kf_stat.initialize(system,noise,initial,kfType,horizon);
```

The Markov parameters are calculated then:

```
% The Markov parameters
fighand = figure;
plot(kf_stat.Markov(1:2:end,1))
hold on
plot(kf_stat.Markov(1:2:end,2))
plot(kf_stat.Markov(2:2:end,1))
plot(kf_stat.Markov(2:2:end,2))
hold off
xlabel('Time [s]')
ylabel('Deduced impulse response, h [cm]')
legend({'$h_{11}$','$h_{12}$','$h_{21}$','$h_{22}$'},'Interpreter','latex','FontSize',15)
saveas(fighand,[pwd '\figures\f09_animpulse'],'png')
```



As we can see, both the dynamic behaviours and the initial values are almost identical to the identified ones. This idicates two important points: the linearization is a good approach at this case, until we do not deviate from the linearization point too much; and that the identification can be a good approach, if the dynamics of the system are known but the exact parameters are not.

However, the larger deviations cause higher step responses, as we saw at the outputs; this can be found, when analysing the initial values as well; they are slightly larger for the identified system.

We can also see that the chosen sampling time of 1 second is 1-2 orders of magnitude smaller than the quickest dynamics of the system, so discretizing the state space model is also reliable.

Problem 5

The Kalman filter object is able to estimate constant parameter dynamical systems with dynamical, stationary Kalman filters, or if the horizon is given, can predict with Kalman or Markov predictors. In the former case it also keeps track of the covariances. Its main functions, apart from initialization function, are output predictor (with Kalman) and Markov predictor. The former includes time update, measurement update, and separate iterative state and output predicton functions. It is designed to support a controller, therefore the Markov predictior gives out the zero-input transient of the system, as well as the Markov (impulse response) parameter matrix for the given horizon.

Question 5.1

If we do not significantly deviate from the chosen linearisation point, the error that we commit during estimation is negligible (this is the basic motivation of using Taylor series).

If we have a constant sampling time, then the way of connecting the instants of a linear system is multiplying by $e^{A(t-t_0)} = e^{AT_s} = A_d$, and B_d can be deduced from

$$\begin{pmatrix} A_d & B_d \\ 0 & I \end{pmatrix} = \exp \begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix} T_s$$

These theoretical assumptions are supported by the comparison of the models from Problem 3 and Problem 4 (identification and analytical linearization).

Question 5.2

Kalman filters are the solutions of optimization problems, where the variance of the innovation signal (or residual, $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{C}\hat{\mathbf{x}}$) is minimized; It sets a gain, with which this signal is going to converge to zero as fast as possible. In case of a dynamic filter, this gain is calculated based on the current variance, therefore it starts to trust the model until enough feedback has arrived. In case of a static filter, the steady state covariance of the state is assumed, and then the gain is calculated by solving the discrete-time Riccati equation. This means that the filter innovation converges slowly to zero; however, less computation is necessary.

The filters have usually two parts: time update and measurement update. The former one increases, the latter one decreases the covariane of the states, as the latter one brings extra information to the signal. The time update can be changed to longer horizon prediction as well, having applied it recursively. A compact version of it, which calculates the output from a schedule of input, is the Markov predictor:

$$Z_k = b_k + \Gamma U_k$$
$$b_k = O\hat{x}_{k|k} + O_{-1} G \hat{w}_{k|k}$$
$$\Gamma = \begin{pmatrix} H_i & 0 & \dots \\ H_i & H_{i+1} & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

where O is the extended observability matrix, and O_{-1} is the observibility matrix divided by the state matrix. Matrix Γ is the Markov parameter matrix, which gives the impulse responses of the system for a given horizon, and i is the time instant.

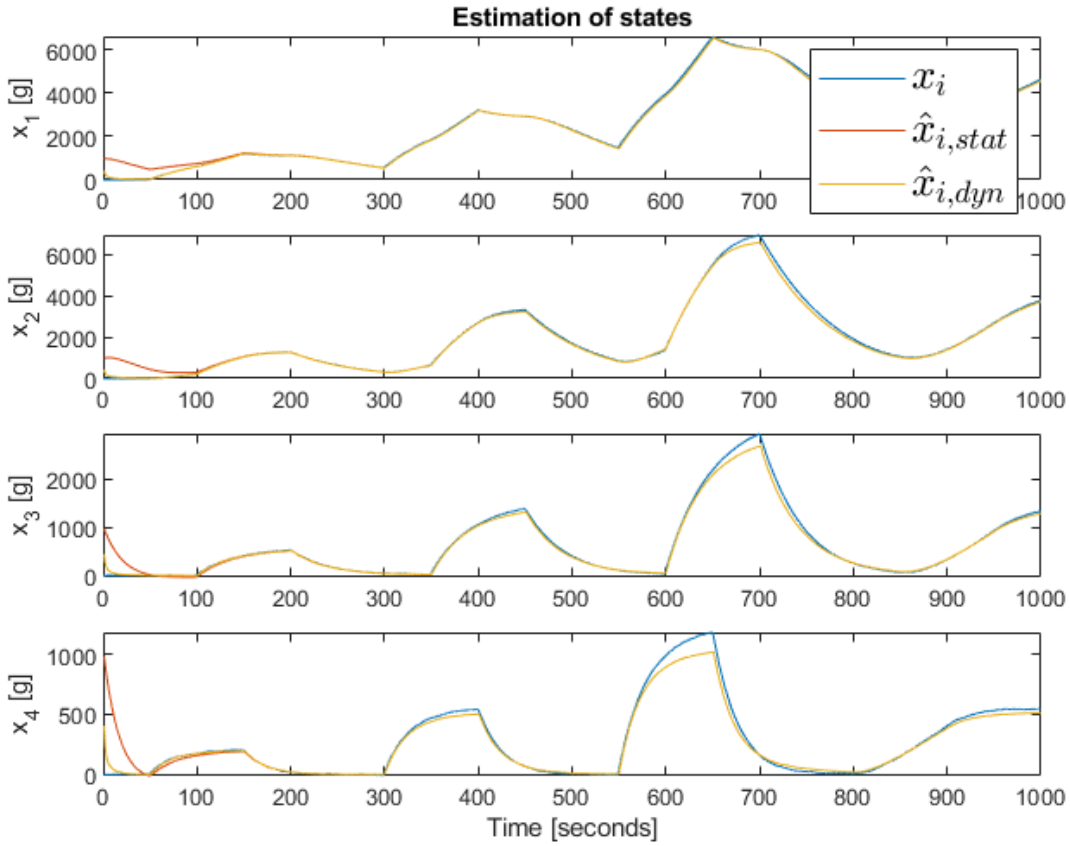
The initialization of the stationary Kalman filter object is already done for deducing the Markov parameters. The initialization of the dynamic Kalman filter:

```
kf_dyn = KalmanFilter;
% Dynamic Kalman Filter
kfType = 'timeinvariant';
kf_dyn.initialize(system,noise,initial,kfType,horizon);
```

The simulation is done for both of the filters:

```
% Reinitialization
fts.reinitialize();
initial.x = ones(4,1)*1e3;
initial.P = eye(4)*1e3;
```

```
horizon = 1;
kf_stat.reinitialize(initial,horizon);
kf_dyn.reinitialize(initial,horizon);
% Declaration of the estimation vectors
xf_stat = zeros(4,length(t)-1);
xf_dyn = zeros(4,length(t)-1);
% Simulation with Kalman estimators
for it = 1:length(t)-1
    fts.timestep([t(it); t(it+1)],ssInput.*deviation(:,it));
    y = fts.record.meas(end,:)' - fts_vlow.hs;
    u = ssInput.*(deviation(:,it) - 1);
    xf_stat(:,it) = kf_stat.outputPredictor(u(1:2,1),y,noise.Q,horizon);
    xf_dyn(:,it) = kf_dyn.outputPredictor(u(1:2,1),y,noise.Q,horizon);
end
% Plots of the noise corrupted measurements:
fighand = figure;
subplot(411)
plot(fts.record.t,fts.record.x(:,1)-fts.ms(1))
hold on
plot(xf_stat(1,:)')
plot(xf_dyn(1,:)')
hold off
ylabel('x_1 [g]')
legend({'$x_i$','$\hat{x}_{i,stat}$','$\hat{x}_{i,dyn}$'},'Interpreter','latex','FontSize',15)
title('Estimation of states')
subplot(412)
plot(fts.record.t,fts.record.x(:,2)-fts.ms(2))
hold on
plot(xf_stat(2,:)')
plot(xf_dyn(2,:)')
hold off
ylabel('x_2 [g]')
subplot(413)
plot(fts.record.t,fts.record.x(:,3)-fts.ms(3))
hold on
plot(xf_stat(3,:)')
plot(xf_dyn(3,:)')
hold off
ylabel('x_3 [g]')
subplot(414)
plot(fts.record.t,fts.record.x(:,4)-fts.ms(4))
hold on
plot(xf_stat(4,:)')
plot(xf_dyn(4,:)')
hold off
xlabel('Time [seconds]')
ylabel('x_4 [g]')
saveas(fighand,[pwd '\figures\f10_est'],'png')
```



Question 5.3

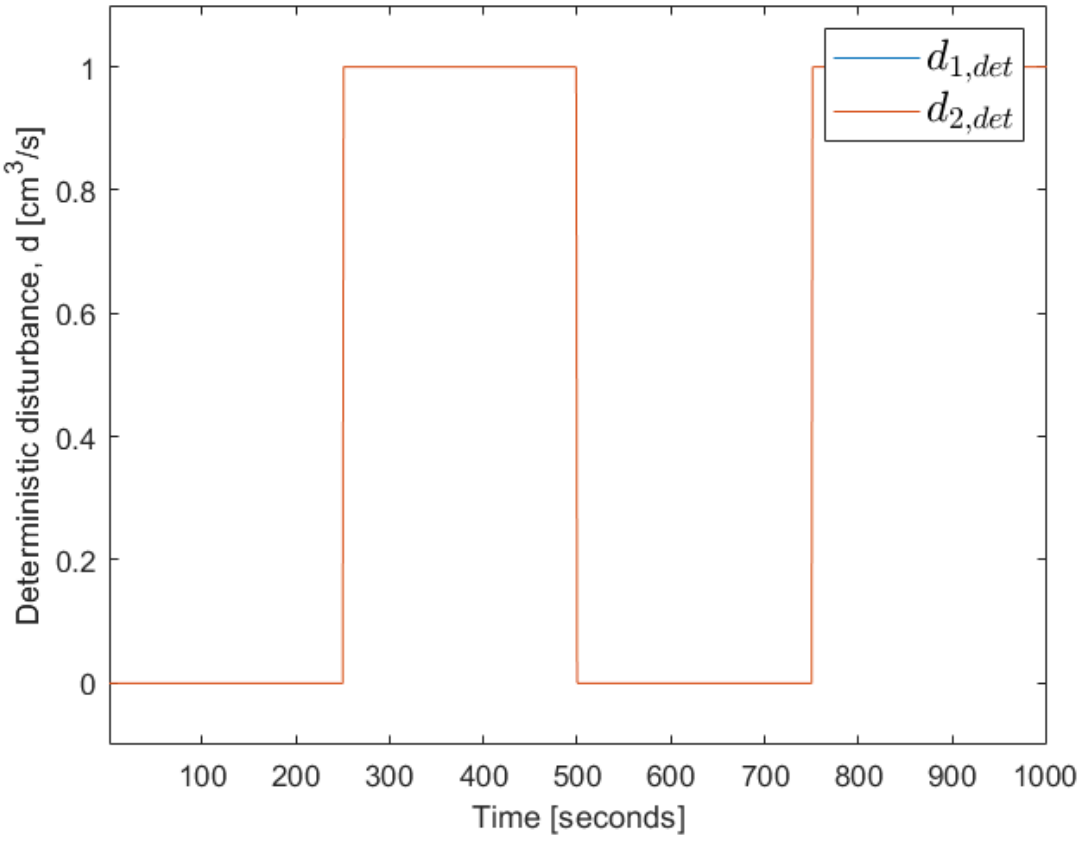
Since the Kalman Filter should not have knowledge of the disturbance until it is augmented, the estimator object will not receive the same inputs as the true plant.

```
% Reinitialization
fts.reinitialize();
initial.x = ones(4,1)*1e3;
initial.P = eye(4)*1e3;
kf_stat.reinitialize(initial,horizon);
kf_dyn.reinitialize(initial,horizon);
% Declaration of the estimation vectors
xf_stat = zeros(4,length(t)-1);
xf_dyn = zeros(4,length(t)-1);
```

As there is a step input for the disturbances, we change the variable ssInput. Let us say that it is a pulse, having the same effect on both tanks (for example modelling raining with open tanks). We assume no deviation in the original inputs from steady state, so the effect of the disturbance can be investigated independently.

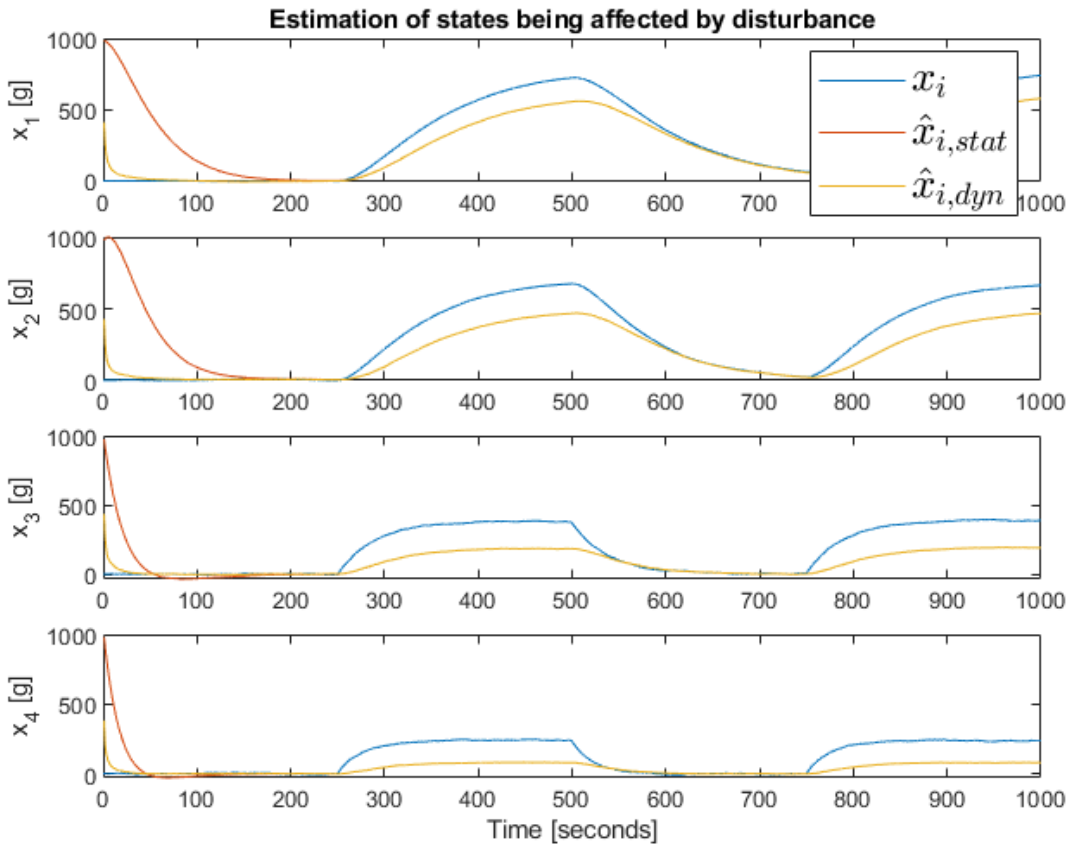
```
% Disturbance base creation
ssInput(3:4,:) = [10; 10];
% Time evolution of disturbance
deviation_dist = deviation;
```

```
deviation_dist(:,:) = 1;
deviation_dist(3:4,1:itmax/4) = 0;
deviation_dist(3:4,itmax/2:itmax/4*3) = 0;
% Plotting disturbance
fighand = figure;
plot(deviation_dist(3:4,:)')
xlim([1 length(t)])
xlabel('Time [seconds]')
legend({'$d_{1,det}$','$d_{2,det}$'},'Interpreter','latex','FontSize',15)
ylabel('Deterministic disturbance, d [cm^3/s]')
ylim([-0.1 1.1])
saveas(fighand,[pwd '\figures\f11_disturbance'],'png')
```




```
% Simulation with Kalman estimators
for it = 1:length(t)-1
    fts.timestep([t(it); t(it+1)],ssInput.*deviation_dist(:,it));
    y = fts.record.meas(end,:)' - fts_vlow.hs;
    u = ssInput.*(deviation_dist(:,it) - 1);
    xf_stat(:,it) = kf_stat.outputPredictor(u(1:2,1),y,noise.Q,horizon);
    xf_dyn(:,it) = kf_dyn.outputPredictor(u(1:2,1),y,noise.Q,horizon);
end
% Plots of the noise corrupted measurements:
fighand = figure;
subplot(411)
plot(fts.record.t,fts.record.x(:,1)-fts.ms(1))
hold on
```

```
plot(xf_stat(1,:)')
plot(xf_dyn(1,:)')
hold off
ylabel('x_1 [g]')
legend({'$x_i$','$\hat{x}_{i,stat}$','$\hat{x}_{i,dyn}$'},'Interpreter','latex','FontSize',15)
title('Estimation of states being affected by disturbance')
subplot(412)
plot(fts.record.t,fts.record.x(:,2)-fts.ms(2))
hold on
plot(xf_stat(2,:)')
plot(xf_dyn(2,:)')
hold off
ylabel('x_2 [g]')
subplot(413)
plot(fts.record.t,fts.record.x(:,3)-fts.ms(3))
hold on
plot(xf_stat(3,:)')
plot(xf_dyn(3,:)')
hold off
ylabel('x_3 [g]')
subplot(414)
plot(fts.record.t,fts.record.x(:,4)-fts.ms(4))
hold on
plot(xf_stat(4,:)')
plot(xf_dyn(4,:)')
hold off
xlabel('Time [seconds]')
ylabel('x_4 [g]')
saveas(fighand,[pwd '\figures\f12_distest'],'png')
```



If we augment the estimators, being prepared for (knows about) the disturbance, the estimation will not have steady state errors anymore. Since we do not know the input, we try to estimate them with assuming hidden states:

$$\begin{pmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{x}}_\mathbf{d} \end{pmatrix} = \begin{pmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{x}_d \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{I} \end{pmatrix} \mathbf{d}$$

```
% Augmented system, continuous
system_aug_cont.A = [fts.ssc.A fts.ssc.G; zeros(2,4) zeros(2)];
system_aug_cont.B = [system.B; zeros(2)];
system_aug_cont.C = [system.C zeros(4,2)];
system_aug_cont.G = [zeros(4,2); eye(2)];
system_aug_cont.Cz = [system.Cz zeros(2)];
system_aug_cont.Rww = diag([0; 0; diag(fts.ssc.Rww)]);
system_aug_cont.Rwv = [fts.ssc.Rwv; zeros(2,4)];
system_aug_cont.Rvv = fts.ssc.Rvv;
% Augmented system, discrete
system_aug = system_aug_cont;
[Ad,Bd] = c2d(system_aug_cont.A,system_aug_cont.B,Ts);
system_aug.A = Ad;
system_aug.B = Bd;
system_aug.G = eye(6);
```

```
system_aug.Rww = fts.c2d_noise(system_aug_cont.A,system_aug_cont.G*fts.Lrww(3:4,3:4),Ts);
```

The discrete time Riccati equation cannot be solved (at least not correctly), if there are states with no variance, since the following has to hold for the subsystem of states unaffected by other states:

$$\begin{pmatrix} \mathbf{GQG'} & \mathbf{S} \\ \mathbf{S'} & \mathbf{R} \end{pmatrix} > 0$$

Therefore the whole system had to be rephrased so that the noise arrives through the disturbance, and is not independent of the deterministic part.

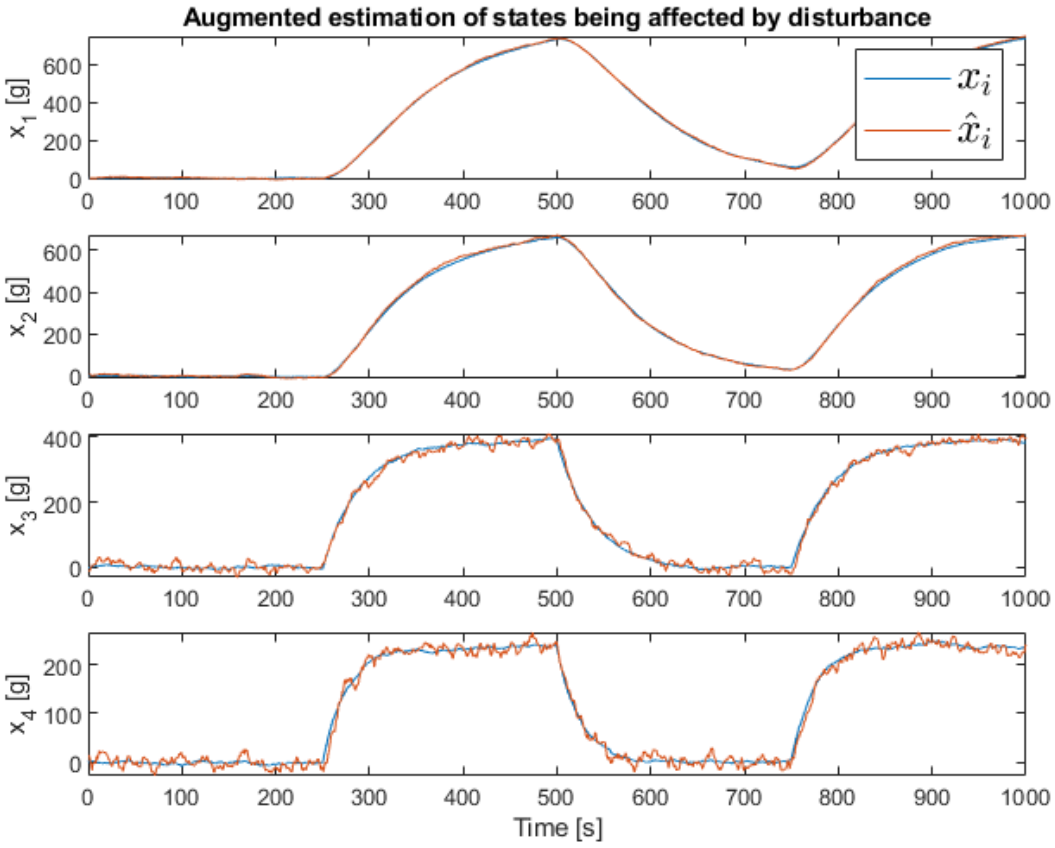
It is not applied now: With empirical tuning, the right variances, with other words, distrust factors can be set for a good estimator convergence:

```
noise_aug.R = system_aug.Rvv;
% Theoretical Q matrix
noise_aug.Q = system_aug.Rww;
% Practical modification to solve dare
%disturbanceDistrust = 1;
%noise_aug.Q(5:6,5:6) = eye(2)*disturbanceDistrust;
noise_aug.S = system_aug.Rwv;
% Augmented Kalman Filter instance
kf_aug = KalmanFilter;
% Stationary Kalman Filter
kfType = 'stationary';
initial.x = ones(6,1);
initial.P = eye(6)*1e3;
kf_aug.initialize(system_aug,noise_aug,initial,kfType,horizon);
```

Reinitialization and simulation with plots:

```
% Reinitialization
fts.reinitialize();
% Declaration of the estimation vectors
xf_aug = zeros(6,length(t)-1);
% Simulation with Kalman estimators
for it = 1:length(t)-1
    fts.timestep([t(it); t(it+1)],ssInput.*deviation_dist(:,it));
    y = fts.record.meas(end,:)' - fts_vlow.hs;
    u = ssInput.*(deviation_dist(:,it) - 1);
    xf_aug(:,it) = kf_aug.outputPredictor(u(1:2,1),y,noise_aug.Q,horizon);
end
fighand = figure;
% Plots of the noise corrupted measurements:
subplot(411)
plot(fts.record.t,fts.record.x(:,1)-fts.ms(1))
hold on
plot(xf_aug(1,:)')
hold off
ylabel('x_1 [g]')
legend({'$x_i$','$\hat{x}_{i}$'},'Interpreter','latex','FontSize',15)
title('Augmented estimation of states being affected by disturbance')
subplot(412)
```

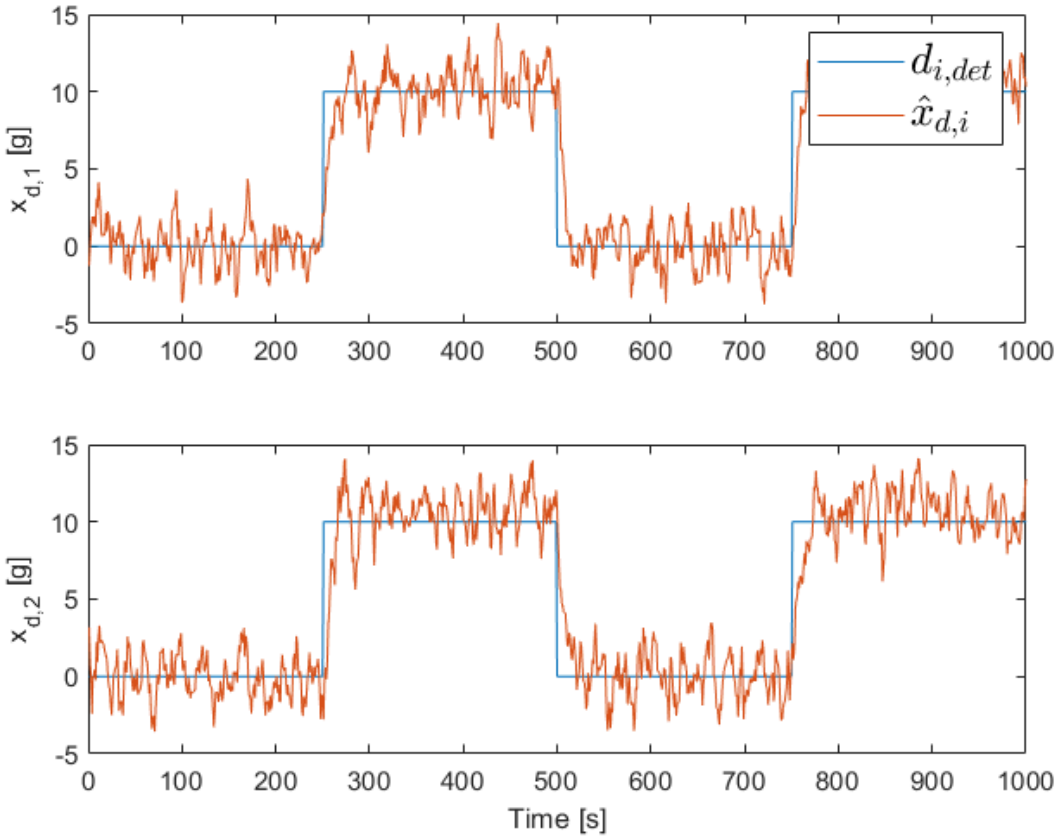
```
plot(fts.record.t,fts.record.x(:,2)-fts.ms(2))
hold on
plot(xf_aug(2,:)')
hold off
ylabel('x_2 [g]')
subplot(413)
plot(fts.record.t,fts.record.x(:,3)-fts.ms(3))
hold on
plot(xf_aug(3,:)')
hold off
ylabel('x_3 [g]')
subplot(414)
plot(fts.record.t,fts.record.x(:,4)-fts.ms(4))
hold on
plot(xf_aug(4,:)')
hold off
ylabel('x_4 [g]')
xlabel('Time [s]')
saveas(fighand,[pwd '\figures\f13_augest'],'png')
```



Plotting the disturbances:

```
% Plotting the disturbances
fighand = figure;
subplot(211)
```

```
plot(ssInput(3)*deviation_dist(3,:))
hold on
plot(xf_aug(5,:)')
hold off
xlim([0 length(t)-1])
ylabel('x_{d,1} [g]')
legend({'$d_{i,det}$','$\hat{x}_{d,i}$'},'Interpreter','latex','FontSize',15)
subplot(212)
plot(ssInput(4)*deviation_dist(4,:))
hold on
plot(xf_aug(6,:)')
hold off
xlim([0 length(t)-1])
ylabel('x_{d,2} [g]')
xlabel('Time [s]')
saveas(fighand,[pwd '\figures\f14_augdistest'],'png')
```



Question 5.4

Note that there is no need for linear simulation: the Kalman filter would have the same behaviour far away from the linearization point as being close to that.

As we can see the Kalman filter has better performance, if the model is close to the plant; we have to be close to the linearization point (else steady state error can be expected due to the disturbance-like appearance of the deviation), and we have to model nonzero mean disturbances that propagate throught the system. If the latter are not modelled, we can see in the first plots, that the estimation is a trade off between

the model and the innovation; the weights decide, what it is going to be closer. In case of the modelling of the disturbance, the uncertainty of the modelled part of the system increases, and the estimator relies more on the innovation. This ends up with noisier estimations, which are, in the end, closer to the true values in the mean.

The disturbances were also successfully estimated.

Problem 6 - Unconstrained MPC

The Model Predictive Controller object is designed to satisfy any kind of objective function, with input and input rate-of-movement constraints, and a set of possible soft output constraints. The way how the objective functions are to be given is with providing the coefficient of the input vector, and the constant that is subtracted, for the quadratic loss function. It has three main functions; initialization, control prepare (initialization of changing parameters, this ususally happens every time in a loop), and control compute (this actually includes control prepare for easier usage). In case of no constrains, it uses the closed formula $x = -H^{-1}g$. In any other case, it leverages the quadprog function.

Question 6.1

Unconstrained Model Predictive Controllers solve the optimization problem of

$$\Phi = \frac{1}{2}\sum_{j=1}^{N} ||Wo||_2^2$$

where j is the horizon, W is the weight (how much we intend to minimize), o is the objective that is to be minimized. This can be rephrased as

$$\Phi = \frac{1}{2}||\bar{W}(KU_k - C)||_2^2$$

where $\bar{W}$ is the augmented weight (we assume that every time instant is weighted equally), K is the coefficient of the control input, for which we are solving for, and C is a constant. For reference tracking, $K = \Gamma, \quad C = c_k$, where $c_k = R_k - b_k$, k is the time instant of the optimization, R_k is the reference and b_k is the zero-input transient vector and Γ is the Markov parameter matrix, made up of the matrices H_i for every horizon instant i. For input minimization (around the steady state), $K = I, \quad C = 0$. For input rate-of-movement minimization, $K = \Lambda, \quad C = I_0\hat{u}_{k-1|k}$, where Λ is the difference matrix, I_0 is an identity matrix followed by zeros for the future time instants, and $\hat{u}_{k-1|k}$ is the previous input. For every time instant k, we solve a quadratic optimization problem of

$$\Phi = \frac{1}{2}U_k^T H U_k + g^T U_k + \rho$$

where the value of ρ does not change the solution of the optimization problem, $H = K^T\bar{W}^T\bar{W}K$, and $g = -K^T\bar{W}^T\bar{W}C$. This gives the solution $U_k = -H^{-1}g$.

Question 6.2

Design of Model Predictive Controller.

The chosen horizon is 60 seconds, since this is a slow system. Generally the bigger the horizon was, the better results we got, but the computation time increased significantly as well.

```
% Reinitialize Kalman Filter with the prediction horizon
horizon = 60;
kf_aug.reinitialize(initial,horizon);
% Creating instance
mpc = ModelPredictiveController;
```

Preinitialization is necessary to determine the dimensions of the controller. For special instances, the preinitialized object instance is copied.

```
% Preinitialization
n = kf_aug.stateSpaceDimensions();
mpc.preinitialize(horizon,n);
```

Creation of unconstrained controller.

```
% Creation of the copy
mpc_unconstrained = copy(mpc);
% Number of objectives
n.no = 2;
n.nscl = 0;
n.nscu = 0;
% Initialization of weights and coefficients of objective functions
W = cell(2,1);
Ucoeff = cell(2,1);
% Unconstrained MPC
ExtraFeatures.useInputConstraints = false;
ExtraFeatures.useInputRateConstraints = false;
ExtraFeatures.useSoftOutputConstraints = false;
% Weight on output
W{1} = diag([1e2 1e2]);
% Weight on input (control signal)
W{2} = diag([1 1]);
```

The objectives are chosen to be identical to the finite horizon LQR controller problem: the reference tracking error and the input deviation from the steady state input are minimized.

```
% Objective phrasing
Ucoeff{1} = kf_aug.Markov;
Ucoeff{2} = kron(eye(n.nu),eye(horizon));
% Initial input signal
u_1 = zeros(2,1);
% Algorithm used in case of constrained optimization
optioptions.Algorithm = 'interior-point-convex';
optioptions.Display = 'off';
mpc_unconstrained.initialize(W,Ucoeff,n,ExtraFeatures,u_1,optioptions)
```

Question 6.3

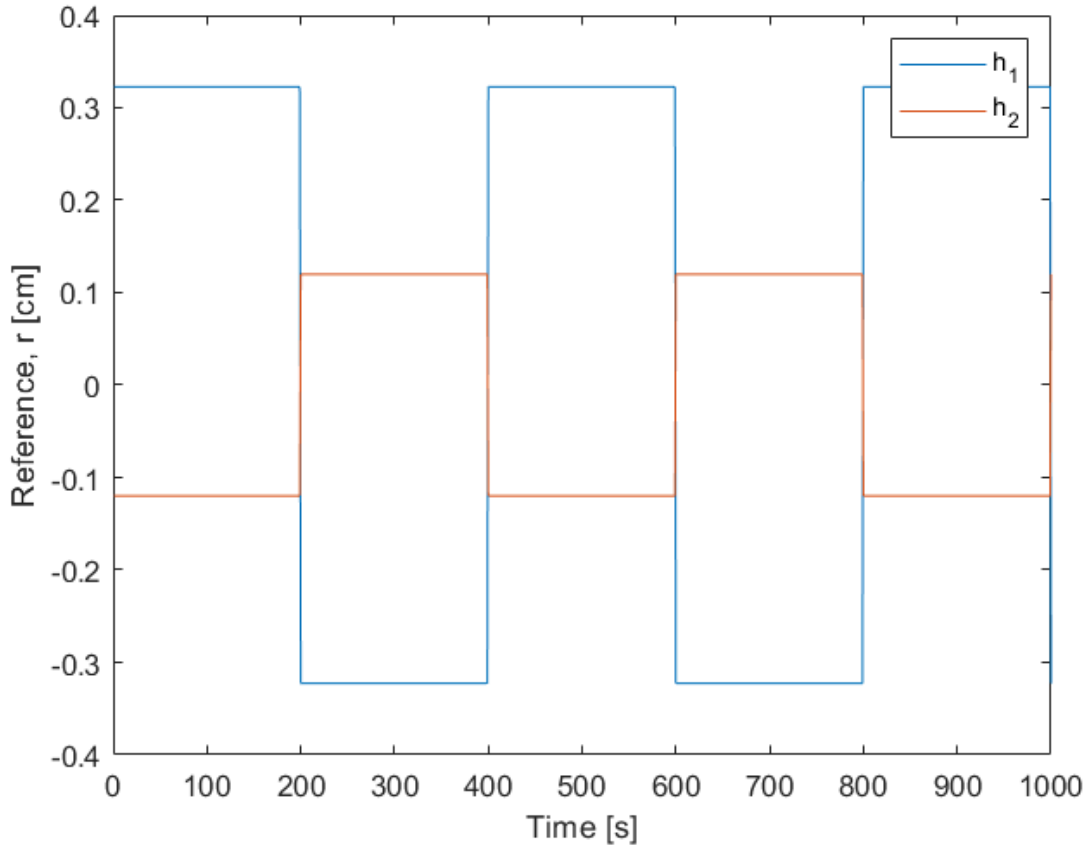
```
% Reinitialization - KF is already reinitialized
fts.reinitialize();
% Constant of second objective function
c = cell(2,1);
% The initial input vector
U = kron(zeros(horizon,1),ones(n.nu,1));
% Initial measurement
y = zeros(4,1);
```

The required reference signal is a pulse function with period itmax/5:

```
% Reference signal
reference = diag([fts.hs(1:2,1)/100])*diag([1 -0.5])
```

```
reference = 2×2
    0.3226         0
         0   -0.1200
```

```
R = reference*ones(2,length(t)-1+horizon);
for it = 1:length(t)-1+horizon
    if rem(floor(it/(itmax/5)),2)
        R(:,it) = -R(:,it);
    end
end
fighand = figure;
plot(R')
xlim([0 length(t)-1])
ylabel('Reference, r [cm]')
xlabel('Time [s]')
legend('h_1','h_2')
saveas(fighand,[pwd '\figures\f15_reference'],'png')
```



Only plotted bounds for later comparison:

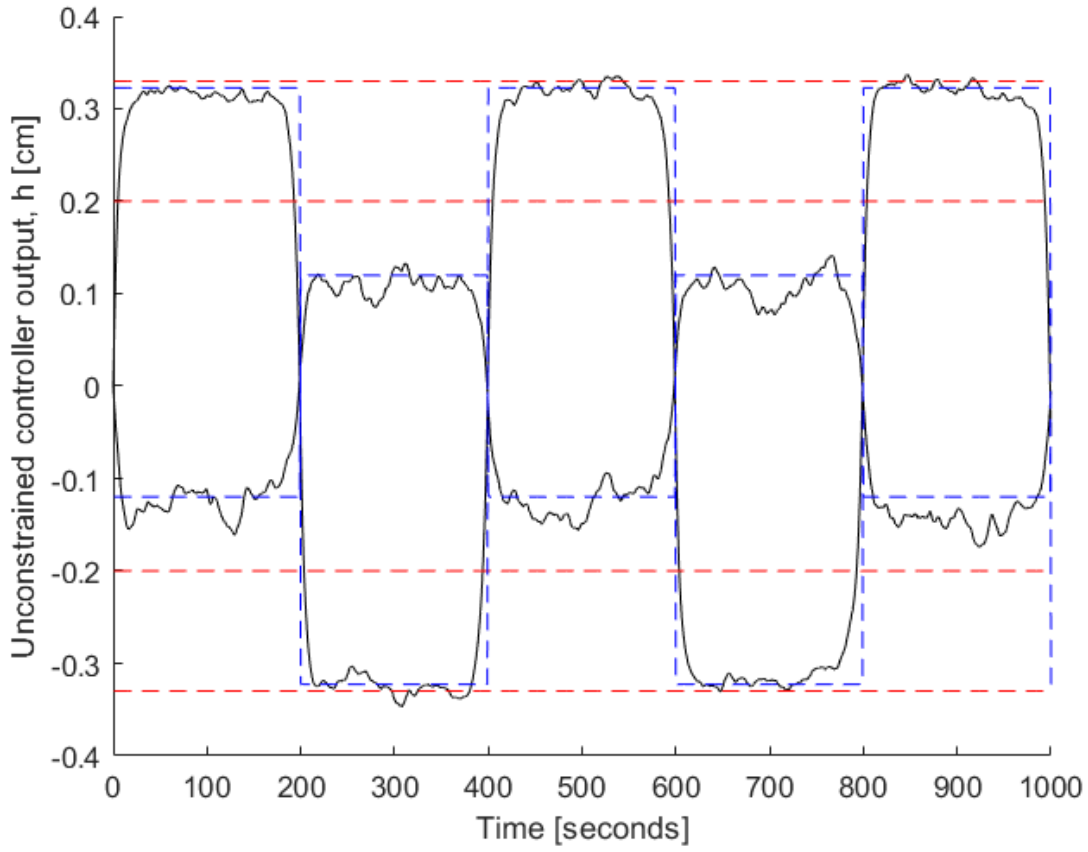
```
% Plotted bounds
heightBounds = [0.33; 0.2]
```

```
heightBounds = 2×1
    0.3300
    0.2000
```

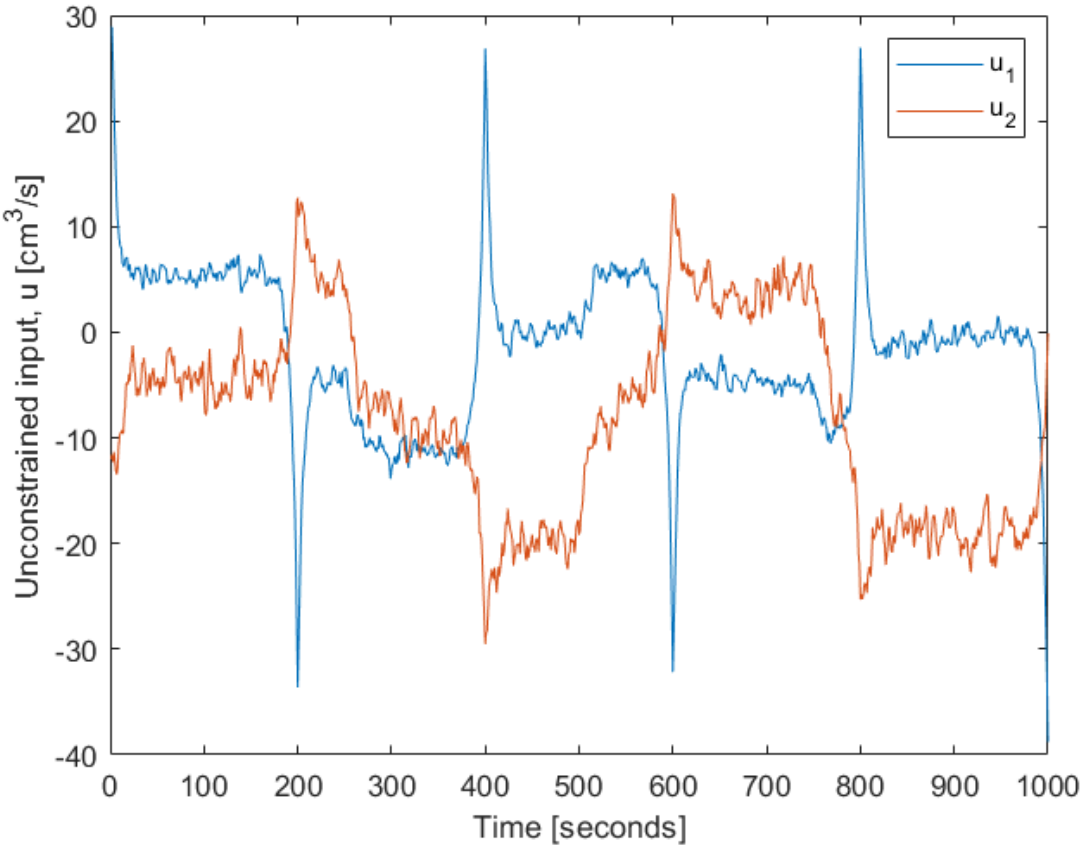
Simulation of estimation based control, having disturbance:

```
% Declaration of record
record.u = zeros(n.nu,length(t)-1);
fighand = figure;
clf
hold on
for it = 1:length(t)-1
    % Kalman Filter, Markov predictor
    kf_aug.markovPredictor(U,y);
    % Model Predictive Control
    Z = reshape(R(:,it:it+horizon-1),kf_aug.nu*kf_aug.j,1);
    c{1} = Z-kf_aug.b;
    c{2} = kron(zeros(horizon,1),ones(2,1));
    ExtraFeatures.b = kf_aug.b;
    [u, U] = mpc_unconstrained.controlCompute(c,ExtraFeatures);
```

```
    % Physical system and measurement
    fts.timestep([t(it); t(it+1)],ssInput.*deviation_dist(:,it)+[u; zeros(2,1)]);
    y = fts.record.meas(end,:)' - fts_vlow.hs;
    % Saving input
    record.u(1:kf_aug.nz,it) = u;
end
plot(fts.record.t,fts.record.x(:,1:2)/fts.rho/fts.A-fts.hs(1:2)','k')
plot(ones(itmax,4)*diag([heightBounds; -heightBounds]),'--r')
plot(R','--b')
hold off
xlabel('Time [seconds]')
ylabel('Unconstrained controller output, h [cm]')
xlim([0 itmax])
saveas(fighand,[pwd '\figures\f16_mpcoutput'],'png')
```




```
fighand = figure;
plot(record.u')
ylabel('Unconstrained input, u [cm^3/s]')
legend('u_1','u_2')
xlabel('Time [seconds]')
saveas(fighand,[pwd '\figures\f17_mpcinput'],'png')
```



Problem 7 - Input Constrained MPC

Question 7.1

In case of input constraints, the optimization problem trades off the best unconstrained solution with the given constraints: $U_{min,k} \leq U_k \leq U_{max,k}$. For input rate-of-movement constraint, we need to have a function of the optimization argument to define the constraints. This function is given as $b_l \leq Ax \leq b_u$, where the bounds are noted by b, and x is the optimization variable. In this case, $\Delta U_{min} + I_0 \hat{u}_{k-1|k} \leq \Gamma U_k \leq \Delta U_{mmax} + I_0 \hat{u}_{k-1|k}$.

Question 7.2

The same object is used to create the input constrained MPC.

Question 7.3

The usage is the following. Apart from the input constraint, input rate constraint is used as well.

```
% Creation of the copy
mpc_inputConstrained = copy(mpc);
% Number of objectives
n.no = 3;
% Initialization of weights and coefficients of objective functions
```

```
W = cell(3,1);
Ucoeff = cell(3,1);
% Input cconstrained MPC
ExtraFeatures.useInputConstraints = true;
ExtraFeatures.useInputRateConstraints = true;
% Weight on output, input, input rate
W{1} = diag([1e2 1e2]);
W{2} = diag([1 1]);
W{3} = diag([1e-2 1e-2]);
% Objective phrasing
Ucoeff{1} = kf_aug.Markov;
Ucoeff{2} = kron(eye(n.nu),eye(horizon));
Ucoeff{3} = mpc.Lambda;
% Initial input signal
mpc_inputConstrained.initialize(W,Ucoeff,n,ExtraFeatures,u_1,optioptions);
% Reinitialization
fts.reinitialize();
kf_aug.reinitialize(initial,horizon);
% Constant of second objective function
c = cell(3,1);
% Algorithm used in case of constrained optimization
optioptions.Algorithm = 'interior-point-convex';
% The initial input vector and measurement
U = kron(zeros(horizon,1),ones(n.nu,1));
y = zeros(4,1);
% Declaration of record
record.u = zeros(n.nu,length(t)-1);
```

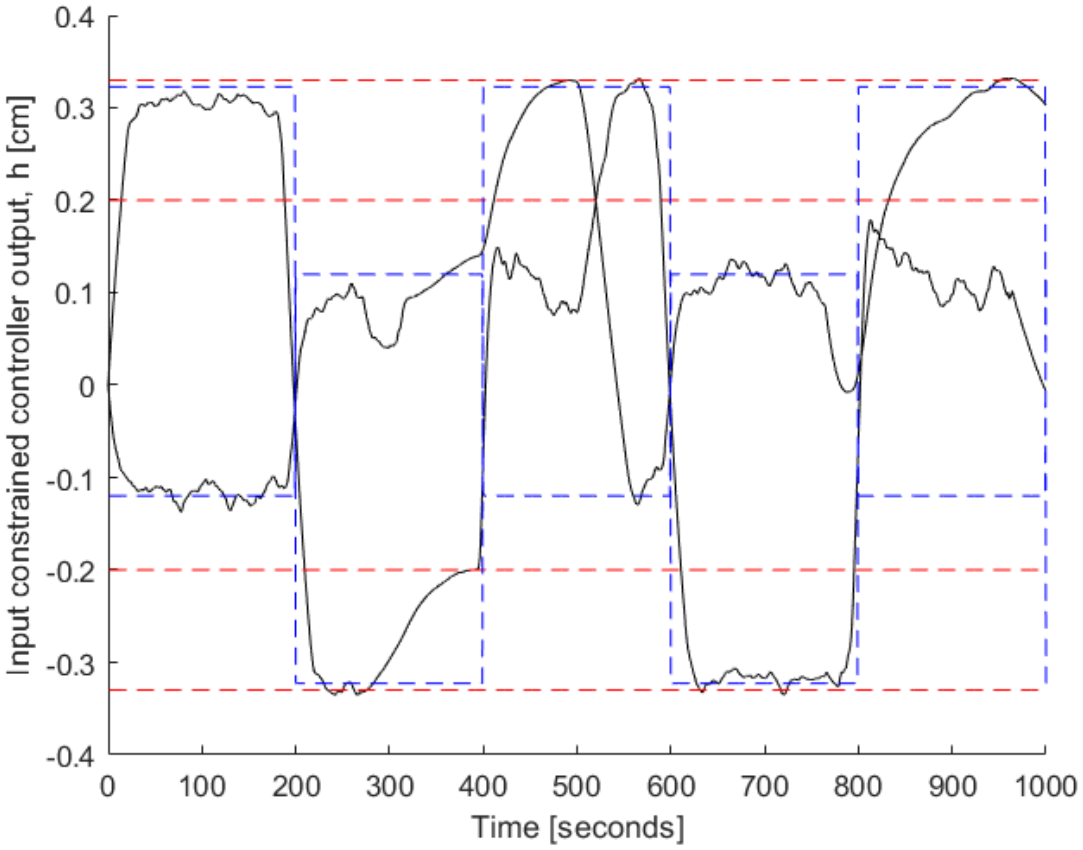
Here are the bounds declared:

```
ExtraFeatures.Bounds.Umin = -10*ones(horizon*kf_aug.nu,1);
ExtraFeatures.Bounds.Umax = 10*ones(horizon*kf_aug.nu,1);
ExtraFeatures.Bounds.DUmin = -100*ones(horizon*kf_aug.nu,1);
ExtraFeatures.Bounds.DUmax = 100*ones(horizon*kf_aug.nu,1);
```

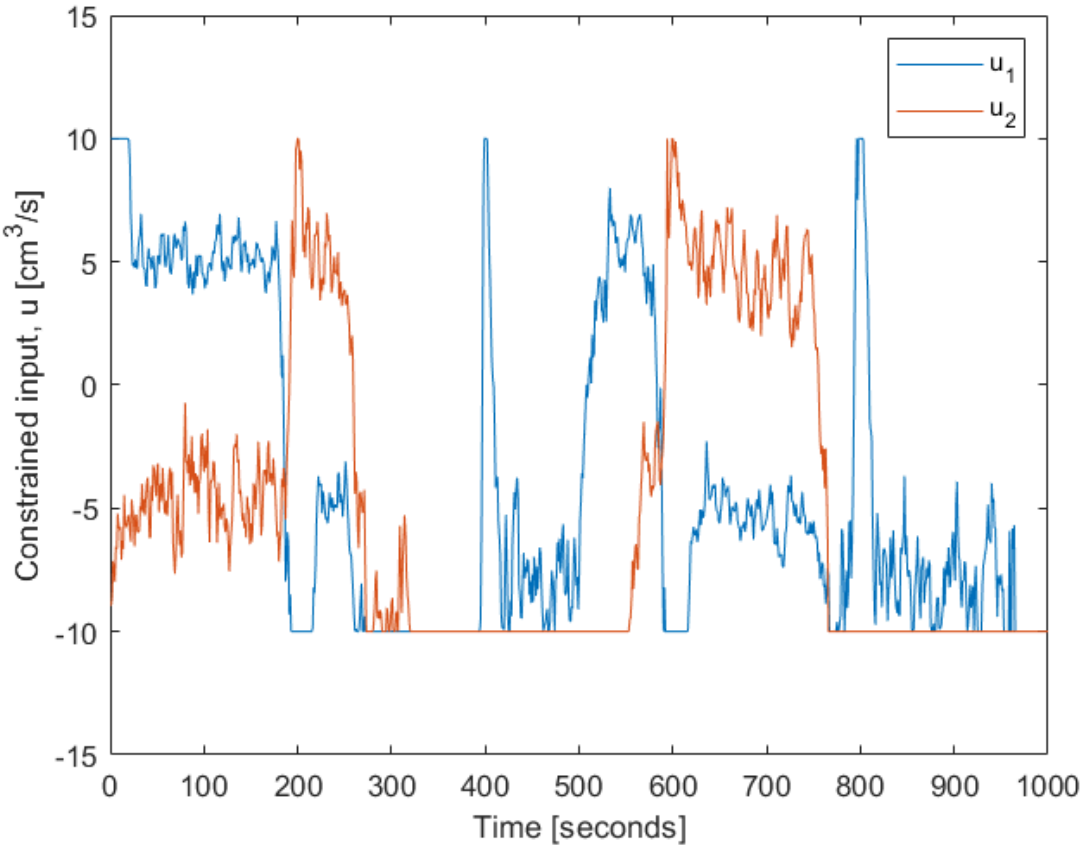
Then the simulation:

```
fighand = figure;
clf
hold on
for it = 1:length(t)-1
    % Kalman Filter, Markov predictor
    kf_aug.markovPredictor(U,y);
    % Model Predictive Control
    Z = reshape(R(:,it:it+horizon-1),kf_aug.nu*kf_aug.j,1);
    c{1} = Z-kf_aug.b;
    c{2} = kron(zeros(horizon,1),ones(2,1));
    c{3} = mpc_inputConstrained.I0*mpc_inputConstrained.u_1;
    ExtraFeatures.b = kf_aug.b;
    [u, U] = mpc_inputConstrained.controlCompute(c,ExtraFeatures);
    % Physical system and measurement
    fts.timestep([t(it); t(it+1)],ssInput.*deviation_dist(:,it)+[u; zeros(2,1)]);
```

```
    y = fts.record.meas(end,:)' - fts_vlow.hs;
    % Print prediction
    % Saving input
    record.u(1:kf_aug.nz,it) = u;
end
plot(fts.record.t,fts.record.x(:,1:2)/fts.rho/fts.A-fts.hs(1:2)','k')
plot(ones(itmax,4)*diag([heightBounds; -heightBounds]),'--r')
plot(R','--b')
hold off
xlim([0 itmax])
xlabel('Time [seconds]')
ylabel('Input constrained controller output, h [cm]')
saveas(fighand,[pwd '\figures\f18_inc_mpcoutput'],'png')
```




```
fighand = figure;
plot(record.u')
hold on
ylim([-15 15])
hold off
ylabel('Constrained input, u [cm^3/s]')
legend('u_1','u_2')
xlabel('Time [seconds]')
saveas(fighand,[pwd '\figures\f19_inc_mpcinput'],'png')
```



We can see that with a certain input constraint, the controller is not able to fulfill its objectives when there is a disturbance affecting the system - as it becomes saturated.

Problem 8 - MPC with input constrainst and soft output constraints

Question 8.1

We use soft optimization for the output, since in that case getting close to the output constraint alreadt starts to "warn" the optimizer about the danger. For this, we introduce new variables, which are the distances from the constraint. Furthermore, it gives the possibility to the controller to operate, while violating this constraint, if no other solution can be found. The constraints are

$$\Gamma U_k + b_k \geq R_{min,k} - b_k$$
$$S_k \geq 0$$
$$\Gamma U_k - T_k \geq R_{max,k} - b_k$$
$$T_k \geq 0$$

We need to augment the unconstrained optimization problem in this case, including

$$H_s = \bar{W}_{s,2}^T \bar{W}_{s,2}$$
$$g_s = \bar{W}_{s,1} e, \qquad e = [1 \quad 1 \quad ...]^T$$
$$H_t = \bar{W}_{t,2}^T \bar{W}_{t,2}$$
$$g_t = \bar{W}_{t,1} e$$

where the numbers refer to the order of the objective function: 2 denotes the part, where a quadratic, 1 denotes a part, where a linear objective term is minimized. The augmentation is straightforward from these information, having the optimized variable

$$\bar{x} = \begin{pmatrix} x \\ S_k \\ T_k \end{pmatrix} = \begin{pmatrix} U_k \\ S_k \\ T_k \end{pmatrix}.$$

Question 8.2

The same objects is used to create the input constrained and soft output constrained MPC.

Question 8.3

The computation time has significantly increased, and the optimization algorithm often got stuck, while struggling with the various constraints, and could not fulfill all of them. Therefore, to ease the challenge, the horizon is decreased.

```
% MPC with new horizon
horizon = 10;
kf_aug.reinitialize(initial,horizon);
mpc.preinitialize(horizon,n);
% Creation of the copy
mpc_ioC = copy(mpc);
% Number of objectives
n.nscl = 1;
n.nscu = 1;
% Initialization of weights and coefficients of objective functions
W = cell(5,1);
% Input and Soft-Output Constrained MPC
ExtraFeatures.useSoftOutputConstraints = true;
% Passing on the Markov matrix
ExtraFeatures.Markov = kf_aug.Markov;
% Weight on output, input, input rate
W{1} = diag([1e2 1e2]);     % Reference tracking
W{2} = diag([1 1]);         % Deviation from steady state input
W{3} = diag([1e-2 1e-2]);   % Input rate punishment
W{4} = diag([1e0 1e0]);     % Lower output constraint
W{5} = diag([1e0 1e0]);     % Higher output constraint
% Objective phrasing
Ucoeff{1} = kf_aug.Markov;
Ucoeff{2} = kron(eye(n.nu),eye(horizon));
Ucoeff{3} = mpc.Lambda;
% Initial input signal
mpc_ioC.initialize(W,Ucoeff,n,ExtraFeatures,u_1,optioptions);
```

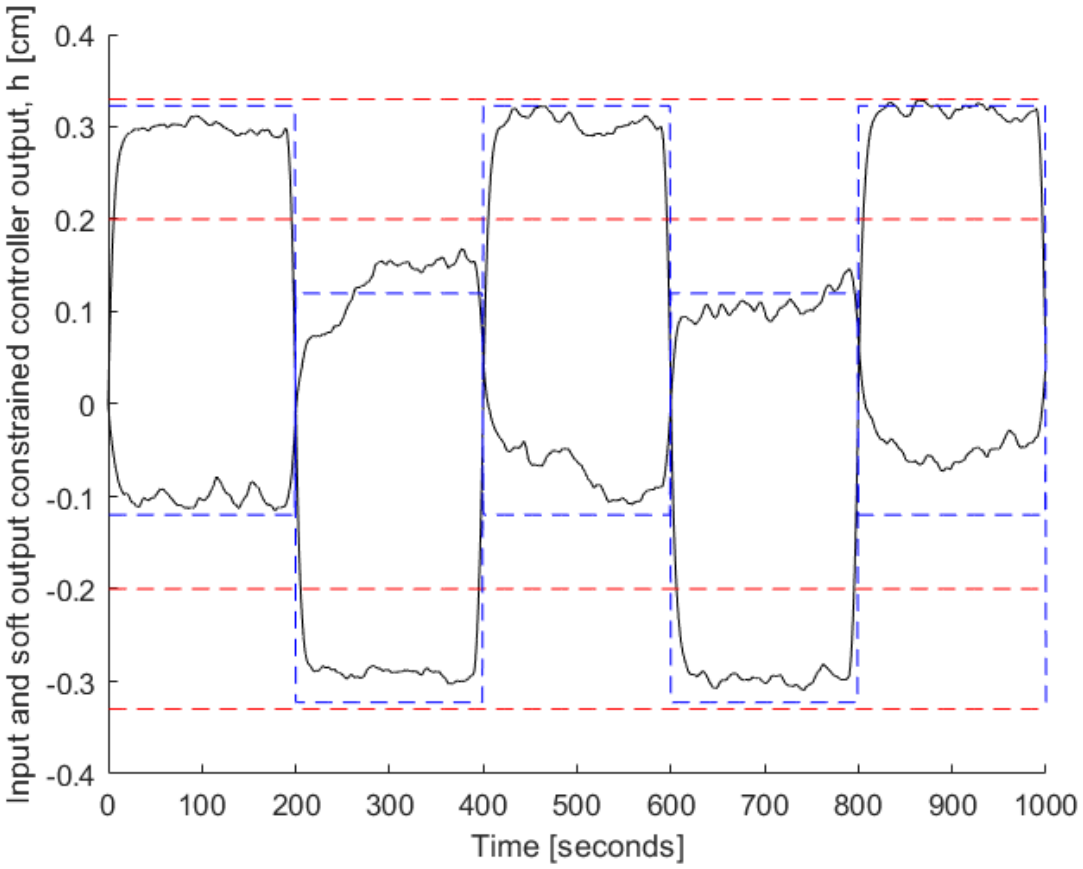
```
% Reinitialization
fts.reinitialize();
kf_aug.reinitialize(initial,horizon);
% The initial input vector and measurement
U = kron(zeros(horizon,1),ones(n.nu,1));
y = zeros(4,1);
% Declaration of record
record.u = zeros(n.nu,length(t)-1);
```

The input constraints are not so restrictive anymore:

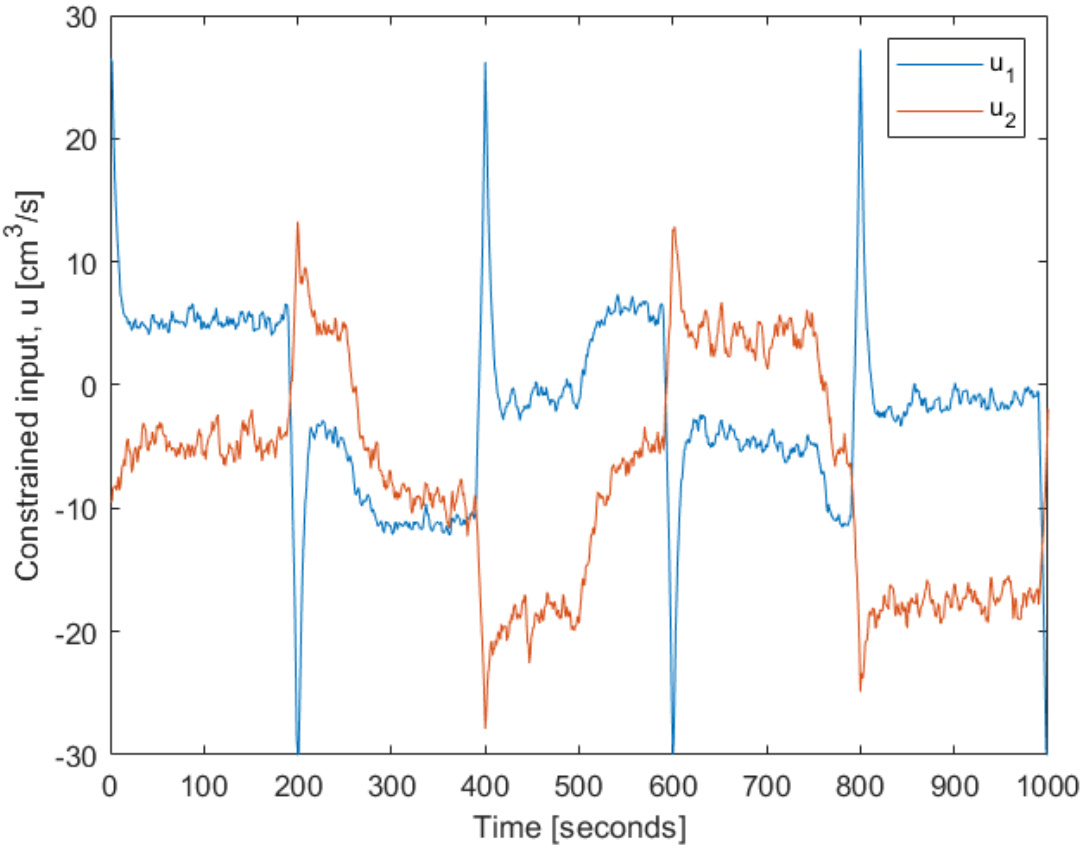
```
% Eased input constraints
ExtraFeatures.Bounds.Umin = -30*ones(horizon*kf_aug.nu,1);
ExtraFeatures.Bounds.Umax = 30*ones(horizon*kf_aug.nu,1);
ExtraFeatures.Bounds.DUmin = -300*ones(horizon*kf_aug.nu,1);
ExtraFeatures.Bounds.DUmax = 300*ones(horizon*kf_aug.nu,1);
```

Then we declare the output constraints:

```
% Declaration
ExtraFeatures.Bounds.Rmin = {-kron(ones(horizon,1),heightBounds)};
ExtraFeatures.Bounds.Rmax = {kron(ones(horizon,1),heightBounds)};
clf
hold on
for it = 1:length(t)-1
    % Kalman Filter, Markov predictor
    kf_aug.markovPredictor(U,y);
    % Model Predictive Control
    Z = reshape(R(:,it:it+horizon-1),kf_aug.nu*kf_aug.j,1);
    c{1} = Z-kf_aug.b;
    c{2} = kron(zeros(horizon,1),ones(2,1));
    c{3} = mpc_ioC.I0*mpc_ioC.u_1;
    ExtraFeatures.b = kf_aug.b;
    [u, U] = mpc_ioC.controlCompute(c,ExtraFeatures);
    % Physical system and measurement
    fts.timestep([t(it); t(it+1)],ssInput.*deviation_dist(:,it)+[u; zeros(2,1)]);
    y = fts.record.meas(end,:)' - fts_vlow.hs;
    % Print prediction
    %plot(it*Ts:Ts:(it-1+kf_aug.j)*Ts,kf_aug.zj(1:2:end),'k')
    %plot(it*Ts:Ts:(it-1+kf_aug.j)*Ts,kf_aug.zj(2:2:end),'k')
    % Saving input
    record.u(1:kf_aug.nz,it) = u;
end
plot(fts.record.t,fts.record.x(:,1:2)/fts.rho/fts.A-fts.hs(1:2)','k')
plot(ones(itmax,4)*diag([heightBounds; -heightBounds]),'--r')
plot(R','--b')
xlim([0 itmax])
hold off
xlabel('Time [seconds]')
ylabel('Input and soft output constrained controller output, h [cm]')
saveas(fighand,[pwd '\figures\f20_inoutc_mpcoutput'],'png')
```




```
fighand = figure;
plot(record.u')
ylabel('Constrained input, u [cm^3/s]')
legend('u_1','u_2')
xlabel('Time [seconds]')
saveas(fighand,[pwd '\figures\f21_inoutc_mpcinput'],'png')
```



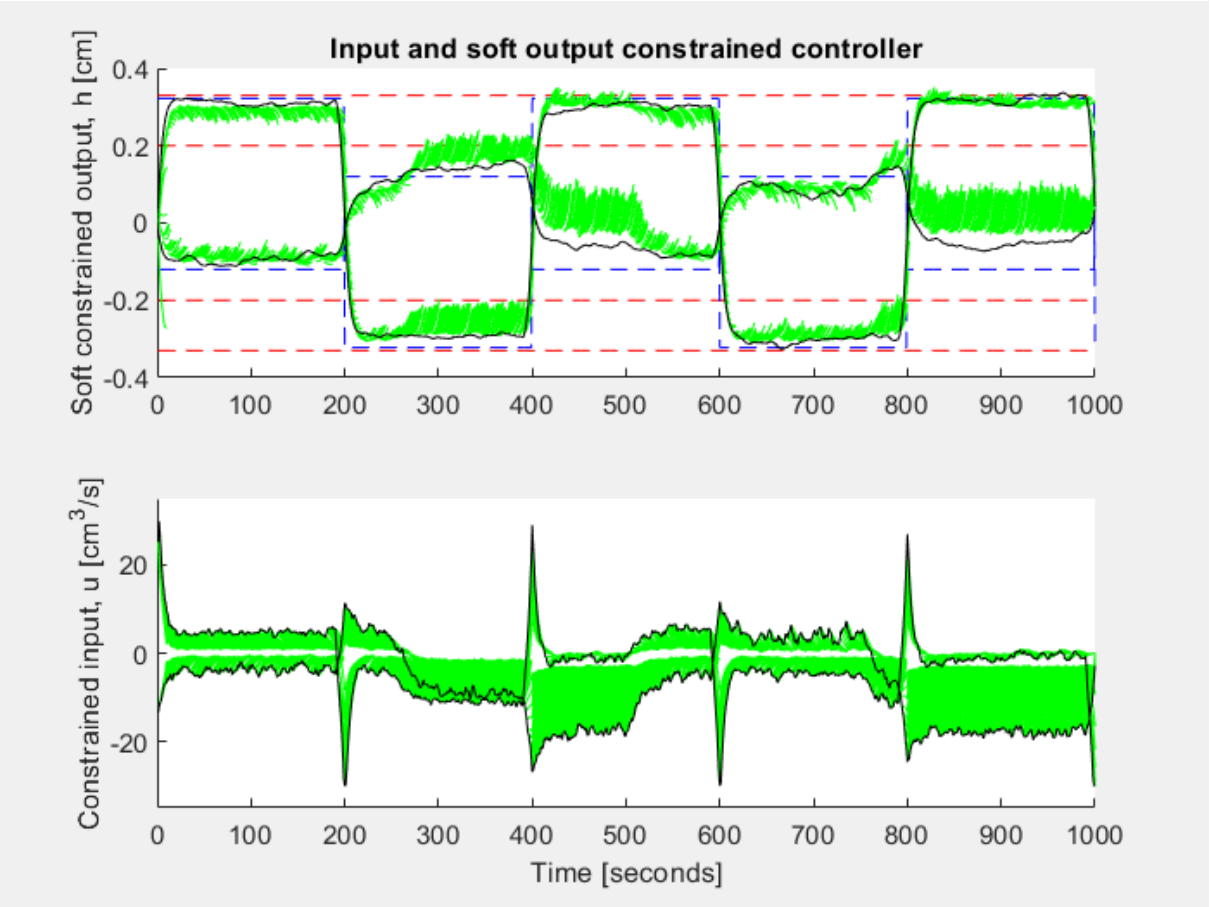
We can see that finally the avoidance of extreme values can be avoided with soft output constraints (though this means small steady state error in the reference tracking, not only in the height that is close to a bound, but a pair of it).

Problem 9 - Closed Loop Simulations

All plots were already produced. It is time to make a movie. Simulations on the nonlinear system is almost identical to the linear, when staying closely to the linearization point.

```
% Reinitialization
fts.reinitialize();
kf_aug.reinitialize(initial,horizon);
% Simulation
fighand = figure;
set(gcf,'Visible','on')
subplot(211)
hold on
plot(heightBounds(1)*ones(itmax,1),'--r')
plot(-heightBounds(1)*ones(itmax,1),'--r')
plot(heightBounds(2)*ones(itmax,1),'--r')
plot(-heightBounds(2)*ones(itmax,1),'--r')
plot(R','--b')
hold off
```

```
axis([0 itmax -0.4 0.4])
ylabel('Soft constrained output, h [cm]')
title('Input and soft output constrained controller')
% Initializing animations for the outputs
hand11 = animatedline('Color','k');
hand12 = animatedline('Color','k');
subplot(212)
axis([0 itmax -35 35])
ylabel('Constrained input, u [cm^3/s]')
xlabel('Time [seconds]')
% Initializing animations for the inputs
hand21 = animatedline('Color','k');
hand22 = animatedline('Color','k');
for it = 1:length(t)-1
    % Kalman Filter, Markov predictor
    kf_aug.markovPredictor(U,y);
    % Model Predictive Control
    Z = reshape(R(:,it:it+horizon-1),kf_aug.nu*kf_aug.j,1);
    c{1} = Z-kf_aug.b;
    c{2} = kron(zeros(horizon,1),ones(2,1));
    c{3} = mpc_ioC.I0*mpc_ioC.u_1;
    ExtraFeatures.b = kf_aug.b;
    [u, U] = mpc_ioC.controlCompute(c,ExtraFeatures);
    % Physical system and measurement
    fts.timestep([t(it); t(it+1)],ssInput.*deviation_dist(:,it)+[u; zeros(2,1)]);
    y = fts.record.meas(end,:)' - fts_vlow.hs;
    % Animation
    pause(0.001)
    addpoints(hand11,it,fts.record.x(end,1)/fts.A/fts.rho-fts.hs(1)')
    addpoints(hand12,it,fts.record.x(end,2)/fts.A/fts.rho-fts.hs(2)')
    addpoints(hand21,it,u(1))
    addpoints(hand22,it,u(2))
    % Print prediction
    subplot(211)
    hold on
    plot(it*Ts:Ts:(it-1+kf_aug.j)*Ts,kf_aug.zj(1:2:end),'g-')
    plot(it*Ts:Ts:(it-1+kf_aug.j)*Ts,kf_aug.zj(2:2:end),'g-')
    hold off
    subplot(212)
    hold on
    plot(it*Ts:Ts:(it-1+kf_aug.j)*Ts,U(1:2:end),'g-')
    plot(it*Ts:Ts:(it-1+kf_aug.j)*Ts,U(2:2:end),'g-')
    hold off
    % Saving input
    record.u(1:kf_aug.nz,it) = u;
end
uistack(hand12,'top');
uistack(hand11,'top');
uistack(hand21,'top');
uistack(hand22,'top');
hold off
saveas(fighand,[pwd '\figures\f22_movie'],'png')
```



We can see how the Markov predictions and the MPCs interact in the diagrams. The optimization of the controller usually predicts system behaviour, for which the input should go to zero quickly; however, then the Markov predictor signs that the system behaves slightly differently than the expected, due to process and measurement noises; hence the controller often cannot go to zero as quickly as it would like. Also, note that since the soft output control has high weights, when, in the middle of the diagram, the passing of it is predicted (due to the disturbance), the controller becomes more active than usually.

Problem 10 - Discussion and Conclusion

In this assignment, the design problems of a control engineer were faced from modelling a nonlinear, continuous time system, to designing a model predictive controller. Through these steps, several assumptions were made, where the right choice of design parameters could be crucial.

It was shown that in case of certain parameter sets, the sampled system becomes non-minimal phase, if the sampling time is kept constant, as zeros appear with time-wise discretizing the system. Therefore the sampling time has to be chosen to be small enough (relevantly smaller than the time constant of the quickest system dynamics) to be able to control the system.

It was shown that noise has different amount of distortion of information, if it effects different signal levels (in this case it was the water height). We could see that linearization is a good approach only until the system deviates too much from the linearization point, where in our case, both the transients and the steady states changed. Identification can only be used with a proper model, and if a transfer function is identified, it also

relies on linearity.It was shown that analytical linearization, in other words, Taylor expansion gives almost the same result as identification around the chosen steady state point.

Then linear Kalman filters were tested. The assumption was made, that if the sampling time is relevantly smaller than the time constant of the smallest system dynamics, the continuous system is well-estimated. It was shown that a dynamic Kalman filter adapts to an erronous situation better than a stationary one. We saw that the filter accummulates error away from the linearization point due to its linear design. It also turned out that a proper model of the system is crucial, since when we experienced disturbance with non-zero mean, the Kalman filter gave wrong estimations, until it was augmented with the knowledge about the disturbances.

The Model Predictive Controller turned to be a good approahc, however, it gave small steady state errors away from the linearization point. It was shown that too strict input constraints can cause the controller to saturate, failing to achieve the control objectives. It was also found that setting soft output contraints close to the reference signal, the controller has to trade off and does not track the reference so precisely anymore. Finally, it was shown in a movie, that the Kalman filter, the Markov predictor and the optimization based controllers are in continuous and strong interaction with each other, to fulfill their objectives.