02619 Model Predictive Control
Exam Assignment
Modified Four Tank System

Autumn 2021

Eduard Maximilian Fiedler s210134

Table of Contents

Problem 1. Control Structure.

1. What are the states, the measurement, the manipulated variables (MVs), the measured disturbance variables (DVs), and the controlled variables (CVs) for this system?

The states (x) will be based on the respective masses of the water in the tanks. It is assumed that the levels (y) in tanks 1 and 2 are measured, and that the levels in them are to be controlled (u) by the MVs F_1 and F_2. Disturbing the system (d) are the DVs F_3 and F_3. The CVs in the system are the set points ($\bar{z}$) in tanks 1 and 2, setting the height of water in the them. Thus, they are equal to the measured levels (y), when measurement noise is ignored.

The variables in the system may be described as follows:

$$\boldsymbol{x}(t) = \begin{bmatrix} m_1(t) \\ m_2(t) \\ m_3(t) \\ m_4(t) \end{bmatrix} [\mathrm{g}], \qquad \boldsymbol{y}(t) = \bar{\boldsymbol{z}}(t) = \begin{bmatrix} h_1(t) \\ h_2(t) \end{bmatrix} [\mathrm{cm}],$$
$$\boldsymbol{u}(t) = \begin{bmatrix} F_1(t) \\ F_2(t) \end{bmatrix} [\mathrm{cm}^3\,\mathrm{s}^{-1}], \qquad \boldsymbol{d}(t) = \begin{bmatrix} F_3(t) \\ F_4(t) \end{bmatrix} [\mathrm{cm}^3\,\mathrm{s}^{-1}]. \tag{1.1}$$

2. Draw a block diagram of the Modified-Four-Tank System with an MPC system. The MPC block should illustrate both the state estimator and regulator of the MPC.

The plant described by the given diagram is to be controlled by the MPC regulator. This regulator receives the estimated state, the desired set points, constraints on the output, as well as, the constraints on the input. The estimated state comes from a state estimator taking in the calculated input and the sensor output. The sensor output measures the tank heights from the plant, but experiences measurement noise. Finally, the plant receiving the regulator output, is having its output controlled whilst being disturbed.

These interactions are visualised in Figure 1.1.

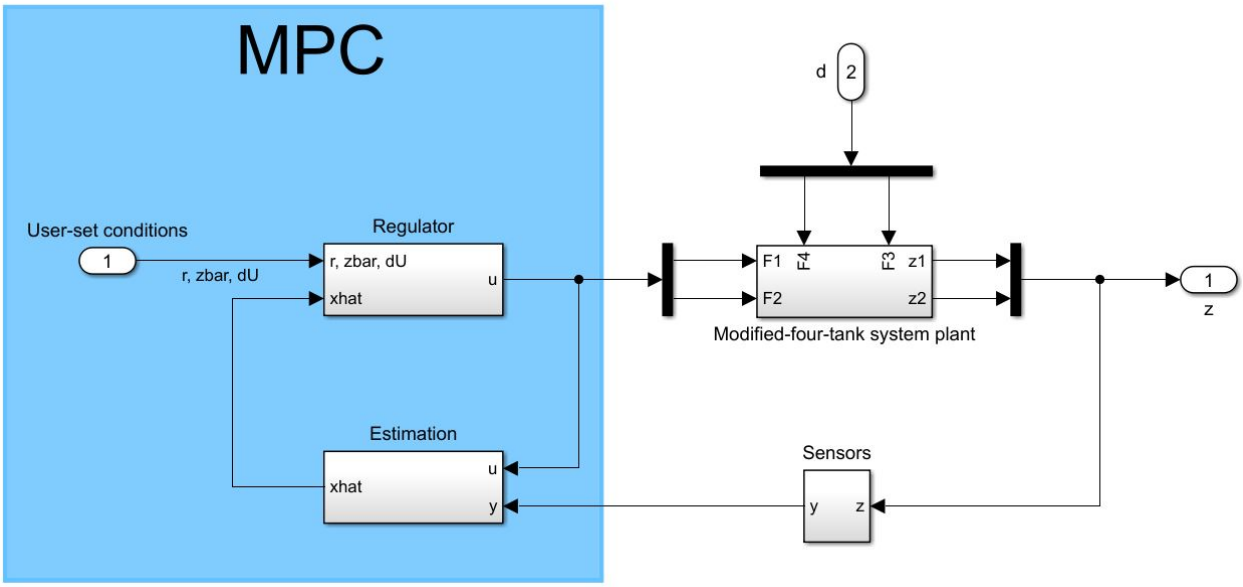

Figure 1.1: Block diagram of the MPC acting upon the modified-four-tank system.

Problem 2. Deterministic and Stochastic Nonlinear Modelling.

The modified-four-tank system is able to control the levels in the tanks 1 and 2 based on the amount of water being pumped into them. However, the difficulty in the control lies in how the respective inputs directly pump into a desired tank (1 or 2) and the tank above the opposing desired tank (4 or 3). The amount of water split between upper and lower levels by the control input is based on the distribution factor γ. Additionally, disturbances interfere with the control objective by directly influencing the levels in the upper tanks, and thus, the level in the lower tanks.

The physical characteristics of the tanks influence the amount of flow out of the tanks, as they affect the outlet pipe cross-sectional area (higher means more outflow), and the head (vertical outflow is directly correlated with the head of water). Thus, a tank with a higher cross-sectional area holding the same mass of water than a lower cross-sectional area tank, leads to a lower outflow rate in comparison.

The parameters in the system, their value, and unit are described in the following Table 2.1.

Table 2.1: Parameters used in the modified-four-tank systems.

Variable	Description	Value	Unit
$a_{1,2,3,4}$	Outlet pipe cross-sectional area	1.2272	cm^2
$A_{1,2,3,4}$	Tank cross-sectional area	380.1327	cm^2
γ_1	Flow distribution between tank 1 and tank 4	0.6	
γ_2	Flow distribution between tank 2 and tank 3	0.75	
g	Acceleration due to gravity	981	$\mathrm{cm\,s}^{-2}$
ρ	Density of water	1	$\mathrm{g\,cm}^{-3}$

1. Develop a deterministic nonlinear model.

The dynamics of the deterministic system were developed to be as follows:

$$\frac{\dot{\boldsymbol{x}}(t)}{\rho} = \begin{bmatrix} -a_1\sqrt{2gm_1(t)/(A_1\rho)} + a_3\sqrt{2gm_3(t)/(A_3\rho)} \\ -a_2\sqrt{2gm_2(t)/(A_2\rho)} + a_4\sqrt{2gm_4(t)/(A_4\rho)} \\ -a_3\sqrt{2gm_3(t)/(A_3\rho)} \\ -a_4\sqrt{2gm_4(t)/(A_4\rho)} \end{bmatrix} + \begin{bmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \\ 0 & 1-\gamma_2 \\ 1-\gamma_1 & 0 \end{bmatrix} \boldsymbol{u}(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{d}(t)\ . \tag{2.1}$$

The measurements of the tank 1 and 2 heights, which are equal to the output, were modelled as follows:

$$\boldsymbol{y}(t) = \boldsymbol{z}(t) = \begin{bmatrix} 1/(A_1\rho) & 0 & 0 & 0 \\ 0 & 1/(A_2\rho) & 0 & 0 \end{bmatrix} \boldsymbol{x}(t)\ , \tag{2.2}$$

they are considered the same in a deterministic model.

2. Develop a stochastic nonlinear model.

The mathematical model with the piecewise constant stochastic disturbances was developed to be:

$$\frac{\dot{\boldsymbol{x}}(t)}{\rho} = \begin{bmatrix} -a_1\sqrt{2gm_1(t)/(A_1\rho)} + a_3\sqrt{2gm_3(t)/(A_3\rho)} \\ -a_2\sqrt{2gm_2(t)/(A_2\rho)} + a_4\sqrt{2gm_4(t)/(A_4\rho)} \\ -a_3\sqrt{2gm_3(t)/(A_3\rho)} \\ -a_4\sqrt{2gm_4(t)/(A_4\rho)} \end{bmatrix} + \begin{bmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \\ 0 & 1-\gamma_2 \\ 1-\gamma_1 & 0 \end{bmatrix} \boldsymbol{u}(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{d}_k\,, \tag{2.3}$$

where the value of the constant disturbance within the time intervals was based on:

$$\boldsymbol{d}_k \sim N\left(\begin{bmatrix} F_3 \\ F_4 \end{bmatrix}, \begin{bmatrix} \sigma_{F_3} & 0 \\ 0 & \sigma_{F_4} \end{bmatrix}\right). \tag{2.4}$$

The measurement model was given by:

$$\boldsymbol{y}(t) = \begin{bmatrix} 1/(A_1\rho) & 0 & 0 & 0 \\ 0 & 1/(A_2\rho) & 0 & 0 \end{bmatrix} \boldsymbol{x}(t) + \boldsymbol{v}(t) \ , \tag{2.5}$$

where:

$$\boldsymbol{v}(t) \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, R_{vv}(p)\right). \tag{2.6}$$

The output of the system is based on the physical level of the tanks. Therefore, it is essentially the measurement without the noise.

$$\boldsymbol{z}(t) = \begin{bmatrix} 1/(A_1\rho) & 0 & 0 & 0 \\ 0 & 1/(A_2\rho) & 0 & 0 \end{bmatrix} \boldsymbol{x}(t) \ . \tag{2.7}$$

3. Develop a stochastic nonlinear model (SDE).

To develop the dynamics of the SDE, the disturbance $\boldsymbol{d}(t)$ from Problem 2.1. had to first be represented as a disturbance model. For this assignment's purpose, this was chosen to be as follows:

$$\mathrm{d}\boldsymbol{d}(t) = 0 \cdot \mathrm{d}t + \begin{bmatrix} \sigma_{F_3} & 0 \\ 0 & \sigma_{F_4} \end{bmatrix} \mathrm{d}\omega_d(t) \ , \tag{2.8}$$

$$\boldsymbol{d}(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} F_3(t) \\ F_4(t) \end{bmatrix} , \tag{2.9}$$

note that σ_{F_3} and σ_{F_4} are not the same as in Problem 2.2.

The stochastic term was defined as a Brownian motion, therefore:

$$\mathrm{d}\omega_d(t) \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathrm{d}t\right) , \tag{2.10}$$

The system was then augmented such that it incorporated $\boldsymbol{d}(t)$ as a stochastic variable as follows:

$$\mathrm{d}\begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{d}(t) \end{bmatrix} = \rho \left(\begin{bmatrix} -a_1\sqrt{2gm_1(t)/(A_1\rho)} + a_3\sqrt{2gm_3(t)/(A_3\rho)} \\ -a_2\sqrt{2gm_2(t)/(A_2\rho)} + a_4\sqrt{2gm_4(t)/(A_4\rho)} \\ -a_3\sqrt{2gm_3(t)/(A_3\rho)} \\ -a_4\sqrt{2gm_4(t)/(A_4\rho)} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \\ 0 & 1-\gamma_2 \\ 1-\gamma_1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \boldsymbol{u}(t) \right.$$

$$\left. + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \boldsymbol{d}(t) \right) \mathrm{d}t + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \sigma_{F_3} & 0 \\ 0 & \sigma_{F_4} \end{bmatrix} \mathrm{d}\omega_d(t) \ , \tag{2.11}$$

note, that $\boldsymbol{x}(t)$ kept the same structure as in previous questions (as the masses in each tank) for this and subsequent equations.

The measurement model was, therefore, given by:

$$\boldsymbol{y}(t) = \begin{bmatrix} 1/(A_1\rho) & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/(A_2\rho) & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{d}(t) \end{bmatrix} + \boldsymbol{v}(t) \ , \tag{2.12}$$

where $\boldsymbol{v}(t)$ was described in the same manner as in Equation 2.6.

The introduction of the disturbance model, required the output to be changed as follows:

$$\boldsymbol{z}(t) = \begin{bmatrix} 1/(A_1\rho) & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/(A_2\rho) & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{d}(t) \end{bmatrix} . \tag{2.13}$$

Problem 3. Nonlinear Simulation – Step Responses.

For these and subsequent simulations, the system was initialised with the following inputs and the resulting states at the operating point:

$$\boldsymbol{u}(t) = \begin{bmatrix} 250 \\ 325 \end{bmatrix} \mathrm{cm}^3\,\mathrm{s}^{-1}, \quad \boldsymbol{d}(t) = \begin{bmatrix} 100 \\ 100 \end{bmatrix} \mathrm{cm}^3\,\mathrm{s}^{-1}, \quad \boldsymbol{x}(t) = \begin{bmatrix} 14.116 \\ 24.333 \\ 4.226 \\ 5.146 \end{bmatrix} \cdot 10^3\,\mathrm{g}. \tag{3.1}$$

1. Simulate the step responses for 10%, 25%, and 50% steps in the manipulated variables. Do this for the deterministic model.

To simulate the system, the system dynamics were first described in a function that could then be called by `ode15s()` for evaluation (see Appendix A). A loop was formed such that over a set simulation time, would call the evaluation of the nonlinear system dynamics using the information of the previous loop evaluation. This wasn't important for the deterministic response, however, for the noisy response, this allowed the injection of stochastic variables in set intervals.

The time frame of the overall simulation, as well as, the evolution of the output were recorded for the set percentage changes in the input. This meant that 6 simulations were made, 3 each for changing one of the inputs based on the given percentage and keeping the other constant. The step change in the input always occurred at $t = 0\,\mathrm{s}$. The result is shown in Figure 3.1.

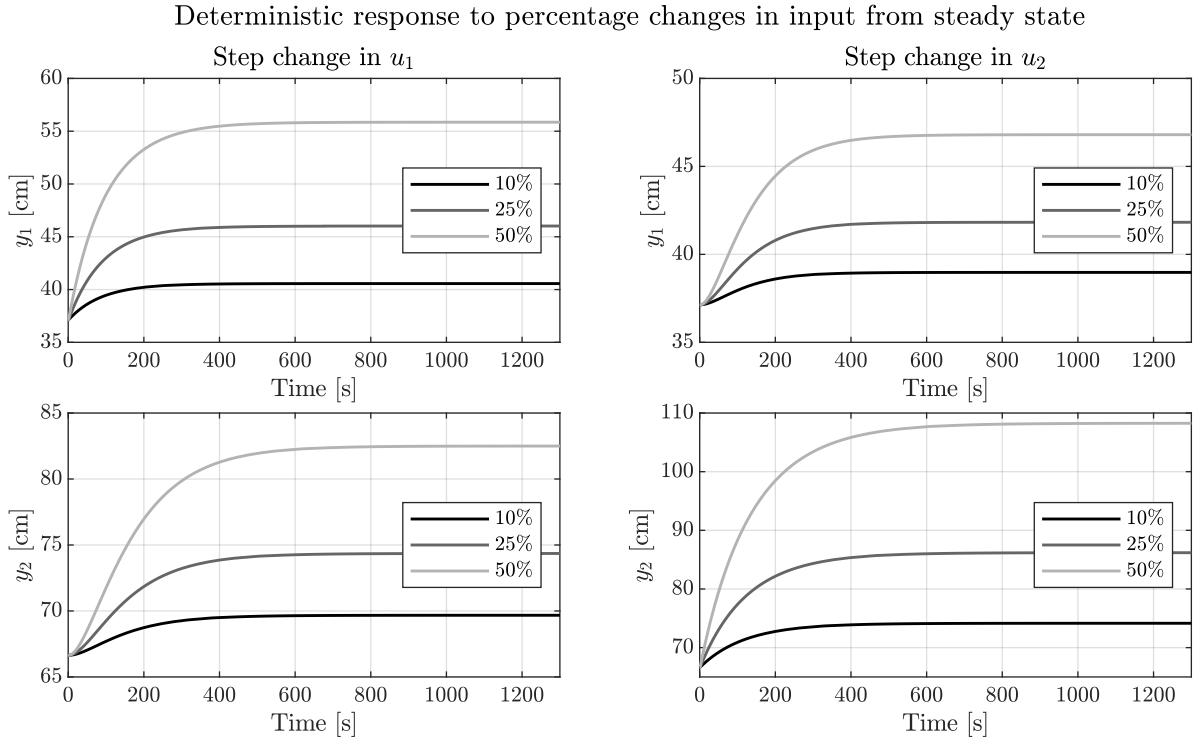

Figure 3.1: Plots on the left show the response of a step change in the first input while keeping the second input constant. The plots on the right show the result of the opposite occurring.

2. Simulate the step responses for 10%, 25%, and 50% steps in the manipulated variables. In this case you should include process and measurement noise. Try 3 different noise levels.

To simulate noise occurring in the process, as well as in the output, when the function `ode15s()` called the modified-four-tank system dynamics, a changing d was injected at every loop interval. Then, the output of the system was disturbed by adding a random variable to the evaluated output every step.

Within the loop, this was achieved by doing the following:

```
[T,X] = ode15s(@ModifiedFourTankSystem,[time time+1],x0,[],u,d+chol
    (10*rvv)'*randn(2,1),p);
noise = chol(rvv)'*randn(2,1);
y = [y;(1/(p(5)*p(12))).*X(end,1)+noise(1) (1/(p(6)*p(12))).*X(end,2)+
    noise(2)];
```

where `rvv` was the covariance matrix determining the amount of noise. It was decided, that the noise in the process would be 10 times bigger than that in the output as the states would fluctuate at a higher scale due to the unit conversion.

For the presented case in Figure 3.2, the chosen covariance matrix for the noise was:

$$\texttt{rvv} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}, \tag{3.2}$$

this corresponded to what was the medium noise level. The higher noise level used a covariance matrix 10 times larger, the lower noise level used a covariance matrix 10 times lower. The responses for these noise levels are given in Appendix B.

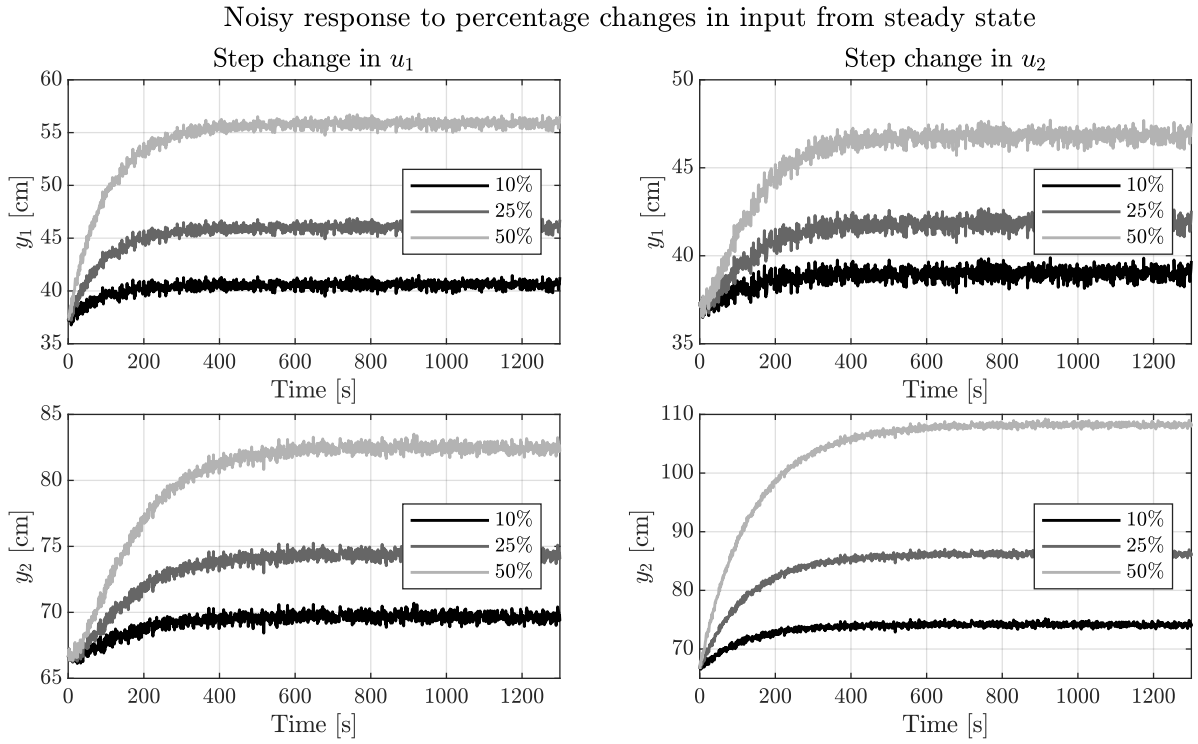

Figure 3.2: Plots on the left show the response of a step change in the first input while keeping the second input constant. The plots on the right show the result of the opposite occurring. Here, a medium level noise was injected in the process as well as in the output.

3. In all cases compute and plot the normalized steps.

The plots shown in Figure 3.1 & 3.2 could also be presented as a normalized step, by dividing the output by the initial output. This is shown Figures 3.3 & 3.4, respectively.

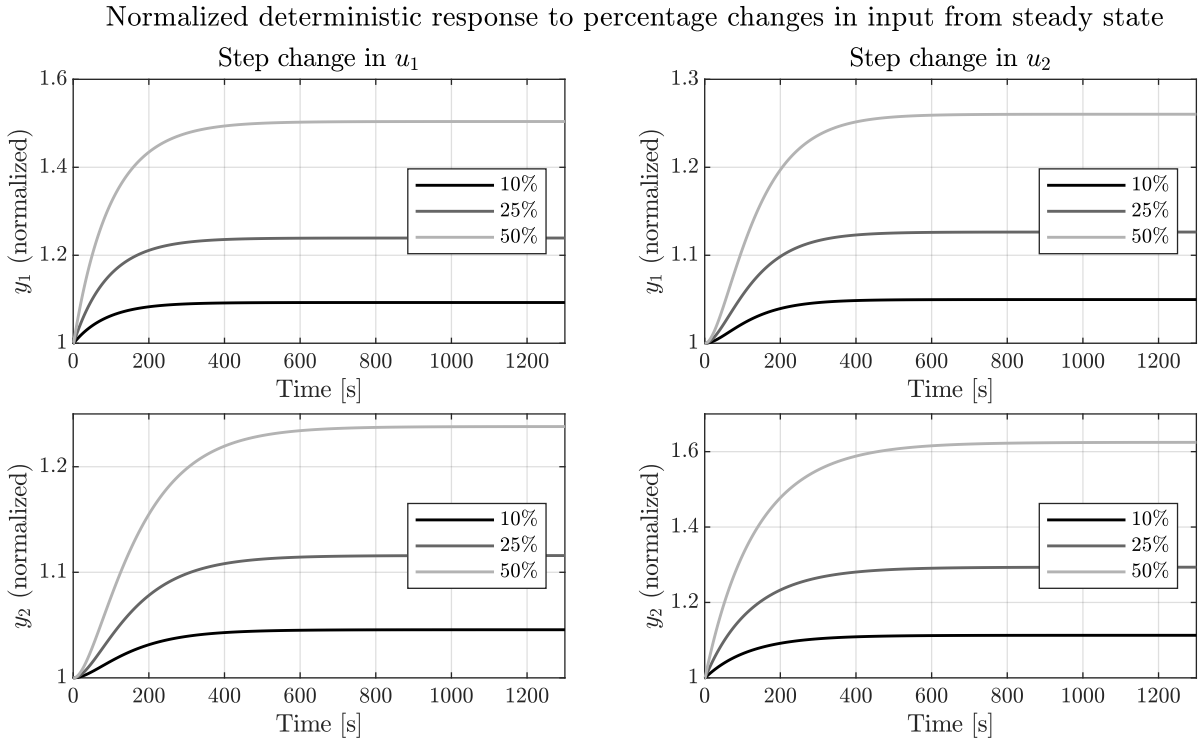

Figure 3.3: Normalized plots on the left show the response of a step change in the first input while keeping the second input constant. The plots on the right show the result of the opposite occurring.

4. From the normalized steps, identify a transfer function for the four tank system.

Observing the step responses, it can be seen (e.g. from Figure 3.1) that when the step change is in the input that directly flows into the measured tank ($u_1 \to y_1$ or $u_2 \to y_2$) then the level in the tank evolves with a first-order response. However, when the step change in the input feeds into the tank above it first ($u_1 \to y_2$ or $u_2 \to y_1$), then the response is slightly second order without oscillations.

For simplicity, all responses were modelled as first-order transfer functions. Thus, obtaining the transfer function was based on matching the gain to the response and identifying the time to reach 63% of the steady-state value.

As the time constant was simply found by using the function `find()` with the correct conditions, the gain was found based on the following formula:

$$K = \frac{y_\infty - y_0}{\Delta u}\,, \tag{3.3}$$

where y in this case is based on the tank being inspected and Δu being based on the change in the input being stepped in.

It then followed that a first-order transfer function is given by:

$$G\left(s\right) = \frac{K}{\tau s + 1}\,, \tag{3.4}$$

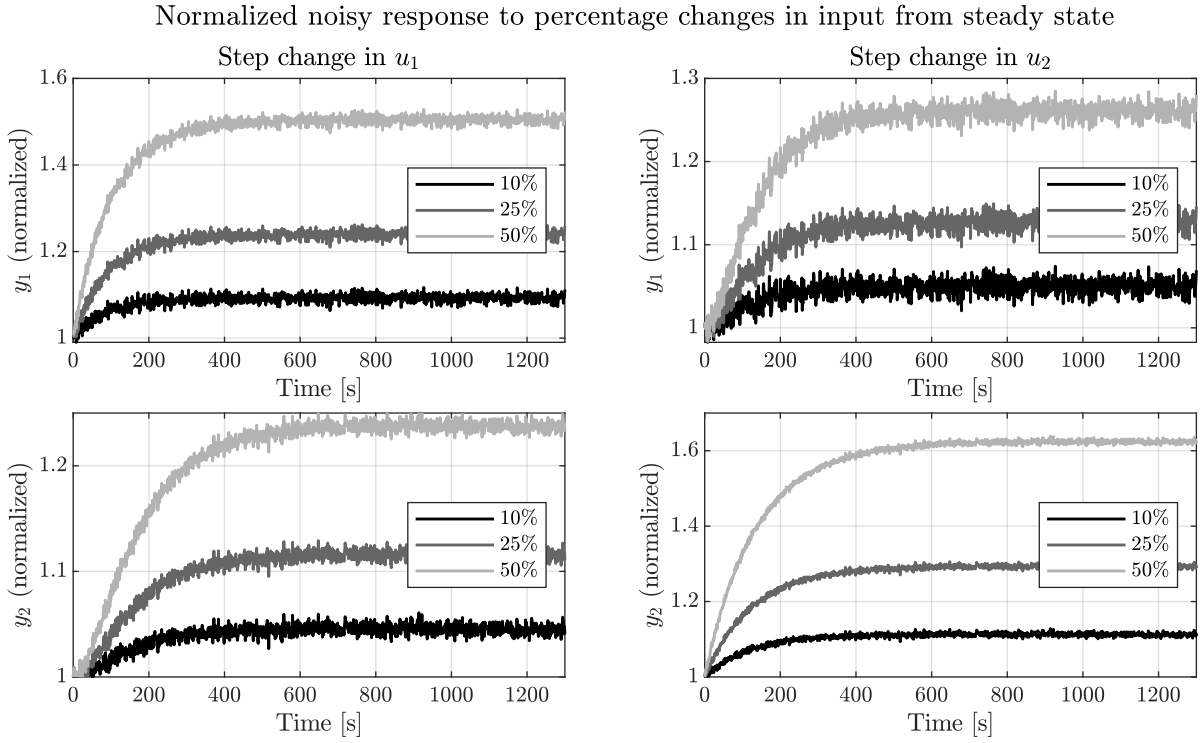

Figure 3.4: Normalized plots on the left show the response of a step change in the first input while keeping the second input constant. The plots on the right show the result of the opposite occurring. Here, a medium level noise was injected in the process as well as in the output.

allowing the identification of the full system based on the relation of the measured outputs to the inputs as follows:

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \end{bmatrix} = \begin{bmatrix} \dfrac{0.1432}{93.23s+1} & \dfrac{0.058}{148.2s+1} \\ \dfrac{0.124}{184.3s+1} & \dfrac{0.2428}{126.7s+1} \end{bmatrix} \begin{bmatrix} U_1(s) \\ U_2(s) \end{bmatrix}. \tag{3.5}$$

It was found that the gains and time constants changed to a limited extent based on the percentage change in the input, when the identified transfer functions were compared to the deterministic response. However, this was due to the nonlinear system being described in the linear transfer function form. As a result, the gains and time constants were identified for the 10%, 25%, and 50% step changes and averaged to obtain a reasonable approximated system.

5. Report the identified linear model estimate from the step responses. Discuss the accuracy of the model and the requirements of a step experiment.

From the transfer-function model in Equation 3.5, the characteristics were recorded in a manner such that the provided MATLAB function *mimoctf2dss* calculated the system (see Appendix C).

The sampling time used was based on the sampling time found in Problem 4.5., which was 5 s.

The resulting system was found to approximately be:

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} 0.9626 & -0.0012 & -0.0080 & -0.0029 \\ 0.0062 & 0.9415 & -0.0126 & 0.0061 \\ -0.0060 & 0.0184 & 0.9788 & -0.0036 \\ -0.0036 & -0.0030 & -0.0038 & 0.9662 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} -0.0589 & -0.0873 \\ 0.0673 & -0.0357 \\ -0.0131 & -0.0016 \\ -0.0008 & -0.0055 \end{bmatrix} \boldsymbol{u}_k \,, \tag{3.6}$$

$$\boldsymbol{y}_k = \begin{bmatrix} -0.0495 & 0.0679 & 0.0008 & -0.0052 \\ -0.0930 & -0.0347 & -0.0134 & -0.0017 \end{bmatrix} \boldsymbol{x}_k \,. \tag{3.7}$$

This system was simulated and compared to the noisy response subjected to the same conditions, the result is shown in Figure 3.5.

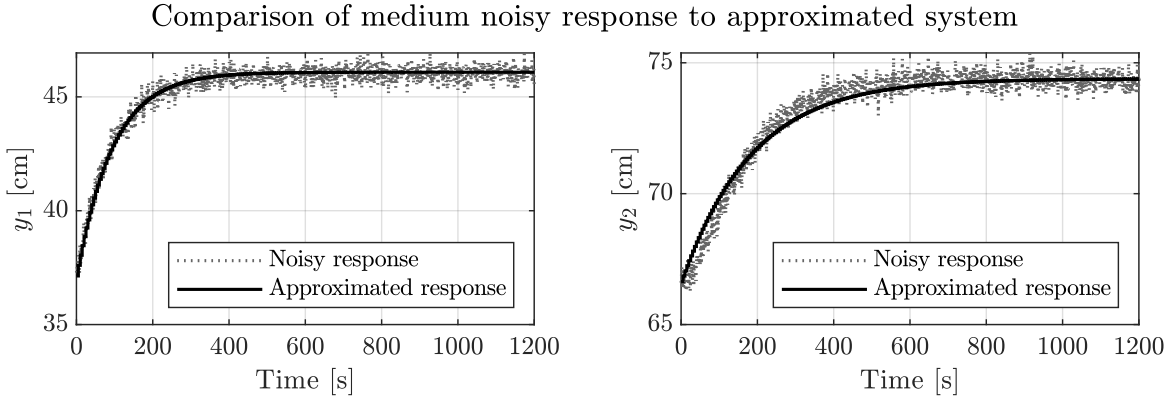

Figure 3.5: The approximated system was simulated for a 25% step change in u_1. This was compared to the medium noisy response to the same change in input.

Given that u_1 feeds directly into tank 1 and indirectly to tank 2, it can be seen that the approximated system closely follows the response in y_1. Expectedly, for y_2, the slightly second-order response of the nonlinear system has the approximated response deviate slightly. Nevertheless, the gains were selected correctly, as the final values were very close between the systems for the 25% step change. The first-order approximation of the second-order response rose slightly faster and settled slightly lower. Overall, the approximation using first-order transfer functions is reasonably close to the nonlinear system given that the original nonlinear system evolves predominantly in a first-order fashion (no oscillations).

A physical step experiment requires an instantaneous change in the manipulated variables. Therefore, it would not be physically possible in most real systems. Instead, the step would have to be approximated over a finite amount of time, reducing the accuracy of an approximated system using this method.

6. Compute the corresponding impulse response coefficients and plot them in appropriate plots.

Knowing the approximated linear system (Equation 3.7), the approximated Markov parameters based on the zero-state input response could be calculated. This involved calculating a vector of the successive Markov parameters, which was also used in the computation of the MPC in later problems.

```
function H = markovvec(A,B,C,n)

si = size(C,1);
H = zeros(si*n,si);

% Intermediate variable
```

```
H_i = C;
for i = 1:si:n*si
    H(i:i+si-1,:) = H_i*B;
    H_i = H_i*A;
end

end
```

The coefficients within each Markov parameter were plotted through the respective position in the vector. The result is shown in Figure 3.6.

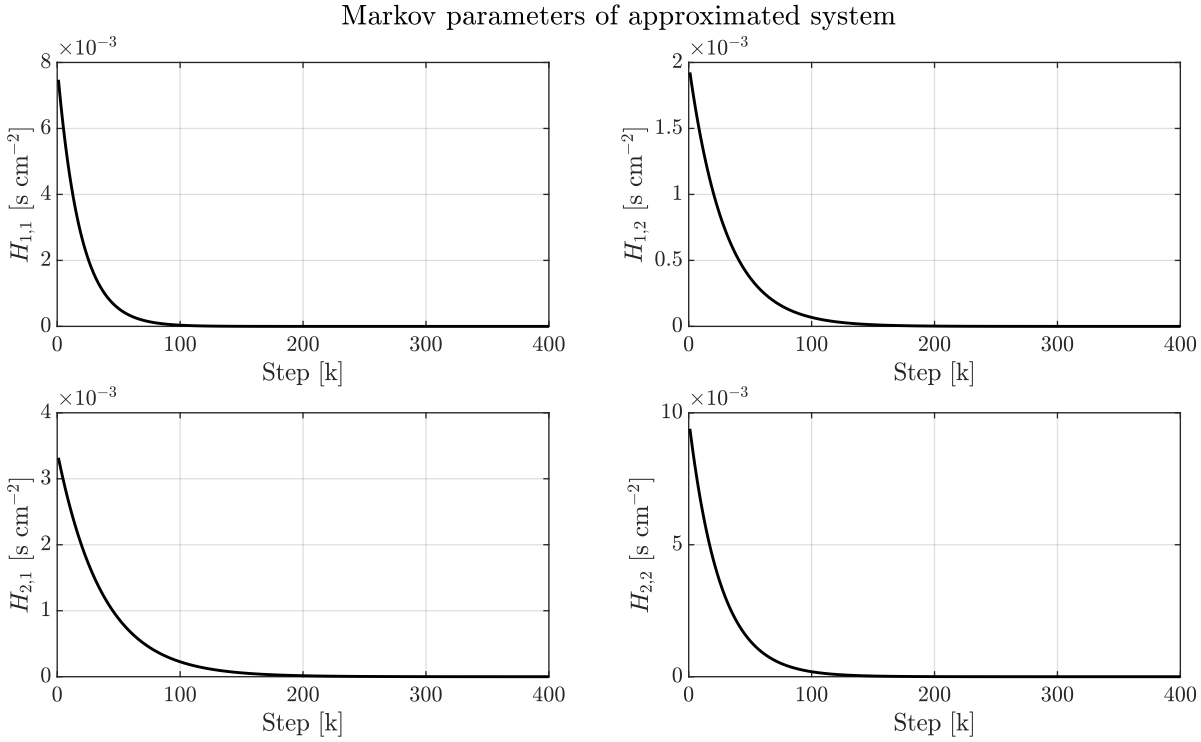

Figure 3.6: Evolution of Markov parameters calculated over 400 steps.

The result is a decreasing exponential in each respective coefficient evolution over successive steps. This is due to each response being modelled as a first-order transfer function. Additionally, it was found that the parameters in the diagonals decrease at a faster rate than those in the off diagonal. This is explained by the fact that these evolutions were based on the direct step change input into the respective lower tank, whereas the off-diagonals characterise the flow that passes through the upper tanks first, and then into the measured tank. These responses had a relatively higher gain, and lower time constant in comparison to the responses based in the off-diagonals (see Equation 3.5).

Problem 4. Linearisation and Discretisation.

1. Compute continuous-time linearised models for the 3 models developed in Problem 2.

1.1. Linearised deterministic model

The matrices of the nonlinear deterministic model could be differentiated accordingly giving the following result:

$$\boldsymbol{A} = \rho \begin{bmatrix} -\frac{a_1 g}{A_1 \rho}\sqrt{\frac{A_1 \rho}{2 g m_{10}}} & 0 & \frac{a_3 g}{A_3 \rho}\sqrt{\frac{A_3 \rho}{2 g m_{30}}} & 0 \\ 0 & -\frac{a_2 g}{A_2 \rho}\sqrt{\frac{A_2 \rho}{2 g m_{20}}} & 0 & \frac{a_4 g}{A_4 \rho}\sqrt{\frac{A_4 \rho}{2 g m_{40}}} \\ 0 & 0 & -\frac{a_3 g}{A_3 \rho}\sqrt{\frac{A_3 \rho}{2 g m_{30}}} & 0 \\ 0 & 0 & 0 & -\frac{a_4 g}{A_4 \rho}\sqrt{\frac{A_4 \rho}{2 g m_{40}}} \end{bmatrix},$$

$$\boldsymbol{B} = \rho \begin{bmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \\ 0 & 1-\gamma_2 \\ 1-\gamma_1 & 0 \end{bmatrix}, \qquad \boldsymbol{B_v} = \rho \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \boldsymbol{C} = \begin{bmatrix} \frac{1}{A_1 \rho} & 0 & 0 & 0 \\ 0 & \frac{1}{A_2 \rho} & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{D} = 0\,, \tag{4.1}$$

where $m_{(\cdot)0}$ denotes the respective stationary state used to linearise the system.

Using this, the stationary definitions given in Equation 3.1 were substituted to obtain the deterministic linearised system as:

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -0.0117 & 0 & 0.0214 & 0 \\ 0 & -0.0088 & 0 & 0.0194 \\ 0 & 0 & -0.0214 & 0 \\ 0 & 0 & 0 & -0.0194 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0.60 & 0 \\ 0 & 0.75 \\ 0 & 0.25 \\ 0.40 & 0 \end{bmatrix} \boldsymbol{u}(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{d}(t)\,, \tag{4.2}$$

$$\boldsymbol{y}(t) = \boldsymbol{z}(t) = \begin{bmatrix} 0.0026 & 0 & 0 & 0 \\ 0 & 0.0026 & 0 & 0 \end{bmatrix} \boldsymbol{x}(t)\,. \tag{4.3}$$

However, it must be noted that the states, inputs, disturbances, and outputs are now based on the deviation variables. Meaning that everything is offset by the set operating point conditions from Equation 3.1.

1.2. Linearised stochastic model

The stochastic model with the sampled disturbance was modelled as follows:

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -0.0117 & 0 & 0.0214 & 0 \\ 0 & -0.0088 & 0 & 0.0194 \\ 0 & 0 & -0.0214 & 0 \\ 0 & 0 & 0 & -0.0194 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0.60 & 0 \\ 0 & 0.75 \\ 0 & 0.25 \\ 0.40 & 0 \end{bmatrix} \boldsymbol{u}(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{d}_k\,, \tag{4.4}$$

$$\boldsymbol{y}(t) = \begin{bmatrix} 0.0026 & 0 & 0 & 0 \\ 0 & 0.0026 & 0 & 0 \end{bmatrix} \boldsymbol{x}(t) + \boldsymbol{v}(t)\,, \tag{4.5}$$

$$\boldsymbol{z}(t) = \begin{bmatrix} 0.0026 & 0 & 0 & 0 \\ 0 & 0.0026 & 0 & 0 \end{bmatrix} \boldsymbol{x}(t)\,, \tag{4.6}$$

where the measurement noise $\boldsymbol{v}(t)$ could be kept the same as in 2.2., but the process noise had to be altered, as the linearised model was not based on the deviation variable. Therefore:

$$\boldsymbol{d}_k \sim N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\sigma_{F_3} & 0\\ 0 & \sigma_{F_4}\end{bmatrix}\right). \tag{4.7}$$

1.3. Linearised stochastic model (SDE)

The SDE from Problem 2.3. developed into the following linearised model:

$$\begin{aligned}\mathrm{d}\boldsymbol{x}(t) = &\left(\begin{bmatrix}-0.0117 & 0 & 0.0214 & 0 & 0 & 0\\ 0 & -0.0088 & 0 & 0.0194 & 0 & 0\\ 0 & 0 & -0.0214 & 0 & 1 & 0\\ 0 & 0 & 0 & -0.0194 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\boldsymbol{x}(t) + \begin{bmatrix}0.60 & 0\\ 0 & 0.75\\ 0 & 0.25\\ 0.40 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\boldsymbol{u}(t)\right)\mathrm{d}t\\ &+\begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ \sigma_{F_3} & 0\\ 0 & \sigma_{F_4}\end{bmatrix}\mathrm{d}\omega_d(t)\,,\end{aligned} \tag{4.8}$$

$$\boldsymbol{y}(t) = \begin{bmatrix}0.0026 & 0 & 0 & 0 & 0 & 0\\ 0 & 0.0026 & 0 & 0 & 0 & 0\end{bmatrix}\boldsymbol{x}(t) + \boldsymbol{v}(t)\,, \tag{4.9}$$

$$\boldsymbol{z}(t) = \begin{bmatrix}0.0026 & 0 & 0 & 0 & 0 & 0\\ 0 & 0.0026 & 0 & 0 & 0 & 0\end{bmatrix}\boldsymbol{x}(t)\,, \tag{4.10}$$

where the Brownian motion term and measurement noise remained the same as shown in in 2.3. It must be noted, however, that the states used are not the same as in the previous linearised models. They are given as follows:

$$\boldsymbol{x}(t) = \begin{bmatrix}m_1(t)\\ m_2(t)\\ m_3(t)\\ m_4(t)\\ F_3(t)\\ F_4(t)\end{bmatrix}. \tag{4.11}$$

2. Compute the gains, poles and zeros of these models.

Using the MATLAB function `zpk`, the gains, poles, and zeros of the models could be found (deterministic is only being considered as it defines the core relations from inputs to outputs).

The gains based on u_1 onto y_1 and y_2 respectively were given as:

$$K = 0.0016,\, 2.0448 \cdot 10^{-5}\,, \tag{4.12}$$

furthermore, the gains based on u_2 onto y_1 and y_2 respectively were given as:

$$K = 1.4102 \cdot 10^{-5},\, 0.002\,. \tag{4.13}$$

The poles of the system were found to be:

$$p = -0.117,\ -0.0088,\ -0.0214,\ -0.0194\,, \tag{4.14}$$

The zeros based on u_1 onto y_1 were found to be:

$$z = -0.0088,\ -0.0194,\ -0.0214\,, \tag{4.15}$$

the zeros based on u_1 onto y_2 were found to be:

$$z = -0.0117,\ -0.0214\,, \tag{4.16}$$

the zeros based on u_2 onto y_1 were found to be:

$$z = -0.0088,\ -0.0194\,, \tag{4.17}$$

the zeros based on u_2 onto y_2 were found to be:

$$z = -0.0117,\ -0.0214,\ -0.0194\,. \tag{4.18}$$

It is observed, that all the input to output relations share the same poles, but different zeros that cancel the poles.

3. Compute the continuous-time transfer functions for the continuous-time linearised models.

Seeing as the poles and zeros cancel, the input-output relation in terms of transfer functions can be described as follows:

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \end{bmatrix} = \begin{bmatrix} \dfrac{0.1345}{85.25s+1} & \dfrac{0.056}{3974.56s^2+131.88s+1} \\ \dfrac{0.120}{5875.44s^2+165.63s+1} & \dfrac{0.2253}{114.2s+1} \end{bmatrix} \begin{bmatrix} U_1(s) \\ U_2(s) \end{bmatrix}. \tag{4.19}$$

4. Compare the gains and time constants to the gains and time constants obtained from the step response experiments in Problem 3.

The approximated gains were found to be quite close to the gains based on the linearised system. In the case of the transfer functions in the diagonal, the time constants are quite similar as well.

The off-diagonal transfer functions were found to be second order. However, this was mentioned when the system was approximated.

5. Compute discrete-time state-space models using a sampling time of your choice.

Inspecting the time constants of the linearised deterministic model, it was found that the smallest time constant was 51.5 s. Therefore, it was decided, that the appropriate sampling time for this system was 5 s.

5.1. Discrete-linearised deterministic model

Using the MATLAB function `c2d()`, the system was discretised using the chosen sampling time of 5 s. This resulted in the following system:

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} 0.9430 & 0 & 0.0987 & 0 \\ 0 & 0.9572 & 0 & 0.0906 \\ 0 & 0 & 8.8983 & 0 \\ 0 & 0 & 0 & 0.9074 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 2.9137 & 0.0634 \\ 0.0927 & 3.6691 \\ 0 & 1.1853 \\ 1.9059 & 0 \end{bmatrix} \boldsymbol{u}_k + \begin{bmatrix} 0.2537 & 0 \\ 0 & 2.318 \\ 4.7413 & 0 \\ 0 & 4.7648 \end{bmatrix} \boldsymbol{d}_k \,, \tag{4.20}$$

$$\boldsymbol{y}_k = \boldsymbol{z}_k = \begin{bmatrix} 0.0026 & 0 & 0 & 0 \\ 0 & 0.0026 & 0 & 0 \end{bmatrix} \boldsymbol{x}_k \,. \tag{4.21}$$

5.2. Discrete-linearised stochastic model

The linearised stochastic model was discretised based on the sampling time of 5 s, resulting in the following model:

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} 0.9430 & 0 & 0.0987 & 0 \\ 0 & 0.9572 & 0 & 0.0906 \\ 0 & 0 & 8.8983 & 0 \\ 0 & 0 & 0 & 0.9074 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 2.9137 & 0.0634 \\ 0.0927 & 3.6691 \\ 0 & 1.1853 \\ 1.9059 & 0 \end{bmatrix} \boldsymbol{u}_k + \begin{bmatrix} 0.2537 & 0 \\ 0 & 2.318 \\ 4.7413 & 0 \\ 0 & 4.7648 \end{bmatrix} \boldsymbol{d}_k \,, \tag{4.22}$$

$$\boldsymbol{y}_k = \begin{bmatrix} 0.0026 & 0 & 0 & 0 \\ 0 & 0.0026 & 0 & 0 \end{bmatrix} \boldsymbol{x}_k + \boldsymbol{v}_k \,, \tag{4.23}$$

$$\boldsymbol{z}_k = \begin{bmatrix} 0.0026 & 0 & 0 & 0 \\ 0 & 0.0026 & 0 & 0 \end{bmatrix} \,, \tag{4.24}$$

where $\boldsymbol{d}_k$ is described by the same covariance as in Problem 4.1.2. as it was already expressed as a discrete variable. The measurement noise was described as:

$$\boldsymbol{v_k} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, Rvv\,(p)_k \right) \,. \tag{4.25}$$

5.3. Discrete-linearised stochastic model (SDE)

The SDE was discretised using the Euler-Maruyama discretisation (explicit-explicit) method. Note, that the state vector $\boldsymbol{x}_k$ used here will be based on the one defined in Problem 4.1.3. This gave the

following model, also based on the sampling time of 5 s:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \left(\begin{bmatrix} 0.9430 & 0 & 0.0987 & 0 & 0.2537 & 0 \\ 0 & 0.9572 & 0 & 0.0906 & 0 & 0.2318 \\ 0 & 0 & 8.8983 & 0 & 4.7413 & 0 \\ 0 & 0 & 0 & 0.9074 & 0 & 4.7658 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 2.9137 & 0.0634 \\ 0.0927 & 3.6691 \\ 0 & 1.1853 \\ 1.9059 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \boldsymbol{u}_k \right) \cdot 5 + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \sigma_{F_3} & 0 \\ 0 & \sigma_{F_4} \end{bmatrix} \Delta\omega_k \,, \tag{4.26}$$

$$\boldsymbol{y}_k = \begin{bmatrix} 0.0026 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0026 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x}_k + \boldsymbol{v}_k \,, \tag{4.27}$$

$$\boldsymbol{z}_k = \begin{bmatrix} 0.0026 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0026 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x}_k \,, \tag{4.28}$$

where the measurement noise was the same as in 4.5.2. The Brownian motion has become based on the sampling time. Therefore, it was described by:

$$\Delta\omega_k \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \right) . \tag{4.29}$$

6. Compute the Markov parameters for these discrete-time state-space models and compare them to the Markov parameters obtained from the step response experiments.

Using the state and input matrices, which remained the same through the discrete-linearised models, the Markov parameters could be obtained in the same way as for the approximated system.

A comparison between the Markov parameters is shown in Figure 4.1.

Expectedly, the responses in the diagonals of Figure 4.1, are very close, given that it was correct to assume that these responses were based on a first-order transfer function. The responses in the off diagonals are different as the linearised system has shown, that the responses are of second order. However, after the parameters peak at around the 25^{th} step, the parameters decrease at almost the same rate.

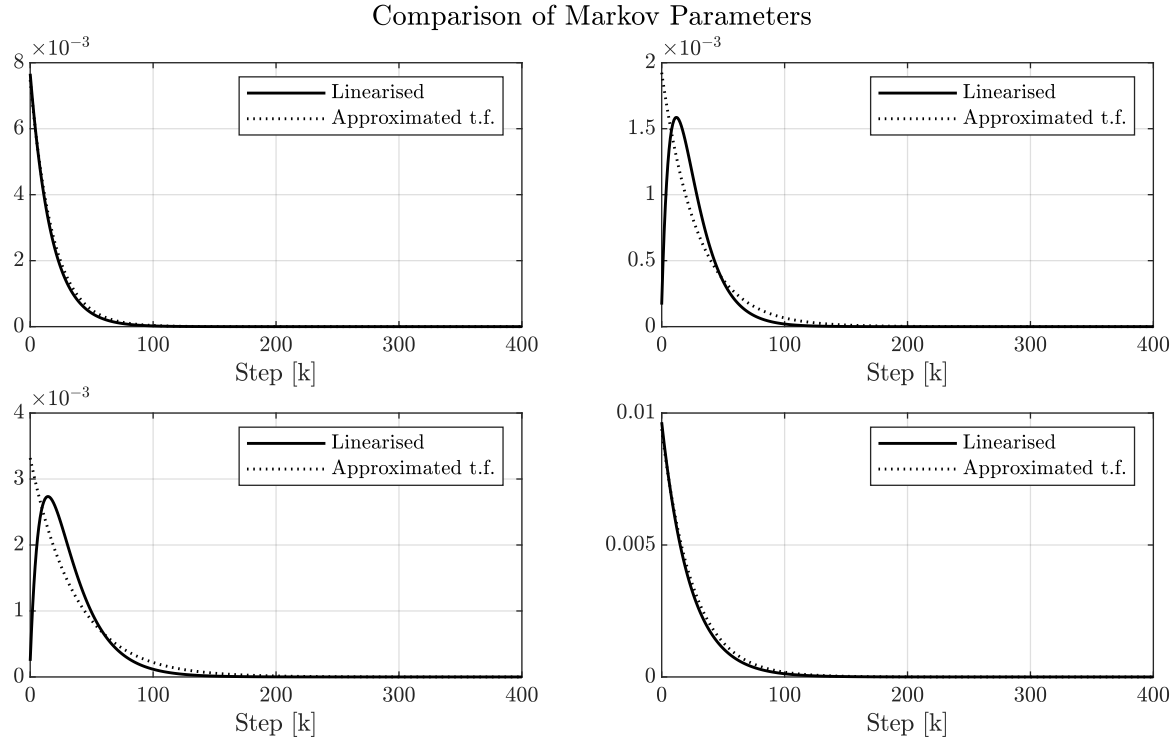

Figure 4.1: Markov parameters were calculated and plotted successively based on the depth. Shown is the discrete-linearised system derived from the nonlinear dynamics, and the coefficients based off of the approximated transfer functions from the step response using the same sampling time.

7. Discuss and comment on the linearisation approach for obtaining discrete-time linear state-space models.

The linearisation approach of a nonlinear system requires the determination of a stationary state of the given system dynamics. Practically, the system dynamics will not always be fully known, therefore, a nonlinear description of a given system would still be an approximation of a system.

Linearising this system, however, makes its implementation for controller calculations much easier. Controllers based on a linearised system will work, to an extent, on its nonlinear description. However, it requires the dynamics of the system to remain near the stationary state to remain accurate.

Problem 5. State Estimation for the Discrete-Time Linear System.

1. Show how the models in Problem 3 and Problem 4 can be represented as linear state-space models in discrete time.

The linear-state-space models based on the approximated system and the linearised system were given in Problem 3.5.& 4.5.

2. Design and evaluate static and dynamic Kalman filters for the linear models identified in Problem 3 and Problem 4. Simulate with unknown disturbances as stochastic variables, but without step changes.

The Kalman filter was set up in a MATLAB function as follows:

```
function [xh,Pk] = statestimator(xh,Pk,A,B,Bv,C,u,y,R,Q)
    Rek = C*Pk*C'+R;
    Kf = Pk*C'*inv(Rek);
    ek = y - C*xh;
    xh = xh + Kf*ek;
    Pk = Pk - Kf*Rek*Kf';

    xh = A*xh+B*u;
    Pk = A*Pk*A'+Bv*Q*Bv';
end
```

it took in the previously estimated states and covariance, the system matrices, input, measurement output, and the disturbance covariances.

Therefore, the relevant matrices of either the approximated or linearised system could be used in the state estimation. However, it can directly be seen, that the approximated system doesn't contain a B_v matrix, making it unable to factor in the disturbance resulting from Q into the estimation.

Additionally, in the case of the static Kalman filter, $P_k = P_\infty$ at each time step. Thus, within the simulation loop, the `statestimator` function could be called as follows:

```
Pk = Pss;
[xh,Pk] = statestimator(xh,Pk,Ad,Bd,Bvd,Cd,u,y,Rk,Qk);
```

This assured, that the precomputed static covariance matrix (calculated via `kalman()`) was used at each step, making the Kalman filter static.

The Kalman filter was implemented for a system with no step changes, i.e. no step changes in d, statically and dynamically for the linearised model in Figures 5.1 & 5.2, respectfully. The following covariance matrices were used for the disturbances in the process and measurement, respectfully:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \quad R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{5.1}$$

In this and following cases, the estimated states were initialised at 50 kg with the covariance $P_0 = 10$. Here, the states were found to approach the real states within around 500 s.

The same evaluation using the approximated system can be found in Appendix D.

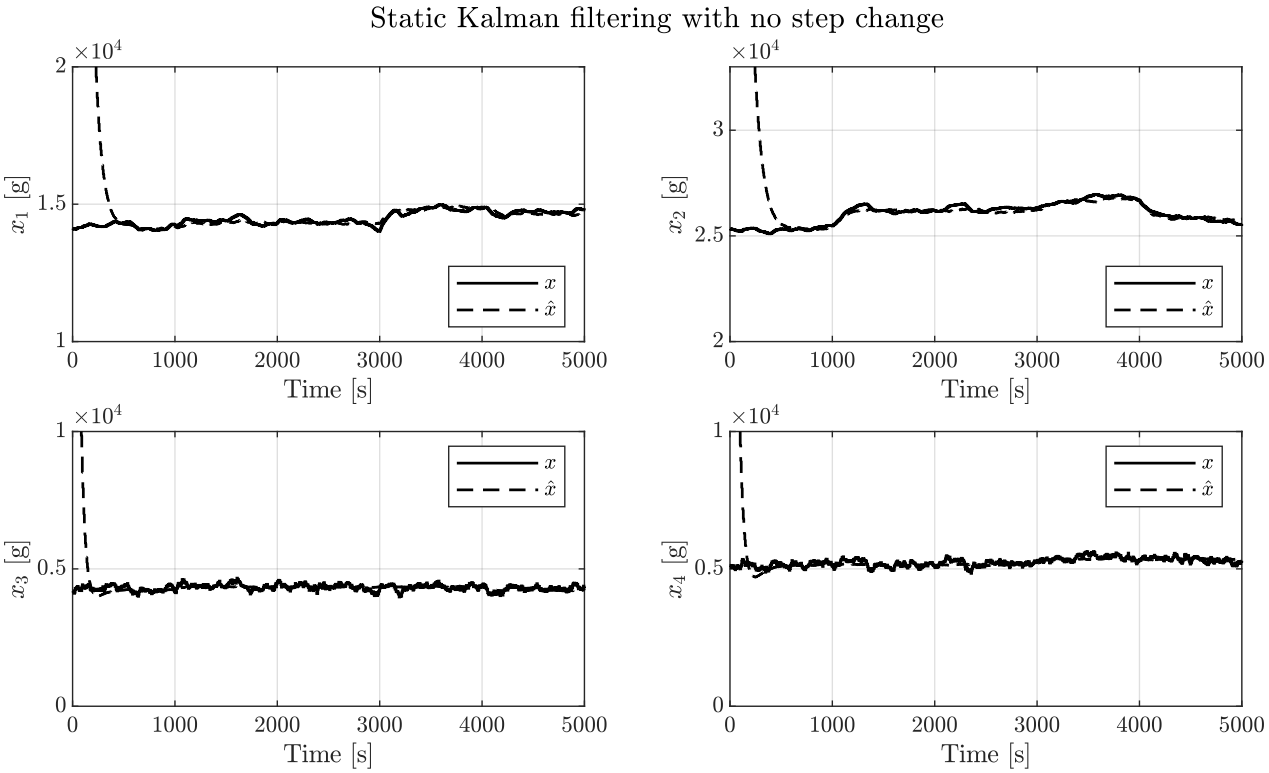

Figure 5.1: Using a static Kalman filter, the estimated states approach the real state within around 500 s despite step changes in u.

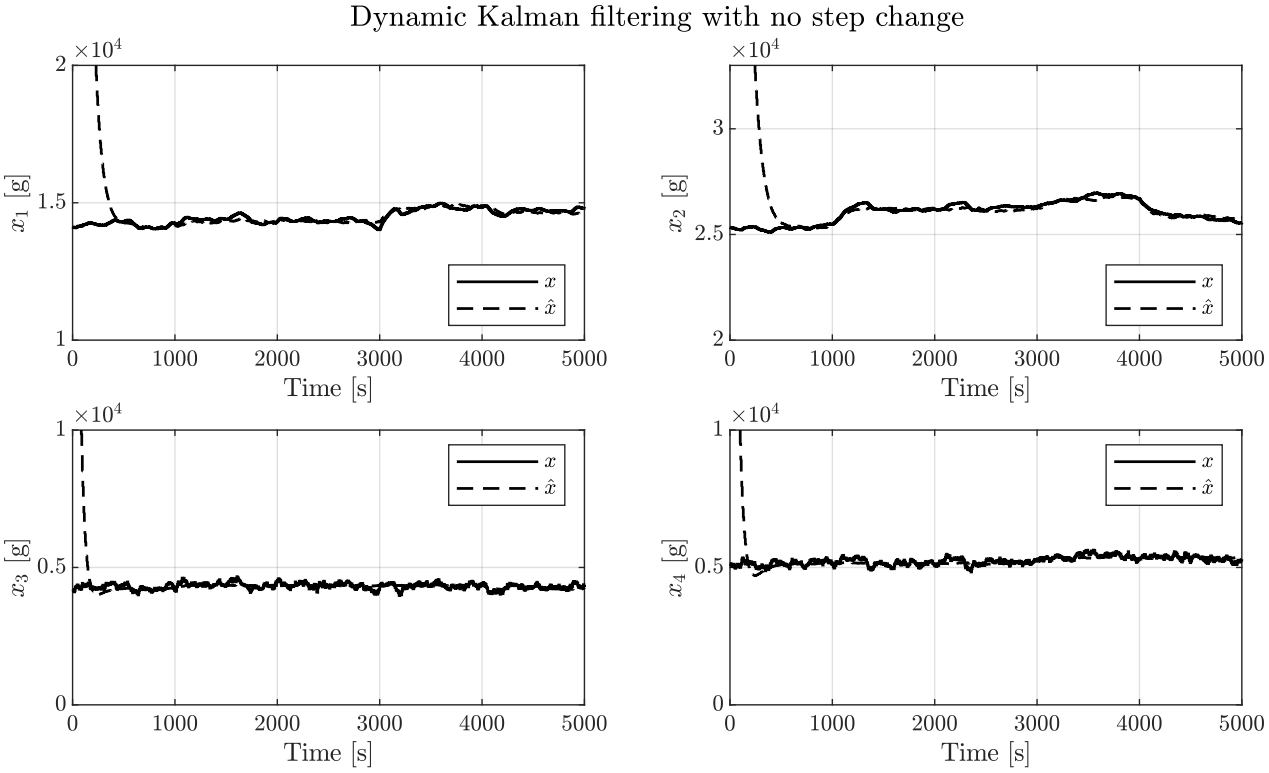

Figure 5.2: Using a dynamic Kalman filter, the estimated states approach the real state within around 500 s despite step changes in u.

3. Design and evaluate static and dynamic Kalman filters for the linear models identified in Problem 3 and Problem 4. Simulate with unknown disturbances as stochastic variables, but with step changes.

The same Kalman filter from Problem 5.2. was implemented with a model experiencing step changes in the input, as well as, in the process noise. The step change in the process noise occurred at 1000 s and had a magnitude of $25\,\mathrm{cm}^3\,\mathrm{s}^{-1}$. The result is shown in Figure 5.3 & 5.4.

Again, the estimated states approach the real states in the same way, until the step change in the process noise occurs. As the implemented Kalman filter doesn't estimate the process noise, it assumes that it remains around the operating point, resulting in a constant error between the estimated and real state.

4. Discuss and evaluate the Kalman filters by simulation on the linear and non-linear models.

Plots of the Kalman filters applied to the linear system were shown. The Kalman filter was also applied to the nonlinear system in simulation. To do this, the input and output of the simulation had to be offset by the stationary state, before inputting this information to the Kalman filter function. The result is shown in Figures 5.5 & 5.6.

It can be seen, that the Kalman filter based on the linearised discrete-time system is able to accurately estimate the states, so long as the disturbance in the system doesn't experience a step change. Additionally, it is clear that the Kalman filter based on the linearised system, has an accuracy limited to a range near the stationary state from which the linearisation stems from.

From these simulations, two additional things were observed:

1. The implemented Kalman filter doesn't estimate the disturbance, resulting in a constant offset if the disturbance experiences a constant step change in mean.
2. The difference in the static and dynamic Kalman filter is only apparent when the Kalman states are initialised at the operating point (see nonlinear system simulations). If the system runs for long enough, the dynamic Kalman filter approaches the same parameters as the static Kalman filter.

An extended-Kalman filter would be able to counteract the step changes in process noise, as it would approximate the disturbance as an additional state in the filter. However, for the purpose of this assignment, the disturbances are going to be kept as disturbances that vary about a constant mean. In this case, the Kalman filter shown for this Problem is sufficient.

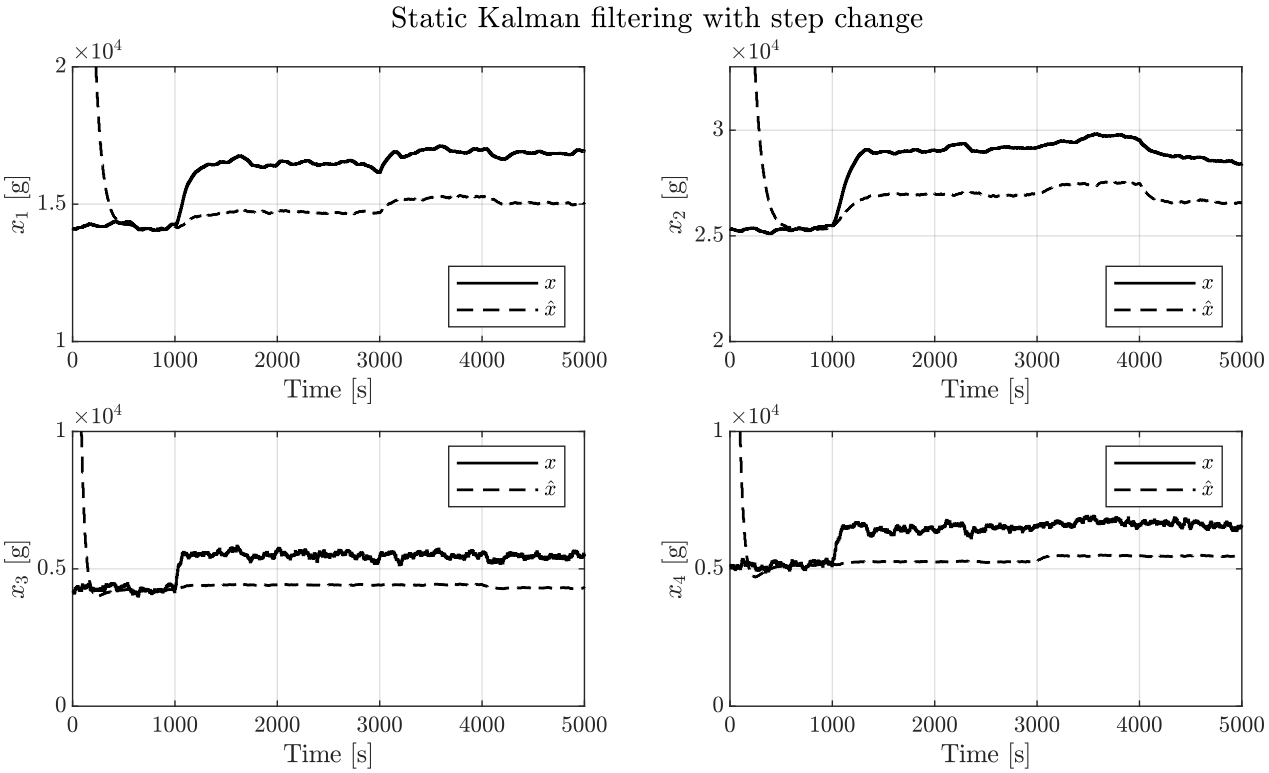

Figure 5.3: Using a static Kalman filter, the estimated states approach the real state within around 500 s (s). However, the step change in the process noise at 1000 s creates a constant error.

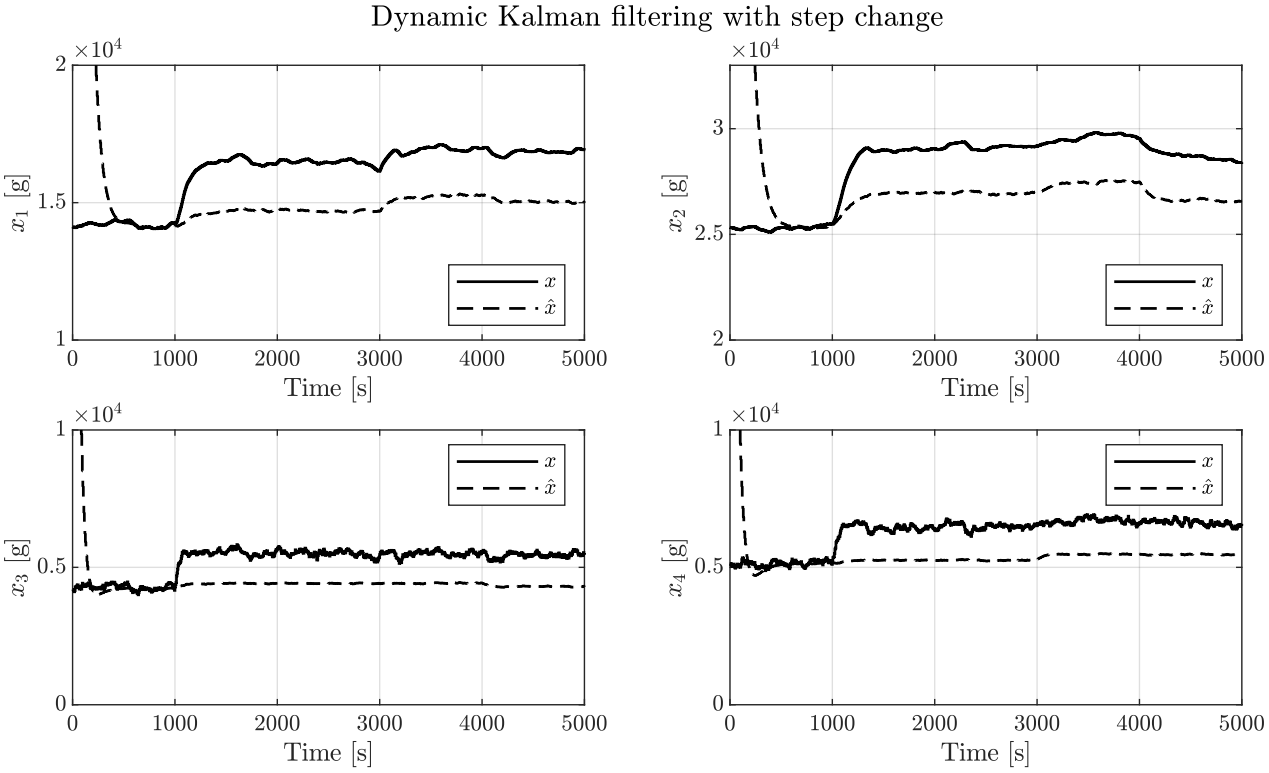

Figure 5.4: Using a dynamic Kalman filter, the estimated states approach the real state within around 500 s and are able to follow the disturbed states. However, the step change in the process noise at 1000 s creates a constant error.

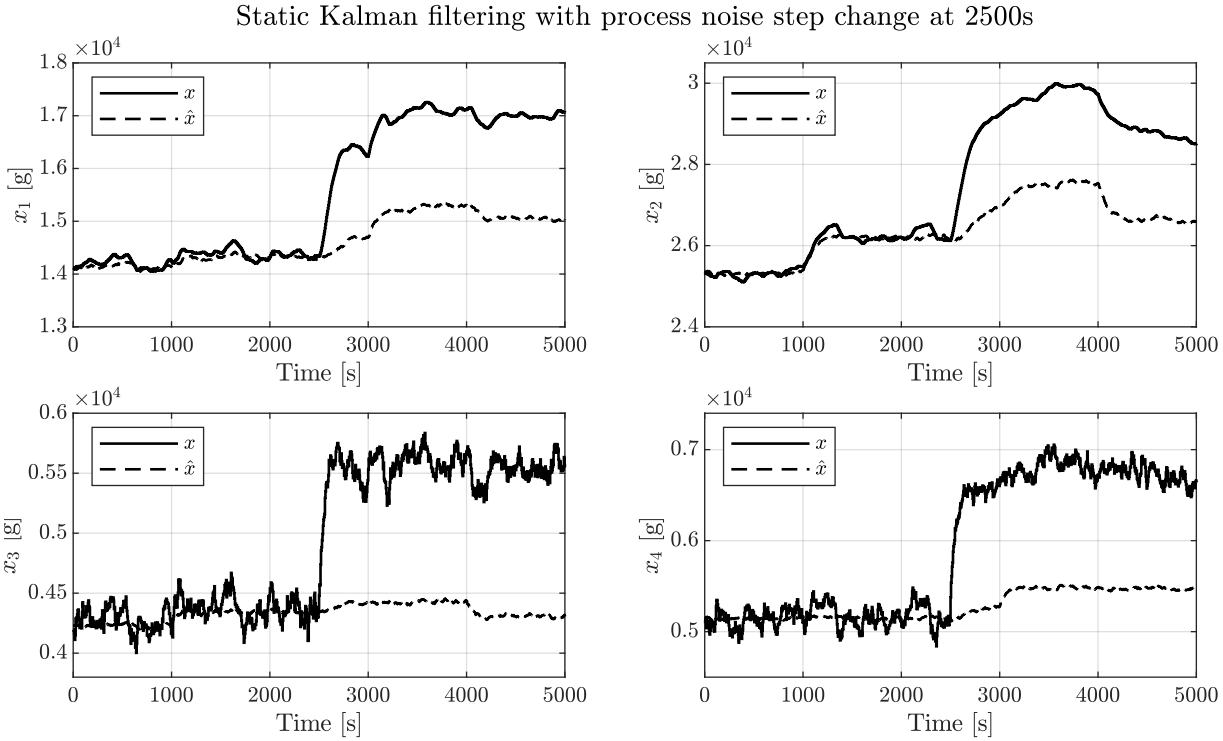

Figure 5.5: State estimation of the nonlinear system using a static Kalman filter based on the linearised discrete-time system. The process noise experiences a step at 2500 s.

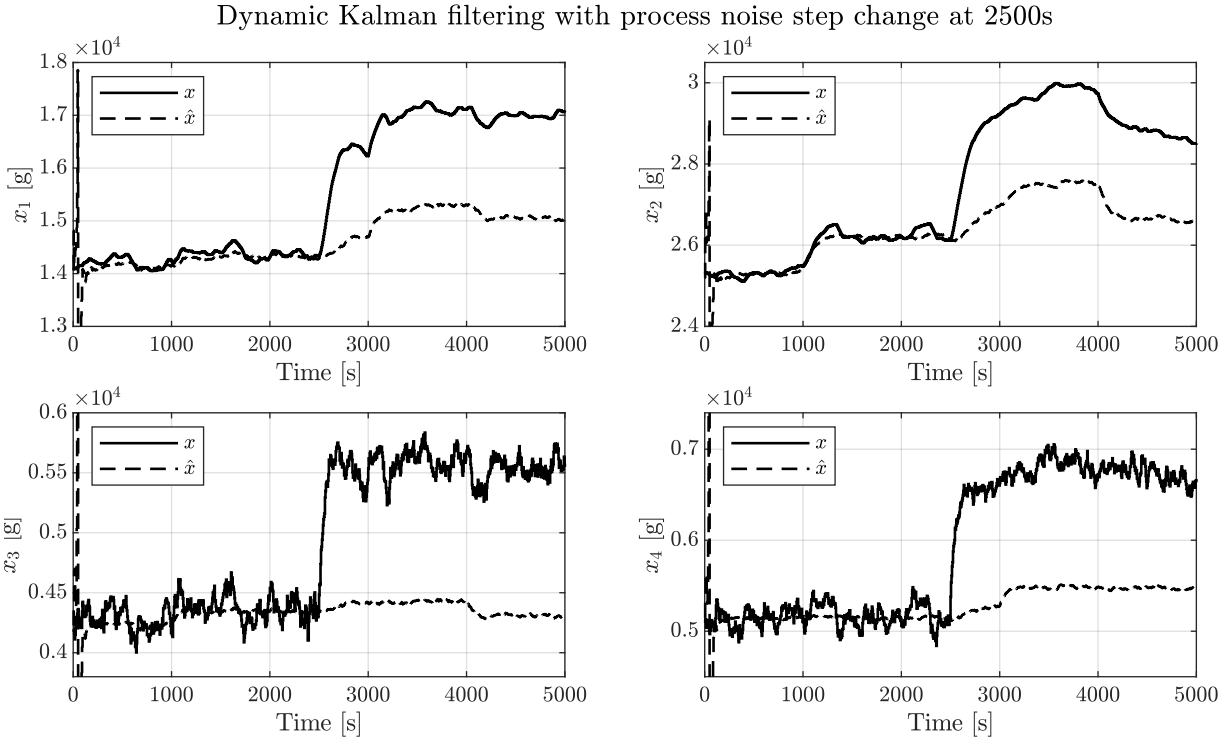

Figure 5.6: State estimation of the nonlinear system using a dynamic Kalman filter based on the linearised discrete-time system. The process noise experiences a step at 2500 s.

Problem 6. QP Solver Interface.

1. Implement a QP solver interface for the solution of the convex quadratic program using the QP solver in MATLAB.

MATLAB contains a function that can be used to solve quadratic objective functions with linear constraints. For the purpose of this assignment, the quadratic objective function to be solved takes on the general form of:

$$\min_x \phi = \frac{1}{2}x^\top Hx + g^\top x \,, \text{ such that } \begin{cases} l \leq x \leq u \\ b_l \leq Ax \leq b_u \end{cases} . \tag{6.1}$$

The in-built function `quadprog`, however, only takes constraints onto Ax in one direction. Therefore, an interface that allows the implementation of constraints as in Equation 6.1 into the in-built `quadprog` was constructed as a function as follows:

```
function [x,info] = qpsolver(H,g,l,u,A,bl,bu,xinit)
    A = [A;-A];
    b = [bu;-bl];
    [x,info] = quadprog(H,g,A,b,[],[],l,u,xinit);
end
```

Problem 7. Unconstrained MPC.

1. Implement a function for the design of an Unconstrained MPC based on the discrete-time state-space models. Explain how the MATLAB functions work and their theoretical background.

For this and the following Problems 8 and 9, the manner in which the following matrices are calculated for the quadratic objective function remains the same. Therefore, how they are obtained will only be shown here. What will change, are the linear constraints that will be specified in Problem 8 and Problem 9.

As shown in Equation 6.1, the main equation that is solved to obtain the control law is based on a quadratic objective function. This is obtained from a weighted-least-squares problem. In the unconstrained case, the equation is formulated as follows:

$$\min_{U} \psi = \frac{1}{2} U^\top H U + g^\top U \,, \tag{7.1}$$

where:

$$H = \boldsymbol{H} + \boldsymbol{H_s} \,, \tag{7.2}$$

$$g = M_{\hat{x}_0}\hat{x}_0 + M_R R + M_{u_{-1}} u_{-1} + M_D D \,. \tag{7.3}$$

The matrices used here are precomputed before the simulation, except for the estimated current state $\hat{x}_0$ and the control input u_{-1} of the previous step.

To obtain an optimal solution, the control law must predict the evolution of the output based on the given state and input. This is made up of the zero-input and zero-state solution for a discrete-time system where:

$$\boldsymbol{Z} = \boldsymbol{\Phi}\hat{x}_0 + \boldsymbol{\Gamma} U + \boldsymbol{\Gamma_d} D \,, \tag{7.4}$$

This equation states the successive outputs based on the estimated initial states and the inputs used in the future. Therefore, the matrices are defined as follows:

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{CA} \\ \boldsymbol{CA}^2 \\ \vdots \\ \boldsymbol{CA}^n \end{bmatrix}, \quad \boldsymbol{\Gamma} = \begin{bmatrix} \boldsymbol{CB} & 0 & 0 & \cdots & 0 \\ \boldsymbol{CAB} & \boldsymbol{CB} & 0 & \cdots & 0 \\ \boldsymbol{CA}^2\boldsymbol{B} & \boldsymbol{CAB} & \boldsymbol{CB} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{CA}^n\boldsymbol{B} & \boldsymbol{CA}^{n-1}\boldsymbol{B} & \boldsymbol{CA}^{n-2}\boldsymbol{B} & \cdots & \boldsymbol{CB} \end{bmatrix},$$

$$\boldsymbol{\Gamma_d} = \begin{bmatrix} \boldsymbol{CB_v} & 0 & 0 & \cdots & 0 \\ \boldsymbol{CAB_v} & \boldsymbol{CB_v} & 0 & \cdots & 0 \\ \boldsymbol{CA}^2\boldsymbol{B_v} & \boldsymbol{CAB_v} & \boldsymbol{CB_v} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{CA}^n\boldsymbol{B_v} & \boldsymbol{CA}^{n-1}\boldsymbol{B_v} & \boldsymbol{CA}^{n-2}\boldsymbol{B_v} & \cdots & \boldsymbol{CB_v} \end{bmatrix}, \tag{7.5}$$

where n expresses the depth of the prediction. It can be seen, that the matrices $\boldsymbol{\Gamma}$ and $\boldsymbol{\Gamma_d}$ are based on the Markov parameters of the discrete-time system.

Using this information, the objective function can be expressed in its discrete form:

$$\min \phi = \frac{1}{2}\sum_{k=0}^{n} ||z_k - r_k||^2_{Q_z} + \frac{1}{2}\sum_{k=0}^{n-1} ||\Delta u_k||^2_S \,. \tag{7.6}$$

The first term of the right-hand side of Equation 7.6 evaluates the output compared to the reference at every step based on the chosen weighting Q_z. The term develops as follows:

$$\phi_z = \frac{1}{2}\sum_{k=0}^{n} ||z_k - r_k||^2_{Q_z}\,, \tag{7.7}$$

$$= \frac{1}{2}||\boldsymbol{Z} - R||^2_{\boldsymbol{Q_z}}\,, \tag{7.8}$$

$$= \frac{1}{2}||\boldsymbol{\Gamma} U - (R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D)\,||^2_{\boldsymbol{Q_z}}\,, \tag{7.9}$$

$$= \frac{1}{2}U^\top \boldsymbol{\Gamma}^\top \boldsymbol{Q_z} \boldsymbol{\Gamma} U - \left[\boldsymbol{\Gamma}^\top \boldsymbol{Q_z}\,(R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D)\right]^\top U + \frac{1}{2}\,(R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D)^\top \boldsymbol{Q_z}\,(R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D)\,, \tag{7.10}$$

$$= \frac{1}{2}U^\top \boldsymbol{H} U + (M_R R + M_{\hat{x}_0}\hat{x}_0 + M_D D)^\top U + \frac{1}{2}\,(R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D)^\top \boldsymbol{Q_z}\,(R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D)\,, \tag{7.11}$$

$$= \frac{1}{2}U^\top H U + g^\top U + \rho\,. \tag{7.12}$$

First, the terms are expressed over the entire depth, then, expressed as a difference between the term U and the rest. It must be noted, that when expressing Q_z over the entire depth, it is expressed as a diagonal matrix of Q_z of size n. The norm is then evaluated, and the equation is simplified into the same form as Equation 7.1.

Next, the second term of the right-hand side of Equation 7.6 evaluates the change in input at every step based on the chosen weighting S. This term develops as follows:

$$\phi_{\Delta u} = \frac{1}{2}\sum_{k=0}^{n-1} ||\Delta u_k||^2_S\,, \tag{7.13}$$

$$= \frac{1}{2}U^\top \begin{bmatrix} 2S & -S & 0 & \cdots & 0 \\ -S & 2S & -S & \ddots & \vdots \\ 0 & -S & 2S & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -S \\ 0 & \cdots & 0 & -S & S \end{bmatrix} U + \left(- \begin{bmatrix} S \\ 0 \\ \\ \vdots \\ \\ 0 \end{bmatrix} u_{-1} \right)^\top U + \frac{1}{2} u_{-1} S u_{-1}\,, \tag{7.14}$$

$$= \frac{1}{2}U^\top \boldsymbol{H_s} U + M_{u_{-1}}^\top U + \frac{1}{2} u_{-1} S u_{-1}\,, \tag{7.15}$$

$$= \frac{1}{2}U^\top H U + g^\top U + \rho\,. \tag{7.16}$$

The terms from ϕ_z and $\phi_{\Delta u}$ making up H and g can then be collected as follows:

$$H = \boldsymbol{\Gamma}^\top \boldsymbol{Q_z} \boldsymbol{\Gamma} + \boldsymbol{H_s}\,, \tag{7.17}$$

$$= \boldsymbol{H} + \boldsymbol{H_s}\,, \tag{7.18}$$

$$g = \left(-\boldsymbol{\Gamma}^\top \boldsymbol{Q_z}\right) R + \left(\boldsymbol{\Gamma}^\top \boldsymbol{Q_z} \boldsymbol{\Phi}\right) \hat{x}_0 + \left(\boldsymbol{\Gamma}^\top \boldsymbol{Q_z} \boldsymbol{\Gamma_d}\right) D + M_{u_{-1}} u_{-1}\,, \tag{7.19}$$

$$= M_R R + M_{\hat{x}} \hat{x}_0 + M_D D + M_{u_{-1}} u_{-1}\,, \tag{7.20}$$

where the terms for ρ are unimportant. The terms can then be input into Equation 7.1.

A MATLAB function was developed in order to generate these matrices that can be precomputed before the implementation into the simulation. To do this, it had to take in the system matrices, the set weights, and the desired prediction depth.

To generate $\boldsymbol{\Phi}$, the size of the matrix is first determined to be based on the depth times the number of rows in $\boldsymbol{C}$, by the number of columns in $\boldsymbol{A}$. Then, it was noticed, that the successive terms in $\boldsymbol{\Phi}$

just post-multiply $\boldsymbol{A}$ in each next step. Thus, by initialising with $\boldsymbol{C}$, consecutive entries just had to multiply the previous entry with $\boldsymbol{A}$. The resulting function is shown below.

```
Phi = zeros(n*size(C,1),size(A,2));
Phi_i = C;
for i = 1:size(C,1):n*size(C,1)
    Phi(i:i+size(C,1)-1,:) = Phi_i*A;
    Phi_i = Phi(i:i+size(C,1)-1,:);
end
```

To generate $\boldsymbol{\Gamma}$, first a vector of all the consecutive Markov parameters was generated like in Problem 3.6. Then, the size of $\boldsymbol{\Gamma}$ was determined to be a square matrix with the same size as the vector. Within a for loop, the Markov parameter vector was inserted into the columns, padding with zeros from the top and only selecting as many Markov parameters as necessary for the given column, in order to create the lower triangular matrix as the loop progressed through the $\boldsymbol{\Gamma}$ matrix from left to right. The $\boldsymbol{\Gamma_d}$ matrix could be created in the same manner by substituting $\boldsymbol{B}$ for $\boldsymbol{B_v}$. The MATLAB function used is shown below.

```
Gamma_vec = markovvec(A,B,C,n);
Gamma = zeros(size(Gamma_vec,1));
% Bulid Gamma Matrix
k = 0;
for i = 1:size(C,1):length(Gamma)
    Z = zeros(k*size(C,1),size(C,1));
    Gamma(:,i:i+size(C,1)-1) = [Z;Gamma_vec(1:end-size(Z,1),:)];
    k = k + 1;
end
```

As mentioned, the $\boldsymbol{Q_z}$ matrix is simply a diagonal matrix of the decided weighting. This was achieved by knowing the size of the matrix and appropriately padding the matrix with zeros as the for loop progressed through the matrix from left to right. The result is shown below.

```
Qz = zeros(length(Gamma));
k = 0;
for i = 1:length(Q):length(Gamma)
    Z_b = zeros(k*length(Q),length(Q));
    Z_a = zeros(length(Gamma)-size(Z_b,1)-length(Q),length(Q));
    Qz(:,i:i+length(Q)-1) = [Z_b; Q; Z_a];
    k = k + 1;
end
```

Next, the $\boldsymbol{H_s}$ matrix was developed by knowing its size is the same as the $\boldsymbol{\Gamma}$ matrix and knowing its pattern. It was noticed that other than the first and last column, the pattern is consistent. Therefore, the first and last columns were constructed separately from the columns in between them. The columns in between were constructed in the same manner as $\boldsymbol{Q_z}$, the only difference being the size of the non-zero matrix being implemented. The result is shown below.

```
Hs = zeros(length(Gamma));
k = 0;
for i = 1:length(S):length(Gamma)
    if i == 1
        Hs_i = [2*S;-S];
        Hs(:,i:i+length(S)-1) = [Hs_i;zeros(length(Gamma)-size(Hs_i,1)
            ,length(S))];
    else
```

```
        if i == length(Gamma)-1
            Hs_i = [-S;S];
            Hs(:,i:i+length(S)-1) = [zeros(length(Gamma)-size(Hs_i,1),
               length(S));Hs_i];
        else
            Hs_i = [-S;2*S;-S];
            Z_b = zeros(k*length(S),length(S));
            Z_a = zeros(length(Gamma)-size(Z_b,1)-size(Hs_i,1),length(
               S));
            Hs(:,i:i+length(S)-1) = [Z_b;Hs_i;Z_a];
            k = k + 1;
        end
    end
end
```

Finally, $M_{u_{-1}}$ was constructed by knowing it is a vector with a length equal to the dimension of Γ. Thus, the weighting S was simply placed as the first element, and the rest were zeros. The function used is shown below.

```
Mu_1 = -[S;zeros(length(Gamma)-length(S),length(S))];
```

With these matrices precomputed, the unconstrained MPC can be implemented within a simulation with the modified-four-tank system.

2. Design an unconstrained MPC for the models identified in Problem 3 and Problem 4.

To obtain the unconstrained MPC using the models from Problem 3 and 4, the matrices outlined in Problem 7.1. were constructed using the matrices of the discrete-time systems (see Equations 3.7 & 4.21), a chosen depth of 150 s, and the weighting matrices chosen as follows:

$$Q_z = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{7.21}$$

In the MATLAB program, the calculation of the matrices were aggregated in a single function as follows:

```
[H,Mx0,Mr,Mu_1,Md,Aqp,Phi,Gamma_d] = MPCmat(Ad,Bd,Cd,Bvd,Q,S,n);
```

where `H` corresponds to Equation 7.18, and the matrices `Mx0, Mr, Mu_1, Md` correspond to Equation 7.20. The matrix `Aqp` was not required for the unconstrained MPC.

3. Implement and discuss a compute and prediction function for this MPC.

Based on the computed matrices, the unconstrained MPC could be implemented in the simulation loop as follows:

```
% Select reference for current step k
i_ref = 2*k+1;
R = ref(i_ref:(i_ref+2*n-1));
% Control law for current step k
gu = Mx0*xh+Mr*R+Mu_1*u+Md*d;
[u,info] = qpsolver(H,gu,[],[],[],[],[],[]);
u = [u(1);u(2)];
```

The control law is calculated based on the computed matrices, the estimated state, the reference being selected from the predetermined path (defined before the simulation) based on the selected depth, the previous step's input, and the assumed mean disturbance value. Then, the control input is selected for the respective inputs as the first two elements of the `qpsolver` output.

To verify that the unconstrained MPC worked as intended, it was applied to the deterministic case where there was no noise, $\hat{x} = x$, and the following weighting matrices:

$$Q_z = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{7.22}$$

where Q_z defines that the reference tracking is equally important for either output and S defines that both inputs may be changing equally as much between steps. The result of this is shown in Figure 7.1.

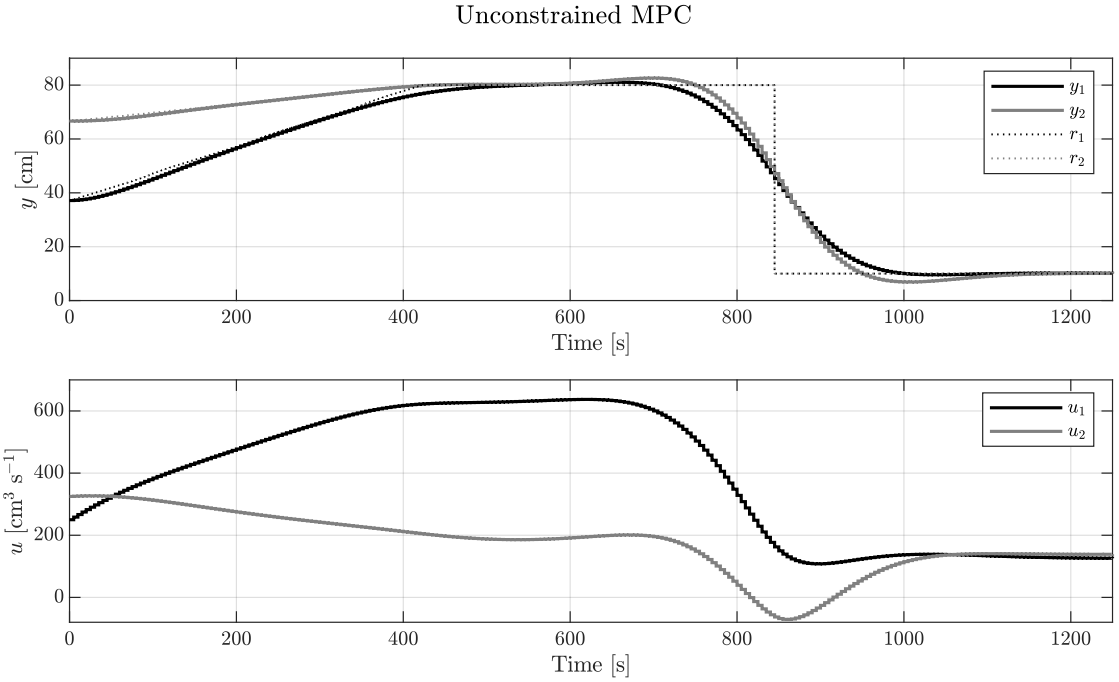

Figure 7.1: Unconstrained MPC applied to the linearised deterministic system.

It can be seen, that the MPC chooses inputs to follow the reference as close as possible. This means that it changed over steps as much as required (largest change of around 20), and the maximum/minimum amount of input and output, were unconstrained. In Problems 8 and 9, the same model will be controlled with constraints to see how the responses differ.

Problem 8. Input Constrained MPC.

1. Implement a function for the design of an input constrained MPC based on the discrete-time state-space models. Explain how the MATLAB functions work and their theoretical background.

As stated previously, the matrices in the quadratic objective function are used in the same way for this case as well. However, to create an input-constrained MPC, linear constraints are used to limit the scope of the input. Practically, this is important because based on the physical limitations of a real system, the input mechanisms aren't always fully capable of producing the desired input. Therefore, the limitations of the system can be acknowledged and implemented within the MPC calculation allowing for a more optimal solution within the limits of the real system.

In the constrained case, the objective function is formulated as follows:

$$\min_{U} \psi = \frac{1}{2}U^\top HU + g^\top U \text{ , such that } \begin{cases} U_{\min} \leq U \leq U_{\max} \\ \Delta U_{\min} \leq \Lambda U \leq \Delta U_{\max} \end{cases} , \tag{8.1}$$

where the calculation of H and g remain the same as in Problem 7.

The constraint on the minimum and maximum output is formulated as a vector defining the minimum and maximum possible input at every step in the prediction. Therefore, it is defined as follows:

$$\begin{bmatrix} u_{\min} \\ \vdots \\ u_{\min} \end{bmatrix} \leq \begin{bmatrix} u_0 \\ \vdots \\ u_{n-1} \end{bmatrix} \leq \begin{bmatrix} u_{\max} \\ \vdots \\ u_{\max} \end{bmatrix} . \tag{8.2}$$

For the purpose of this assignment, minimum and maximum outputs remained the same over the entire simulation. Therefore, the matrices were simply a vector containing a repeating constant with a length relative to the desired depth. This was implemented as follows:

```
U_min = repmat(u_min,n,1);
U_max = repmat(u_max,n,1);
```

Evidently, there are cases where the bounds on the input would have to change over time. For example, when a noise intensive system would have to run overnight, but must remain under a certain noise level within those hours, $u_{\max}$ would then change accordingly (if the input contributes to the noise). In the modified-four-tank system, the lower bound would have to be capped at 0, because it would be unreasonable to assume that the pumps would be able to suck water out of the tanks.

Next, the constraint on the change in input at every step is defined as follows:

$$\begin{bmatrix} \Delta u_{\min} + u_{-1} \\ \Delta u_{\min} \\ \vdots \\ \Delta u_{\min} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{I} & 0 & \cdots & 0 \\ -\boldsymbol{I} & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -\boldsymbol{I} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{bmatrix} \leq \begin{bmatrix} \Delta u_{\max} + u_{-1} \\ \Delta u_{\max} \\ \vdots \\ \Delta u_{\max} \end{bmatrix} . \tag{8.3}$$

The matrix $\boldsymbol{\Lambda}$ (premultiplying the predicted input matrix) is used to map consecutive predicted inputs against each other such that they don't exceed the set constraints. The first entries of the $\boldsymbol{\Delta}$ matrices contain the previous input, as the prediction starts with the state $\hat{x}_0$, thus this is accounted for in the matrices to keep the set differences.

Again, for the purpose of this assignment, the minimum and maximum change in input between a steps remained the same over the entire simulation. Therefore, the matrices could be constructed similar to the $U_{\min}$ and $U_{\max}$ vectors. However, their calculations had to be performed within the loop as they required an updating input for their first entry. In MATLAB, the constraints were called in a function within the loop as follows:

```
function DU = MPCdu(r,n,u)

DU = zeros(n*size(r,1),1);
for i = 1:size(r,1):size(DU,1)
    DU(i:i+size(r,1)-1,1) = [r(1);r(2)];
end

DU(1:size(r,1)) = DU(1:size(r,1))+u;

end
```

The function takes in the desired $\Delta u_{\min}$ or $\Delta u_{\max}$ as `r`, then, based on the desired depth and previous step input, created the matrices.

The $\mathbf{\Lambda}$ matrix can be constructed before the simulation. The function used, took a matrix with the same dimensions as $\mathbf{\Gamma}$ and placed the identity matrices by padding the zeros accordingly above and below, as the for loop progressed from left to right of the matrix. Additionally, it is known, that the dimensions of the identity matrices would be equal to the number of outputs, which was equal to the size of the weighting matrix Q_z. When the loop reached the last column, it placed the last identity matrix at the very bottom. This was implemented in MATLAB as follows:

```
Lam = zeros(length(Gamma));
k = 0;
for i = 1:length(Q):length(Gamma)
    if i == length(Gamma)-length(Q)+1
        Lam(:,i:i+length(Q)-1) = [zeros(length(Gamma)-length(Q),length
            (Q));eye(length(Q))];
    else
        Lam_i = [eye(length(Q)); -eye(length(Q))];
        Z_b = zeros(k*length(Q),length(Q));
        Z_a = zeros(length(Gamma)-size(Z_b,1)-size(Lam_i,1),length(Q))
            ;
        Lam(:,i:i+length(Q)-1) = [Z_b; Lam_i ; Z_a];
        k = k + 1;
    end
end
```

Knowing how to build these matrices, the MPC can now be applied with input constraints based on user-defined parameters.

2. Design an input constrained MPC for the models identified in Problem 3 and Problem 4.

To obtain the input-constrained MPC using the models from Problem 3 and 4, the matrices as constructed in Problem 7 were constructed, in addition to the matrices introduced in 8.1. The same depth, and weighting matrices were used as in Problem 7.2.

The same MATLAB function was used as follows:

```
[H,Mx0,Mr,Mu_1,Md,Aqp,Phi,Gamma_d] = MPCmat(Ad,Bd,Cd,Bvd,Q,S,n);
```

where this time, `Aqp` was used. Within the program, `Aqp` outputs the following matrix:

$$A_{qp} = \begin{bmatrix} \mathbf{\Lambda} \\ \mathbf{\Gamma} \end{bmatrix}, \tag{8.4}$$

this matrix is useful for Problem 9. However, for the input-constrained MPC, only $\mathbf{\Lambda}$ was required (as outlined in Problem 8.1.).

Additionally, the matrices for the constraints were constructed. For the absolute constraints, the upper and lower limits were set for either input, and if the system is based on the deviation variable, the stationary input used for linearisation had to be subtracted. Then, the matrix was repeated according to the chosen depth `n`. For the maximum rate of change in the input over steps, the function as shown in Problem 8.1. was implemented within the simulation loop.

3. Implement and discuss a compute and prediction function for this MPC.

Based on the computed matrices, the input-constrained MPC could be implemented in the simulation loop as follows:

```
% Control Law
        % Delta U
D_U_min = MPCdu(d_u_min,n,u);
D_U_max = MPCdu(d_u_max,n,u);
        % solving QP
gu = Mx0*xh+Mr*R+Mu_1*u+Md*d;
[u,info] = qpsolver(H,gu,U_min,U_max,Lam,D_U_min,D_U_max,[]);
u = [u(1);u(2)];
```

First, the $\boldsymbol{\Delta U}$ constraints have to be updated based on the previous step's input and the selected constraint defined outside the loop, then the `qpsolver` can be called with the constraints (refer to Problem 6). Lastly, the input is selected for the next step.

To verify that the input-constraints worked as intended, it was applied on the deterministic case with no noise and $\hat{x} = x$. Based on the response shown in Problem 7.3. the maximum possible input was set to $600\,\mathrm{cm^3\,s^{-1}}$, the minimum at $0\,\mathrm{cm^3\,s^{-1}}$ as it is assumed the pumps can't suck water out of the system, and a maximum change of $\pm 10\,\mathrm{cm^3\,s^{-1}}$ across steps. The result is shown in Figure 8.1.

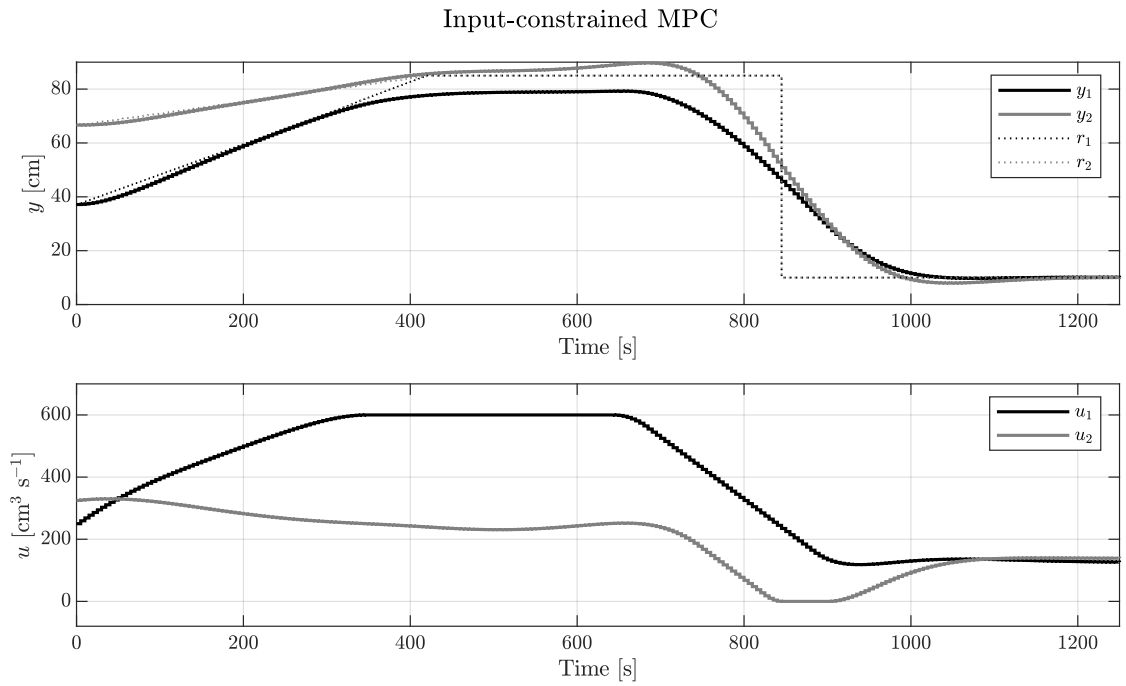

Figure 8.1: Input-constrained MPC applied to the linearised deterministic system.

It can be seen, that the inputs are clearly staying within 0 and $600\,\mathrm{cm^3\,s^{-1}}$. Additionally, the

maximum change over steps was kept at $10\,\mathrm{cm}^3\,\mathrm{s}^{-1}$. The introduction of the input constraints made it impossible for the first tank height to reach the set reference at 80 cm like in Figure 7.1. However, it can be seen, that input 2 is still far from saturation. Therefore, if it were critical for tank 1 to reach this reference, the weighting matrix Q could be altered such that reference tracking for tank 2 is somewhat sacrificed in favour for tank 1.

To show this, the weighting matrix onto the reference tracking was changed to:

$$Q = \begin{bmatrix} 50 & 0 \\ 0 & 1 \end{bmatrix}, \tag{8.5}$$

resulting in the response shown in Figure 8.2.

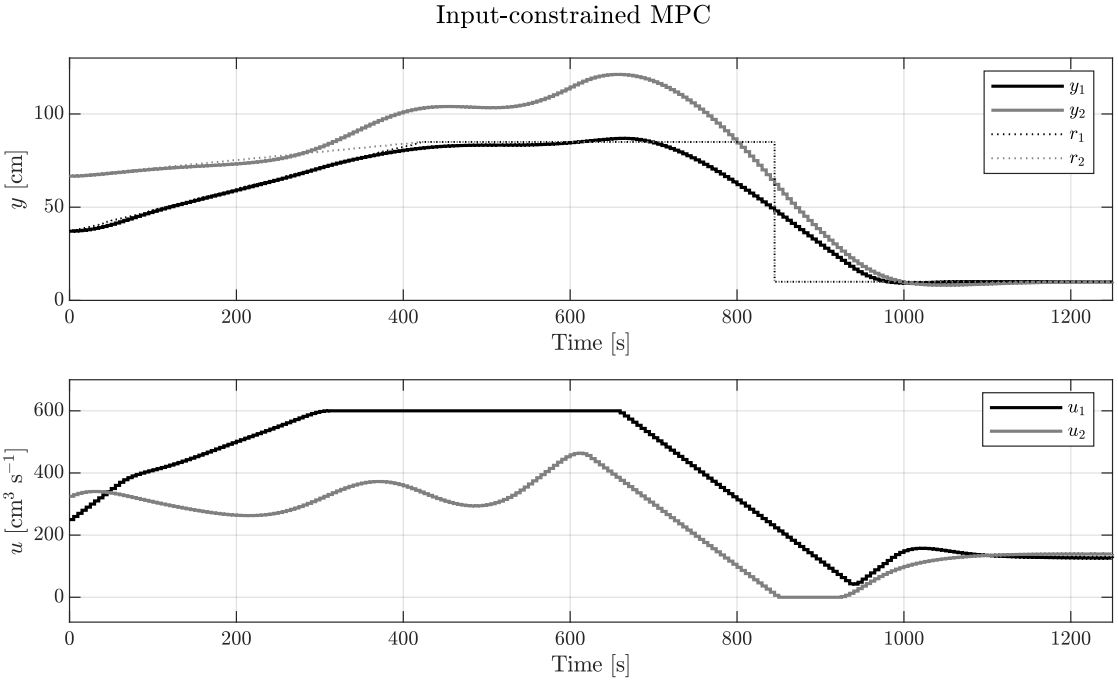

Figure 8.2: Input-constrained MPC applied to the linearised deterministic system. Reference tracking for tank 1 was selected to be 50 times more important than in tank 1.

As it can be seen, the output of tank 1 is now able to follow the reference at the height of 80 cm despite its direct input reaching saturation, due to its indirect input (u_2) compensating. However, this resulted in the height of tank 2 to exceed the set reference by a lot.

Problem 9. MPC with Input and Soft-Output Constraints.

1. Implement a function for the design of an MPC with input constraints and soft output constraints based on the discrete-time state-space models. Explain how the MATLAB functions work and their theoretical background.

To include soft-output constraints, the quadratic objective function used in Problems 8 & 9 (Equation 7.6) has to be altered to include a new term that introduces a separate penalty. This penalty based on w_k, would introduce a penalty based on being too close to the set upper and lower output constraints. The objective function becomes:

$$\min_{\{u_k, w_{k+1}\}_{k=0}^{n-1}} \phi = \frac{1}{2}\sum_{k=0}^{n}\left(||z_k - r_k||_{Q_z}^2 + ||w_k||_{S_e}^2\right) + \frac{1}{2}\sum_{k=0}^{n-1}||\Delta u_k||_S^2\,,$$

$$\text{such that } \begin{cases} U_{\min} \leq U \leq U_{\max} \\ \Delta U_{\min} \leq \Lambda U \leq \Delta U_{\max} \\ z_k + w_k \geq z_{\min} \\ z_k - w_k \geq z_{\max} \\ w_k \geq 0 \end{cases}. \tag{9.1}$$

The new term develops in the following way:

$$\phi_w = \frac{1}{2}\sum_{k=0}^{n}||w_k||_{Se}^2\,, \tag{9.2}$$

$$= \frac{1}{2}W^\top \boldsymbol{H_{ww}} W\,, \tag{9.3}$$

$$\boldsymbol{H_{ww}} = \begin{bmatrix} S_e & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & S_e \end{bmatrix}. \tag{9.4}$$

Combining this with the findings from Problems 7 & 8, the objective function can be formulated as follows:

$$\min_{U,W} \phi = \frac{1}{2}\begin{bmatrix} U \\ W \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{H} + \boldsymbol{H_s} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{H_{ww}} \end{bmatrix} \begin{bmatrix} U \\ W \end{bmatrix} + \begin{bmatrix} g \\ \boldsymbol{0} \end{bmatrix}^\top \begin{bmatrix} U \\ W \end{bmatrix},$$

$$\text{such that } \begin{cases} \begin{bmatrix} U_{\min} \\ 0 \end{bmatrix} \leq \begin{bmatrix} U \\ W \end{bmatrix} \leq \begin{bmatrix} U_{\max} \\ \infty \end{bmatrix} \\ \begin{bmatrix} \Delta U_{\min} \\ Z_{\min} - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D \\ -\infty \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{\Lambda} & \boldsymbol{0} \\ \boldsymbol{\Gamma} & \boldsymbol{I} \\ \boldsymbol{\Gamma} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} U \\ W \end{bmatrix} \leq \begin{bmatrix} \Delta U_{\max} \\ \infty \\ Z_{\max} - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D \end{bmatrix} \end{cases}, \tag{9.5}$$

where the maximum and minimum Z are simply vectors with the respective outputs built the same way as the $U_{\min}$ and $U_{\max}$ vectors from Problem 8.

Introducing the new term adds a penalty onto the output coming too close to the set output constraint. It achieves this without setting a hard output constraint which, at times, may create an infeasibility in the evaluation of the objective function. This is important when working with stochastic systems that evolve, to an extent, unpredictably.

Knowing the setup of this MPC, the matrices can be input to the QP solver and the system can be controlled optimally based on user-defined input and soft-output constraints.

2. Design an input constrained MPC with soft output constraints for the models identified in Problem 3 and Problem 4.

To obtain the input and soft-output-constrained MPC using the models from Problem 3 and 4, the matrices as constructed in Problem 7 were constructed, in addition to those introduced in Problem 8, and the ones introduced in Problem 9.1. The same depth, and the following weighting matrices were used:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad S_e = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \tag{9.6}$$

the high values in S_e were chosen to ensure a high enough penalty for exceeding the set output constraint, in comparison to the weighting matrix Q.

The same MATLAB function was used as follows:

```
[H,Mx0,Mr,Mu_1,Md,Aqp,Phi,Gamma_d] = MPCmat(Ad,Bd,Cd,Bvd,Q,S,n);
```

where this time, different from Problem 7.2., the entire vector `Aqp` was used.

Additionally, on top of the matrices based on the input constraints (as constructed in Problem 7.2.), the matrices from Problem 9.1. have to be constructed. First, the matrix premultiplying U and W were constructed based on `Aqp` as follows:

```
Aqp_int = [zeros(2*n);eye(2*n)];
Aqp = [Aqp Aqp_int; Aqp(2*n+1:end,:) -eye(2*n)];
```

The matrices giving the output constraints are built in the simulation loop and are shown in the following section.

3. Implement and discuss a compute and prediction function for this MPC.

Based on the computed matrices, the input and soft-output-constrained MPC could be implemented in the simulation loop as follows:

```
% Control Law
        % Delta U
D_U_min = MPCdu(d_u_min,n,u);
D_U_max = MPCdu(d_u_max,n,u);
        % Zbar
c = Phi*xh+Gamma_d*D;
Zbarmin = Zmin - c;
Zbarmax = Zmax - c;
        % solving QP
gu = Mx0*xh+Mr*R+Mu_1*u+Md*d;
g = [gu;zeros(length(gu),1)];
[u,info] = qpsolver(H,g,U_min,U_max,Aqp,[D_U_min;Zbarmin;-inf*ones(2*n
   ,1)],[D_U_max;inf*ones(2*n,1);Zbarmax],[]);
u = [u(1);u(2)];
```

Here, the $\bar{Z}$ has to be computed based on the estimated state and disturbance which has to be given as a vector with a size proportional to the depth. Then, the g vector is constructed with g_u, and a vector with zeros as derived in Problem 9.1. This was then plugged into the `qpsolver` to express what is shown in Equation 9.5. The first two results are then selected as the inputs to the system.

To verify that the soft-output constraints worked as intended, they were applied on the deterministic case with no noise where, $\hat{x} = x$. The response shown in Figure 8.1 showed a slight overshoot before and after the step down.

Therefore, to avoid this, the constraints were chosen such that: $9.5\,\text{cm} \leq y \leq 80.5\,\text{cm}$. The input constraints were chosen to be the same as in Problem 8.3. The result is shown in Figure 9.1.

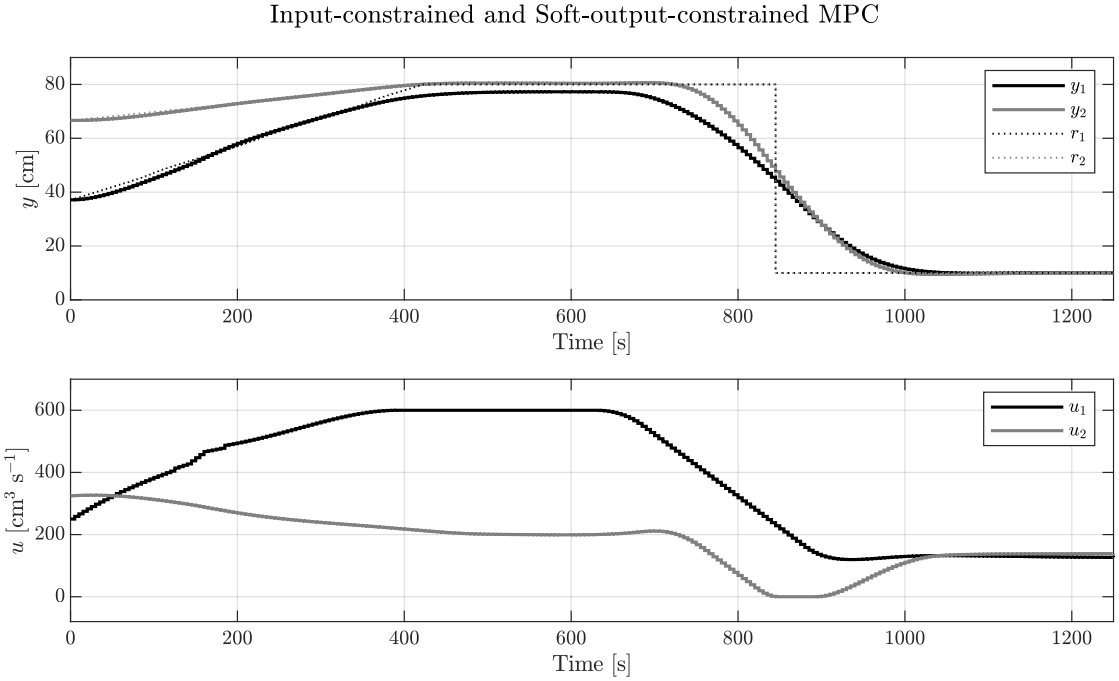

Figure 9.1: Input and soft-output-constrained MPC applied to the linearised deterministic system.

In comparison to Figure 8.1, the output doesn't overshoot the reference before and after the step down much any more. Additionally, the constraint could be verified as soft, as the output had exceeded the set limits only marginally by $\leq 0.17\,\text{cm}$. To achieve a harder output constraint, the weighting matrix on the penalty S_e would have to be increased even more. However, increasing this weighting also brings the calculation of the control input closer to infeasibility, as the output constraint becomes harder, and making the computation time comparatively longer.

Problem 10. Closed-loop Simulations.

1. Do closed-loop simulations of the MPCs for both the linear and nonlinear models. Discuss the results.

Closed-loop simulations of the MPCs on the linearised deterministic model were shown in Problem 7–9 to verify the implementation. For this system, refer to the figures presented there.

For this Problem, the simulations were carried out using the following weighting matrices:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad S_e = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \tag{10.1}$$

where required. Additionally, the depth, the sampling time, the input and soft-output constraints were all kept the same. This is to ensure that the systems remained consistent for comparison.

The covariance matrices for the process and measurement noise were chosen based on the following parameters:

$$\sigma_{F_3} = \sigma_{F_4} = 100\,, \quad R_{vv}\,(p) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{10.2}$$

(except for the discrete-linearised SDE, clarification is provided in Problem 10.1.3.) it is assumed that the measurement is always sampled based on the chosen sampling time for every model.

1.1. Approximated model

For the approximated model, the simulation was treated as deterministic. Therefore, there was no noise such that $\hat{x} = x$.

The following figures show the unconstrained, input-constrained, and input and output-constrained MPC applied to the approximated model based on the stated conditions.

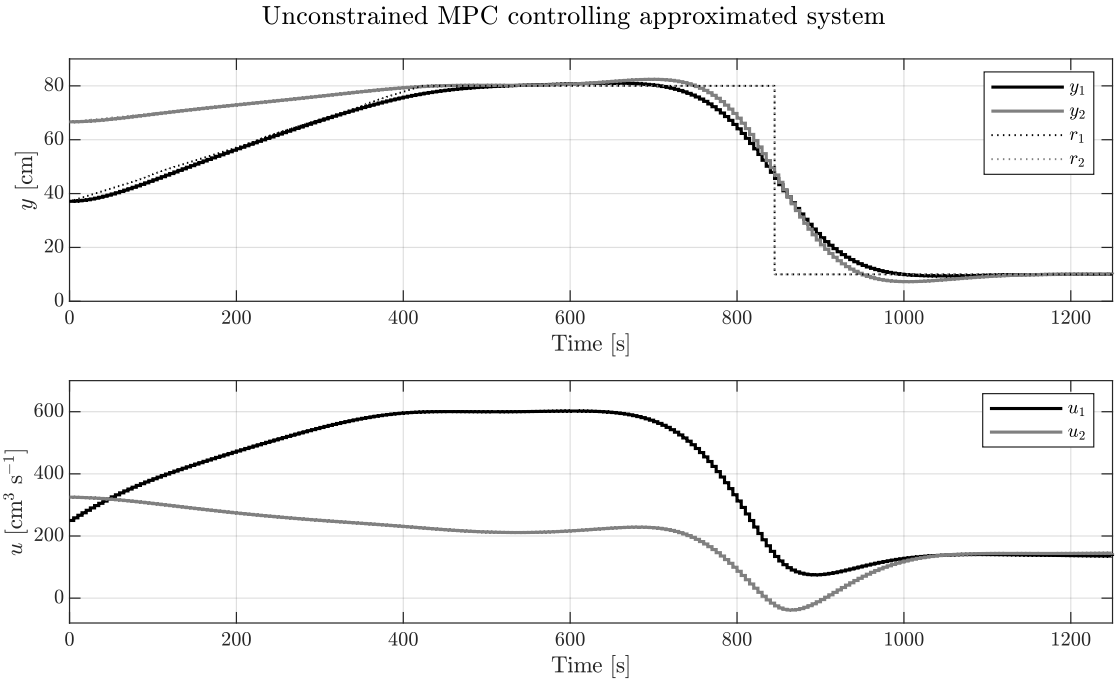

Figure 10.1: Unconstrained MPC applied to the approximated system acquired from analysing the step response.

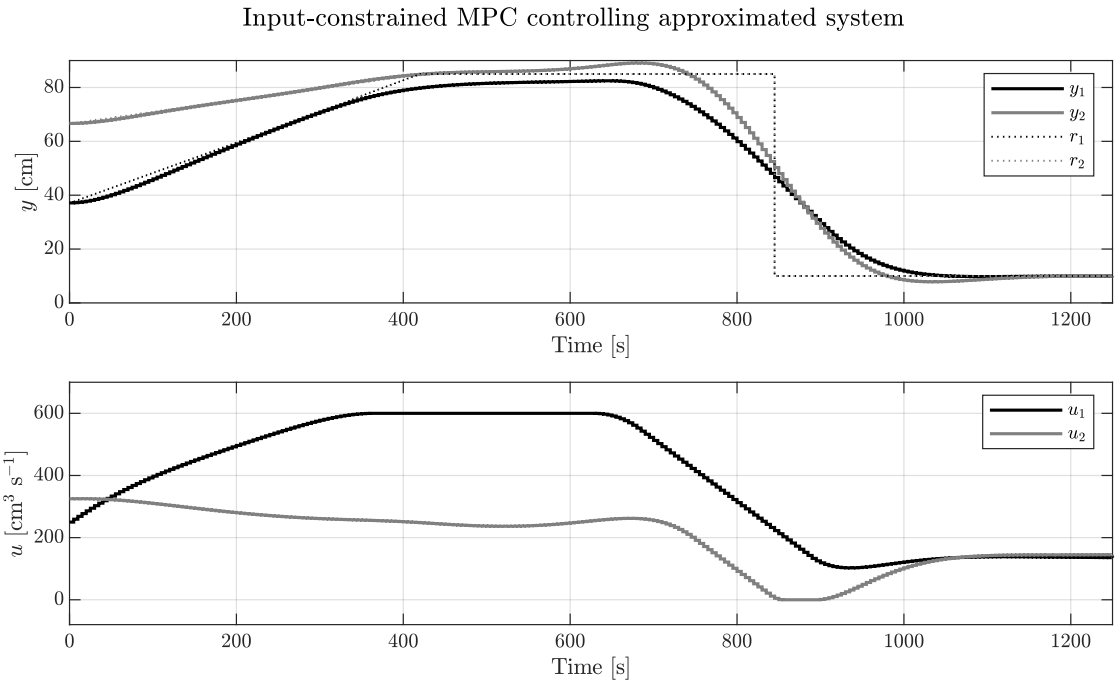

Figure 10.2: Input-constrained MPC applied to the approximated system acquired from analysing the step response.

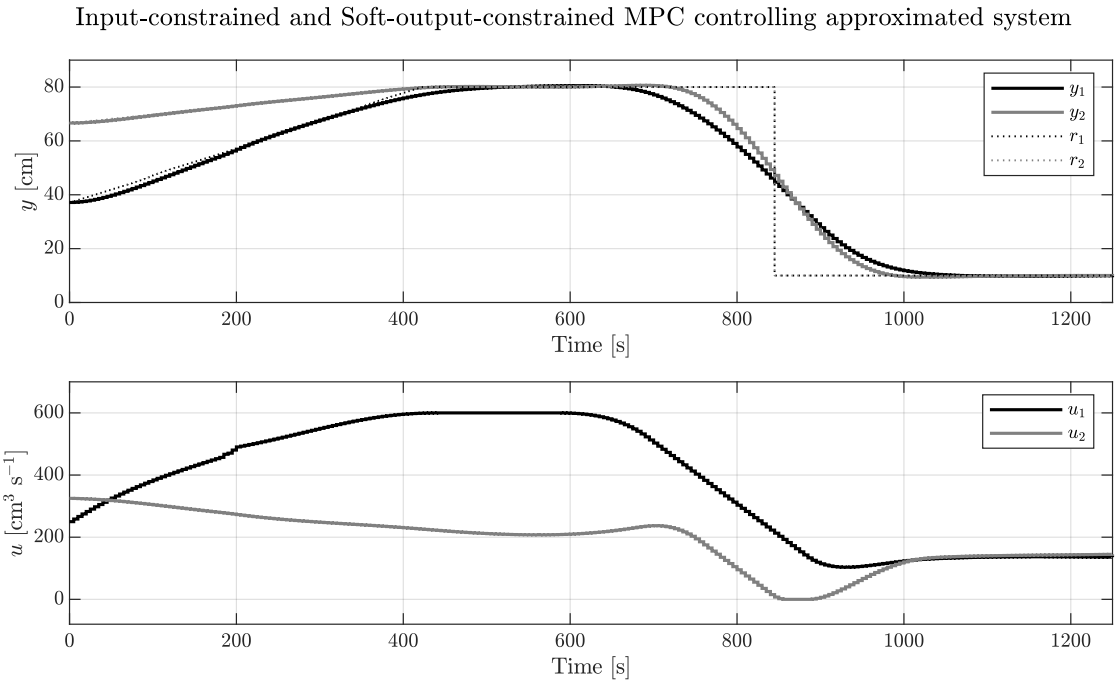

Figure 10.3: Input and output-constrained MPC applied to the approximated system acquired from analysing the step response.

Compared to the linearised deterministic system, the approximated model simulations show that the inputs to the system remained fairly similar. The most obvious difference, however, is that the input required is reduced at the extremes for achieving the same reference tracking in the unconstrained case. Therefore, reference tracking seems slightly improved in the input-constrained case, as the upper limit in the input is fairly close to the maximum input used in the unconstrained case. As a result, contrary to the linear deterministic system, reference tracking is achieved more readily when the soft-output constraints are applied.

1.2. Discrete linearised stochastic model

In this case, noise was added to the linearised deterministic system in a manner that agrees with the representation in Problem 4.5.2. The Kalman filter was used to infer the states using the output.

The following Figures show the unconstrained, input constrained, and input and output-constrained MPC applied to the discrete-linearised stochastic model based on the stated conditions.

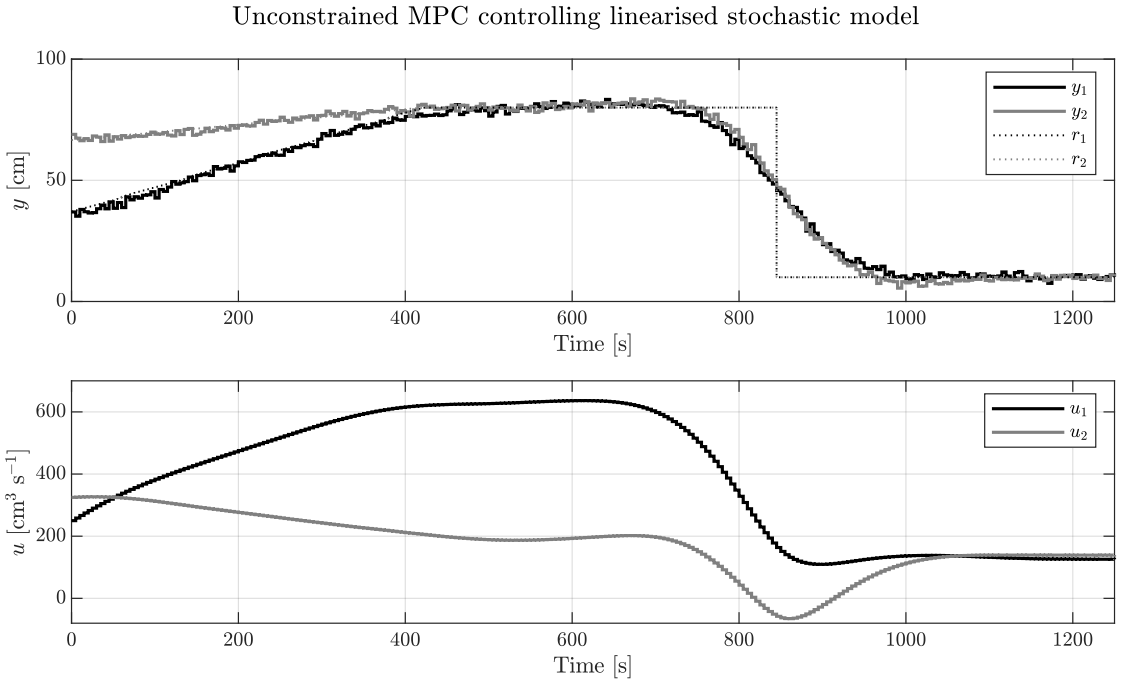

Figure 10.4: Unconstrained MPC applied to the discrete-linearised stochastic system.

It can be seen, that adding process and measurement noise to the system did not alter the input pattern used to control the system significantly. However, it does show that despite the noise, this system representation was able to be controlled using the MPC implementation. Importantly, the soft-output-constrained MPC did not experience infeasibility despite the output response exceeding the set limits due to the noise. Meaning that the penalties to the system worked as intended.

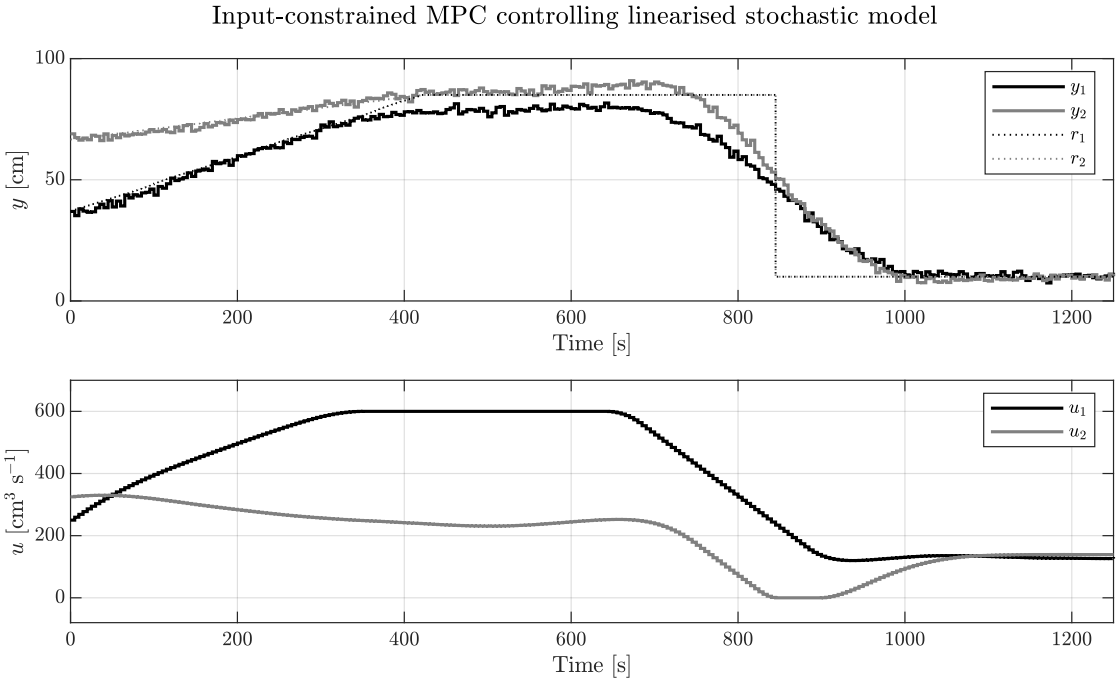

Figure 10.5: Input-constrained MPC applied to the discrete-linearised stochastic system.

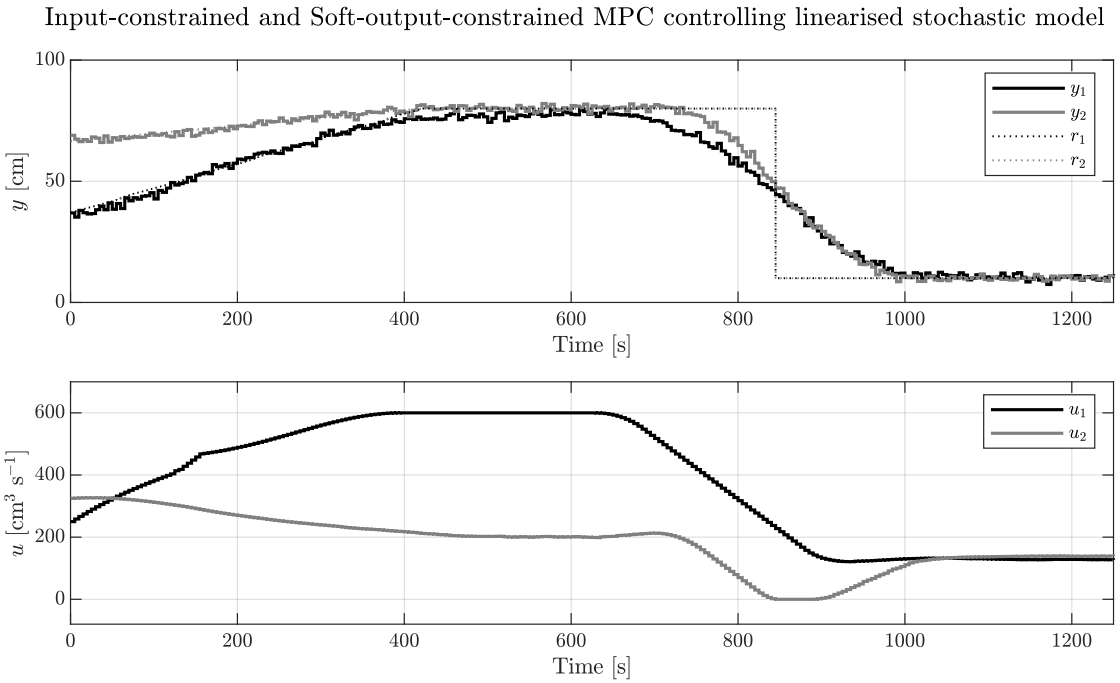

Figure 10.6: Input and soft-output-constrained MPC applied to the discrete-linearised stochastic system.

1.3. Discrete linearised SDE model

To implement the SDE model, the augmented representation shown in Problem 4.5.3. was implemented in the simulation loop. It was found, that in order to obtain a comparable noise profile as in the simulations shown in Problem 10.1.2., the process noise had to be defined with the following parameters:

$$\sigma_{F_3} = \sigma_{F_4} = 0.1\,. \tag{10.3}$$

Furthermore, the Kalman filter was used to infer the states using the output of the model at every step.

The following figures show how the different MPC configurations controlled the SDE model in the simulation.

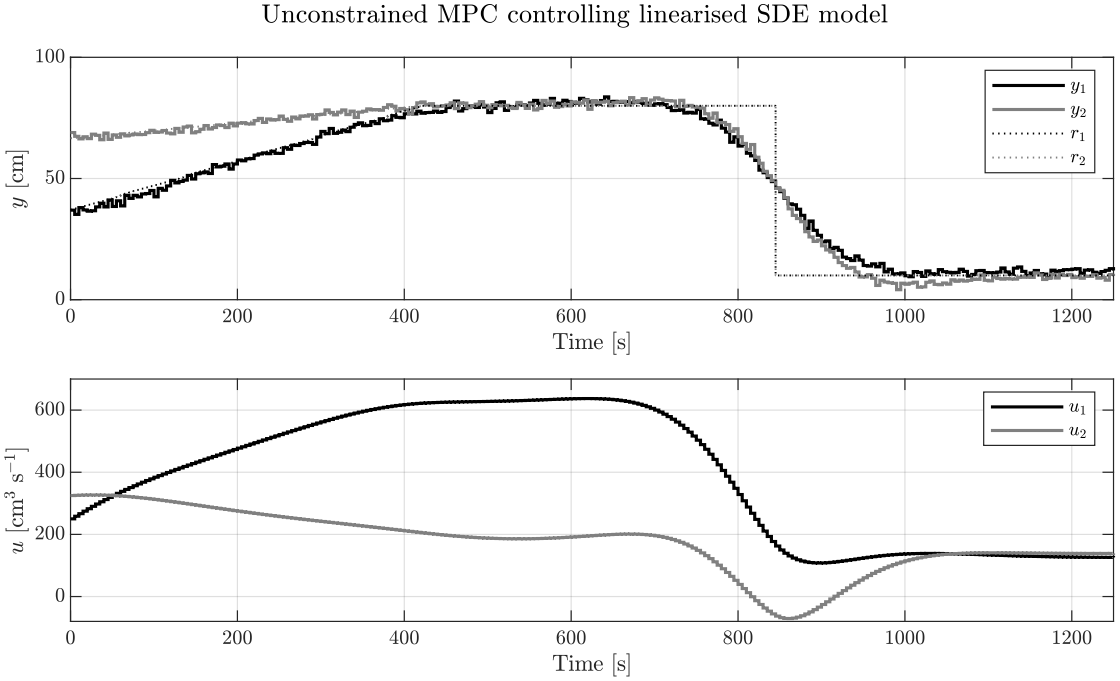

Figure 10.7: Unconstrained MPC applied to the discrete-linearised SDE system.

Overall, the input and response patterns were not significantly different from what was seen in the plots shown in 10.1.2. Therefore, it was shown, that the MPC could be applied to the Euler-Maruyama discretised system.

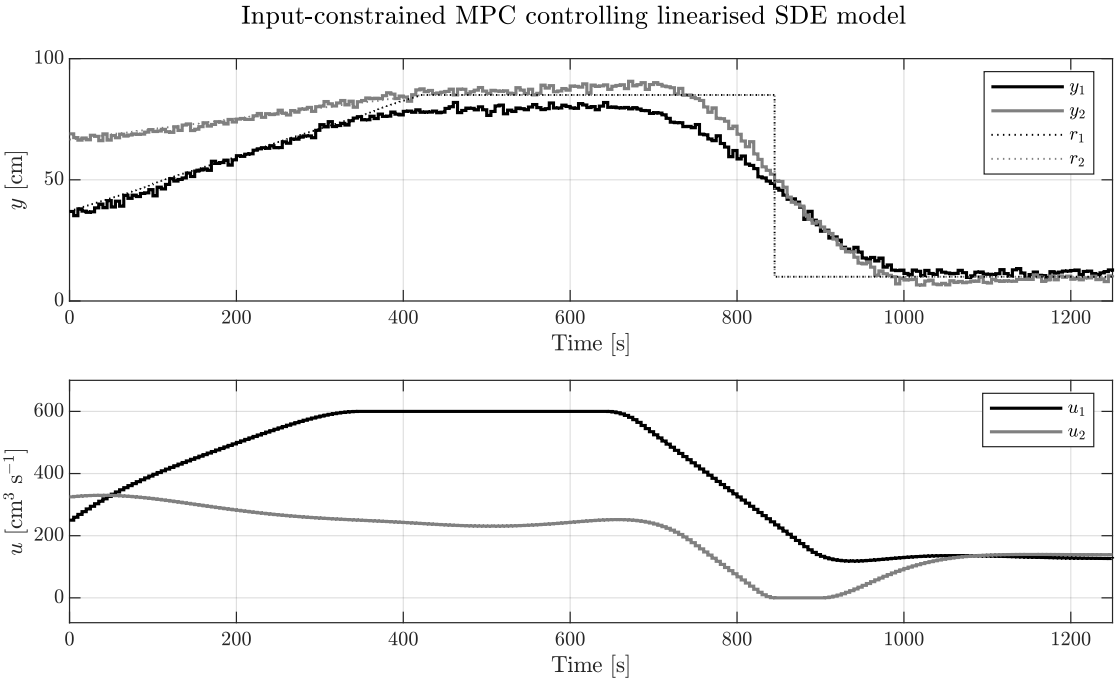

Figure 10.8: Input-constrained MPC applied to the discrete-linearised SDE system.

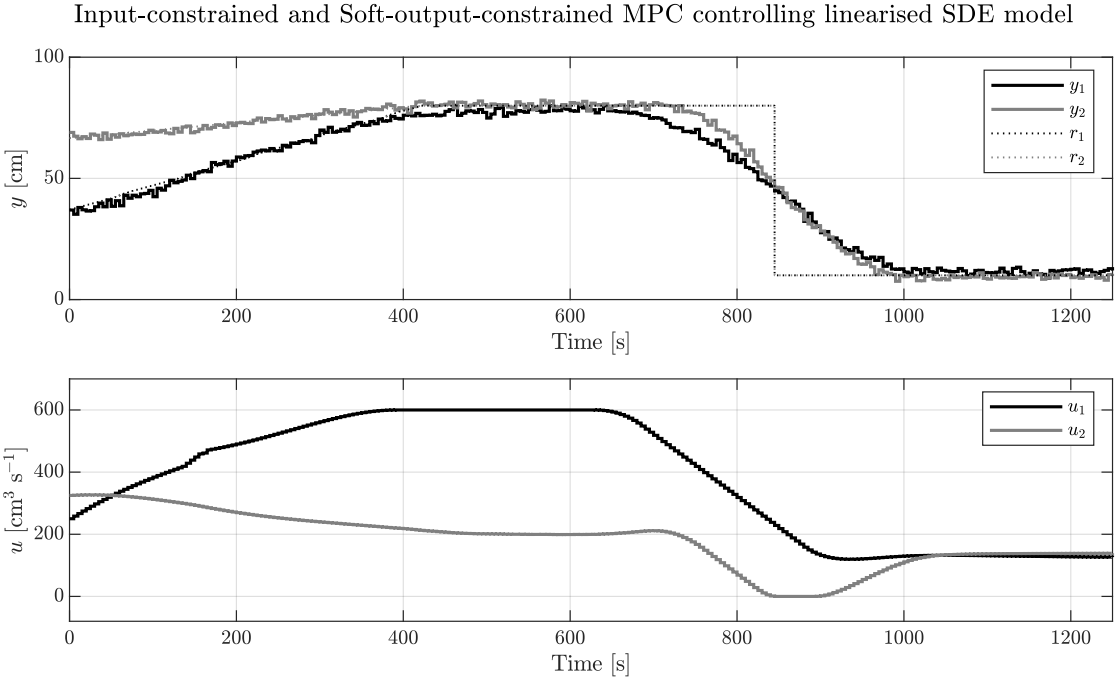

Figure 10.9: Input and soft-output-constrained MPC applied to the discrete-linearised SDE system.

1.4. Nonlinear deterministic model

To allow the MPC to control the nonlinear model, it had to be realized, that the MPC is calculated based on the discrete-linearised model of the system. Therefore, the Kalman filter and the MPC control law were based on the stationary variables used to define the linearised model. As a result, the MPC was implemented in the same way as previously, only the output of the nonlinear model had to be offset based on the identified stationary states. Then, after the calculation of the control law, the control input had to be offset based on the stationary control input.

This allowed the nonlinear model to be controlled. The following figures show how the different MPC configurations controlled the nonlinear deterministic model. Note, that the Kalman filter was used despite this being the deterministic model to verify its validity.

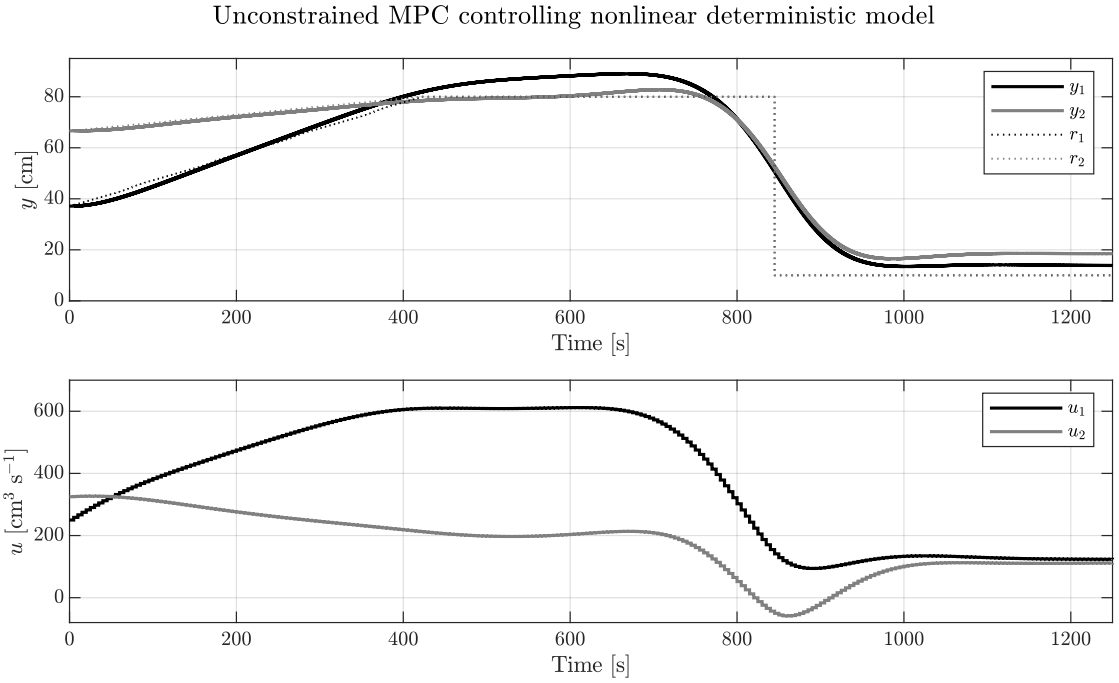

Figure 10.10: Unconstrained MPC applied to the nonlinear-deterministic system.

Find that, the input pattern remained very similar to all the previous models which is expected. However, the output response didn't follow the reference at its extremes as well as it did in the linearised models. This steady-state error is a result of the state estimation and control law being evaluated based on the linearised model.

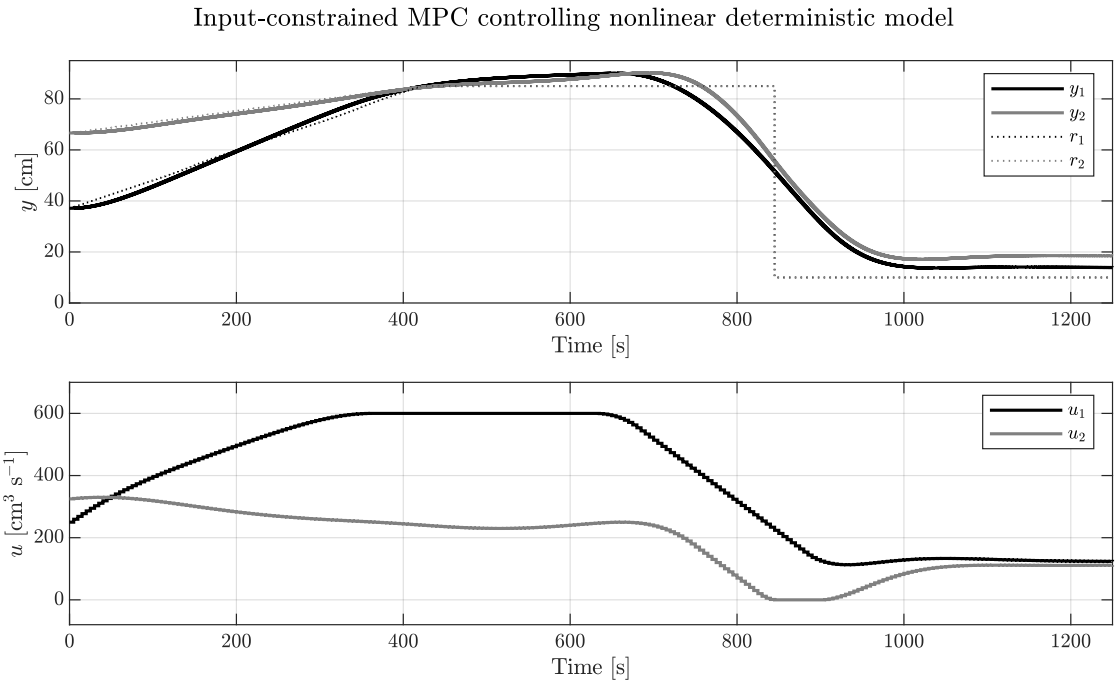

Figure 10.11: Input-constrained MPC applied to the nonlinear-deterministic system.

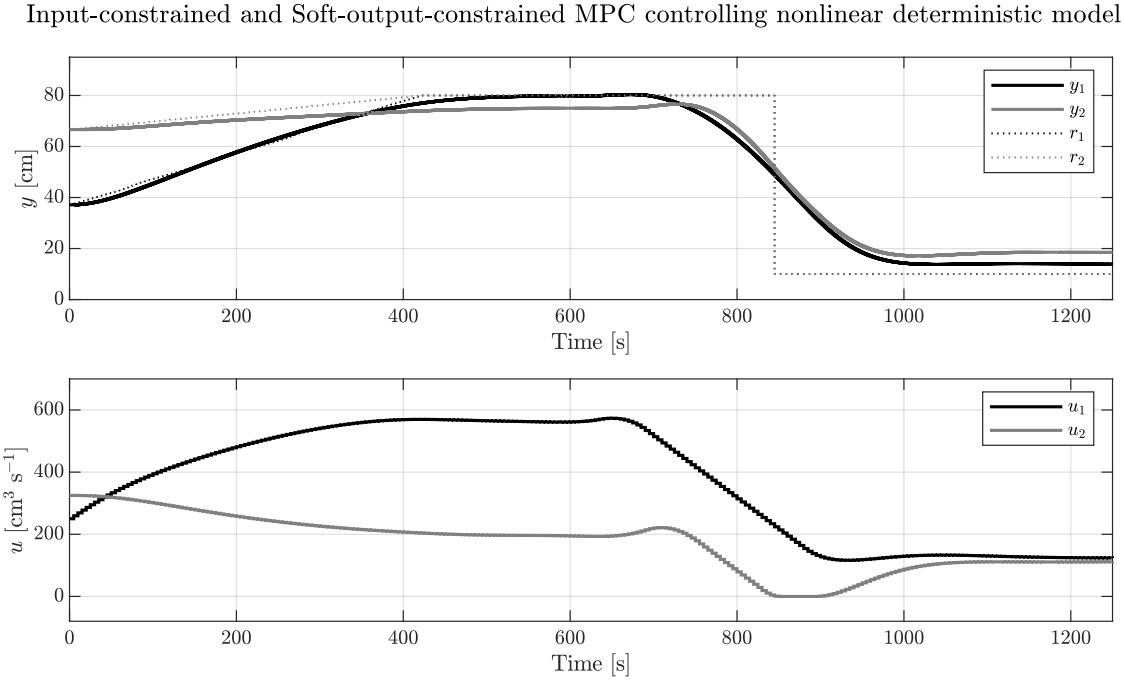

Figure 10.12: Input and soft-output-constrained MPC applied to the nonlinear-deterministic system.

1.5. Nonlinear stochastic model

For the nonlinear stochastic model, the implementation was the same as the nonlinear deterministic model. The only difference was, that the disturbance according to the parameters in Equation 10.2 were imposed at every step.

The following figures display how despite the disturbance, the different MPC configurations were able to control the system to the desired reference.

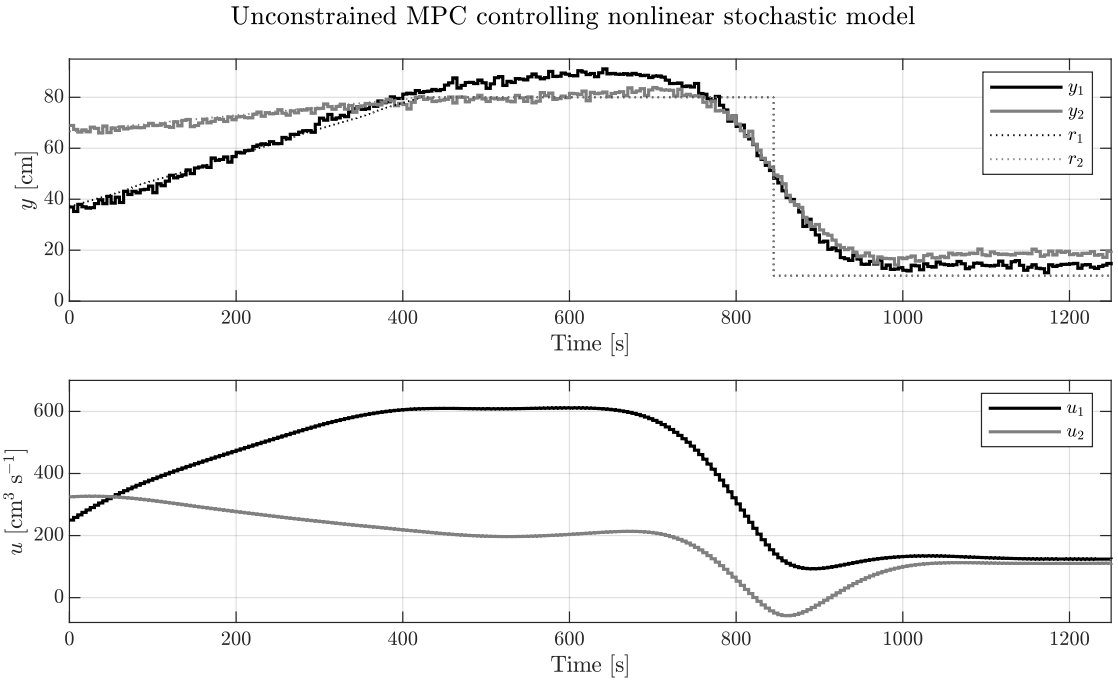

Figure 10.13: Unconstrained MPC applied to the nonlinear stochastic system.

Similarly to what was observed in Problem 10.1.4., there was a constant error at the extremes of the reference due to the same issues based on the state estimation and MPC calculation using the linearised system.

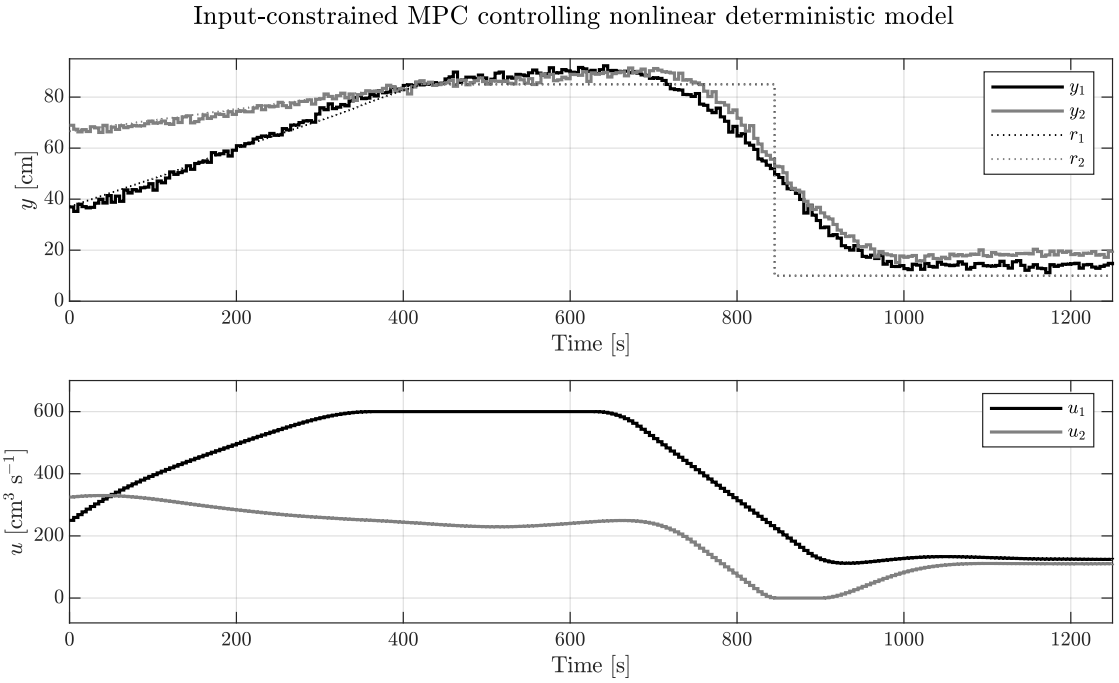

Figure 10.14: Input-constrained MPC applied to the nonlinear stochastic system.

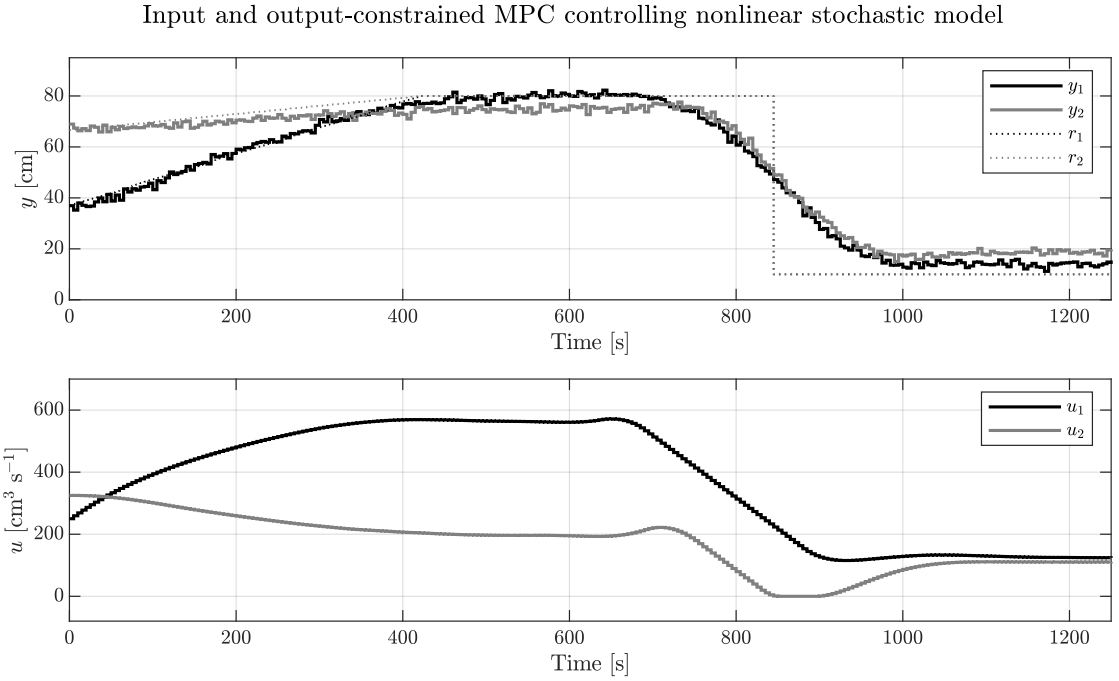

Figure 10.15: Input and soft-output-constrained MPC applied to the nonlinear stochastic system.

1.6. Nonlinear SDE model

2. Discussion

Across all models, the input pattern, expectedly, remained consistent. This was even observed with the approximated model, indicating that it may be possible to control a physical system of the modified-four-tank system, based off approximations of the step input.

The other discrete model systems were consistent with each other, despite noise being added to some of the systems.

It was apparent, that there was a kink in input 1, in the linearised model simulations occurring at around 180 s in the input and soft-output-constrained MPCs. This was, most likely, a result of `quadprog()` not being able to find an optimal solution within its iteration limit. However, the input-constraints weren't violated and it can be seen, that the output remained behaved. In general, the hardware used to evaluate the input-constrained and soft-output-constrained MPC was insufficient as simulations would take up to 4 times longer (up to 30 mins) than the purely input-constrained system.

Interestingly, the shortcomings of the MPC based on the linearised models were exposed in the nonlinear model simulations, seeing as the tracking performance deteriorated the further the output response had to be controlled from the stationary state. As a result, if a control law were to be based on a linear model of a physical system, the stationary point from which it is to be calculated from should be chosen wisely.

Problem 11. Nonlinear MPC.

1. Provide a continuous-discrete mathematical model for the modified-four-tank system.

A continuous-discrete mathematical model of the system would mean that the process is continuous, while the measured output is sampled. For the modified-four-tank system, this is given by:

$$\mathrm{d}\begin{bmatrix}\boldsymbol{x}(t)\\ \boldsymbol{d}(t)\end{bmatrix} = \rho\left(\begin{bmatrix}-a_1\sqrt{2gm_1(t)/(A_1\rho)}+a_3\sqrt{2gm_3(t)/(A_3\rho)}\\ -a_2\sqrt{2gm_2(t)/(A_2\rho)}+a_4\sqrt{2gm_4(t)/(A_4\rho)}\\ -a_3\sqrt{2gm_3(t)/(A_3\rho)}\\ -a_4\sqrt{2gm_4(t)/(A_4\rho)}\\ 0\\ 0\end{bmatrix} + \begin{bmatrix}\gamma_1 & 0\\ 0 & \gamma_2\\ 0 & 1-\gamma_2\\ 1-\gamma_1 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\boldsymbol{u}(t) + \begin{bmatrix}0 & 0\\ 0 & 0\\ 1 & 0\\ 0 & 1\\ 0 & 0\\ 0 & 0\end{bmatrix}\boldsymbol{d}(t)\right)\mathrm{d}t + \begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ \sigma_{F_3} & 0\\ 0 & \sigma_{F_4}\end{bmatrix}\mathrm{d}\omega_d(t)\,, \tag{11.1}$$

$$\boldsymbol{y}_k = \begin{bmatrix}1/(A_1\rho) & 0 & 0 & 0 & 0 & 0\\ 0 & 1/(A_2\rho) & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_k\\ \boldsymbol{d}_k\end{bmatrix} + \boldsymbol{v}_k\,, \tag{11.2}$$

$$\boldsymbol{z}(t) = \begin{bmatrix}1/(A_1\rho) & 0 & 0 & 0 & 0 & 0\\ 0 & 1/(A_2\rho) & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{x}(t)\\ \boldsymbol{d}(t)\end{bmatrix}\,. \tag{11.3}$$

Note, that this is based on the nonlinear stochastic model from Problem 2.3.

2. Provide and implement a continuous-discrete extended Kalman filter for the modified-four-tank system and test it by simulation.

3. Implement a prediction-error method for the modified-four-tank system. Do a stochastic simulation and use the data to identify parameters using the prediction-error method. Compare the identified parameters to the true parameters. Compare the predictions of the identified model to the predictions with the true model.

4. Implement a bound-constrained NMPC for the modified-tank system.

5. Test the NMPC by closed-loop simulations.

6. Compare the closed-loop simulations of the NMPC to the closed-loop simulations of the linear MPC of Problem 9.

The use of a nonlinear MPC would allow the errors occurring as a result of reference tracking at values far from the stationary states, used to calculate the linearised model, to vanish.

Problem 12. Economic Linear MPC and Nonlinear MPC.

1. Linear Economic MPC

1.1. Formulate the optimal control problem for the linear economic MPC.

The linear economic MPC for the purpose of this assignment is to keep the minimum level in the lower tanks above a certain reference. To do this, the input to the system will be associated with a pumping cost and constraints based on the levels in the tanks will be imposed. Additionally, as to not cause infeasibility, a certain amount of slack will be tolerated. The resulting optimal control problem and the constraints related to it is given by:

$$\min_{\{u_k, v_{k+1}\}_{k=0}^{n-1}} \phi = \sum_{k=1}^{n} \gamma v_k + \sum_{k=0}^{n-1} c_k^\top u_k \,,$$
$$\text{such that} \begin{cases} u_{\min} \leq u_k \leq u_{\max} \\ \Delta u_{\min} \leq \Delta u_k \leq \Delta u_{\max} \\ y_k \geq r_k - v_k \\ v_k \geq 0 \end{cases} . \tag{12.1}$$

where the constraints on the input are built up in the same manner as in the MPC in Problem 8 and the output y_k is based on the output of the system. The variable c_k represents the cost of the input to the system, while γ represents the cost of violating the minimum reference.

As it can be seen, the optimal control problem for the linear economic MPC is a linear cost function and will be implemented as such in MATLAB.

1.2. Implement the optimal control problem for the linear economic MPC.

To implement the control problem in MATLAB, the control problem from Equation 12.1 was reformulated as follows:

$$\min_{[UV]^\top} \phi = \begin{bmatrix} g_u \\ g_v \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} ,$$
$$\text{where,} \quad g_u = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} , \quad g_v = \gamma \cdot \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{n \times 1} . \tag{12.2}$$

As mentioned previously, the cost c over the depth n was chosen arbitrarily and γ was chosen chosen to be sufficiently large, such that the cost of violation would force the control law to keep the tank levels above the set reference.

Next, the constraints on u_k and Δu_k were constructed in the same manner as in Problem 8. Additionally, the introduction of the slack variable made the implementation the same as to what was found in Problem 9. Therefore, the implementation of the input constraints was as follows:

$$U_{\min} \leq \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \leq U_{\max} \,, \tag{12.3}$$

$$\Delta U_{\min} \leq \begin{bmatrix} \boldsymbol{\Lambda} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \leq \Delta U_{\max} \,. \tag{12.4}$$

Following this, the constraint on the output based on the reference could be formed. This had to be determined from the state transition, and the zero-state input response. As stated previously, the

economic linear controller is to keep the minimum tank level above the reference height, minus, the slack variable. Therefore,

$$R - V \leq \boldsymbol{\Phi}\hat{x}_0 + \boldsymbol{\Gamma_u} U + \boldsymbol{\Gamma_d} D\,, \tag{12.5}$$
$$R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D \leq \boldsymbol{\Gamma_u} U + V\,, \tag{12.6}$$

thus,

$$R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D \leq \begin{bmatrix} \boldsymbol{\Gamma_u} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \leq \infty\,. \tag{12.7}$$

Lastly, as the slack variable is to give leniency to the set minimum, it is desirable for this value to remain greater than 0. Therefore, it was implemented as follows:

$$0 \leq \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \leq \infty\,. \tag{12.8}$$

Collectively, the function and its constraints were implemented in the MATLAB function `linprog()`, which is used to solve linear programming problems. However, it takes constraints in the form $Ax \leq b$, i.e. only from one side. Thus, to implement the above obtained constraints, the following matrices were defined:

$$\begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{I} \\ -\boldsymbol{\Lambda} & \boldsymbol{0} \\ \boldsymbol{\Lambda} & \boldsymbol{0} \\ -\boldsymbol{\Gamma_u} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \leq \begin{bmatrix} -U_{\min} \\ U_{\max} \\ \boldsymbol{0} \\ -\Delta U_{\min} \\ \Delta U_{\max} \\ R - \boldsymbol{\Phi}\hat{x}_0 - \boldsymbol{\Gamma_d} D \end{bmatrix}, \tag{12.9}$$

Allowing the MATLAB function to evaluate the input to the system within the loop as follows:

```
[u,info] = linprog([gu;gv],A_u,b_u);
u = [u(1);u(2)];
```

1.3. Do closed-loop simulations with a Kalman filter for the linear economic MPC.

The economic linear MPC tries to optimise the use of lower cost input effort that achieves the same minimum tank level. For the modified-four-tank system, either input is able to affect the level in both tanks. Therefore, it was decided to also observe the responses when either input was associated with a different cost.

The program set up to simulate the system worked as follows:

1. Obtain the discrete-time linearised system for a 5 second sampling time.
2. Set the depth to 30, i.e. 150 s.
3. Pre-compute the static matrices required for the control objective.
4. Define the cost variable (see Figures 12.1 & 12.3).
5. Define slack variable factor (set to $1 \cdot 10^6$). Note, that it must be high enough to penalize not achieving the minimum level.
6. Start simulation loop:
   i. Evaluate the linearised model.
   ii. Estimate states from the noisy output.
   iii. Select reference and cost based on depth at current step.
   iv. Evaluate dynamic matrices for control objective (ΔU).
   v. Evaluate `linprog()`.
   vi. Select first two outputs of `linprog()` as the input to the system.

For the first simulation, the cost for the second input was chosen to be higher first, equal, and then lower than the first input. This is shown in Figure 12.1.

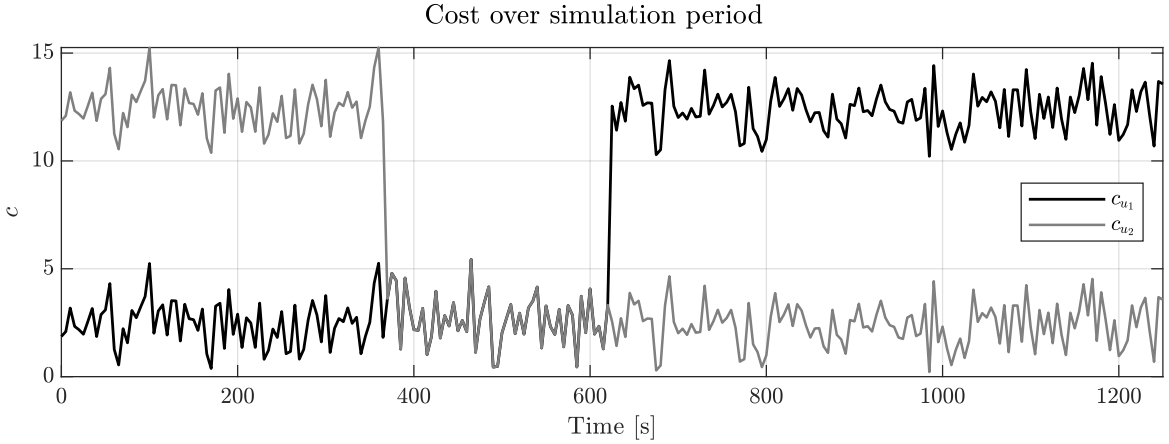

Figure 12.1: Evolution of the cost associated with either input.

Based on the cost evolution, the inputs were selected accordingly resulting in the heights in the tanks to change as seen in Figure 12.2.

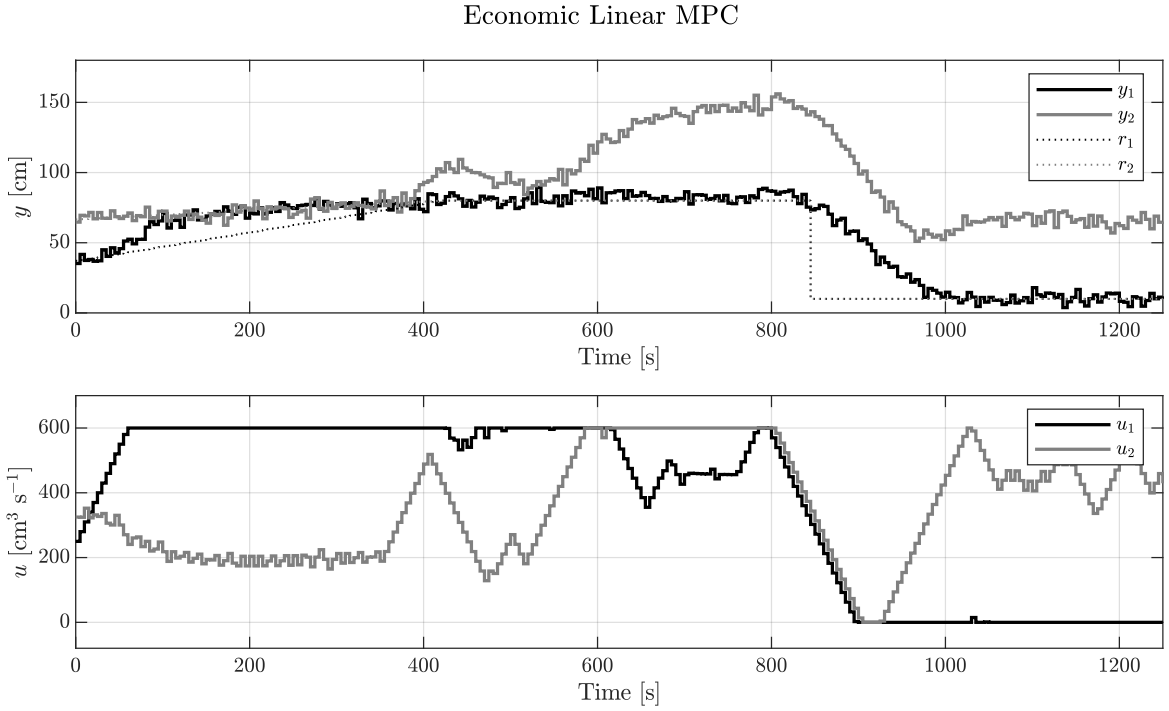

Figure 12.2: Measured heights in tanks 1 and 2, and the inputs required to achieve the minimum heights set by the reference.

It can be seen, that since the input cost is so high for tank 2, the controller tries to keep its evolution exactly on the reference. Tank 1 on the other hand, is allowed to exceed the reference and is encouraged to do so due to its lower price. However, it can only do this as long as the input remains within the set upper limit of $600\,\mathrm{cm}^3\,\mathrm{s}^{-1}$. After around 600 s the cost for input 1 becomes higher, and since input 2 hasn't been in saturation, it assumes the maximum value in order to relieve work being done by input 1 due to its high cost. This is then reflected in the much higher tank height to compensate.

To verify the opposite relation, the cost evolution was inverted. This is seen in Figure 12.3.

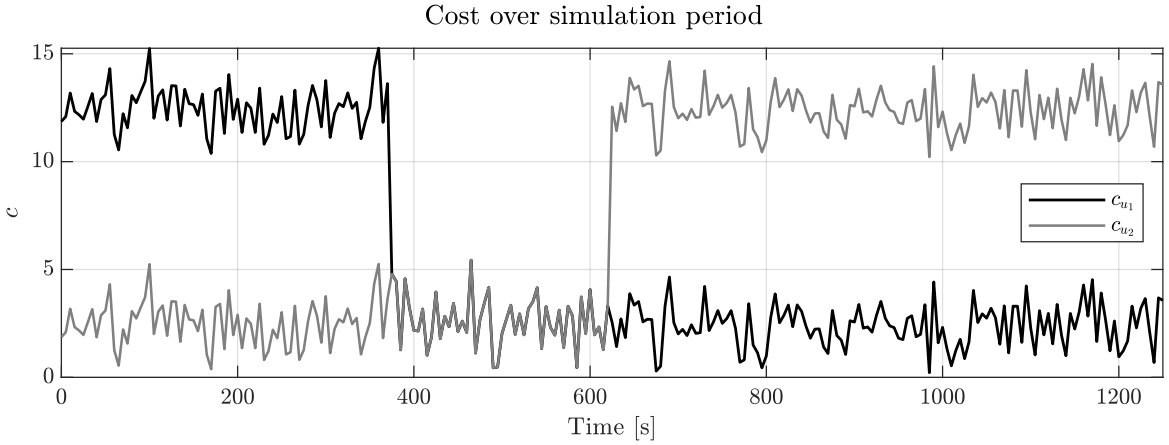

Figure 12.3: Evolution of the cost associated with either input.

This resulted in the following tank height evolution (see Figure 12.4).

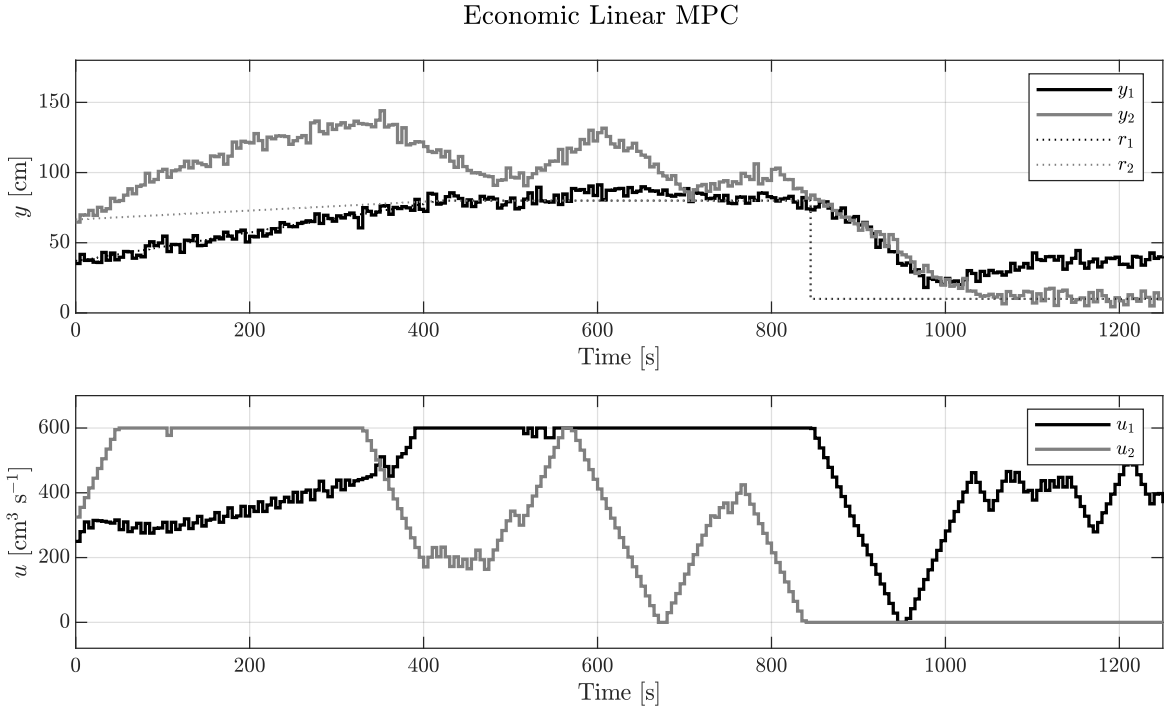

Figure 12.4: Measured heights in tanks 1 and 2, and the inputs required to achieve the minimum heights set by the reference.

Similarly, the controller tries to maximise inputs coming from relatively cheaper inputs when they are available. As a result, tank heights rise in order to keep the more expensively run tank at the minimum. In Figure 12.4, input 2 becomes expensive near the end, and as it was possible to keep both heights above the reference by only using input 1, the MPC kept input 2 at the lowest possible value, 0.

1.4. Compare and discuss the linear economic MPC to other controllers.

Compared to other controllers, the linear economic MPC will try to ensure that the reference remains the minimum output, in accordance with the cost. Therefore, if like in the modified-four-tank system there are multiple possibilities to achieve the required output, it will select an input pattern that reduces cost as much as possible. This could mean making one level in the tank much higher, because of its relative lower cost, so the other tank is also above the minimum reference for an overall low input cost.

Conventional controllers on the other hand, aim to reduce the error in the response to the reference. Thus, dependent on e.g. the weighting matrix, a given output might not follow the reference due to the priority being set on the other output following its reference.

2. Nonlinear Economic MPC

2.1. Formulate the optimal control problem for the nonlinear economic MPC.

2.2. Implement the optimal control problem for the nonlinear economic MPC.

2.3. Do closed-loop simulations with an extended Kalman filter for the linear economic MPC.

2.4. Compare and discuss the nonlinear economic MPC to other controllers.

Problem 13. PID Control.

1. Discuss the pairing of inputs and outputs for the four-tank system.

The system is set up such that both inputs have influence on either output. This is because input 1 directly affects output 1 and the height of the tank above output 2. For input 2, the direct effect is in output 2 and in the height of the tank above output 1. As a result, the height of the water in the tanks above output 1 and 2 trickle down into the lower tanks, indirectly affecting outputs 1 and 2.

This adds complexity to the system as there exist many input patterns to end up with the same output in one of the tanks. This allows, as shown in the economic MPC, the use of e.g. input 1 to manipulate output 2.

2. Implement a P-, a PI- and a PID-controller for the four-tank system.

The controller parameters were found by tuning the parameters available, and simulating on the linearised-deterministic system, such that an acceptable tracking performance was achieved. The P-, PI- and PID-controllers were found to be as follows:

$$D_P(z) = 200\,, \quad D_{PI}(z) = \frac{270 + 90z^{-1}}{1 - 0.1z^{-1}}\,, \quad D_{PID}(z) = \frac{180 + 60z^{-1} + 30z^{-2}}{1 + 0.5z^{-1} + 0.1z^{-2}}\,. \tag{13.1}$$

The controllers were implemented in the simulation as follows:

1. Define the values of the parameters used
2. Start the simulation loop
3. Simulate the system
4. Estimate the states
5. Use the estimated states to obtain the estimated output
6. Evaluate the error based on: $e = r - \hat{y}$
7. Apply the error to a direct realization of the discrete-time controllers
8. Update past error values and past input values used in the PI and PID controllers.

As an example, the implementation of the PID-controller was as follows:

```
% Controller parameters
b0 = 180; b1 = 60; b2 = 30; a0 = -0.1; e2 = 0; e3 = 0;
for k = sim_length
    % Simulation
    d = chol(Qk)'*randn(2,1);
    x = Ad*x+Bd*u+Bvd*d;
    v = chol(Rk)'*randn(2,1);
    y = Cd*x+v;
    z = Cd*x;

    % State Estimation
    [xh,Pk] = statestimator(xh,Pk,Ad,Bd,Bvd,Cd,u,y,Rk,Qk);
    xhd = [xhd xh];

    % Output estimation
    y = Cd*xh;
```

```
    % Select reference
    i_ref = 2*k+1;
    R = ref(i_ref:(i_ref+2*n-1));
    % Control law
    e1 = R(1:2)-y;
    ed = [ed;e1'];
    u = b0*e1+b1*e2+b2*e2+a0*u;
    e3 = e2;
    e2 = e1;

    % Imposing a lower bound
    if u(1) < -F1
        u(1) = -F1;
    end
    if u(2) < -F2
        u(2) = -F2;
    end
end
```

The estimated output is used instead of the actual output of the system, because the addition of noise to the system made the control input very erratic. The Kalman filter, filters the noise such that it is beneficial to use the estimated output instead. Furthermore, a lower-bound input constraint was imposed that ensured that the input to the system could not fall below 0.

3. Test the controllers by closed-loop simulation.

The following figures show the implementation of the P-, PI-, and PID-controller, respectively, on the discrete-linearised stochastic model.

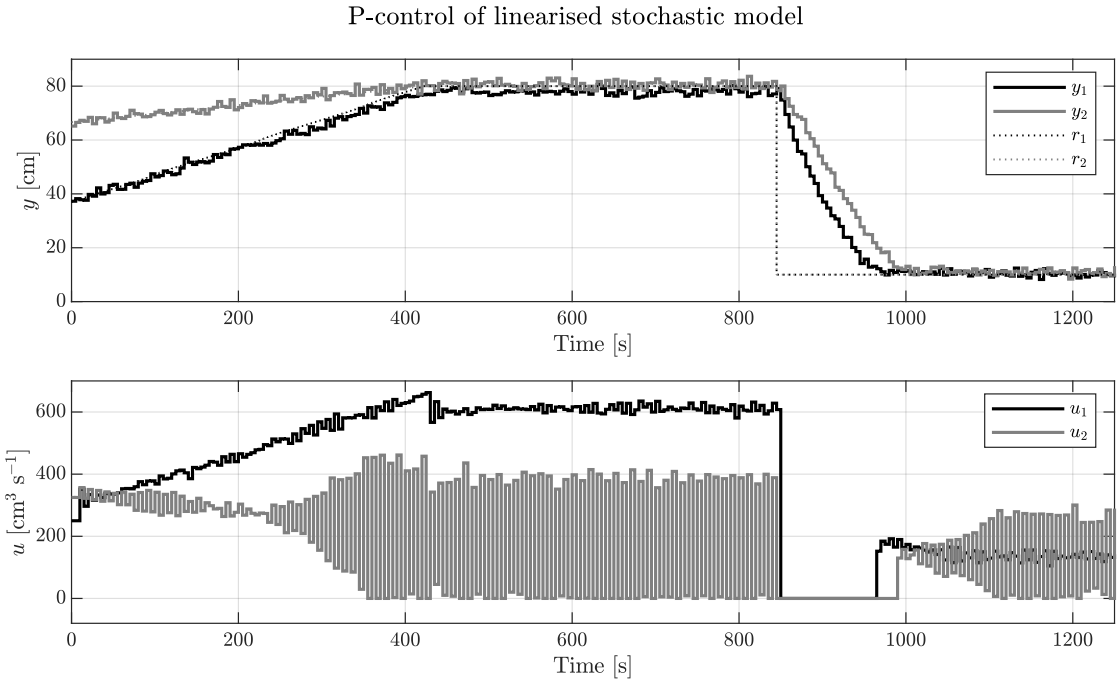

Figure 13.1: P controller applied to the discrete-stochastic model.

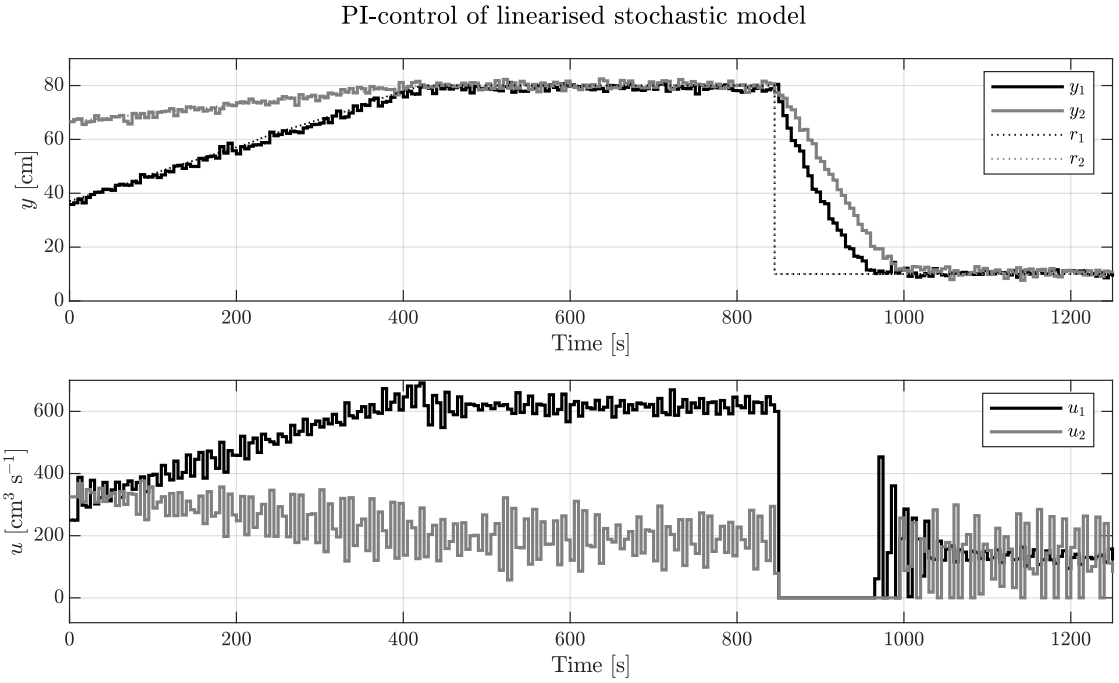

Figure 13.2: PI controller applied to the discrete-stochastic model.

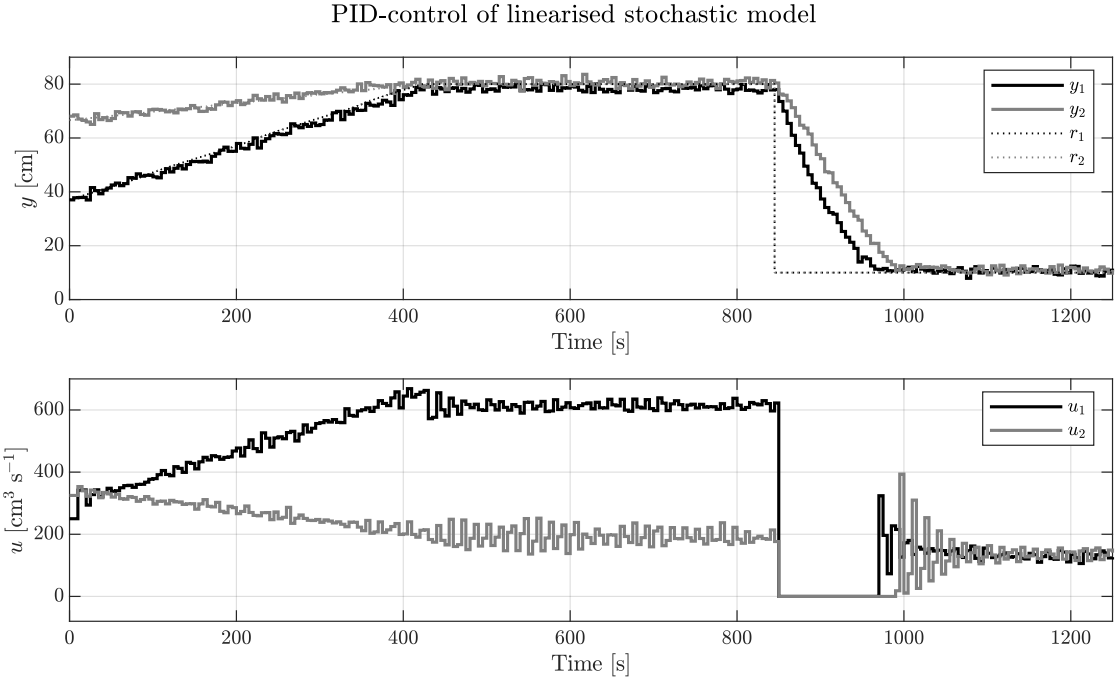

Figure 13.3: PID controller applied to the discrete-stochastic model.

All the PID-type controllers were able to generate a control law able to follow the set reference. However, it can be seen that the PID-controller uses the least amount of control effort for an output response that is insignificantly different from the P- and PI-controllers.

4. Compare the PID type controllers to other controllers e.g. MPC.

To compare the performances, the PID controller (best performing) was compared to an input-constrained MPC applied to the same system. The input-constrained MPC was implemented with no rate of movement constraints and only a lower bound constraint such that the input didn't fall below 0. This allowed it to operate under similar conditions as the PID-controller shown in Problem 13.3.

The resulting response of the system being controlled by this MPC is shown in Figure 13.4.

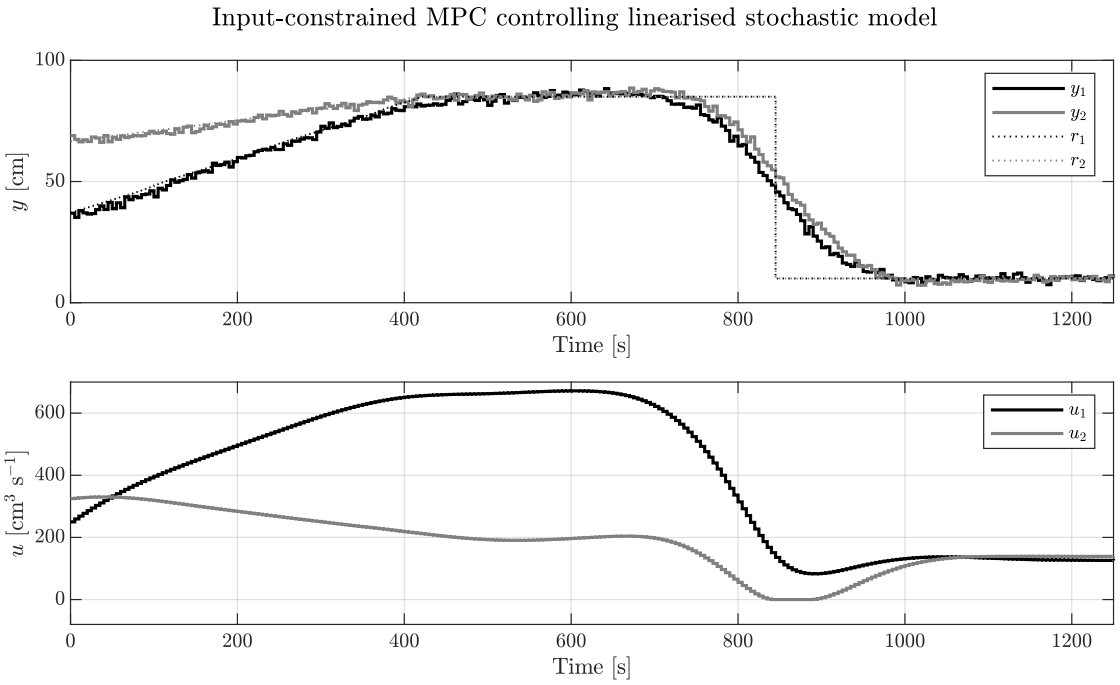

Figure 13.4: Input-constrained MPC used to compare with the PID controller.

The difference between the reference and the outputs were recorded and compared to determine which controller performed better. This is shown in Figure 13.5.

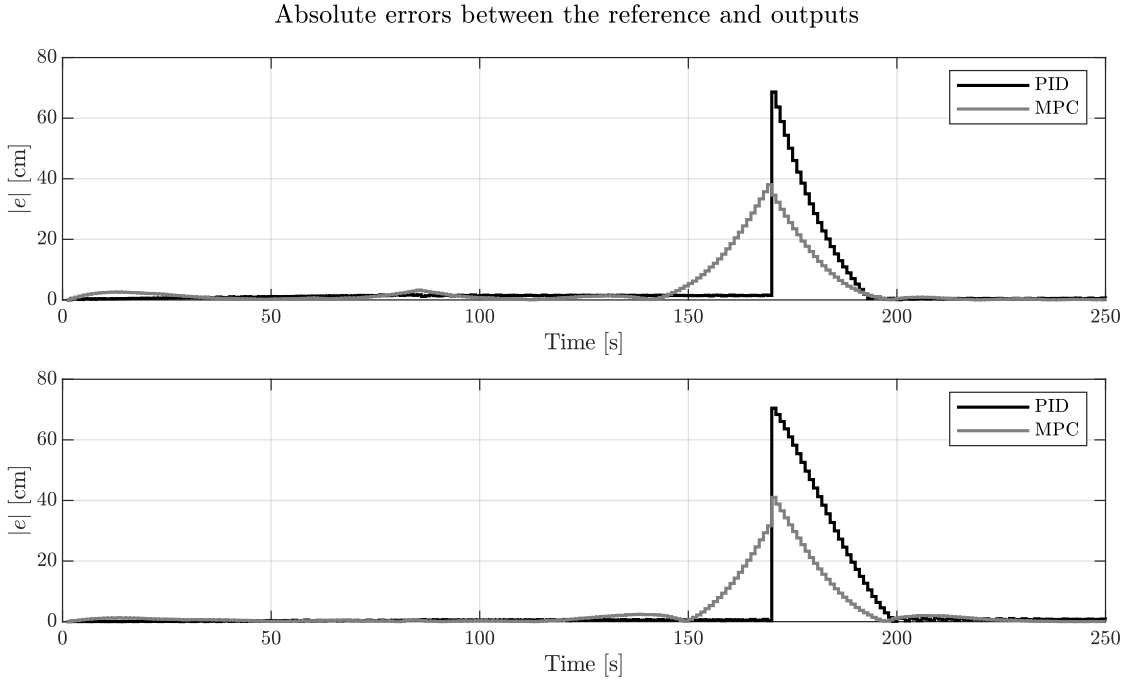

Figure 13.5: Comparison of the absolute errors resulting from the use of the PID and MPC controller.

It can be seen, that the MPC gives a better tracking performance, as it keeps a lower overall error than the PID controller. This is a result of the MPC predicting and using the information it is provided with, to lower the tank heights prematurely, before the reference is being stepped down. The PID on the other hand, only knows about the change of reference when it occurs, resulting in a high error. Additionally, it was found from summing the absolute errors over the entire simulation, that the PID controller had a higher overall error at 1991 cm, whereas the MPC had 1778 cm.

The MPC also displayed a lower control effort in that it wasn't changing erratically over the simulation period. This comes from its calculation of its objective functions at every step, allowing it to find an optimal control law at each step.

It must be said, however, that the PID controller could be implemented very quickly and is able to calculate its control law quickly as well. The MPC has a higher inherent complexity and requires more computing power, as well as, preparation.

When it comes to a physical implementation where computing power is available, the MPC would be preferred. This is due to the ability of being able to control constraints onto the system, and being able to obtain an optimal control law that isn't erratic despite potential noise in the system.

Problem 14. Discussion and Conclusion.

This report has shown the principles used to construct various MPCs on the given modified-four-tank system. These MPCs were tested and their differences were evaluated.

Using MPCs to form the control law of a system is a powerful tool, as seen in the vastly more efficient manner of control compared to a PID-type controller. By anticipating the response of the system, the MPC is able to generate a better control law. This was implemented based on the linearised system dynamics, allowing the formulation of the unconstrained, input-constrained, input-constrained with soft-output constraints, and economic MPCs. Where the latter controller's objective is to keep the response above the given level and the other controllers try to track the reference. Within the construction of these controllers, it was possible to implement a program that could alter the implemented constraints very easily.

Unfortunately, with my current knowledge of MATLAB and unfamiliarity with NMPCs, it wasn't possible to simulate the nonlinear SDE and construct a nonlinear MPC. The nonlinear MPC is in theory able to achieve a much better performance, as it would be based more closely on the actual dynamics of a given system.

Overall, this report has shown how to go from modelling a system, to how to estimate the states of the system, to building the basis for MPCs and introducing how these can be varied based on the control objective and limitations of a system. All of these methods were verified through implementations and simulations with MATLAB. Therefore, this report and its related programs, can be used as a reference for MPC-related subject matter in the future.

Appendix.

A MATLAB function of the modified-four-tank system dynamics.

```
function xdot = ModifiedFourTankSystem(t,x,u,d,p)
m = x;
F(1:2) = u;
F(3:4) = d;
a = p(1:4,1);
A = p(5:8,1);
gam = p(9:10,1);
g = p(11,1);
rho = p(12,1);

% Inflow
qin = zeros(4,1);
qin(1,1) = gam(1)*F(1);
qin(2,1) = gam(2)*F(2);
qin(3,1) = (1-gam(2))*F(2)+F(3);
qin(4,1) = (1-gam(1))*F(1)+F(4);

% Outflow
h = m./(rho*A);
qout = a.*sqrt(2*g*h);

% Differential equations
xdot = zeros(4,1);
xdot(1,1) = rho*(qin(1,1)+qout(3,1)-qout(1,1));
xdot(2,1) = rho*(qin(2,1)+qout(4,1)-qout(2,1));
xdot(3,1) = rho*(qin(3,1)-qout(3,1));
xdot(4,1) = rho*(qin(4,1)-qout(4,1));
end
```

B Low and high noise level responses.

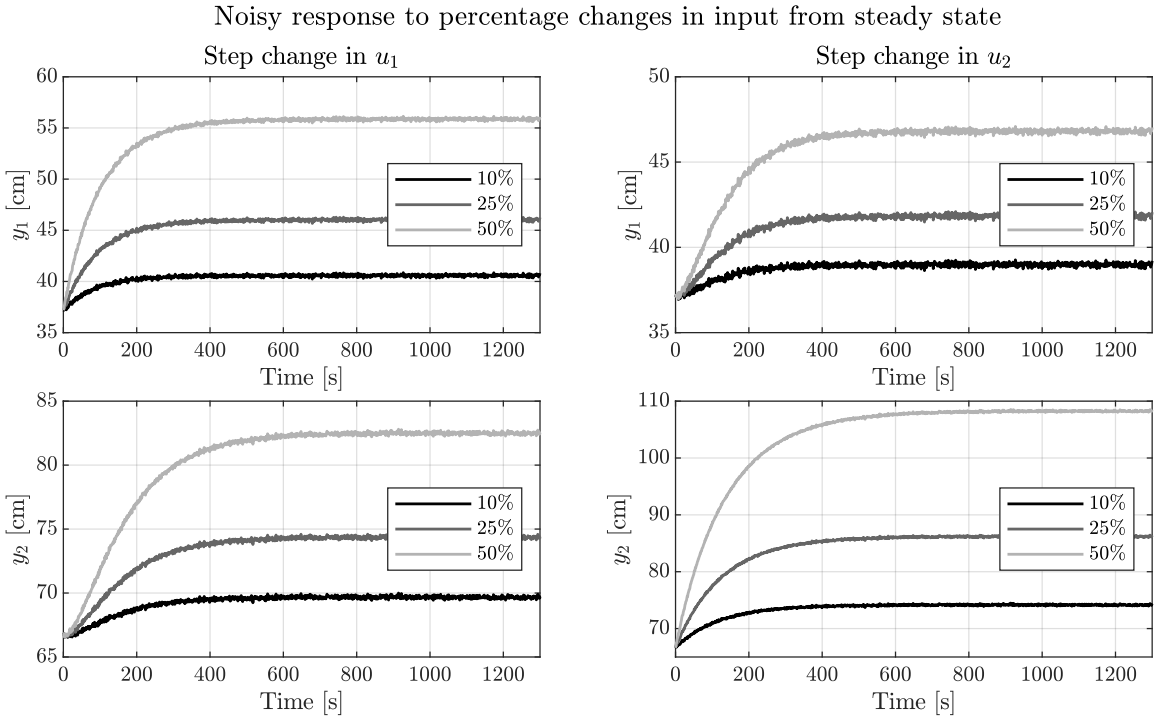

Figure 15.1: Low noise step responses.

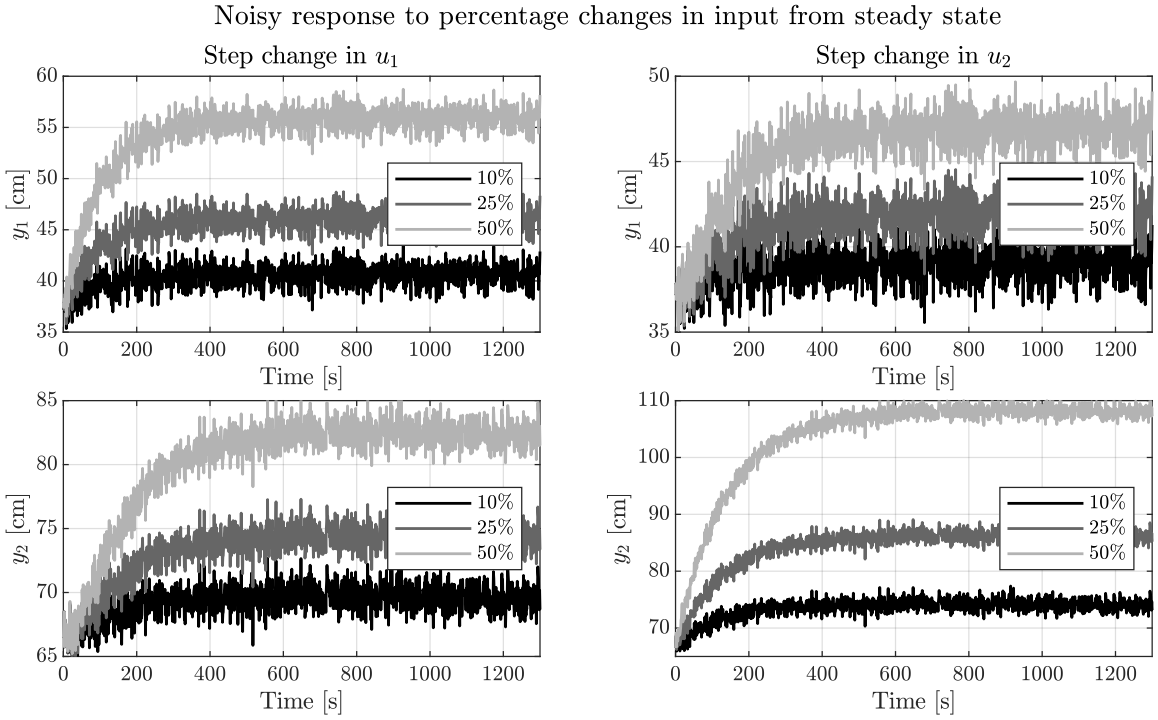

Figure 15.2: High noise step responses.

C Using mimoctf2dss.

The function takes in the transfer functions `TF` and the set sampling time `Ts`, to bring them in required form to apply the provided function *mimoctf2dss*.

```
function sysd = tf2syslin(TF,Ts);
addpath('Realization')

si = length(TF)/2;

% Allocate memory
num = cell(si,si);
den = cell(si,si);
lambda = zeros(si,si);

% Build Array
[n1,d1] = tfdata(TF(1));
[n2,d2] = tfdata(TF(2));
[n3,d3] = tfdata(TF(3));
[n4,d4] = tfdata(TF(4));

num(1,1) = n1;
den(1,1) = d1;

num(1,2) = n2;
den(1,2) = d2;

num(2,1) = n3;
den(2,1) = d3;

num(2,2) = n4;
den(2,2) = d4;

Nmax = 2*1200;
tol = 1e-8;

[Ad,Bd,Cd,Dd,sH] = mimoctf2dss(num,den,lambda,Ts,Nmax,tol);

sysd = ss(Ad,Bd,Cd,Dd,Ts);
end
```

D State estimation in the approximated system.

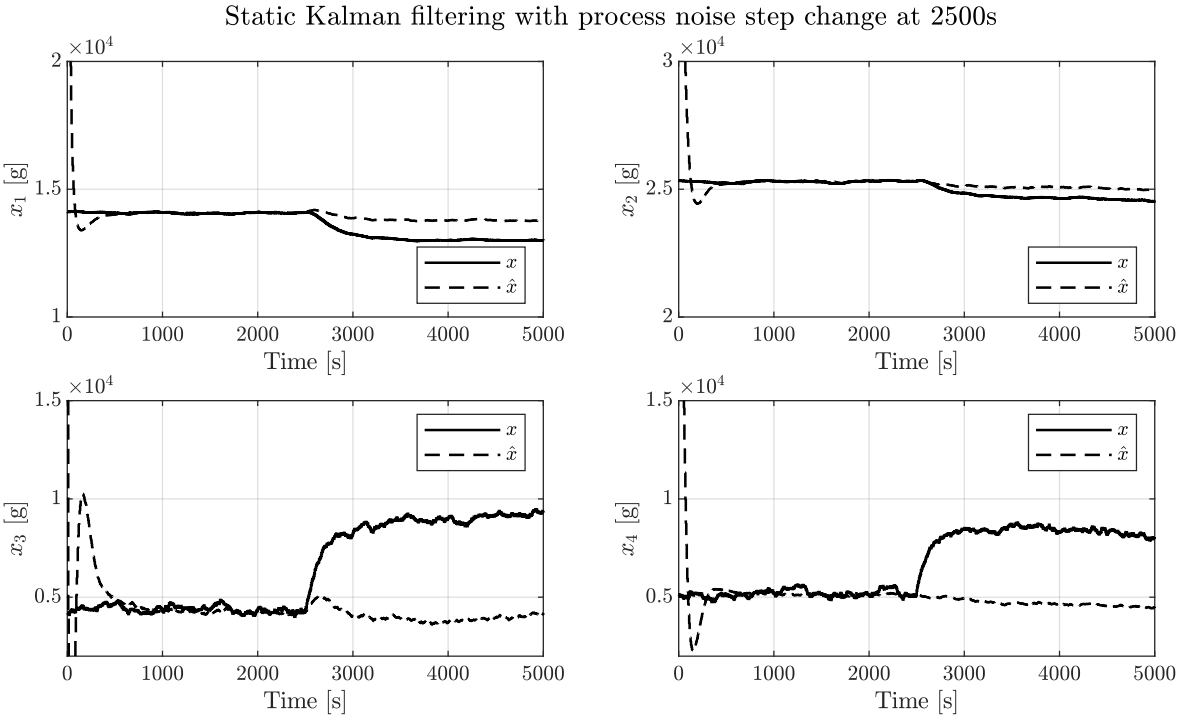

Figure 15.3: The approximated system can be estimated with the static Kalman filter, however, step changes in the process noise create a constant error between the real state and the estimate.

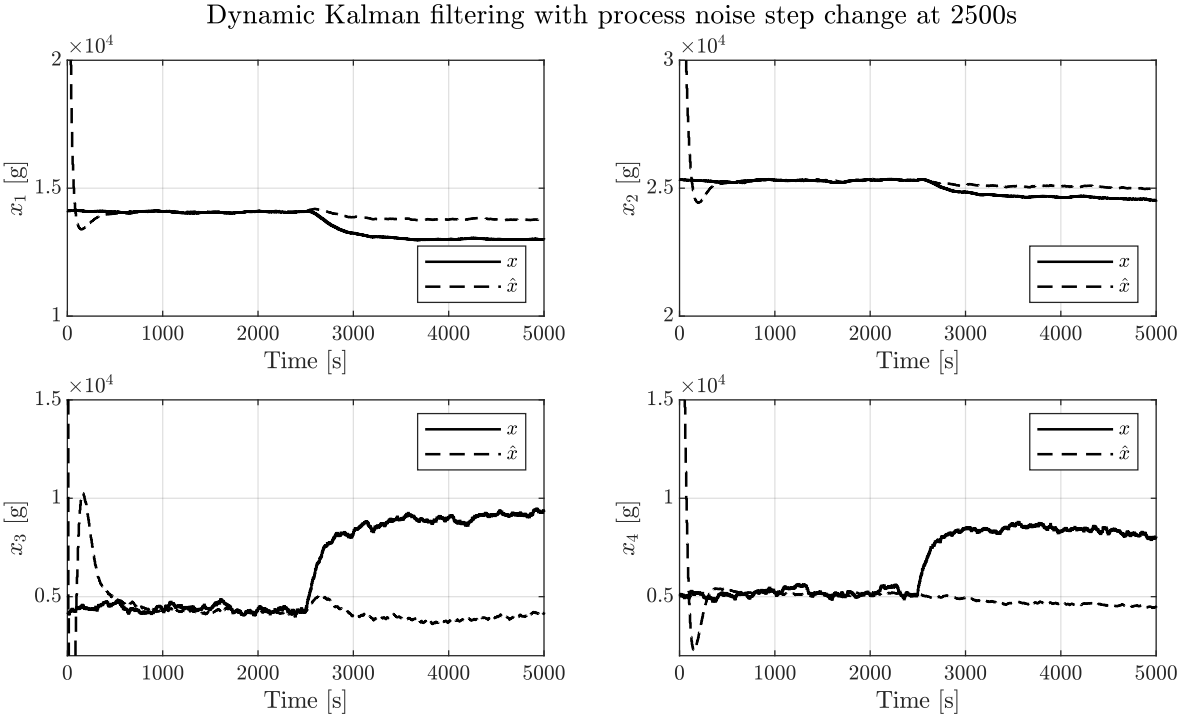

Figure 15.4: The approximated system can be estimated with the dynamic Kalman filter, however, step changes in the process noise create a constant error between the real state and the estimate.