Assignment 5: Introduction to Unconstrained MPC

Charl van de Merwe, 20804157
Optimal Control, EBO 780

May 3, 2020

1 Assignment Overview

The purpose of this assignment is to implement a simple unconstrained linear model predictive controller (MPC), without using MATLAB's MPC toolbox, for the following plant model:

$$G(s) = \begin{bmatrix} \frac{12.8}{16.7s+1}e^{-1s} & \frac{-18.9}{21s+1}e^{-3s} \\ \frac{6.6}{10.9s+1}e^{-7s} & \frac{-19.4}{14.4s+1}e^{-3s} \end{bmatrix}. \tag{1}$$

2 MPC Description

This report is structured to follow the assignment numbering, as follows:

Problem 1. Define a discrete-time linear unconstrained MPC by giving the objective function, optimization problem as well as all the variables/parameters involved.

At each time instance, the MPC algorithm computes a sequence of future manipulated variables (MV) to optimize the future plant behaviour. The first input of the optimized system is then sent into the plant. The MPC algorithm is computed at every time instance, at a period equal to the sampling frequency (T_s) [1]. MPC is a discrete time algorithm. The optimization problem is described mathematically as

$$\min_{u(k+N_C|k),\ldots,u(k|k)} V(x(k|k), u), \tag{2}$$

where u is the input to the plant, x is the plant states ($x(k|k)$ is the initial state, measured or estimated from the plant) N_C is the amount of control steps and V is the objective function to be minimized. Note that the $|k$ term indicates that the plant behaviour is predicted from the measured states at discrete time instance $|k$. As an example, $x(5|3)$ is the predicted states at instance 5, propagated from the measured states at instance 3. The objective function is

$$\begin{aligned} V(x,u) = &\sum_{i=1}^{N_P} (Y_{sp}(k) - y(k+i|k))^T Q (Y_{sp}(k) - y(k+i|k)) + \\ &\sum_{i=1}^{N_C} \Delta u(k+i|k)^T R\, \Delta u(k+i|k), \end{aligned} \tag{3}$$

subject to

$$x(k+i+1|k) = f(x(k+i|k), u(k+i|k)) \qquad \forall i = 1, ..., N_C \tag{4a}$$
$$x(k+i+1|k) = f(x(k+i|k), u(k+N_C|k)) \quad \forall i = N_C+1, ..., N_P \tag{4b}$$
$$y(k+i|k) = h(x(k+i|k), u(k+i|k)) \qquad \forall i = 1, ..., N_C \tag{4c}$$
$$y(k+i|k) = h(x(k+i|k), u(k+N_C|k)) \quad \forall i = N_C+1, ..., N_P, \tag{4d}$$

where N_P is the prediction horizon, Y_{sp} is the constant setpoint or desired output of the plant, y is the predicted output, Δu is the input (control) step size (from one instance to another) and Q and R is the output and input weighing matrices. From (4) it can be seen that there are N_C unique control steps, whereafter the last control step is maintained throughout the prediction horizon.

In this assignment, the future manipulated variables (MV) behaviour (or output behaviour) is specified using setpoints, as described by $Y_{sp}(k)$ in (3). Future MV behaviour can also be specified by using zones, reference trajectories or a funnel [1].

3 System Description in Discrete Time

Problem 2. Convert the continuous-time model to a discrete-time model with sampling time of 1 second and document the model.

The continuous model in (1) is converted to a discrete-time model using MATLAB's *c2d* function, as seen in Section 10.1, for a sampling time of $T_s = 1\,s$. The resulting model is

$$G(z) = \begin{bmatrix} \frac{0.744}{z-0.9419} z^{-1} & \frac{-0.8789}{z-0.9535} z^{-3} \\ \frac{0.5786}{z-0.9123} z^{-7} & \frac{-1.3015}{z-0.9329} z^{-3} \end{bmatrix}. \tag{5}$$

Problem 3. Convert the discrete-time model to a state-space representation and document the model.

The system's state space representation is found using MATLAB's *ss* and *getDelayModel* functions, as seen in Section 10.1. The discrete state-space system with the dead time (or delay time) incorporated is described by

$$x(k+1) = Ax(k) + B_1 u_{delayed}(k) + B_2 w(k) \tag{6a}$$
$$y(k) = C_1 x(k) + D_{11} u_{delayed}(k) + D_{12} w(k) \tag{6b}$$
$$z(k) = C_2 x(k) + D_{21} u_{delayed}(k) + D_{22} w(k) \tag{6c}$$
$$w(k) = z(k-\tau), \tag{6d}$$

with

$$u_{delayed}(k) = \begin{bmatrix} u_1(k-u_{1Delay}) \\ u_2(k-u_{2Delay}) \\ \vdots \\ u_p(k-u_{pDelay}) \end{bmatrix}, \tag{7}$$

where τ is a component that represents all of the internal delays and p is the amount of inputs. This model, described by (5), has $n = 4$ states, $p = 2$ inputs and $q = 2$ outputs.

The state-space matrices found with the aid of MATLAB are

$$A = \begin{bmatrix} 0.9419 & 0 & 0 & 0 \\ 0 & 0.9123 & 0 & 0 \\ 0 & 0 & 0.9535 & 0 \\ 0 & 0 & 0 & 0.9329 \end{bmatrix} \tag{8a}$$

$$B = \left[\begin{array}{c:c} B_1 & B_2 \end{array}\right] = \left[\begin{array}{cc:c} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{array}\right] \tag{8b}$$

$$C = \begin{bmatrix} C_1 \\ \hdashline C_2 \end{bmatrix} = \left[\begin{array}{cccc} 0.744 & 0 & -0.8789 & 0 \\ 0 & 0 & 0 & -1.3015 \\ \hdashline 0 & 0.5786 & 0 & 0 \end{array}\right] \tag{8c}$$

$$D = \left[\begin{array}{c:c} D_{11} & D_{12} \\ \hdashline D_{21} & D_{22} \end{array}\right] = \left[\begin{array}{cc:c} 0 & 0 & 0 \\ 0 & 0 & 1 \\ \hdashline 0 & 0 & 0 \end{array}\right]. \tag{8d}$$

The delay values are

$$u_{delayed}(k) = \begin{bmatrix} u_1(k-1) \\ u_2(k-3) \end{bmatrix} \tag{9a}$$

$$\tau = 6. \tag{9b}$$

Problem 4. Create a discrete-time model that outputs the states of the model, by using the identity matrix as the C-matrix and document. The time delays can be expressed as explicit states.

The discrete plant, described in (6) and (7), can be simulated by using the Simulink model as seen in Figure 1 to 3. The *Determine States* block in Figure 1 outputs the states, by setting the block parameters A and B to A and B_1, as defined in (8). C and D are set to the identity matrix and a zeroed matrix, with the same sizes as C_1 and D_{11}, as defined in (8).

The *Input Delay* block, as seen in Figure 1 and 2, implements the function in (9a). The plant output is found using the *States to Output* block, as seen in Figure 1 and 3, which implements (6b) to (6d).

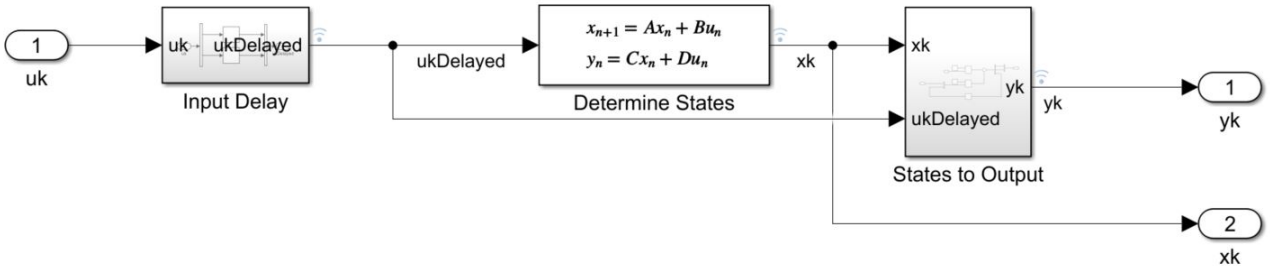

Figure 1: Simulink Discrete Plant Implementation

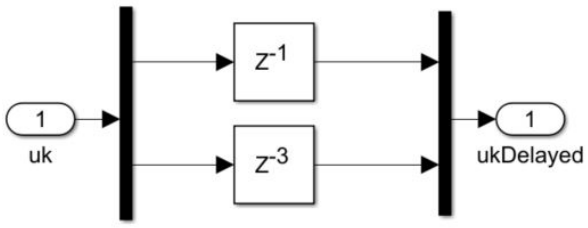

Figure 2: Simulink Input Delay Block

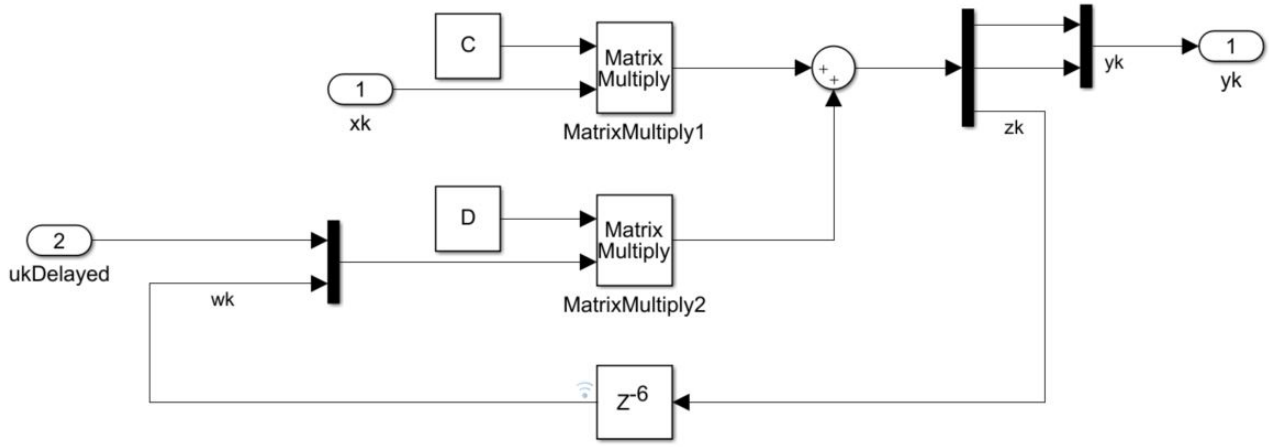

Figure 3: Simulink States to Output Block

4 Prediction and Control Horizon

Problem 5. Document the prediction horizon that will lead to 99% of steady-state for the slowest model to a unit step.

By implementing MATLAB's step function on the plant in (1), as seen in Section 10.1, a step response of each input variable to each output variable is generated, as seen in Figure 4. It is found that for the response of output 1 to a step in input 2 reaches 99% of it's steady state value (-18.9) after $99.7\,s$, which is the slowest observed response.

The prediction horizon, N_P, is chosen to cover the longest plant dynamics and the control horizon. The chosen value for the control horizon and prediction horizon is given under **Problem 6**.

Problem 6. Document the control horizon that you have chosen.

The control steps are computed with blocking, so that control steps are maintained over a few simulation samples. The control horizon is defined as

$$C = N_B \times N_C, \tag{10}$$

where N_B is the amount of samples that each control step needs to be maintained for and C is the control horizon. The prediction horizon is determined from

$$N_P = C + \frac{L}{T_s}, \tag{11}$$

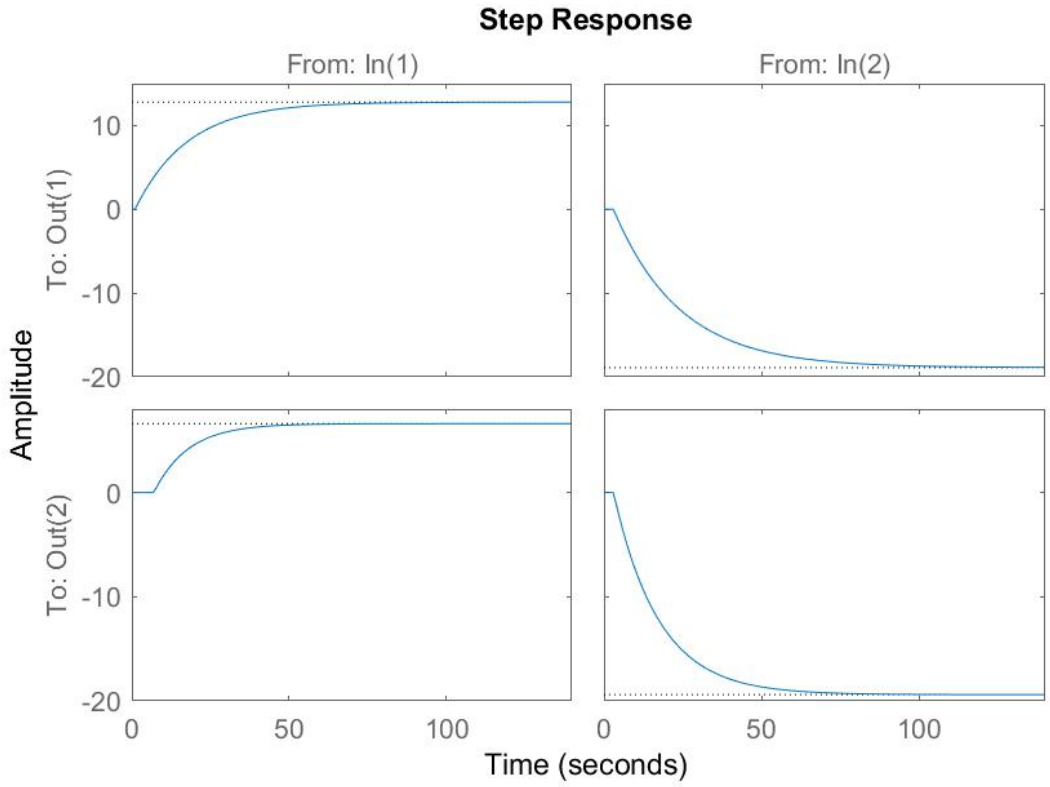

Figure 4: Plant Step Response

where L is the longest plant dynamics in seconds. After the full MPC Simulink implementation was completed, these parameters where tuned to produce the desired plant response. The chosen values are $N_C = 3$, $N_B = 6$ and $N_P = 118$ (longest plant dynamics rounded to $100\,s$ and $T_s = 1\,s$). Maintaining each of the three inputs for six samples allows each input to be maintained almost as long as the longest plant delay (seven samples, as seen in (5)).

Note that these parameters are only relevant to the prediction simulation. In each time instance of the physical system, the MPC algorithm is repeated to compute the optimal control input. The control input sent into the plant is not maintained as it is done in the prediction simulation (blocking implementation).

5 MPC Algorithm Implementation

Problem 7. Code the Matlab function *ObjFunc* that calculates the objective value that will be optimized by *fminunc*. Your MPC controller must implement blocking to allow for a smooth closed loop MV trajectory.

Problem 8. Show the code to simulate the model predictions in your objective function using the A, B, C and D matrices of the discrete time state-space model explicitly and not *lsim*.

This section answers both **Problem 7** and **8**, as they are difficult to explain separately.

5.1 MPC Algorithm Description

As mentioned in Section 2, the MPC algorithm computes the optimal input steps to produce the desired future output (or manipulated variables). The MPC algorithm can be summarized in the following steps:

1. Guess the optimal future inputs.
2. Predict the future output based on the guessed inputs, from (6) and (7).
3. Compute the quadratic penalty (cost) due to the difference of the output to the setpoints and the difference between input steps, as described by the objective function defined in (3).
4. Repeat step 1 to 3 to find the optimal input steps.

These steps are followed within MATLAB's *fminunc* function. This function will numerically compute the optimal input steps to minimize the objective function.

The implementation of the MPC algorithm can be seen in Section 10.3. The computation of the objective function can be seen in Section 10.4. The implementation of the output prediction can be seen in Section 10.5.

5.2 Testing the MPC Algorithm

A script to test the MPC algorithm, for the plant in (1) starting from rest with a [5,-5] setpoint, can be seen in Section 10.6. The result of this test can be seen in Figure 5.

The *fminunc* function computes the optimal input steps for this test script to be

$$\begin{aligned} U^* &= \begin{bmatrix} u_1^*(k|k) & u_1^*(k+1|k) & u_1^*(k+3|k) \\ u_2^*(k|k) & u_2^*(k+1|k) & u_2^*(k+3|k) \end{bmatrix} \\ &= \begin{bmatrix} 2.6358 & 1.7465 & 1.5409 \\ 1.2987 & 1.1848 & 0.7853 \end{bmatrix}. \end{aligned} \tag{12}$$

Note that the first two input steps are maintained for six samples each, as specified by the blocking parameter $N_B = 6$, and the last input step is maintained for the full duration of the prediction horizon. The plot is limited to 60 samples, but the prediction horizon is $N_P = 118$, as specified.

5.3 Delayed Input Implementation

From the prediction algorithm in (6) and (9a) it is seen that the past input steps are needed. A circular buffer is used to store the past implemented inputs, as follows:

$$U_{stored} = \begin{bmatrix} u_1^*(k|k) & u_1^*(k-1|k) & u_1^*(k-2|k) & u_1^*(k-3|k) \\ u_2^*(k|k) & u_2^*(k-1|k) & u_2^*(k-2|k) & u_2^*(k-3|k) \end{bmatrix}, \tag{13}$$

from which $u_{delayed}(k)$, as seen in (9a), can be determined from:

$$u_{delayed}(k) = \begin{bmatrix} U_{stored}(row\,1, column\,2) \\ U_{stored}(row\,2, column\,4) \end{bmatrix}. \tag{14}$$

Before the next iteration of the prediction algorithm, the buffer is shifted by one sample (or one column to the right). At the start of the next iteration, the first column of the buffer is filled with the input step of that iteration. A buffer created and shifted similarly for the $z(k)$ parameter in (6). These implementations can be seen in Section 10.5.

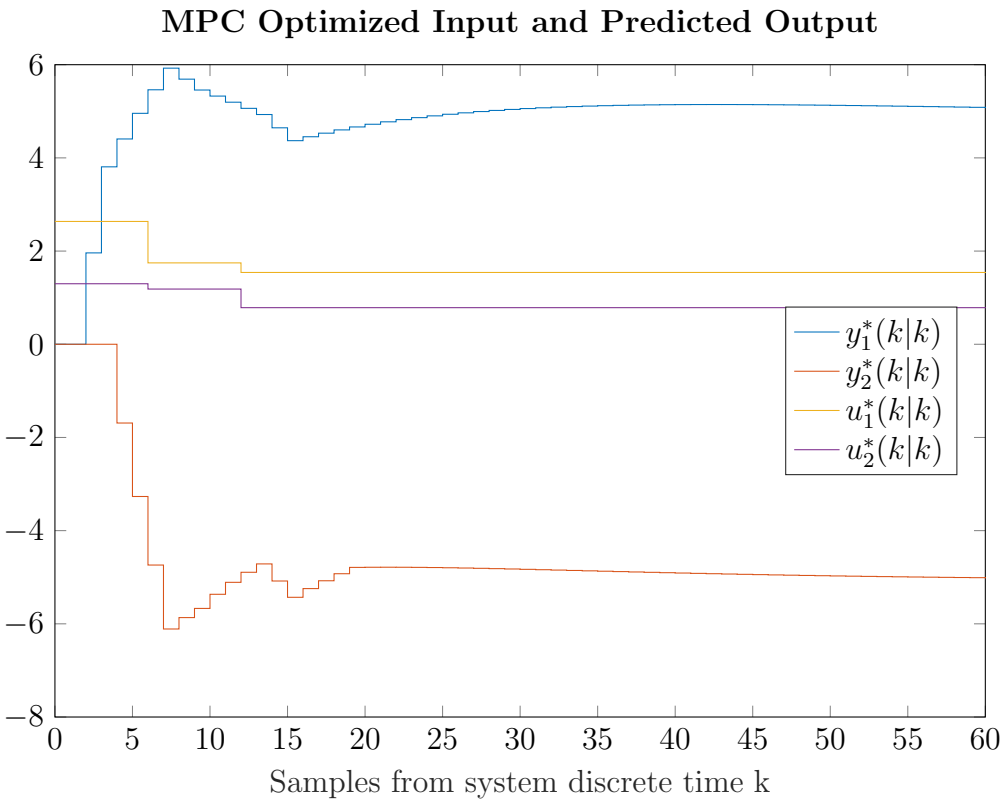

Figure 5: MPC Test Result

6 Simulink Model Implementation

Problem 9. Code a MATLAB Simulink s-function m-file to call *fminunc* on your function ObjFunc and connect it to the original continuous-time model G(s).

6.1 Simulink Model Description

A Simulink model was created to model MPC control of the plant, as can be seen in Figure 6. The continuous plant block is an implementation of (1), as can be seen in Figure 7. The MPC algorithm reads the states of the plant at each step. The states are determined from the discrete plant block, which is a discrete state space implementation of the plant, as seen in Figure 1. The MPC algorithm is implemented within the MPC block.

The sampling time for the discrete plant and for the MPC block is 1 second. The Simulink *solver options* are therefore set to use *Runge-Kutta* with a *fixed-step size* of 1 second.

6.2 MPC Block Coding

The MPC algorithm is run in the Simulink model by using a level-2 MATLAB s-function block, as seen in Figure 6. This block allows the use of MATLAB language to create a custom block. A template for creating a m-file to define the workings of the block can be invoked by typing *edit msfuntmpl_basic* in the *Command Window*. The definition of the MPC block can be seen in Section 10.2.

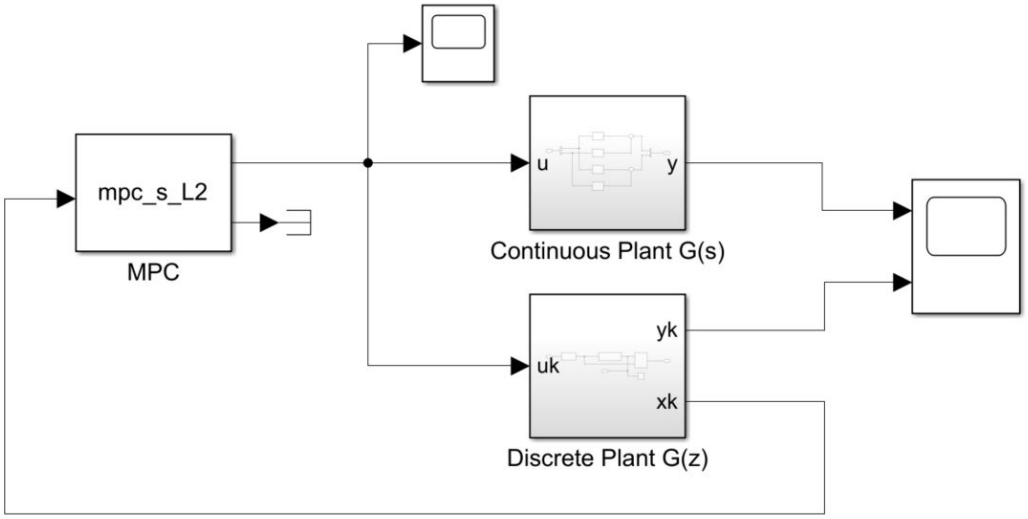

Figure 6: Full Simulink Model

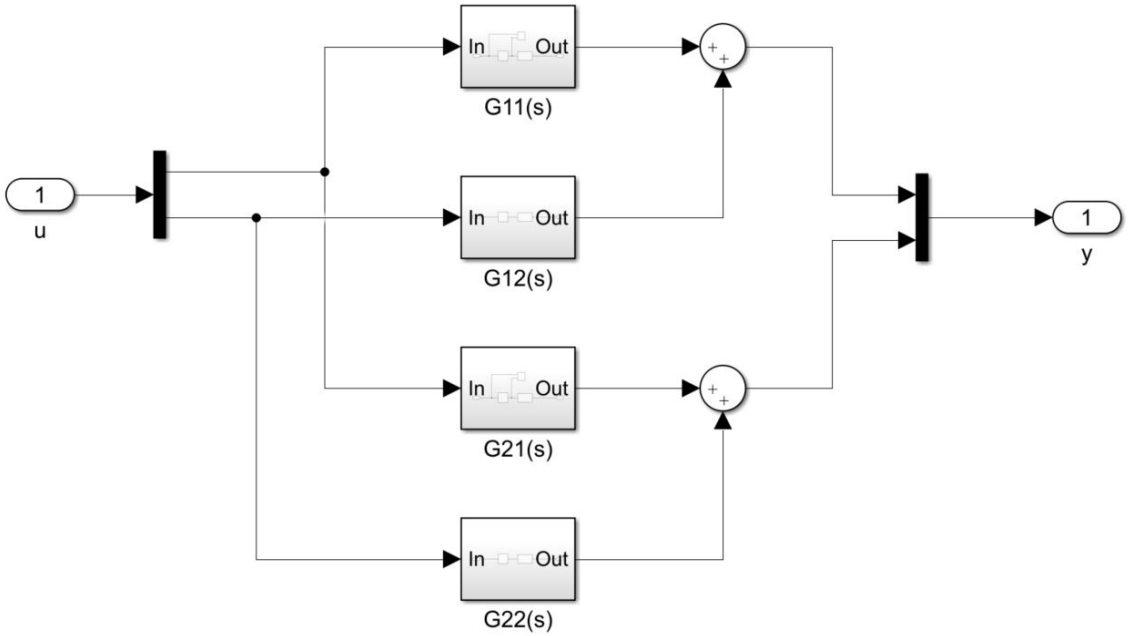

Figure 7: Simulink Continuous Plant Block

6.2.1 Block Setup

At the start of the Simulation, the *SetupParameters* function (Section 10.1) is called to create and define the discrete model in (5), the setpoint vector array, the control & prediction horizon, weighing matrices and the sample time. This is done by setting the Simulink *InitFcn* callback to the *SetupParameters* function. These parameters are passed to the MPC block as dialog parameters.

The basic characteristics of the MPC block, such as the amount of inputs and outputs, are defined in the *setup* function, as seen in Section 10.2. Storage of data within the block is done in the block's *Dwork* vectors. The size and amount of *Dwork* vectors have to be defined in the *DoPostPropSetup* function.

At the start of the Simulink simulation, the MPC block's *start* function is called. In

this function, the state space parameters in (8) and (9) are determined, flattened and stored in the *Dwork* vectors.

At the start of the prediction simulation, the values for previously implemented parameters $u(k)$ and $z(k)$ are needed, as can be seen from (6d) and (9a). The required storage sizes are set in the *DoPostPropSetup* function and the vectors are initialized to zero at the start of the simulation in the *start* function.

6.2.2 Code Execution During Simulation

During each Simulink simulation step, the *Outputs* function is called in the MPC block, as seen in Section 10.2. This function reads the current state from the Simulink environment and determines the output setpoint based on the current simulation time. These parameters, as well as the plant model parameters, system parameters and stored parameters are passed to the *mpc* function. The *mpc* function, as seen in Section 10.3, then calculates the optimal control steps to deliver the desired future response, as was described in Section 5.

The difference between the MPC implementation during simulation and for the test function (as in Section 10.6, result in Figure 5), is that the current states and previously implemented parameters $u(k)$ and $z(k)$, as can be seen from (6d) and (9a), are not zero. For the full simulation, the output setpoint changes from [5,-5] to [-5,5] at simulation discrete time of 100 (step 101), as described in **Problem 10**. The optimal future input and output calculated by the MPC algorithm at this step can be seen in Figure 8.

At the start of the prediction algorithm for the Simulink time of 100 (step 101), the input buffer in (13) must have the previously implemented simulation step inputs stored, as follows:

$$U_{stored} = \begin{bmatrix} u_1^*(100|100) & u_1^*(99|100) & u_1^*(98|100) & u_1^*(97|100) \\ u_1^*(100|100) & u_1^*(99|100) & u_1^*(98|100) & u_1^*(97|100) \end{bmatrix}. \tag{15}$$

This is done by saving the current input (to be implemented for the current Simulink step) in the first column of the buffer after the MPC calculation is complete. At the start of the next Simulink step, the buffer is shifted one column to the right. Similar logic is followed with the storage of $z(k)$. This implementation can be seen in Section 10.2.

To increase the speed with which *fminunc* finds the optimal input steps, as described by (12), the optimal input steps found by the previous Simulink simulation step is used as a first guess for the optimal inputs of the current simulation step, as seen in Section 10.2 and 10.3.

At the end of the *Outputs* function, the block output is set to the plant input to implement in the current Simulink step.

6.3 Testing the Plant Model Implementations

The implementation of the model of the physical system needs to be very accurate for MPC to be effective. A mistake in the model used in the prediction algorithm, as seen in Section 10.5, would be costly. The output of the discrete plant block is compared to the continuous plant block to verify that (6) to (9) produces the desired response, as seen in Figure 6. Both the continuous and discrete plant outputs are plotted on the simulation result plot in Figure 9 in Section 7.3.

To test the implementation of (6) to (9) within the MPC block, the second output of the block was set to $y(k|k)$, as seen in Section 10.2. It was found that this output

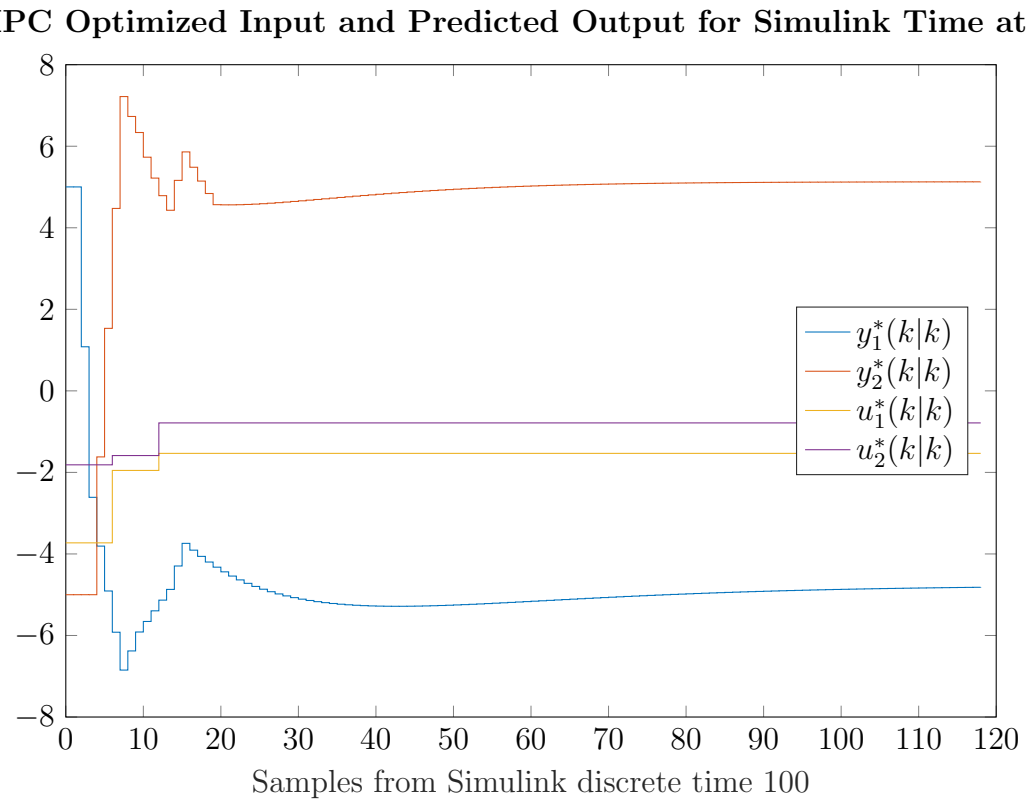

Figure 8: MPC Prediction During Simulation at Simulation Step 101

produced the exact same result as the discrete plant block output, confirming that the MPC implementation of the plant, as well as storages of past variables, was implemented correctly. The second output was not used thereafter, as seen in Figure 6, because it does not contribute any new information to the simulation.

7 Simulation and Results

Problem 10. Run the Simulink model for 200 seconds and document the plot for the controlled variables and the manipulated variables. Start with the initial condition of all the states at 0 and then make a setpoint changes from [0;0] to [5;-5] at time 20 seconds and then to [-5;5] at time 100 seconds.

7.1 Setpoint

The output setpoints described in **Problem 10** are stored in a vector array, Y_{sp}, as seen in Section 10.1. The setpoint relevant to the simulation time is extracted by the MPC block. To do so, the MPC block keeps track of the simulation time, as seen in the *Update* function in Section 10.2.

7.2 Tuning Q and R

The weighing matrices in the objective function (3) are diagonal matrices defined by

$$Q = \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \tag{16}$$

$$R = \begin{bmatrix} R_{11} & 0 \\ 0 & R_{22} \end{bmatrix}. \tag{17}$$

If Q_{22} is larger than Q_{11}, then a deviation in y_2 from the setpoint will be penalized greater by the objective function, as seen in (3), than for a deviation in y_1. The assignment does not specify a preference in the performance of either output, but it is chosen to favour y_2 slightly, due to it's slower response. The final values of $Q_{11} = 1$ and $Q_{22} = 1.4$ are the result of tuning (testing various values).

By testing various values for R relative to Q, it was found that choosing R_{11} so that a step change in u_1 of one ($\Delta u_1 = 1$) contributes the same amount to the objective function as a 2% deviation in the steady state of $y_{1ss} = 5$, delivers a smooth plant response. This relationship can be expressed mathematically as

$$\Delta u_1 \times R_{11} \times \Delta u_1 = (2\%\, y_{1ss}) \times Q_{11} \times (2\%\, y_{1ss}), \tag{18}$$

from which $R_{11} = 0.01$ is found.

If R_{22} is larger than R_{11}, the MPC control will be biased to move the system more with u_1. From the step response in Figure 4 (or from the plant in (1)), it is seen that the contribution of u_2 to the end state of the plant is approximately double that of u_1:

$$\frac{18.9 + 19.4}{12.8 + 6.6} = 1.97 \approx 2. \tag{19}$$

R_{22} is therefore chosen so that a step of $u_2 = 1$ contributes roughly the same to the objective function as a step of $u_1 = 2$. This relationship can be expressed mathematically as

$$\Delta u_2 \times R_{22} \times \Delta u_2 = \Delta u_1 \times R_{11} \times \Delta u_1, \tag{20}$$

from which $R_{22} = 0.04$ is found.

The chosen matrices are defined in the *SetupParameters* function, as seen in Section 10.1.

7.3 Results

By running the Simulink model, as seen in Figure 6, for 200 seconds, the output and input results are found, as seen in Figure 9 and 10. The plots are generated at the end of the simulation by calling the *PlotResults* script, as seen in Section 10.7, from Simulink's *StopFcn* callback function.

From Figure 9, it can be concluded that the plant is controlled effectively. Note that y_2 only responds to a change in u_1 after seven seconds, due to the dead-time, as seen in (1). This dead-time causes sudden changes in the output response of y_2 once large changes in input u_1 takes effect (after the dead-time). Even with these large delays, the MPC controller is able to control the plant's output response effectively.

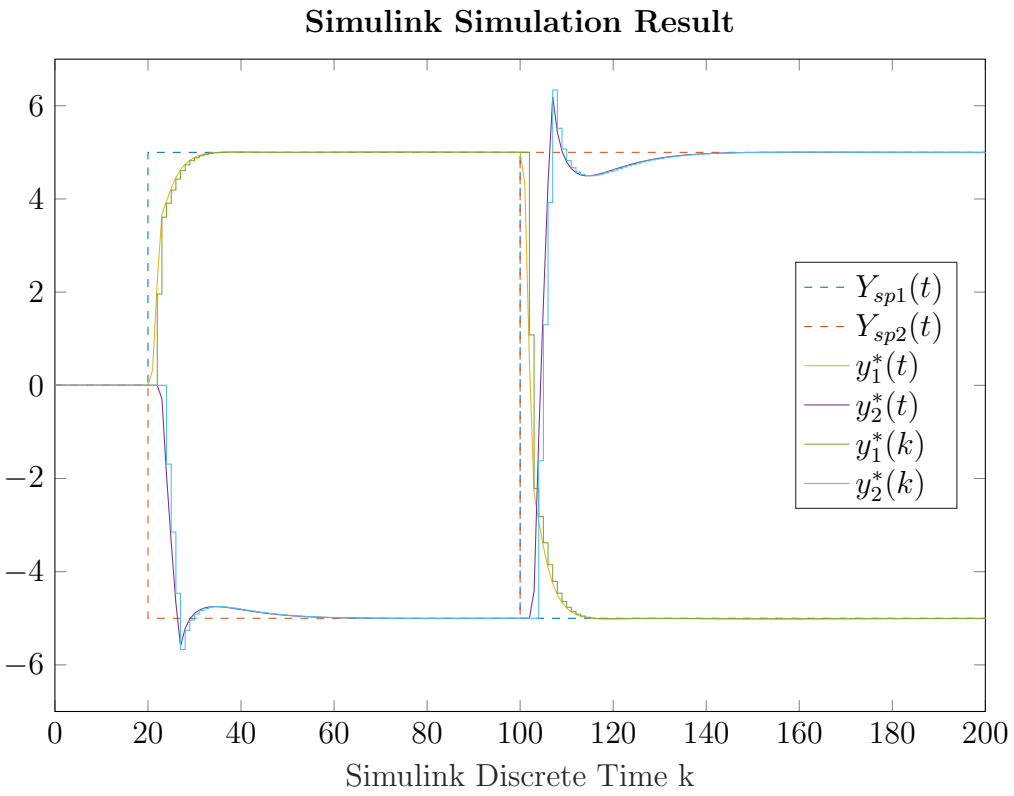

Figure 9: Simulation Result: Outputs

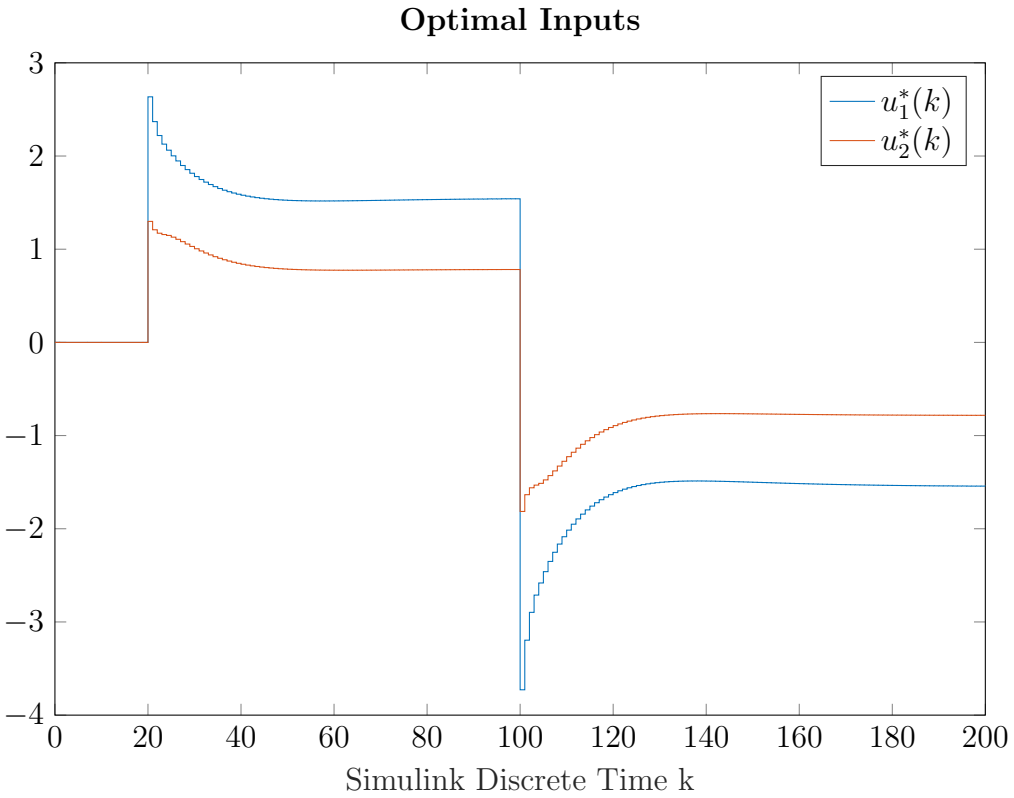

Figure 10: Simulation Result: Inputs

8 Other

Problem 11. Attach your MATLAB code to the end of the report.

All of the relevant code can bee seen in the Appendix (Section 10).

Problem 12. Assume you have setup a MPC controller for a 2-by-2 system with $N_P = 20$ and $N_C = 3$. Determine what Q_1 and Q_2 should be in order for a 1% deviation of y_1 for the steady-state of 150 to contribute 5 times as much as a 7% deviation of y_2 from its steady-state value of 320 to the objective function. Further, determine what R_1 and R_2 should be that a 10% change relative to the range of u_1 of 0 - 400 contribute the same to the objective function as a 20% change relative to the range of u_2 of 25 - 68. Lastly, the contribution of y_1 and y_2 should be 60 times larger for the stated deviations than the contributions of u_1 and u_2, after the objective function was corrected for prediction and control horizon. Document your calculations in your report.

The mathematical relationship that will ensure that a 1% deviation of y_1 from the steady-state of 150 will contribute 5 times as much as a 7% deviation of y_2 from its steady-state value of 320 to the objective function, is

$$(1\% \cdot 150)^2 Q_1 = 5(7\% \cdot 320)^2 Q_2. \tag{21}$$

The mathematical relationship that will ensure that a 10% change relative to the range of u_1 of 0 - 400 contribute the same to the objective function as a 20% change relative to the range of u_2 of 25 - 68, is

$$(10\% \cdot 400)^2 R_1 = (20\% \cdot 43)^2 R_2. \tag{22}$$

The mathematical relationship that will ensure that the contribution of y_1 and y_2 is 60 times larger for the stated deviations than the contributions of u_1 and u_2 (without scaling for N_P and N_C), is

$$(1\% \cdot 150)^2 Q_1 + 5(7\% \cdot 320)^2 Q_2 = 60[(10\% \cdot 400)^2 R_1 + (20\% \cdot 43)^2 R_2]. \tag{23}$$

These equations are rewritten as

$$0 = 2.25\, Q_1 - 2508.8\, Q_2 \tag{24}$$
$$0 = 1600\, R_1 - 73.96\, R_2 \tag{25}$$
$$0 = 2.25\, Q_1 + 2508.8\, Q_2 - 96000\, R_1 - 4437.6\, R_2 \tag{26}$$

If $Q_2 = 1$ is chosen, $Q_1 = 1115$, $R_1 = 0.0261$ and $R_2 = 0.5654$ are found from (24) to (26). Scaling R to take into account the difference in N_P and N_C can be done by

$$R_{scaled} = \frac{N_P}{N_C} R \tag{27}$$

Implementing the scaling produces the following final values for Q and R:

$$Q = \begin{bmatrix} 1115 & 0 \\ 0 & 1 \end{bmatrix}$$
$$R = \begin{bmatrix} 0.1742 & 0 \\ 0 & 3.769 \end{bmatrix}.$$

9 References

[1] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, 2003.

10 Appendix: MATLAB Code

10.1 SetupParameters Script

```
% Setup script for  Assignment 5
% This script is called automatically at the start of the
% Simulink MPC simulation, to initialize the plant and
% control parameters. This script can also be run
% independantly to inspect these parameters and to see the
% step response of the plant.

clear;
addpath('matlab2tikz\src'); % library that enables saving
   data for Latex pgf figures
savePlotData = true;

%% setup system variables
Ts = 1;      % sampling time
Nb = 6;      % blocking samples
Nc = 3;      % control moves
C = Nc*Nb;  % control horizon in terms of samples (not time)
Np = 100 + C;     % prediction horizon
Q = [1 0; 0 1.4]; % output weighing matrix
R = [0.01 0; 0 0.04]; % input weighing matrix
n = 4;       % amount of states
p = 2;       % amount of inputs
q = 2;       % amount of outputs

% C and D to output states
CforStateOutput = eye(4);
DforStateOutput = zeros(n, p);

%% Create setpoint array. Note that time t = 0 is at index 1
Ysp = zeros(2,201);
Ysp(:,20+1:99+1) = [5;-5]*ones(1,80);
Ysp(:,100+1:200+1) = [-5;5]*ones(1,101);

% % to test
% Ysp = [5;-5]*ones(1,201);

%% Create plant model
G11 = tf(12.8,[16.7 1],'IODelay', 1);
G12 = tf(-18.9,[21 1],'IODelay', 3);
```

```
G21 = tf(6.6,[10.9 1],'IODelay', 7);
G22 = tf(-19.4,[14.4 1],'IODelay', 3);
G = [G11 G12; G21 G22];
Gd = c2d(G,Ts); % convert to discrete model

plantC = ss(G);
plantD = ss(Gd);    % discrete state-space

% decompose state-space system
[H,tau] = getDelayModel(plantD);
[A,B1,B2,C1,C2,D11,D12,D21,D22,E,tau]=getDelayModel(plantD);

%% Step Response
figure(4); % generates Figure 4 in the report
step(G)

if savePlotData == true
    set(gcf,'Position',[200 200 600 400])
    saveas(gcf,[pwd '\Figures\StepResponse.jpg']);
    %matlab2tikz('Figures\StepResponse.tex');
end
```

10.2 mpc_s_L2 Function (MATLAB Level-2 S-Function)

```
function mpc_s_L2(block)
% s functoin implementation of Model Predictive Control. In
% each iteration, the block receives the current measurements
% of the output. The block calculates the optimal control
% move, which it outputs.
% Dialog input parameters:
% - Gd      discrete plant model
% - Ysp     output setpoints over time
% - Np      prediction horizon
% - Nb      blocking samples
% - Nc      control moves
% - Q       output weighing matrix
% - R       input weighing matrix
% - Ts      sampling time

%% Main body: Call setup function. No other calls should be
%% added to the main body.
setup(block);

%endfunction

%% Set up the basic characteristics of the S-function block
function setup(block)
```

```
% Register number of ports
block.NumInputPorts  = 1;
block.NumOutputPorts = 2;

% Setup port properties to be inherited or dynamic
block.SetPreCompInpPortInfoToDynamic;
block.SetPreCompOutPortInfoToDynamic;

% Override input port properties
block.InputPort(1).Dimensions  = 4;
block.InputPort(1).DatatypeID  = 0;  % double
block.InputPort(1).Complexity  = 'Real';
block.InputPort(1).DirectFeedthrough = true;

% Override output port properties
block.OutputPort(1).Dimensions  = 2;
block.OutputPort(1).DatatypeID  = 0; % double
block.OutputPort(1).Complexity  = 'Real';

block.OutputPort(2).Dimensions  = 2;
block.OutputPort(2).DatatypeID  = 0; % double
block.OutputPort(2).Complexity  = 'Real';

% Register parameters
block.NumDialogPrms     = 8;

% Register sample times
Ts = block.DialogPrm(8).Data;
block.SampleTimes = [Ts 0];  % sample time of 1 s

% Specify the block simStateCompliance.
%    'DefaultSimState', < Same sim state as a built-in block
block.SimStateCompliance = 'DefaultSimState';

% Register all relevant methods
block.RegBlockMethod(...
    'PostPropagationSetup', @DoPostPropSetup);
block.RegBlockMethod('Start', @Start);
block.RegBlockMethod('Outputs', @Outputs);
block.RegBlockMethod('Update', @Update);
block.RegBlockMethod('Terminate', @Terminate);
block.RegBlockMethod(...
    'SetInputPortSamplingMode', @SetInputPortSamplingMode);

%end setup

%% Setup work areas and state variables
function DoPostPropSetup(block)
```

```
block.NumDworks = 15;

%% Iteration counter vector
  block.Dwork(1).Name            = 'SimIteration';
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;      % double
  block.Dwork(1).Complexity      = 'Real'; % real
  block.Dwork(1).UsedAsDiscState = true;

%% Plant model storage
Gd = block.DialogPrm(1).Data;
plantD = ss(Gd);    % discrete plant
[A,B1,B2,C1,C2,D11,D12,D21,D22,E,tau]=getDelayModel(plantD);

  Asize = size(A);
  block.Dwork(2).Name            = 'A';
  block.Dwork(2).Dimensions      = Asize(1) * Asize(2);
  block.Dwork(2).DatatypeID      = 0;      % double
  block.Dwork(2).Complexity      = 'Real'; % real
  block.Dwork(2).UsedAsDiscState = true;

  B1size = size(B1);
  block.Dwork(3).Name            = 'B1';
  block.Dwork(3).Dimensions      = B1size(1) * B1size(2);
  block.Dwork(3).DatatypeID      = 0;      % double
  block.Dwork(3).Complexity      = 'Real'; % real
  block.Dwork(3).UsedAsDiscState = true;

  B2size = size(B2);
  block.Dwork(4).Name            = 'B2';
  block.Dwork(4).Dimensions      = B2size(1) * B2size(2);
  block.Dwork(4).DatatypeID      = 0;      % double
  block.Dwork(4).Complexity      = 'Real'; % real
  block.Dwork(4).UsedAsDiscState = true;

  C1size = size(C1);
  block.Dwork(5).Name            = 'C1';
  block.Dwork(5).Dimensions      = C1size(1) * C1size(2);
  block.Dwork(5).DatatypeID      = 0;      % double
  block.Dwork(5).Complexity      = 'Real'; % real
  block.Dwork(5).UsedAsDiscState = true;

  C2size = size(C2);
  block.Dwork(6).Name            = 'C2';
  block.Dwork(6).Dimensions      = C2size(1) * C2size(2);
  block.Dwork(6).DatatypeID      = 0;      % double
  block.Dwork(6).Complexity      = 'Real'; % real
  block.Dwork(6).UsedAsDiscState = true;
```

```
  D11size = size(D11);
  block.Dwork(7).Name            = 'D11';
  block.Dwork(7).Dimensions      = D11size(1) * D11size(2);
  block.Dwork(7).DatatypeID      = 0;      % double
  block.Dwork(7).Complexity      = 'Real'; % real
  block.Dwork(7).UsedAsDiscState = true;

  D12size = size(D12);
  block.Dwork(8).Name            = 'D12';
  block.Dwork(8).Dimensions      = D12size(1) * D12size(2);
  block.Dwork(8).DatatypeID      = 0;      % double
  block.Dwork(8).Complexity      = 'Real'; % real
  block.Dwork(8).UsedAsDiscState = true;

  D21size = size(D21);
  block.Dwork(9).Name            = 'D21';
  block.Dwork(9).Dimensions      = D21size(1) * D21size(2);
  block.Dwork(9).DatatypeID      = 0;      % double
  block.Dwork(9).Complexity      = 'Real'; % real
  block.Dwork(9).UsedAsDiscState = true;

  D22size = size(D22);
  block.Dwork(10).Name            = 'D22';
  block.Dwork(10).Dimensions      = D22size(1) * D22size(2);
  block.Dwork(10).DatatypeID      = 0;      % double
  block.Dwork(10).Complexity      = 'Real'; % real
  block.Dwork(10).UsedAsDiscState = true;

  block.Dwork(11).Name            = 'tau';
  block.Dwork(11).Dimensions      = 1;
  block.Dwork(11).DatatypeID      = 0;      % double
  block.Dwork(11).Complexity      = 'Real'; % real
  block.Dwork(11).UsedAsDiscState = true;

inputDelay = plantD.inputDelay;

  inputDelaysize = size(inputDelay);
  block.Dwork(12).Name            = 'inputDelay';
  block.Dwork(12).Dimensions      = ...
      inputDelaysize(1) * inputDelaysize(2);
  block.Dwork(12).DatatypeID      = 0;      % double
  block.Dwork(12).Complexity      = 'Real'; % real
  block.Dwork(12).UsedAsDiscState = true;

%% Dynamic storage vectors
  block.Dwork(13).Name            = 'uPrevious';
  p = size(B1,2); % amount of inputs
```

```
  Dimension13 = p*block.DialogPrm(5).Data; % number of inputs
        % timesprediction samples
  block.Dwork(13).Dimensions      = Dimension13;
  block.Dwork(13).DatatypeID      = 0;      % double
  block.Dwork(13).Complexity      = 'Real'; % real
  block.Dwork(13).UsedAsDiscState = true;

  block.Dwork(14).Name            = 'uStored';
  Dimension14 = p*(max(inputDelay)+1);
  block.Dwork(14).Dimensions      = Dimension14;
  block.Dwork(14).DatatypeID      = 0;      % double
  block.Dwork(14).Complexity      = 'Real'; % real
  block.Dwork(14).UsedAsDiscState = true;

  block.Dwork(15).Name            = 'zk';
  block.Dwork(15).Dimensions      = tau+1;
  block.Dwork(15).DatatypeID      = 0;      % double
  block.Dwork(15).Complexity      = 'Real'; % real
  block.Dwork(15).UsedAsDiscState = true;

% end DoPostPropSetup

%% Called at start of model execution to initialize states
function Start(block)

block.Dwork(1).Data = 1; % start simIter at 1

% determine the discrete state space plant parameters
Gd = block.DialogPrm(1).Data;
plantD = ss(Gd);    % discrete plant
[A,B1,B2,C1,C2,D11,D12,D21,D22,E,tau]=getDelayModel(plantD);

% store flattened matrixes and vectors (plant parameters)
block.Dwork(2).Data = reshape(A,[],1);
block.Dwork(3).Data = reshape(B1,[],1);
block.Dwork(4).Data = reshape(B2,[],1);
block.Dwork(5).Data = reshape(C1,[],1);
block.Dwork(6).Data = reshape(C2,[],1);
block.Dwork(7).Data = reshape(D11,[],1);
block.Dwork(8).Data = reshape(D12,[],1);
block.Dwork(9).Data = reshape(D21,[],1);
block.Dwork(10).Data = reshape(D22,[],1);
block.Dwork(11).Data = tau;
inputDelay = plantD.inputDelay;
block.Dwork(12).Data = reshape(inputDelay,[],1);

% initialize dynamic storage vectors
p = size(B1,2); % amount of inputs
```

```
Nc = block.DialogPrm(5).Data;
uGuess = ones(p,Nc);
uGuess = reshape(uGuess,[],1); % flatten matrix
block.Dwork(13).Data = uGuess; % first guess for control

uStored = zeros(p,max(inputDelay)+1);
block.Dwork(14).Data = reshape(uStored,[],1);

zk = zeros(1,tau+1);
block.Dwork(15).Data = zk;

%end Start

%% Called to generate block outputs in simulation step
function Outputs(block)

tic

%% Reshape stored parameters
Aflat = block.Dwork(2).Data;
n = sqrt(length(Aflat));  % amount of states
A = reshape(Aflat,n,n);

B1flat = block.Dwork(3).Data;
B1 = reshape(B1flat,n,[]);
p = size(B1,2); % amount of inputs

B2flat = block.Dwork(4).Data;
B2 = reshape(B2flat,n,1);

C1flat = block.Dwork(5).Data;
C1 = reshape(C1flat,[],n);
q = size(C1,1); % amount of outputs

C2flat = block.Dwork(6).Data;
C2 = reshape(C2flat,1,n);

D11flat = block.Dwork(7).Data;
D11 = reshape(D11flat,q,p);

D12flat = block.Dwork(8).Data;
D12 = reshape(D12flat,q,1);

D21flat = block.Dwork(9).Data;
D21 = reshape(D21flat,1,p);

D22 = block.Dwork(10).Data;
```

```
tau = block.Dwork(11).Data;

inputDelayFlat = block.Dwork(12).Data;
inputDelay = reshape(inputDelayFlat,p,1);

%% Create data structures to pass to functions
delayMod.A = A;
delayMod.B1 = B1;
delayMod.B2 = B2;
delayMod.C1 = C1;
delayMod.C2 = C2;
delayMod.D11 = D11;
delayMod.D12 = D12;
delayMod.D21 = D21;
delayMod.D22 = D22;
delayMod.tau = tau;
delayMod.inputDelay = inputDelay;
delayMod.n = n;
delayMod.p = p;
delayMod.q = q;

% Setpoint
Ysp = block.DialogPrm(2).Data;
simIter = block.Dwork(1).Data;
delayMod.Ysp = Ysp(:,simIter); % update the setpoint based on
    simulation time

sysVar.Np = block.DialogPrm(3).Data;
sysVar.Nb = block.DialogPrm(4).Data;
sysVar.Nc = block.DialogPrm(5).Data;
sysVar.Q = block.DialogPrm(6).Data;
sysVar.R = block.DialogPrm(7).Data;
sysVar.Ts = block.DialogPrm(8).Data;
sysVar.C = sysVar.Nb*sysVar.Nc;

%% Read current state
xk = block.InputPort(1).Data;

%% Reshape stored data and create a structure
uPrevious = block.Dwork(13).Data; % take the last control
        % implemented as the first guess of the new step
uPrevious = reshape(uPrevious,p,[]);
uStoredFlat = block.Dwork(14).Data;
uStored = reshape(uStoredFlat,p,max(inputDelay)+1);
uStored = circshift(uStored,1,2);
zk = block.Dwork(15).Data;
zk = circshift(zk,1);
```

```
storedData.uPrevious = uPrevious;
storedData.uStored = uStored;
storedData.zk = zk;

%% Execute MPC algorithm to find optimal control
uOptimal = mpc(delayMod, sysVar, xk, storedData)

%% Store relevant data
block.Dwork(13).Data = reshape(uOptimal, [], 1);
uStored(:,1) = uOptimal(:,1);
block.Dwork(14).Data = reshape(uStored, [], 1);

% execute prediction once more to extract z1 and for plots
% for debugging
[yk, uk, z1] = ...
    prediction(uOptimal, delayMod, sysVar, xk, storedData);
zk(1) = z1;
block.Dwork(15).Data = zk;

toc

%% Plots for debugging
simTime1 = 20;
simTime2 = 80;
simTime3 = 100;
if simIter == simTime1+1 || simIter == simTime2+1 || ...
        simIter == simTime3+1
    if simIter == simTime1+1 % at time 20 seconds
        figure(1);
    elseif simIter == simTime2+1
        figure(2);
    elseif simIter == simTime3+1
        figure(8); % corresponds with figure number in report
    end
    stairs(0:sysVar.Np, yk');
    hold on
    stairs(0:sysVar.Np, uk');
    hold off
    title(strcat('MPC Optimized Input and Predicted', ...
        ' Output for Simulink Time at', {' '}, ...
        num2str(simIter-1)));
    xlabel(strcat('Samples from Simulink discrete time', ...
        {' '}, num2str(simIter-1)));
    leg = legend('$y_1^*(k|k)$', '$y_2^*(k|k)$', ...
    '$u_1^*(k|k)$', '$u_2^*(k|k)$', ...
    'Location','east');
    set(leg, 'Interpreter', 'latex');
    set(gcf,'Position',[200 200 600 400])
```

```
    if simIter == simTime3+1
        addpath('matlab2tikz\src'); % library that enables
           saving data for Latex pgf figures
        matlab2tikz('Figures\MPCduringSim.tex');
    end
end

%% Output optimal control step
block.OutputPort(1).Data = uOptimal(:,1);

%% Output calculated plant output for debugging
block.OutputPort(2).Data = yk(:,1); % output y(k|k)

%end Outputs

%% Called to update discrete states during simulation step
function Update(block)

block.Dwork(1).Data = block.Dwork(1).Data + 1;

%end Update

%% Set the sampling of the ports
function SetInputPortSamplingMode(block, idx, fd)

    block.InputPort(idx).SamplingMode = fd;
    for i = 1:block.NumOutputPorts
        block.OutputPort(i).SamplingMode = fd;
    end

%end SetInputPortSamplingMode

%% Called at the end of simulation for cleanup
function Terminate(block)

%end Terminate
```

10.3 mpc Function

```
function uOptimal = mpc(delayMod, sysVar, xk, storedData)
% This function implements the model predictive control (MPC)
% algorithm, which computes the optimal input steps to
% minimizes the objective function.

objFunc = @(uFlat) ...
    objectiveFunc(uFlat, delayMod, sysVar, xk, storedData);

options = optimoptions('fminunc',...
```

```
    'Display','Iter',...
    'MaxFunEvals',Inf, 'MaxIterations',4000);

% take the last control implemented as the first guess of the
% new step
uGuess = storedData.uPrevious;
uGuess = reshape(uGuess,[],1); % flatten matrix

% find the optimal control steps
tic
uOptimal = fminunc(objFunc, uGuess, options);
toc

uOptimal = reshape(uOptimal,2,[]);

end
```

10.4 objectiveFunc Function

```
function cost = ...
    objectiveFunc(uFlat, delayMod, sysVar, xk, storedData)

% Extract needed system variables
Np = sysVar.Np;
Nc = sysVar.Nc;
Q = sysVar.Q;
R = sysVar.R;

% predict the future plant behaviour based on iputs uControl
uControl = reshape(uFlat,2,[]);
[yk, uk, z1] = ...
    prediction(uControl, delayMod, sysVar, xk, storedData);

% Current setpoint
Ysp = delayMod.Ysp;

%% Calculate objective function
cost = 0;

% Output cost contribution
for i = 1:Np
    cost = cost + (Ysp-yk(:,i))'*Q*(Ysp-yk(:,i));
end

% Input cost contribution
uDel = zeros(2, Nc);
u0 = storedData.uPrevious(:,1); % last implemented u
```

```
uDel(:,1) = uControl(:,1) - u0;
for i = 2:Nc
    uDel(:,i) = uControl(:,i) - uControl(:,i-1);
end

for i = 1:Nc
    cost = cost + uDel(:,i)'*R*uDel(:,i);
end

end
```

10.5 prediction Function

```
function [yk, uk, z1] = ...
    prediction(uControl, delayMod, sysVar, xk, storedData)

% Extract needed system variables
Np = sysVar.Np;
Nb = sysVar.Nb;
C = sysVar.C;

% Extract needed model parameters
inputDelay = delayMod.inputDelay;
A = delayMod.A;
B1 = delayMod.B1;
B2 = delayMod.B2;
C1 = delayMod.C1;
C2 = delayMod.C2;
D11 = delayMod.D11;
D12 = delayMod.D12;
D21 = delayMod.D21;
D22 = delayMod.D22;
tau = delayMod.tau;
p = delayMod.p;

% Stored data
uStored = storedData.uStored;
zk = storedData.zk;

% Implement blocking
uk = zeros(p,Np+1);
for i = 1:C
    index = floor((i-1)/Nb + 1);
    uk(:,i) = uControl(:,index);
end
uk(:,C+1:Np+1) = uControl(:,end)*ones(1,Np+1-C); % propagate
   last control move
```

```
% Create vector to store delayed inputs
ukDelayed = zeros(p,1);

% Create predicted output vector array
yk = zeros(2, Np+1);

%% Propagate state space model to implement prediction
for i = 1:Np+1
    % execute delay section of the state space model
    uStored(:,1) = uk(:,i); % if inputDelay is zero, then
       ukDelayed = uk(:,i)
    for j = 1:p
        ukDelayed(j) = uStored(j,inputDelay(j)+1);
    end
    zkminusTau = zk(tau+1);
    wk = zkminusTau;
    zk(1) = C2*xk+D21*ukDelayed+D22*wk;

    % store the value of z at thes start of simulation
    % iteration to fill the z buffer for the next simulation
    % iterations
    if i == 1
        z1 = zk(1);
    end

    % calculate states and output
    xkplus1 = A*xk+B1*ukDelayed+B2*wk;
    yk(:,i) = C1*xk+D11*ukDelayed+D12*wk;

    % setup parameters for next iteration
    xk = xkplus1;
    uStored = circshift(uStored,1,2);
    zk = circshift(zk,1);

end

end
```

10.6 TestMPC Script

```
% MPC test script

clear;
addpath('matlab2tikz\src'); % library that enables saving
   data for Latex pgf figures
savePlotData = true;

%% setup system variables
```

```
Ts = 1;      % sampling time
Nb = 6;      % blocking samples
Nc = 3;      % control moves
C = Nc*Nb;  % control horizon in terms of samples (not time)
Np = 100 + C;     % prediction horizon
Q = [1 0; 0 1.4]; % output weighing matrix
R = [0.01 0; 0 0.04]; % input weighing matrix
n = 4;       % amount of states
p = 2;       % amount of inputs
q = 2;       % amount of outputs

% C and D to output states
CforStateOutput = eye(4);
DforStateOutput = zeros(n, p);

% random test setpoints
Ysp = [5;-5]; % random test setpoints

%% Create plant model
G11 = tf(12.8,[16.7 1],'IODelay', 1);
G12 = tf(-18.9,[21 1],'IODelay', 3);
G21 = tf(6.6,[10.9 1],'IODelay', 7);
G22 = tf(-19.4,[14.4 1],'IODelay', 3);
G = [G11 G12; G21 G22];
Gd = c2d(G,Ts); % convert to discrete model

plantC = ss(G);
plantD = ss(Gd);     % discrete state-space

% decompose state-space system
[H,tau] = getDelayModel(plantD);
[A,B1,B2,C1,C2,D11,D12,D21,D22,E,tau]=getDelayModel(plantD);

%% Setup the system
delayMod.A = A;
delayMod.B1 = B1;
delayMod.B2 = B2;
delayMod.C1 = C1;
delayMod.C2 = C2;
delayMod.D11 = D11;
delayMod.D12 = D12;
delayMod.D21 = D21;
delayMod.D22 = D22;
delayMod.tau = tau;
delayMod.inputDelay = plantD.inputDelay;
delayMod.n = n;
delayMod.p = p;
delayMod.q = q;
```

```
delayMod.Ysp = Ysp;

sysVar.Np = Np;
sysVar.Nb = Nb;
sysVar.Nc = Nc;
sysVar.Q = Q;
sysVar.R = R;
sysVar.Ts = Ts;
sysVar.C = C;

storedData.uPrevious = ones(p,Nc);
storedData.uStored = zeros(p,max(plantD.inputDelay)+1);
storedData.zk = zeros(1,tau+1);

xk = [0;0;0;0]; % initial states

%% Test MPC
tic
uOptimal = mpc(delayMod, sysVar, xk, storedData)
toc

[yk, uk, z1] = ...
    prediction(uOptimal, delayMod, sysVar, xk, storedData);

figure(5);
% plot(0:Np, yk, 0:Np, uk);
stairs(0:Np, yk');
% plot(0:Np, yk);
hold on
stairs(0:Np, uk');
hold off
xlim([0,60]);
title('MPC Optimized Input and Predicted Output');
xlabel('Samples from system discrete time k');
leg = legend('$y_1^*(k|k)$', '$y_2^*(k|k)$', ...
'$u_1^*(k|k)$', '$u_2^*(k|k)$', ...
'Location','east');
set(leg, 'Interpreter', 'latex');
set(gcf,'Position',[200 200 600 400])

if savePlotData == true
    matlab2tikz('Figures\TestMPC.tex');
end
```

10.7 PlotResults Script

```
% This script is automatically called at the end of the
% Simulink simulation to plot the results (using the StopFcn
```

```
% callback). The data is extracted the enabled logging of
% the needed variables.
% This script can be run independantly, but only after the
% Simulink simulation has been run at least once (so that the
% variables are available in the Workspace).

y = out.logsout{1}.Values.Data;
yk = out.logsout{2}.Values.Data;
uk = out.logsout{7}.Values.Data;

t = out.tout;
figure(9); % corresponds with figure number in the report
stairs(t, Ysp', '--')
hold on
% stairs(t,uk)
plot(t, y)
stairs(t, yk)
hold off
title('Simulink Simulation Result');
xlabel('Simulink Discrete Time k');
ylim([-7,7]);
leg = legend('$Y_{sp1}(t)$', '$Y_{sp2}(t)$', ...
    '$y_1^*(t)$', '$y_2^*(t)$', ...
    '$y_1^*(k)$', '$y_2^*(k)$', ...
    'Location','east');
set(leg, 'Interpreter', 'latex');
set(gcf,'Position',[200 200 600 400])
matlab2tikz('Figures\Outputs.tex'); % library path added in
   SetupParameters script

figure(10); % corresponds with figure number in the report
stairs(t,uk)
title('Optimal Inputs');
xlabel('Simulink Discrete Time k');
leg = legend('$u_1^*(k)$', '$u_2^*(k)$', ...
    'Location','northeast');
set(leg, 'Interpreter', 'latex');
set(gcf,'Position',[200 200 600 400])
matlab2tikz('Figures\Inputs.tex');
```