Assignment 7: Introduction to Constrained MPC

Charl van de Merwe, 20804157
Optimal Control, EBO 780

June 14, 2020

1 Assignment Overview

The purpose of this assignment is to implement a simple constrained linear model predictive controller (MPC), without using MATLAB's MPC toolbox. A Kalman filter (KF) needs to be implemented to estimate the states and disturbances. The MPC and KF will make use of the prediction model, defined by

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

where $Y(s)$ is the controlled variable (CV) and $U(s)$ is the manipulated variable (MV).

The process model differs from the prediction model and is given by

$$G_P(s) = \begin{bmatrix} \frac{15.36}{16.7s+1}e^{-1s} & \frac{-18.9}{25.2s+1}e^{-3s} \\ \frac{7.92}{10.9s+1}e^{-7s} & \frac{-19.4}{17.28s+1}e^{-3s} \end{bmatrix}. \tag{2}$$

2 MPC and KF Description

2.1 Problem 1: MPC Description

Problem: Define a discrete-time linear constrained MPC by giving the objective function and optimisation problem with constraints. Further, define 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

$$\min_{u(k+N_C|k),\ldots,u(k|k)} V(x(k|k), u), \tag{3}$$

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. The input and states are constrained, as defined by

$$\begin{aligned} x &\in X \\ y &\in U \\ X &\triangleq \{x \in \mathbb{R}^{N_X} | x_l \leq x \leq x_u\} \\ U &\triangleq \{u \in \mathbb{R}^{N_U} | u_l \leq u \leq u_u\} \end{aligned}$$

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

$$\begin{aligned} V(x,u) = &\sum_{i=1}^{N_P} (Y_{sp}(k) - y(k+i|k))^T Q (Y_{sp}(k) - y(k+i|k)) + \\ &\sum_{i=1}^{N_C} \Delta u(k+i|k)^T R \, \Delta u(k+i|k), \end{aligned} \tag{4}$$

subject to

$$x(k+i+1|k) = f(x(k+i|k), u(k+i|k)) \qquad \forall i = 1, ..., N_C \tag{5a}$$
$$x(k+i+1|k) = f(x(k+i|k), u(k+N_C|k)) \qquad \forall i = N_C+1, ..., N_P \tag{5b}$$
$$y(k+i|k) = h(x(k+i|k), u(k+i|k)) \qquad \forall i = 1, ..., N_C \tag{5c}$$
$$y(k+i|k) = h(x(k+i|k), u(k+N_C|k)) \qquad \forall i = N_C+1, ..., N_P, \tag{5d}$$

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 (**??**) 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 (4). Future MV behaviour can also be specified by using zones, reference trajectories or a funnel [1].

2.2 Problem 2: KF Description

Problem: Define a discrete-time Kalman filter in state-space form which can estimate both the process states and process disturbances.

The Kalman filter (KF) takes the implemented CVs, u_k, and measurements of the MVs, y_k, to determine estimated states, $\hat{x}_k$. In this implementation, the Kalman filter is extended to include estimates of the process disturbances. The process disturbances, d_k, can be modelled as random walks. The process model is therefore defined by

$$x_{k+1} = Ax_k + B(u_k + d_k) \tag{6a}$$
$$d_{k+1} = d_k + w_k \tag{6b}$$
$$y_k = Cx_k. \tag{6c}$$

The process model is rewritten in an augmented state space model as

$$z_{k+1} = \bar{A}z_k + \bar{B}p_k \tag{7a}$$
$$y_k = \bar{C}z_k + \bar{D}p_k, \tag{7b}$$

where

$$z_k = \begin{bmatrix} x_k \\ d_k \end{bmatrix} \tag{8a}$$
$$p_k = \begin{bmatrix} u_k \\ w_k \end{bmatrix} \tag{8b}$$
$$\bar{A} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \tag{8c}$$
$$\bar{B} = \begin{bmatrix} B & 0 \\ 0 & I \end{bmatrix} \tag{8d}$$
$$\bar{C} = \begin{bmatrix} C & 0 \end{bmatrix}. \tag{8e}$$
$$\bar{D} = 0. \tag{8f}$$

Note that the random walk variable, w_k, will be set to zero in the model, but the value of the disturbance, d_k, will be determined and update by the Kalman filter.

The Kalman filter estimates the states in two steps. The next time step (at discrete time $k+1$) estimate, $\hat{z}_{k+1|k}$, is determined in the *time update* step:

$$\hat{z}_{k+1|k} = \bar{A}\hat{z}_{k|k} + \bar{B}p_k, \tag{9}$$

where $\hat{z}_{k|k}$ is the current estimate of the states determined from measurements at time k. The current estimate is determined in the *measurement update* step:

$$\hat{z}_{k|k} = \hat{z}_{k|k-1} + M(y_{vk} - \bar{C}\hat{z}_k), \tag{10}$$

where $\hat{z}_{k|k-1}$ is the estimate of the state at time k, determined by the *time update* step at time $k-1$, M is the Kalman gain and y_{vk} is the noisy output measurement. The *time update* step can be rewritten as a state space system, by substituting (10) into (9), as

$$\hat{z}_{k+1|k} = A_{KF}\hat{z}_{k|k-1}B_{KF}u_{KF} \tag{11a}$$
$$y_{KF} = C_{KF}\hat{z}_{k|k-1}, \tag{11b}$$

where

$$u_{KF} = \begin{bmatrix} p_k \\ y_{vk} \end{bmatrix} \tag{12a}$$
$$A_{KF} = \bar{A}(I - M\bar{C}) \tag{12b}$$
$$B_{KF} = \begin{bmatrix} \bar{B} & \bar{A}M \end{bmatrix} \tag{12c}$$
$$C_{KF} = I. \tag{12d}$$

Note that the output of (11) is $\hat{z}_{k|k-1}$. To determine $\hat{z}_{k|k}$, the *measurement update* has to be implemented. The *measurement update* step is rewritten as

$$\hat{z}_{k|k} = (I - M\bar{C})\hat{z}_{k|k-1} + My_{vk}. \tag{13}$$

From the output of the *measurement update*, the filtered (estimated) output can be determined from

$$\hat{y}_{k|k} = \bar{C}\hat{z}_{k|k}. \tag{14}$$

2.3 Problem 3: Process Model

Problem: Based on the specifications above, give the process model $G_P(s)$ in continuous LTI format.

The continuous process model is given in (2).

3 System Description in Discrete Time

3.1 Problem 4: Discrete State-Space Prediction Model

Problem: Create a discrete state-space prediction based on the continuous model, by discretizing it with a sampling time of 1 second.

The continuous model in (1) is converted to a discrete-time model using MATLAB's *c2d* function, with a sampling time of $T_s = 1\,s$. The discrete-time model is converted to a state-space model by using MATLAB's *ss* function. The time-delays are then mapped to discrete-time state by using the *absorbDelay* function, as seen in Section 9.1. The resulting model is

$$x_{k+1} = Ax_k + Bu_k \tag{15a}$$
$$y_k = Cx_k + Du_k. \tag{15b}$$

with the A, B and C matrices corresponding to the matrices in (6) and (8). These matrices are

$$A = \begin{bmatrix} 0.9419 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0.9123 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.9535 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.9329 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0.5786 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{16a}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{16b}$$

$$C = \begin{bmatrix} 0.74400 & -0.878900000000000 \\ 000 & -1.30151000000000 \end{bmatrix} \tag{16c}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{16d}$$

3.2 Problem 5: KF Model

Problem: Create a discrete state-space model for the Kalman filter, based on your discrete-time model for the MPC. The KF should also estimate the process disturbances.

The answer to this problem is included in Section 2.2 (Problem 2). This is implemented in code, as can be seen in Section 9.1.

3.3 Problem 6: KF Gain

Problem: Create a discrete-time linear KF using the MATLAB command "kalman", based on the state-space KF model.

The augmented state-space prediction model defined in (7) and (8) are used in MATLAB's *kalman* function to determine the Kalman gain, as can be seen in Section 9.1. In addition to the process noise, w_k, measurements of the output includes noise, v_y. The augmented process model for the Kalman filter is therefore given by (note that $\bar{D} = 0$)

$$z_{k+1} = \bar{A} z_k + \bar{B} \begin{bmatrix} u_k \\ w_k \end{bmatrix} \tag{17a}$$

$$y_k = \bar{C} z_k + v_k, \tag{17b}$$

where w_k is the process noise with a covariance of $E(w_k w_k^T) = Q_{KF}$ and v_k is the measurement noise with a covariance of $E(v_k v_k^T) = R_{KF}$. It may not always be possible to physically measure the noise covariances. The Q_{KF} and R_{KF} matrices are chosen based on prior knowledge of the process and measurement noise and tuned to produce the best possible control. The absolute value of Q_{KF}, R_{KF} or elements within the matrices do not matter, but their values with respect to each other do.

In the simulation done in this assignment, there is no reason to expect that the noise will be greater on one of the inputs or outputs than of the other. Therefore $Q_{KF11} = Q_{KF22}$

and $R_{KF11} = R_{KF22}$ where chosen. The output noise in this simulation contributes more to the disturbance of the system than the input noise, therefore $R_{KF} = 50Q_{KF}$ was chosen. The chosen noise covariance matrices are

$$Q_{KF} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{18a}$$

$$R_{KF} = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix} \tag{18b}$$

Based on the augmented process matrices defined in (8) and the noise covariance matrices in (18), the Kalman gain is

$$M = \begin{bmatrix} 0.4218 & -0.2423 \\ 0.3965 & -0.2233 \\ -0.0608 & -0.3204 \\ -0.0618 & -0.3101 \\ 0.0210 & -0.0227 \\ 0.0405 & -0.0335 \\ 0.0668 & -0.0489 \\ 0.0999 & -0.0677 \\ 0.1394 & -0.0883 \\ 0.1836 & -0.1092 \\ 0.1070 & -0.0511 \\ -0.0348 & -0.0989 \\ -0.0348 & -0.0989 \\ -0.0348 & -0.0989 \\ 0.1070 & -0.0511 \\ -0.0348 & -0.0989 \end{bmatrix} \tag{19}$$

4 Simulink Model

The complete Simulink model can be seen in Figure 1. The plant, G_p is as defined in (2). The MPC block is implemented by using a level-2 MATLAB s-function block, as seen in Figure 1. This block allows the use of MATLAB language to create a custom block. The MPC block reads the estimated state and filtered disturbances. The MPC algorithm is implemented within the MPC block, so that the output of the block is the optimal control steps for the plant. The implementation of the code for this block can be seen in Section 9.2.

The state and disturbance estimates are done by the *Kalman Filter* block, as can be seen in Figure 2. The *Kalman Filter* block implements (11) to (14), to determine a current estimate of the states (including disturbance) and output based on the implemented input/control, measured output and previous predicted states ($x_{k|k-1}$).

The disturbance estimate is filtered by the *State Disturbance Filter* block seen in Figure 3. This filter is explained in Section 6.1.

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.

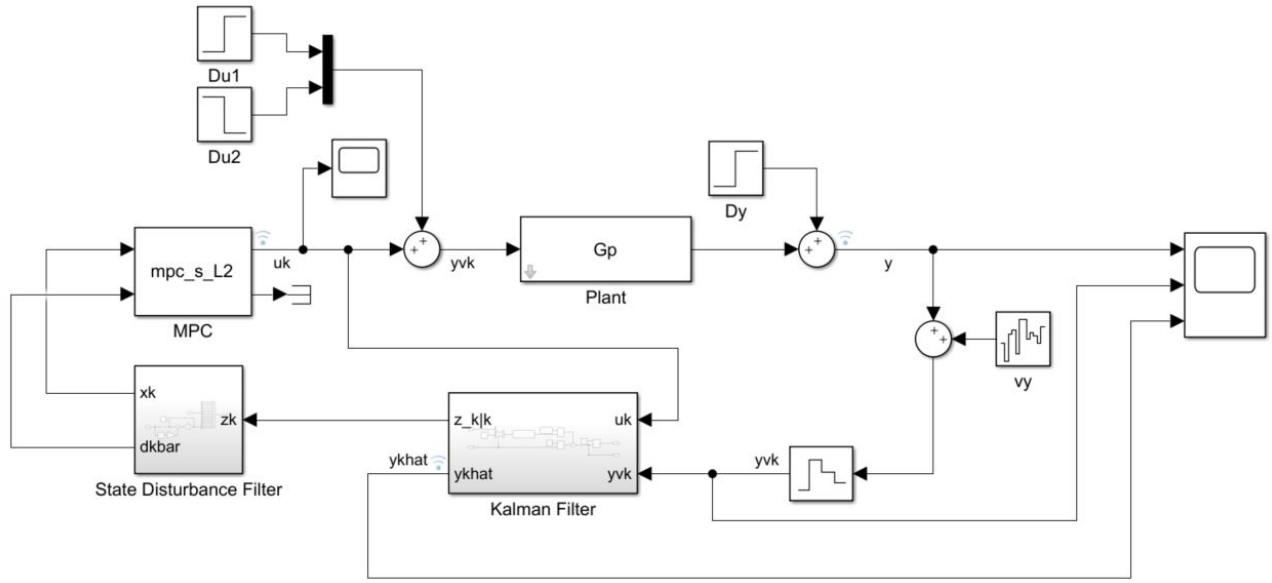

Figure 1: Simulink Model

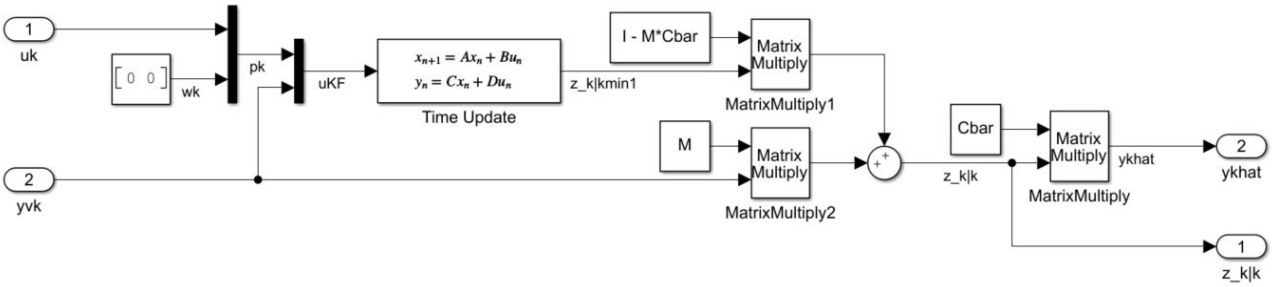

Figure 2: Kalman Filter Block

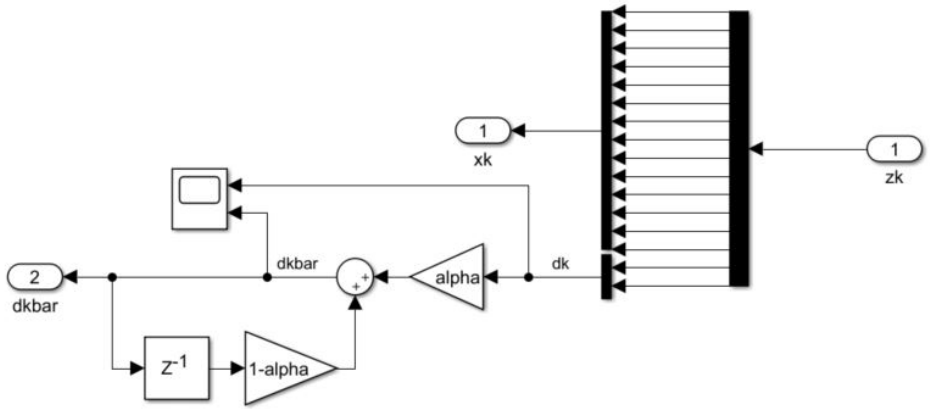

Figure 3: State Disturbance Filter Block

5 MPC Controller

5.1 Problem 7: MPC Constraints

Problem: Code the MATLAB function "constraints" that enforces constraints on the inputs. This function will be included as part of the optimization function in the MPC. The constraints for $G_C(s)$ on the manipulated variable (MV) 1 are $(-\infty, 1]$ and on MV 2 are $[-4, \infty)$.

Constrained MPC finds the optimal controlled variables (CV) for which the objective function

$$\min_{u(k+N_C|k),\ldots,u(k|k)} V(x(k|k), u), \tag{20}$$

is minimized such that

$$c(x, u) \leq 0 \tag{21a}$$
$$c_{eq}(x, u) = 0, \tag{21b}$$

where linear and non-linear inequality and equality constraints are captured in $c(x, u)$ and $c_{eq}(x, u)$. The inequality constraints defined in the problem can be formulated as

$$c(u) = \begin{bmatrix} u_1 - 1 \\ -u_2 - 4 \end{bmatrix} \leq 0. \tag{22}$$

As stated in Section 2.1, there are N_C unique control steps in the MPC prediction. The constraints needs to be imposed on each of these control steps. The inequality constraints vector implemented is therefore

$$c(u) = \begin{bmatrix} u_1(k|k) - 1 \\ \vdots \\ u_1(k + N_C|k) - 1 \\ -u_2(k|k) - 4 \\ \vdots \\ -u_2(k + N_C|k) - 4 \end{bmatrix}. \tag{23}$$

The constraints are defined in the *getConstraints* function, as can be seen in Section 9.4. These constraints are passed to the *fmincon* function in the *nonlcon* argument, as can be seen in Section 9.3.

5.2 Problem 8: MPC Algorithm Description

Problem: Code the Matlab function *ObjFunc* that calculates the objective value that will be optimized by *fmincon*. Your MPC controller must implement blocking to allow for a smooth closed-loop MV trajectory. The function "*ObjFunc*" should use the disturbance estimate from your KF as part of the predictions. Show the algorithm "*ObjFunc*" uses to calculates the predictions for the MPC.

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 (15).

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 (4).

4. Repeat step 1 to 3 to find the optimal input steps.

These steps are followed within MATLAB's *fmincon* function. This function will numerically compute the optimal input steps, within the constraints defined in the constraints function, to minimize the objective function.

The implementation of the MPC algorithm can be seen in Section 9.3. The constraints are defined in the *getConstraints* function as seen in Section 9.4. The computation of the objective function can be seen in Section 9.5. The implementation of the output prediction can be seen in Section 9.6.

The output and input weighing matrices, Q and R, as seen in the objective function in (4), were chosen to be

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1.4 \end{bmatrix} \tag{24a}$$

$$R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.04 \end{bmatrix}. \tag{24b}$$

5.3 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 9.7. The result of this test can be seen in Figure 4.

The *fmincon* function computes the optimal input steps for this test script to be ($N_C = 3$ was chosen):

$$\begin{aligned} U^* &= \begin{bmatrix} u_1^*(k|k) & u_1^*(k+1|k) & u_1^*(k+3|k) \\ u_2^*(k|k) & u_2^*(k+1|k) & u_2^*(k+3|k) \end{bmatrix} \\ &= \begin{bmatrix} 1.0000 & 0.9390 & 0.5442 \\ 2.0758 & 2.1836 & 1.7889 \end{bmatrix}. \end{aligned} \tag{25}$$

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. None of the input values exceed the constraints defined in (22). The plot is limited to 60 samples, but the chosen prediction horizon is $N_P = 100$.

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).

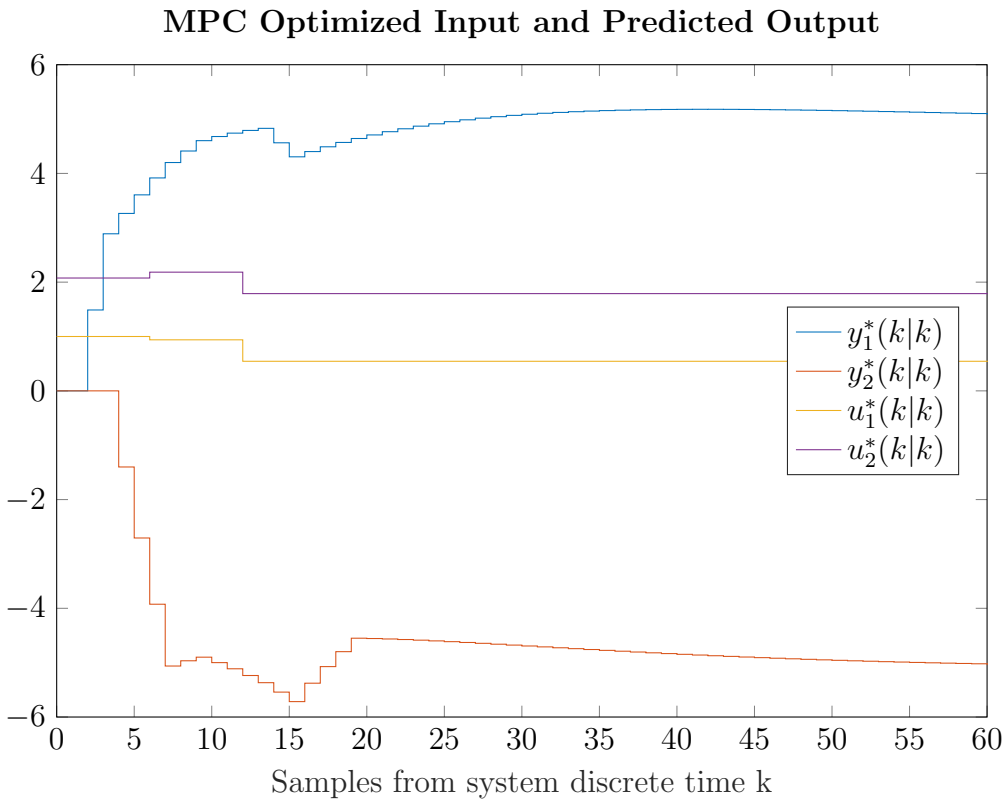

Figure 4: MPC Test Result

6 Simulation

6.1 Problem 9: Disturbance Filter

Problem: Add an extra input to your s-function m-file to read in your disturbance estimate from your KF. Exponentially filter the disturbance estimate to remove unwanted noise from the estimate. Add the filtered disturbance estimate to the MV values of your prediction model in such a way that the MV constraints will still be honoured by your MPC controller.

The Kalman filter includes process disturbances, d_k, as a state, as can be seen in (8a). The KF therefore estimates the process disturbance. An exponential filter is used to filter the estimated disturbance. The filter is described by

$$\bar{d}_k = \alpha d_k + (1 - \alpha)\bar{d}_{k-1}, \tag{26}$$

where $\bar{d}_k$ is the filtered disturbance, $\bar{d}_{k-1}$ is the previous filtered value, d_k is the KF estimate of the disturbance and α is a fraction between 0 and 1 that controls the amount of filtering. The implementation of the filter can be seen in Figure 3. The filter with $\alpha = 0.3$ produces the result in Figure 5.

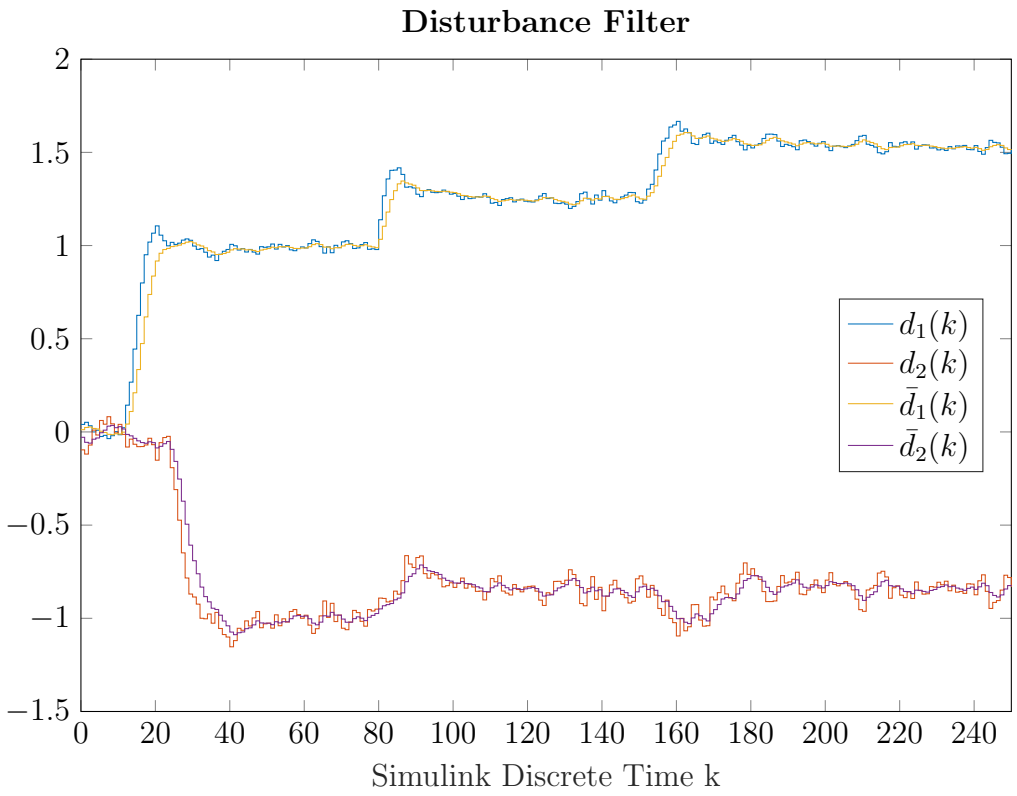

Figure 5: Estimated and Filtered Disturbance

The filtered disturbance estimate is read by the MPC block, as can be seen in Figure 1. The disturbance is added to the prediction algorithm as follows (see the code implementation in Section 9.6):

$$x_{k+1} = Ax_k + B(u_k + d_k) \tag{27a}$$
$$y_k = Cx_k + D(u_k + d_k). \tag{27b}$$

This allows the MPC to take the disturbance into account when calculating the optimal control steps.

6.2 Problem 10: Simulation Setup

Problem: Ensure that the plant is simulated with $G_P(s)$. Therefore, there will be model-plant-mismatch between the plant and the model used for the MPC and KF. Create the following events in the simulation:

- Add a step disturbance to MV 1 that will trigger at time 10 seconds with magnitude 1.
- Add a step disturbance to MV 2 that will trigger at time 20 seconds with magnitude -1.
- Add a step disturbance to the output of $G_P(s)$ that will add 1 to CV 1 and -1 to CV 2 at time 80 seconds.
- Further, add band-limited white noise to the CVs of $G_P(s)$ with a noise power of $\begin{bmatrix}0.1 & 0.1\end{bmatrix}^T$ and a sampling time of 1 second.

- Make a set-point step change of $\begin{bmatrix}5 & -5\end{bmatrix}^T$ at time 150 seconds.
- Tune your controller to maintain the set-point if the constraints become active.

At the start of the Simulation, the *SetupParameters* function (Section 9.1) is called to create and define the process model, the discrete prediction model, the augmented KF model, the Kalman gain, 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. The relevant parameters are passed to the MPC block as dialog parameters.

The output setpoints are stored in a vector array, Y_{spArr}, as seen in Section 9.1. 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 9.2.

The specified input disturbance, D_{u1} and D_{u2}, output disturbance, D_y, and measurement noise, v_y, are added to the Simulink model directly, as can be seen in Figure 1.

6.3 Problem 11: Simulation Results

Problem: Run the Simulink model for 250 seconds. Start with the initial condition of all the states at 0. Document the plots for the CVs, the setpoint changes, and the MVs. Document the plots of the disturbance estimates from the KF. Properly label the plots.

By running the Simulink model, as seen in Figure 1, for 250 seconds, the output and input results are found, as seen in Figure 6 and 7. In Figure 6, $y^*(t)$ is the optimally controlled CVs without measurement noise and $\hat{y}(k)$ is the Kalman filter's estimates of the CVs. The disturbance estimate plot can be seen in Figure 5. The plots are generated at the end of the simulation by calling the *PlotResults* script, as seen in Section 9.8, from Simulink's *StopFcn* callback function.

7 Conclusion

7.1 Problem 12: Performance of MPC and KF

Problem: Comment on the performance of the MPC and KF.

From Figure 6, 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 (2). 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). Despite these large delays, plant model mismatch, MV constraints, input disturbance, output disturbance and measurement noise the MPC controller is able to control the plant's output response (CVs) effectively and track the setpoints.

Without the Kalman filter, the system would not be able to track the setpoints, due to the mismatch between the prediction model and process model, as seen in (1) and (2). In the presence of disturbance, the system would not be able to track the setpoints without the disturbance estimates (determined by the KF and disturbance filter).

From the result in Figure 6 it can be seen that the input and output disturbances causes the CVs to briefly move away from the setpoints. As soon as the Kalman filter

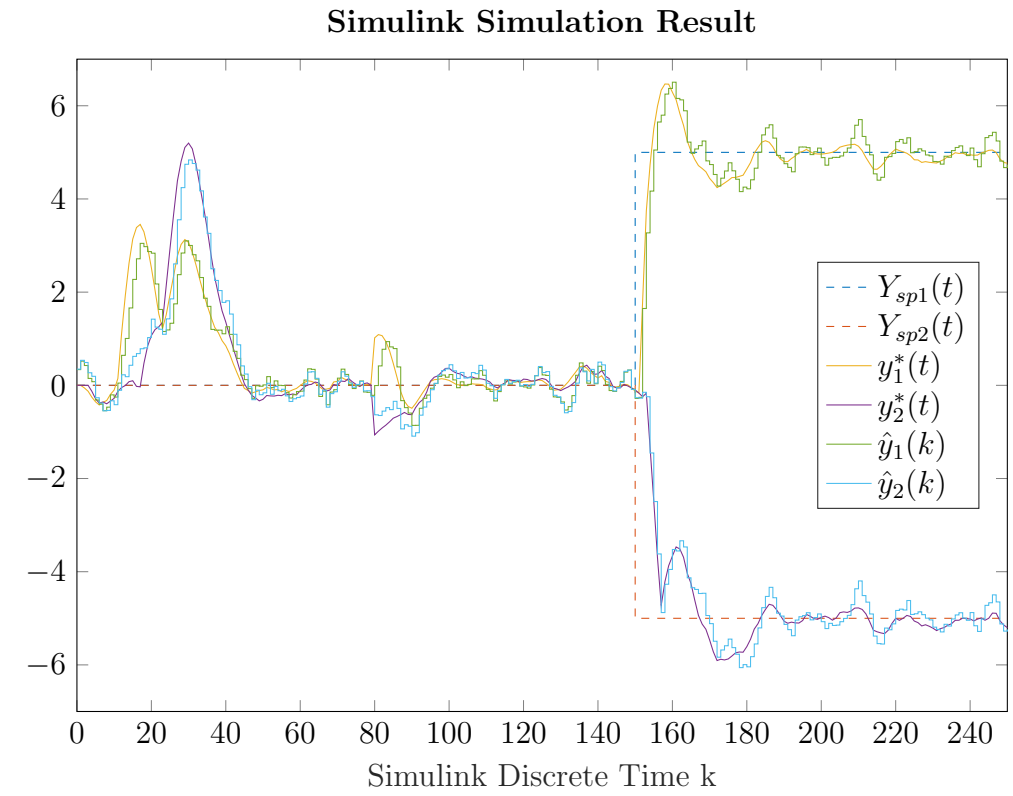

Figure 6: Simulation Result: Controlled Variables and Setpoints

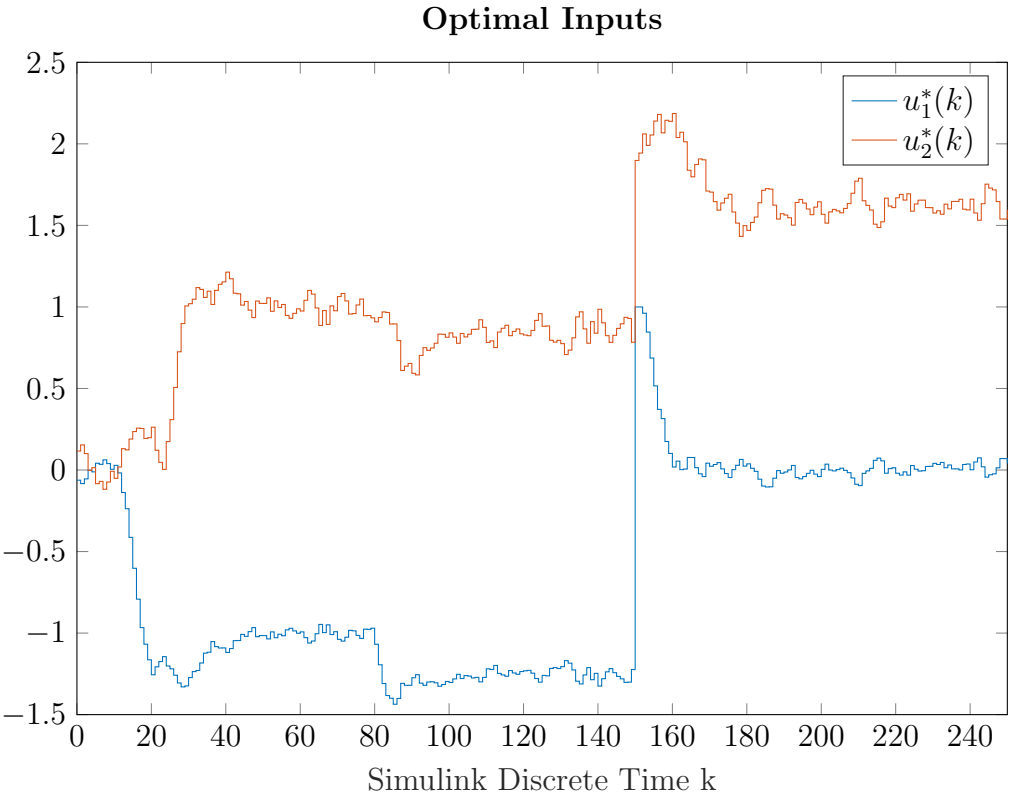

Figure 7: Simulation Result: Manipulated Variables

accurately estimates these disturbances, the controller counteracts its effect (see Figure 5 and 7), so that the setpoints are tracked once again. The effect of input disturbance has a larger effect on the CVs than output disturbance. The effect of D_{u2} is larger than that of D_{u1}, but it seems that the system responds to (suppresses) the effect of either disturbance in roughly the same amount of time.

In this assignment, the prediction model is the same as the plant model in Assignment 5. The output and input weighing matrices, Q and R, are therefore kept the same. R could possibly be decreased to counteract the effect of measurement noise, but doing so did not improve the control in simulation significantly.

As mentioned, the output noise in this simulation contributes more to the disturbance of the system than the input noise, therefore $R_{KF} = 50Q_{KF}$ was chosen. Increasing R_{KF} further decreases the amount of noise is the estimate, but increases estimation lag. This would result in the state estimate to be inaccurate initially under a new disturbances, but the steady-state estimate could be improved (due to less noise in the estimate).

7.2 Problem 13: MATLAB Code

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

All of the relevant code can bee seen in the Appendix (Section 9).

8 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.

9 Appendix: MATLAB Code

9.1 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;     % prediction horizon
Q = [1 0; 0 1.4]; % output weighing matrix
R = [0.01 0; 0 0.04]; % input weighing matrix
% R = [0.1 0; 0 0.4]; % input weighing matrix
% n = 4;       % amount of states
p = 2;       % amount of inputs
q = 2;       % amount of outputs
tend = 250; % simulation end time
alpha = 0.3; % disturbance filter control

%% Create setpoint array. Note that time t = 0 is at index 1
YspArr = zeros(2,tend+1);
YspArr(:,150+1:tend+1) = [5;-5]*ones(1,tend-150+1);

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

%% Create prediction 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

% create a state space model which absorbs the delay
plantD = absorbDelay(ss(Gd));
n = size(plantD.A, 1);          % amount of states

%% Kalman filter
% This process model model includes disturbance as a state
Abar = [plantD.A, plantD.B; zeros(p,n), eye(p)];
Bbar = [plantD.B, zeros(n,p); zeros(p,p), eye(p)];
Cbar = [plantD.C, zeros(q,p)];
Dbar = zeros(q, 2*p);

plantD_KF_ss = ss(Abar,Bbar,Cbar,Dbar,Ts);

% Determine Kalman gain
Q_KF = [1 0; 0 1];
R_KF = [50 0; 0 50];
[kalmf,L,P,M] = kalman(plantD_KF_ss,Q_KF,R_KF);

% State Space matrices for KF
I = eye(n+p);
A_KF = Abar * (I-M*Cbar);
B_KF = [Bbar, Abar*M];
C_KF = eye(16);
```

```
D_KF = zeros(16,6);
% C_KF = I - M*Cbar;
% D_KF = [zeros(n+p, 2*p), M];

%% Create process plant model
Gp11 = tf(1.2*12.8,[16.7 1],'IODelay', 1);
Gp12 = tf(-18.9,[1.2*21 1],'IODelay', 3);
Gp21 = tf(1.2*6.6,[10.9 1],'IODelay', 7);
Gp22 = tf(-19.4,[1.2*14.4 1],'IODelay', 3);
Gp = [Gp11 Gp12; Gp21 Gp22];

% %% 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
```

9.2 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  = 2;
block.NumOutputPorts = 2;

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

% Override input port properties
block.InputPort(1).Dimensions  = 14;
% block.InputPort(1).Dimensions  = 16;
block.InputPort(1).DatatypeID  = 0;  % double
block.InputPort(1).Complexity  = 'Real';
block.InputPort(1).DirectFeedthrough = true;

block.InputPort(2).Dimensions  = 2;
block.InputPort(2).DatatypeID  = 0;  % double
block.InputPort(2).Complexity  = 'Real';
block.InputPort(2).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 = 6;

%% 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 = absorbDelay(ss(Gd));   % discrete state-space,
   delays absorbed
A = plantD.A;
B = plantD.B;
C = plantD.C;
D = plantD.D;

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

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

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

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

%% Dynamic storage
```

```
  block.Dwork(6).Name            = 'uPrevious';
  p = size(B,2); % amount of inputs
  Dimension6 = p*block.DialogPrm(5).Data; % number of inputs
        % timesprediction samples
  block.Dwork(6).Dimensions      = Dimension6;
  block.Dwork(6).DatatypeID      = 0;      % double
  block.Dwork(6).Complexity      = 'Real'; % real
  block.Dwork(6).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 = absorbDelay(ss(Gd));   % discrete state-space,
   delays absorbed
A = plantD.A;
B = plantD.B;
C = plantD.C;
D = plantD.D;

% store flattened matrixes and vectors (plant parameters)
block.Dwork(2).Data = reshape(A,[],1);
block.Dwork(3).Data = reshape(B,[],1);
block.Dwork(4).Data = reshape(C,[],1);
block.Dwork(5).Data = reshape(D,[],1);

% initialize dynamic storage vector
p = size(B,2); % amount of inputs
Nc = block.DialogPrm(5).Data;
uGuess = ones(p,Nc);
uGuess = reshape(uGuess,[],1); % flatten matrix
block.Dwork(6).Data = uGuess; % first guess for control

%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);

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

Cflat = block.Dwork(4).Data;
C = reshape(Cflat,[],n);
q = size(C,1); % amount of outputs

Dflat = block.Dwork(5).Data;
D = reshape(Dflat,q,p);

%% Create data structures to pass to functions
plantD.A = A;
plantD.B = B;
plantD.C = C;
plantD.D = D;

% Setpoint
YspArr = block.DialogPrm(2).Data;
simIter = block.Dwork(1).Data;
Ysp = YspArr(:,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;
sysVar.n = n;
sysVar.p = p;
sysVar.q = q;

%% Read current state
% zk = block.InputPort(1).Data;
% xk = zk(1:n);       % state estimate
% dk = zk(n+1:n+p); % disturbance estimate
xk = block.InputPort(1).Data;
dk = block.InputPort(2).Data;

%% Reshape previous control
uPrevious = block.Dwork(6).Data; % take the last control
        % implemented as the first guess of the new step
uPrevious = reshape(uPrevious,p,[]);
```

```
%% Execute MPC algorithm to find optimal control
uOptimal = mpc(plantD, Ysp, sysVar, xk, dk, uPrevious)

%% Store control data
block.Dwork(6).Data = reshape(uOptimal, [], 1);

%% Do prediction once more for plotting and debugging
[yk, uk] = prediction(uOptimal, plantD, sysVar, xk, dk);

toc

%% Plots for debugging
simTime1 = 20;
simTime2 = 80;
simTime3 = 150;
if simIter == simTime1+1 || simIter == simTime2+1 || ...
        simIter == simTime3+1

%     [yk, uk] = prediction(uOptimal, plantD, sysVar, xk);

    if simIter == simTime1+1 % at time 20 seconds
        figure(1);
    elseif simIter == simTime2+1
        figure(2);
    elseif simIter == simTime3+1
        figure(3); % 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
```

9.3 mpc Function

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

objFunc = @(uFlat) objectiveFunc(...
    uFlat, plantD, Ysp, sysVar, xk, dk, uPrevious);

nonLinCon = @(uFlat) getConstraints(uFlat, sysVar);

% options = optimoptions('fminunc',...
%     'Display','Iter',...
%     'MaxFunEvals',Inf, 'MaxIterations',4000);

% options = optimoptions('fmincon','Display','Iter',...
```

```
%     'Algorithm','sqp',...
%     'MaxFunEvals',Inf, 'MaxIterations',4000);

options = optimoptions('fmincon','Display','Iter',...
    'MaxFunEvals',Inf, 'MaxIterations',4000);

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

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

% x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon) subjects the
% minimization to the nonlinear inequalities c(x) or
% equalities ceq(x) defined in nonlcon. fmincon optimizes
% such that c(x) <= 0 and ceq(x) = 0.
uOptimal = fmincon(objFunc, uGuess,...
    [],[],[],[],[],[], nonLinCon, options);
toc

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

end
```

9.4 getConstraints Function

```
function [C, Ceq] = getConstraints(uFlat, sysVar)

Nc = sysVar.Nc; % number of control steps
p = sysVar.p;   % number of inputs, p=2

uControl = reshape(uFlat,2,[]);

C = zeros(p,Nc);

for i = 1:p
    for j = 1:Nc
        if i == 1
            C(1,j) = uControl(1,j)-1;
        else
            C(2,j) = -uControl(2,j)-4;
        end
    end
end
```

```
C = reshape(C,[],1);

% C = [];
% this method is less preferred, because the
% size of C changes
% for i = 1:Nc
%     C = [C; uControl(1,i)-1; -uControl(2,i)-4];
% end

Ceq = [];
end
```

9.5 objectiveFunc Function

```
function cost = objectiveFunc(...
    uFlat, plantD, Ysp, sysVar, xk, dk, uPrevious)

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

% predict the future plant behaviour based on uControl
uControl = reshape(uFlat,2,[]);
[yk, uk] = prediction(uControl, plantD, sysVar, xk, dk);

%% 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 = 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
```

9.6 prediction Function

```
function [yk, uk] = ...
    prediction(uControl, plantD, sysVar, xk, dk)
% Extract needed system variables
Np = sysVar.Np;
Nb = sysVar.Nb;
CH = sysVar.C; % control horizon
p = sysVar.p;

% Extract model parameters
A = plantD.A;
B = plantD.B;
C = plantD.C;
D = plantD.D;

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

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

%% Propagate state space model to implement prediction
for i = 1:Np+1
    % calculate states and output
    xkplus1 = A*xk+B*(uk(:,i)+dk);
    yk(:,i) = C*xk+D*(uk(:,i)+dk);

    % setup states for the next iteration
    xk = xkplus1;

end

end
```

9.7 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

% random test setpoints
Ysp = [5;-5]; % random test setpoints
dk = [1; -1]; % disturbance

%% 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)
   ;

% create a model which absorbs the delay
plantD = absorbDelay(ss(Gd));   % discrete state-space,
   delays absorbed
n = size(plantD.A, 1);          % amount of states

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

%% Setup the system
sysVar.Np = Np;
```

```
sysVar.Nb = Nb;
sysVar.Nc = Nc;
sysVar.Q = Q;
sysVar.R = R;
sysVar.Ts = Ts;
sysVar.C = C;
sysVar.n = n;
sysVar.p = p;
sysVar.q = q;


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
xk = zeros(n,1); % initial states

%% Test MPC
tic
uOptimal = mpc(plantD, Ysp, sysVar, xk, dk, uPrevious)
toc

[yk, uk] = prediction(uOptimal, plantD, sysVar, xk, dk);

figure(4);
% 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
```

9.8 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).

yk = out.logsout{1}.Values.Data;
uk = out.logsout{2}.Values.Data;
dk = out.logsout{3}.Values.Data;
dkbar = out.logsout{4}.Values.Data;
y = out.logsout{5}.Values.Data;

t = out.tout;
figure(6); % corresponds with figure number in the report
stairs(t, YspArr', '--')
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('$Y_{sp1}(t)$', '$Y_{sp2}(t)$', ...
    '$y_1^*(t)$', '$y_2^*(t)$', ...
    '$\hat{y}_1(k)$', '$\hat{y}_2(k)$', ...
    'Location','east');
set(leg, 'Interpreter', 'latex');
set(gcf,'Position',[200 200 600 400])
if savePlotData == true
    matlab2tikz('Figures\Outputs.tex');
        % library path added in SetupParameters script
end

figure(7); % 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])
if savePlotData == true
    matlab2tikz('Figures\Inputs.tex');
end

figure(5); % corresponds with figure number in the report
```

```
stairs(t,dk)
hold on
stairs(t,dkbar)
hold off
title('Disturbance Filter');
xlabel('Simulink Discrete Time k');
leg = legend('$d_1(k)$', '$d_2(k)$', ...
    '$\bar{d}_1(k)$', '$\bar{d}_2(k)$', ...
    'Location','east');
set(leg, 'Interpreter', 'latex');
set(gcf,'Position',[200 200 600 400])
if savePlotData == true
    matlab2tikz('Figures\Disturbance.tex');
end
```