Shanghai Jiao Tong University

Predictive Control

AU7015

Simulation of DMC algorithm for MIMO systems

Dingran Yuan

121032910048

April 2, 2022

Contents

1 Abstract

This report contains theoretical explanation and practical implementation of the DMC algorithm for MIMO systems, as well as the results and analysis of the simulation. First, in Section 2, we introduce the theories starting from the DMC for SISO systems. Next, in Section 3, we talk about the detailed implementation steps. Then in Section4, we analyse the simulation results through MATLAB and the source codes are attached at the end of this report.

2 Theory

2.1 Dynamic Matrix Control

The Dynamic Matrix Control (DMC) is an iterative control algorithm which utilises the model of the target system to predict the future behaviour of the model within a predefined prediction horizon, also decides the controller behaviour in the control horizon based on the difference between reference values and prediction values. The DMC algorithm was the first Model Predicted Control (MPC) algorithm introduced in early 1980s, and it first appeared in petroleum industry as an optimal controller for oil production. We then separate the theoretical analysis of DMC for Single Input Single Output (SISO) systems and Multi Inputs Multi Outputs (MIMO) systems, and explicate them respectively.

2.2 DMC for unconstrained SISO systems

To understand the DMC for MIMO systems, we first analyse the working principle of the DMC for SISO systems. For SISO system there is no coupling between different input variables, thus, we only need to consider the iterative tracking and the behaviour of a single output variable.

The DMC algorithm consists of three parts: prediction model, rolling optimisation, and feedback correction. *(This is also true for MIMO systems)*

2.2.1 Prediction model

For the system model, DMC uses the sampled unit step response of the system to represent the model

$$a_i = a(iT), \quad i = 1, 2, \dots \tag{1}$$

where T is the sampling period, and a_i is the sampled system output under the unit step input at time $t = iT$. For the asymptotically stable system, the output step response will converge to a constant after a certain amount of time $t_N = NT$, s.t. $a_i - a_{i+1} \to 0, \forall i > N$. Thus, for the stable plant model, we construct a set of parameters $\{a_1, a_2, \dots, a_N\}$ to describe the system, and this set of parameters determines the model parameter of the plant, where

$$\mathbf{a} = [a_1 \dots a_N]^T \tag{2}$$

is called the model vector, and N is called the length of the model. According to the superposition property of step response, we can predict the future plant output through the model vector: if the control input has an increment $\Delta u(K)$, the future output of the model length can be estimated as

$$\widetilde{y}_1(k+i|k) = \widetilde{y}_0(k+i|k) + a_i \Delta u(k), \quad i = 1, \dots, N \tag{3}$$

and if there are M successive control input increments starting from time t_k, the output can be predicted as

$$\widetilde{y}_M(k+i|k) = \widetilde{y}_0(k+i|k) + \sum_{j=1}^{\min(M,i)} a_{i-j+1} \Delta u(k-j+1), \quad i = 1, \dots, N \tag{4}$$

which is the prediction formulation we will use when updating our model predictions giving a control increment Δu. Also, the prediction in Eq.4 can be written in vector form as

$$\widetilde{\mathbf{y}}_{PM}(k) = \widetilde{\mathbf{y}}_{P0}(k) + \mathbf{A}\Delta\mathbf{u}_M(k), \tag{5}$$

where

$$\widetilde{\mathbf{y}}_{PM}(k) = \begin{bmatrix} \widetilde{y}_M(k+1|k) \\ \vdots \\ \widetilde{y}_M(k+P|k) \end{bmatrix}, \ \widetilde{\mathbf{y}}_{P0}(k) = \begin{bmatrix} \widetilde{y}_0(k+1|k) \\ \vdots \\ \widetilde{y}_0(k+P|k) \end{bmatrix} \tag{6}$$

$$\mathbf{A} = \begin{bmatrix} a_1 & \dots & a_M & \dots & a_P \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & a_1 & \dots & a_{P-M+1} \end{bmatrix}^T \tag{7}$$

2.2.2 Rolling optimisation

For rolling optimisation of unconstrained SISO system, we are trying to minimise a cost function by forming an DMC optimisation problem considering the cost of the prediction value (Eq.3) w.r.t. the reference value ($w(k+i),\ i=1,\dots,P$) over the prediction horizon (P), and also consider the rate of change of our control input, which can be formulated as

$$\min_{\Delta u} J(k) = \sum_{i=1}^{P} q_i \left[w(k+i) - \widetilde{y}_i(k+i|k)\right]^2 + \sum_{j=1}^{M} r_i \Delta u^2(k+j-1). \tag{8}$$

Rewriting it in matrix form, we derive

$$\min_{\Delta \mathbf{u}_M(k)} J(k) = \|\mathbf{w}_P(k) - \widetilde{\mathbf{y}}_{PM}(k)\|_{\mathbf{Q}}^2 + \|\Delta \mathbf{u}_M(k)\|_{\mathbf{R}}^2, \tag{9}$$

where, $\mathbf{w}_P(k) = [w(k+1),\dots,w(k+P)]^T$ is the reference value over the prediction horizon, the diagonal matrices $\mathbf{Q}$ and $\mathbf{R}$ are error weighting matrix and control weighting matrix respectively, and can be written as

$$\mathbf{Q} = \begin{bmatrix} q_1 & & \\ & \ddots & \\ & & q_P \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} r_1 & & \\ & \ddots & \\ & & r_M \end{bmatrix} \tag{10}$$

Then, solving the Eq.9, we derive

$$\Delta \mathbf{u}_M(k) = \left(\mathbf{A}^T\mathbf{Q}\mathbf{A} + \mathbf{R}\right)^{-1} \mathbf{A}^T\mathbf{Q}[\mathbf{w}_P(k) - \widetilde{\mathbf{y}}_{P0}(k)]. \tag{11}$$

After computing the control vector $\Delta \mathbf{u}_M$, we extract the current control increment $\Delta u(k)$ by selecting the first element of $\Delta \mathbf{u}_M$ using a row vector $\mathbf{c}^T \in \mathbb{R}^M = [1,0,\dots,0]^T$

$$\Delta u(k) = \mathbf{c}^T \Delta \mathbf{u}_M. \tag{12}$$

The Eq.12 can be written as

$$\Delta u(k) = \mathbf{d}^T[\mathbf{w}_P(k) - \widetilde{\mathbf{y}}_{P0}(k)], \tag{13}$$

where $\mathbf{d}^T = \mathbf{c}^T \left(\mathbf{A}^T\mathbf{Q}\mathbf{A} + \mathbf{R}\right)^{-1} \mathbf{A}^T\mathbf{Q}$, which can be calculated offline, and Eq.13 is the online optimisation equation we use during the iteration process.

2.2.3 Feedback correction

The future prediction is performed by implementing the control input derived from Eq.13, and it can be given by

$$\widetilde{\mathbf{y}}_{N1}(k) = \widetilde{\mathbf{y}}_{N0}(k) + \mathbf{a}\Delta u(k). \tag{14}$$

There might exist error for our updated predicted value at t_{k+1} compared to the actual plant output at t_{k+1}, expressed as

$$e(k+1) = y(k+1) - \widetilde{y}_1(k+1|k), \tag{15}$$

therefore, we need to update our predicted plant values based on the actual output. This is done by using the output error $e(k+1)$ as correction criteria, and introducing a N-dimensional vector $\mathbf{h} = [h_1, \dots, h_N]^T$ is to correct the rest of the model outputs, which can be expressed as

$$\widetilde{\mathbf{y}}_{cor} = \widetilde{\mathbf{y}}_{N1} + \mathbf{h}e(k+1). \tag{16}$$

After the correction, we shift the model output $\mathbf{y}_{N0} \in \mathbb{R}^{N\times 1}$ by 1 timestamp, using a shifting matrix

$$\mathbf{S} = \begin{bmatrix} 0 & 1 & & 0 \\ \vdots & \ddots & \ddots & \\ \vdots & & 0 & 1 \\ 0 & \dots & 0 & 1 \end{bmatrix}, \tag{17}$$

thus, the new system outputs $\mathbf{y}_N$ at instant t_k becomes the new starting point at t_{k+1}, and the control algorithm keeps iterating every time it produces a new output

$$\mathbf{y}_{N0} = \mathbf{S}\widetilde{\mathbf{y}}_{cor}(k+1). \tag{18}$$

The overall process of DMC for SISO system can be illustrated in Figure1.

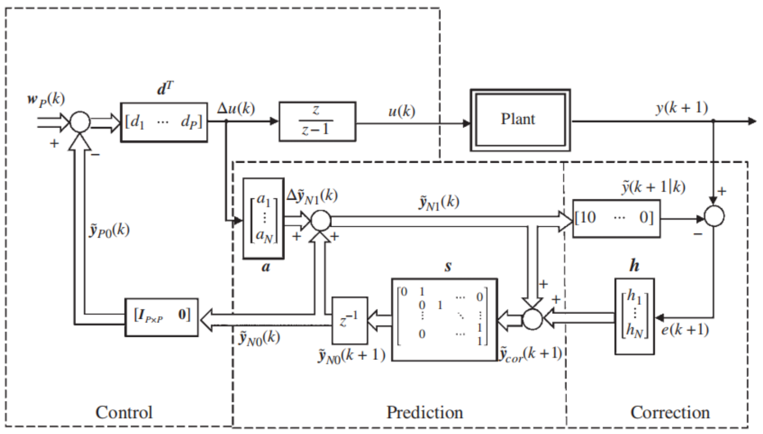

Figure 1: DMC diagram for unconstrained SISO systems

The single-variable DMC is designed based on the following principles:

1. Output prediction is based on the step response model using proportion and superposition properties of linear systems.

2. Rolling optimisation is an online calculation based on optimal output tracking and control input increment suppression.

3. Prediction error and its correlation are based on output measured in real-time.

2.3 DMC for unconstrained MIMO systems

Then we consider the DMC for MIMO systems. Similar to the previous case, the DMC for MIMO systems also contains three parts, the distinction for the MIMO case is that, there are couplings between inputs and outputs, thus, we need to divide the overall system into several subsystems, and then using the superposition property to estimate the total outputs. We will next illustrate the control procedure for the MIMO systems step by step.

2.3.1 Prediction model

Different from SISO systems, to derive the step response of the whole system, we need to consider each input one by one along with their effects on each output. Which means, for a system with m control inputs, and n outputs, we need $m \times n$ step response-based models. We then measure the unit step response from each input u_j to each output y_i as a_{ij}, similar to Eq.2 in SISO systems, we construct a model from u_j to y_i based on its converged model length, and these vectors build up the model vectors for the whole system:

$$\mathbf{a}_{ij}[a_{ij}(1), \dots, a_{ij}(N)]^T, \quad i = 1, \dots, n,\ j = 1, \dots, m \tag{19}$$

where N is the largest model length among all the input output combinations.

Then similar to Eq.14, the prediction for the output y_i when given a new input increment Δu_j is given by

$$\widetilde{\mathbf{y}}_{i,N1}(k) = \widetilde{\mathbf{y}}_{i,N0}(k) + (a)_{ij}\Delta u_j(k), \tag{20}$$

where $\widetilde{\mathbf{y}}_{i,N1}(k) = [\widetilde{\mathbf{y}}_{i,1}(k+1,k), \dots, \widetilde{\mathbf{y}}_{i,1}(k+N,k)]^T$ and $\widetilde{\mathbf{y}}_{i,N0}(k) = [\widetilde{\mathbf{y}}_{i,0}(k+1,k), \dots, \widetilde{\mathbf{y}}_{i,0}(k+N,k)]^T$. Similar to the prediction over the prediction horizon in Eq.4, for MIMO system we have

$$\widetilde{\mathbf{y}}_{i,PM}(k) = \widetilde{\mathbf{y}}_{i,P0}(k) + (A)_{ij}\Delta u_{j,M}(k), \tag{21}$$

where the P is the length of prediction horizon and M is the length of control horizon, and $\mathbf{A}_{ij} \in \mathbb{R}^{P \times M}$ is a matrix constructed by

$$\mathbf{A}_{ij} = \begin{bmatrix} a_{ij}(1) & \dots & a_{ij}(M) & \dots & a_{ij}(P) \\ & \ddots & \vdots & & \vdots \\ \mathbf{0} & & a_{ij}(1) & \dots & a_{ij}(P-M+1) \end{bmatrix}^T. \tag{22}$$

If we consider the effect combining all the inputs from Δu_1 to Δu_M, we then derive the following representation in vector form for Eq.20 and Eq.21

$$\begin{aligned} \widetilde{\mathbf{y}}_{N1} &= \widetilde{\mathbf{y}}_{N0} + \overline{\mathbf{A}}\Delta\mathbf{u}(k) \\ \widetilde{\mathbf{y}}_{PM} &= \widetilde{\mathbf{y}}_{P0} + \mathbf{A}\Delta\mathbf{u}_M(k), \end{aligned} \tag{23}$$

where

$$\overline{\mathbf{A}} = \begin{bmatrix} \mathbf{a}_{11} & \dots & \mathbf{a}_{1m} \\ \vdots & & \vdots \\ \mathbf{a}_{n1} & \dots & \mathbf{a}_{nm} \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \dots & \mathbf{A}_{1m} \\ \vdots & & \vdots \\ \mathbf{A}_{n1} & \dots & \mathbf{A}_{nm} \end{bmatrix}. \tag{24}$$

2.3.2 Rolling optimisation

Similar to Eq.9, the control of the unconstrained MIMO system is also determined by solving an optimisation function, the difference is that, here the variables of interest are all the outputs and inputs, which can be formulated as follow

$$\min_{\Delta \mathbf{u}_M} J(k) = \|\mathbf{w}(k) - \widetilde{\mathbf{y}}_{PM}(k)\|^2_{\mathbf{Q}} + \|\Delta \mathbf{u}_M(k)\|^2_{\mathbf{R}}, \tag{25}$$

the $\widetilde{\mathbf{y}}_{PM}(k) \in \mathbb{R}^{nP}$ is a long column vector contains all the output within the prediction horizon, where n is the number of the outputs, and P is the prediction horizon. $\mathbf{w}(k)$ on the other hand, is the reference values

$$\begin{aligned} &\mathbf{w}_i(k) = [w_i(k+1), \ldots, w_i(k+P)]^T, \quad i = 1, \ldots, n \\ &\mathbf{w}(k) = [\mathbf{w}_1(k), \ldots, \mathbf{w}_n(k)]^T, \end{aligned} \tag{26}$$

and the $\mathbf{Q}$ and $\mathbf{R}$ again are error weighting matrix and control weighting matrix respectively as in Eq.9. But, for MIMO systems, they have the new forms of

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 & & \\ & \ddots & \\ & & \mathbf{Q}_n \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} \mathbf{R}_1 & & \\ & \ddots & \\ & & \mathbf{R}_m \end{bmatrix} \tag{27}$$

The $\mathbf{Q}_i$ and $\mathbf{R}_i$ have the same structure with Eq.10, representing proprietary weights for different inputs and outputs. Solving the optimisation problem in Eq.25 we derive the optimal control input $\Delta \mathbf{u}_M(k) \in \mathbb{R}^{mM}$ at time t_k as

$$\Delta \mathbf{u}_M(k) = \left(\mathbf{A}^T\mathbf{Q}\mathbf{A} + \mathbf{R}\right)^{-1} \mathbf{A}^T\mathbf{Q}[\mathbf{w}(k) - \widetilde{\mathbf{y}}_{P0}(k)], \tag{28}$$

which echos with that of Eq.11 for unconstrained SISO systems. Then we select out the current control increment by multiplying a matrix $\mathbf{L} \in \mathbb{R}^{m \times mM} = \begin{bmatrix} \mathbf{c}^T & & \\ & \ddots & \\ & & \mathbf{c}^T \end{bmatrix}$, where $\mathbf{c}$ is the row vector in Eq.12, and we have

$$\begin{aligned} \Delta \mathbf{u}(k) &= \mathbf{L}\Delta \mathbf{u}_M(k) \\ &= \mathbf{D}[\mathbf{w}(k) - \widetilde{\mathbf{y}}_{P0}(k)], \end{aligned} \tag{29}$$

where $\mathbf{D} = \mathbf{L}\left(\mathbf{A}^T\mathbf{Q}\mathbf{A} + \mathbf{R}\right)^{-1} \mathbf{A}^T\mathbf{Q}$, which can be calculated offline. Through Eq.29, we can calculate the m control input for the system at each time step.

2.3.3 Feedback correction

Again, after applying the control input $\Delta \mathbf{u}(k)$, we can derive the system output output at t_{k+1}, then correcting all the predictions by

$$\widetilde{\mathbf{y}}_{cor} = \widetilde{\mathbf{y}}_{N1} + \mathbf{H}\mathbf{e}(k+1), \tag{30}$$

which is similar to that of Eq.16 in SISO, but here the $\mathbf{e}(k+1) \in \mathbb{R}^n$ is a n-dimensional vector contains the error w.r.t. actual values for all the system outputs. And for the weight matrix $\mathbf{H}$ we have

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_{11} & \dots & \mathbf{h}_{1n} \\ \vdots & & \vdots \\ \mathbf{h}_{n1} & \dots & \mathbf{h}_{nn} \end{bmatrix}, \mathbf{h}_{ij} = \begin{bmatrix} h_{ij}(1) \\ \vdots \\ h_{ij}(N) \end{bmatrix}, \; i, j = 1, \dots, n \tag{31}$$

Finally, similar to Eq.18, we shift the system value by 1 timestamp, and get ready for the next control iteration by multiplying a shifting matrix $\mathbf{S}_0$

$$\widetilde{\mathbf{y}}_{N0}(k+1) = \mathbf{S}_0 \widetilde{\mathbf{y}}_{cor}(k+1), \tag{32}$$

where for MIMO systems, $\mathbf{S}_0 \in \mathbb{R}^{nN \times nN} = \begin{bmatrix} \mathbf{S} & & \\ & \ddots & \\ & & \mathbf{S} \end{bmatrix}$, $\mathbf{S}$ is the matrix in Eq.17.

The overall procedure of DMC for MIMO systems is illustrated in Figure2, which is the process we adopted in the following implementation.

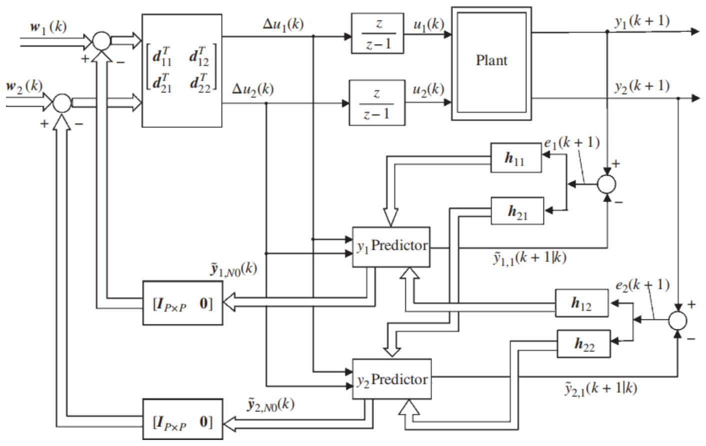

Figure 2: DMC diagram for unconstrained MIMO systems

3 Implementation

DMC algorithm consists of three parts: prediction model, rolling optimisation and, feedback correction.

3.1 System model

Here we pre-define a double input-double output MIMO system using state space representation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{33}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{34}$$

Which can be written in vector form as $\begin{cases} \dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \\ \mathbf{y} = C\mathbf{x} + D\mathbf{u}. \end{cases}$ For DMC controller, the system to be controlled needs to be a stable system *(i.e. all the eigenvalues of matrix A will not have negative real parts)*, therefore, for demonstration, we set our pre-defined model as follow:

$$\begin{aligned} A_{sys} &= \begin{bmatrix} -2 & -1 \\ 6 & -4 \end{bmatrix} \qquad B_{sys} = \begin{bmatrix} 4 & -2 \\ 6 & 3 \end{bmatrix} \\ C_{sys} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad D_{sys} = \begin{bmatrix} -1 & -2 \\ -1 & 1 \end{bmatrix} \end{aligned} \tag{35}$$

Since the DMC is based on the step response model, we first use the MATLAB inbuilt command `step()` to derive the step response model of the system from each input to each output. Below are the results from the step response with the sampling time $T_s = 0.1s$.

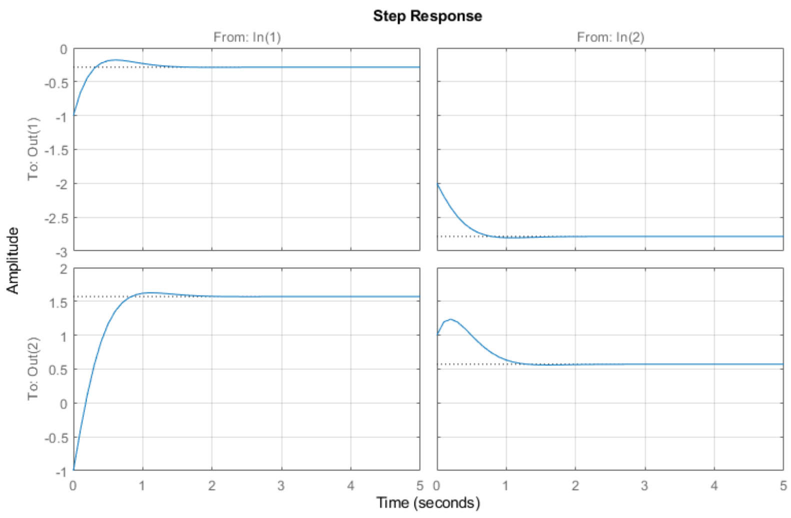

Figure 3: Step response of each input (u_j) to each output (y_i)

From the individual step response of each input to each output in Figure3, we can see that, the system we constructed is asymptotically stable, since all the step responses converge, and they all reach their steady value a_∞ within sample length $N = 50$, thus, our model length is set to $N = 50$.

3.2 System simulation (Runge-Kutta Method)

The system is simulated using the 4^{th} order Runge-Kutta method. We use the `ode45(F, [0, Ts], x0)` function over each timestamp to calculate the value of the state variables after each control input $\Delta\mathbf{u}_M$, where `F` is the differential equation we use to simulate the state evolution, which is obtained from the state space model in Eq.33, `Ts` is the sampling period, and `x0` is the current state values.

During each iteration, the timestamp (simulation time), the current state variables (initial values), and the current control are inputs to the function. The function returns the state variables with the new value calculated using `ode45()` after each sampling period.

3.3 Offline calculation of matrix D

To construct matrix D in Eq.29, we construct its containing matrices (A, S, H, L) based on the system model and the number of input-output signals. Then pre-calculated the value of matrix D before the online controlling process.

For error weighting matrix $\mathbf{Q}$ and control weighting matrix $\mathbf{R}$, after some gain tuning process, we fixed the error weighting matrix $\mathbf{Q}$ to be $\mathbf{Q} \in \mathbb{R}^{nP\times nP} = \begin{bmatrix} 3 & & \\ & \ddots & \\ & & 3 \end{bmatrix} \oplus \begin{bmatrix} 4 & & \\ & \ddots & \\ & & 4 \end{bmatrix}$, where n is the number of outputs, and P is the prediction horizon. In this way we fix the weighting for the output error of our two system outputs to be 3 and 4 respectively, which guarantees the system to be stable within 4 seconds (with overshoot less than 10% when $\mathbf{R} =$`diag(5)`). Then we tried different control weighting for the $\mathbf{R}$ matrix, and the results are analysed in the next section.

After deciding the $\mathbf{Q}$ and $\mathbf{R}$ matrix, we calculate the matrix D offline by

$$\mathbf{D} = \mathbf{L}\left(\mathbf{A}^T\mathbf{Q}\mathbf{A} + \mathbf{R}\right)^{-1}\mathbf{A}^T\mathbf{Q}. \tag{36}$$

3.4 Online update and correction

Using the matrix D computed above, we can perform our online DMC control following the procedure showed in Figure2. The values of system outputs values $[y_1, y_2]^T$ are stored in a $nN \times 1$ column vector `YN` where n is the number of outputs, and N is the length of model vector which is set to 50 in our simulation.

4 Results

The controlled results are shown in the following plots, where the two orange plots on the LHS are inputs u_1 and u_2 under the DMC control, the corresponding blue plots on the RHS are the system output values, and the reference value is highlighted in red dotted line for each output.

4.1 Identical control weightings

As mentioned in Section 3, we first set the control weighting $\mathbf{R}$ for each input to be identical (i.e. $\mathbf{R_1} = \mathbf{R_2}$), and analyse the impact of the control weighting to the system. Note that, the error weighting matrix is fixed to a constant value mentioned in Eq.36.

4.1.1 Control weighting $r = 1$ ($\mathbf{R} = \mathbf{I}_{mM \times mM}$)

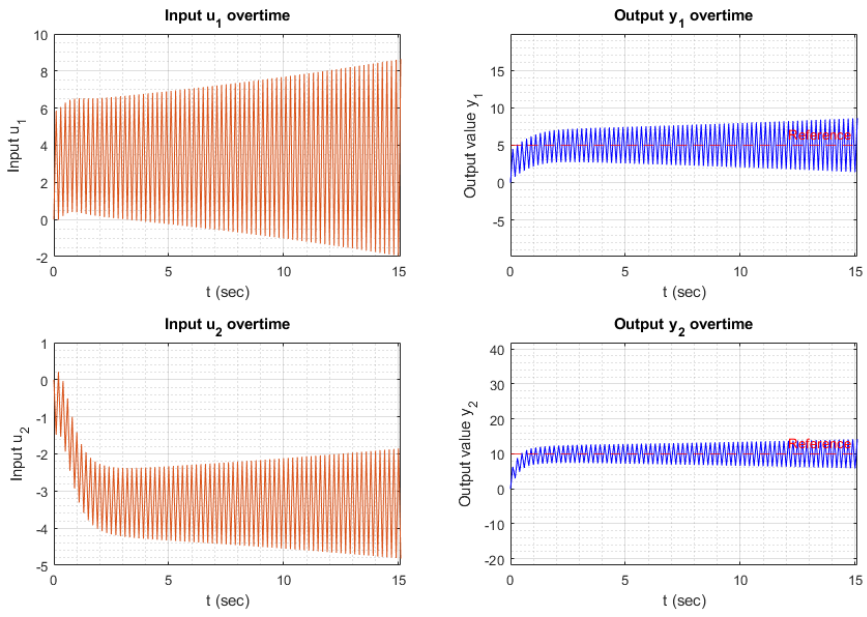

Figure 4: Input and output values under the DMC control with control weighting $\mathbf{R} = \mathbf{I}_{mM \times mM}$

We can see that, when control weighting is relatively small, the system is even unstable, and enormous input changing rate can be observed from the beginning. The high rate of change results in intense oscillation that eventually caused the system to blow up. This can be fixed when we add a little more constraint on the input rate which will be explicated next.

4.1.2 Control weighting $r = 5$ ($\mathbf{R} = 5 \cdot \mathbf{I}_{mM \times mM}$)

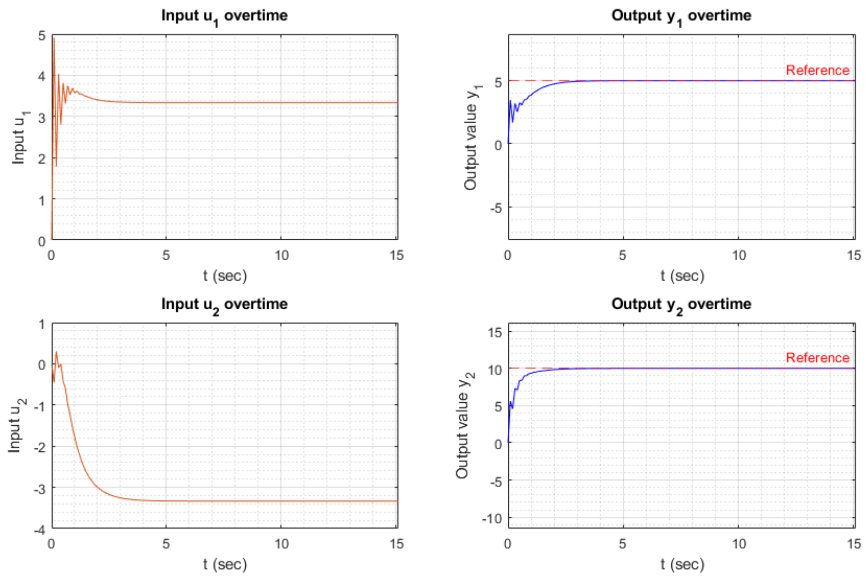

Figure 5: Input and output values under the DMC control with control weighting $\mathbf{R} = 5 \cdot \mathbf{I}_{mM \times mM}$

By penalising all the control inputs a little bit more (from 1 to 5), we can stabilise our system, although it still experiencing a mild oscillating behaviour, it shortly converges to the reference values. This oscillation here is actually due to a dynamic process that when there occurs a strong input change, the optimiser in Eq.25 will suppress the input behaviour, the outputs in return will not reach their optimal at this timestamp, thus, for the next timestamp the impact of the output errors will get larger, pushing the control input to make a substantial change. As the process iterating, this results in an oscillating behaviour.

4.1.3 Control weighting $r = 10$ ($\mathbf{R} = 10 \cdot \mathbf{I}_{mM \times mM}$)

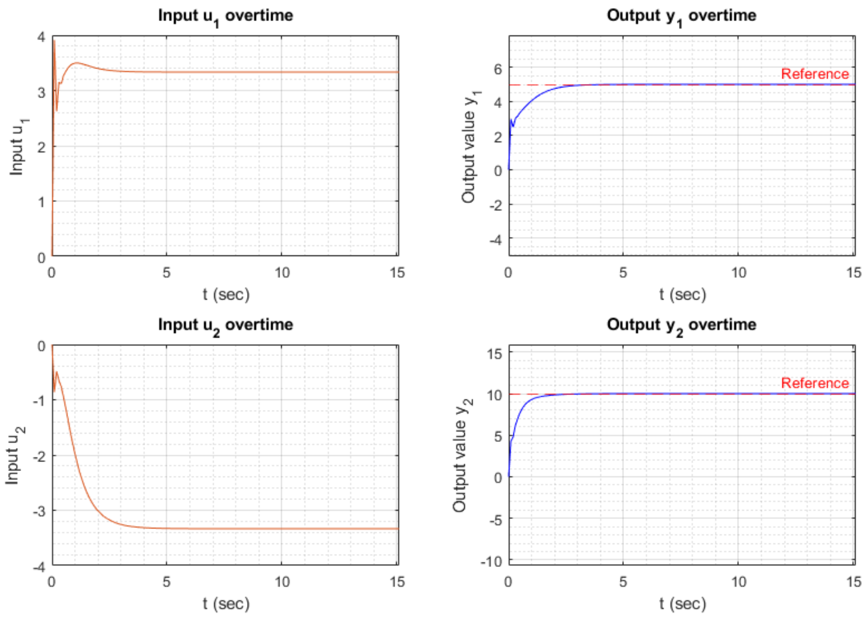

Figure 6: Input and output values under the DMC control with control weighting $\mathbf{R} = 10 \cdot \mathbf{I}_{mM \times mM}$

When we keep increasing the weighting on controls, we can see the system behaviour becomes more stable. It can be seen from the figure above, both input and output converge within 5 seconds (or 50 iterations with $T_s = 0.1$), and the output curves are relatively smooth and perfectly match the reference value its the steady state. Also, due to the suppression on the control input, there is no striking peak at the beginning of the control process, and goes smoothly, while the little drawback might be the slightly slower convergence speed compare to the adjacent case above.

4.2 Independent control weightings

The above scenarios consider identical weighting for all the control inputs, here we set different weightings for different control inputs, specifically, we set $\mathbf{R} = 5 \cdot \mathbf{I}_{M \times M} \oplus 10 \cdot \mathbf{I}_{M \times M}$. From Figure7 we can see that the input

u_1 and output y_2 experiencing a mild oscillating behaviour and, while the input u_2 and output y_1 are relatively smooth, this indicates the coupling relations between the inputs and outputs in the MIMO system, which is the substantial distinction to that of the SISO system.

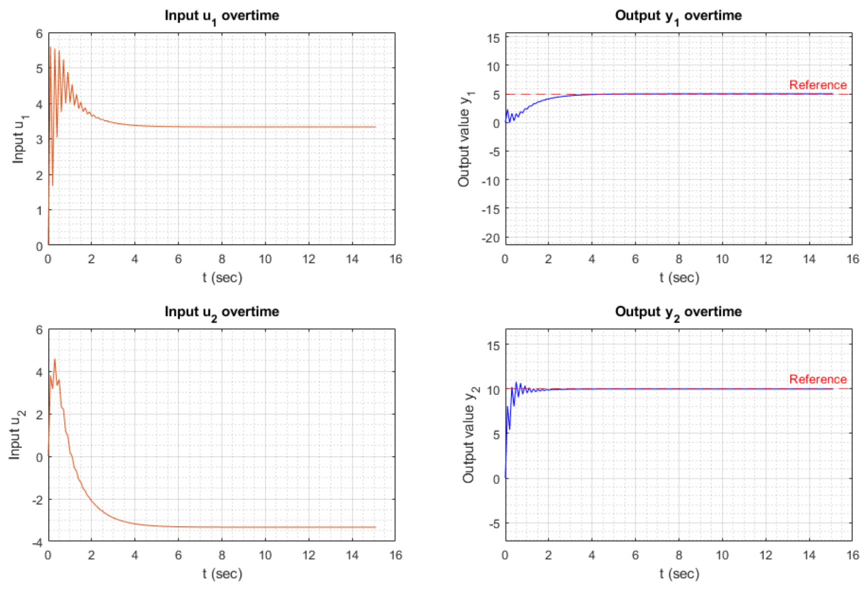

Figure 7: Input and Output values under the DMC control

5 Codes

The implementation of our proposed MIMO DMC control system is shown in the code section below. Where in this demo, a double input-double output continuous-time system is defined using state space model and then simulated using the 4th order *Runge-Kutta Method*.

```
%% MIMO DMC simulation
% —— Author: Dingran Yuan
% —— Date: 29/03/2022

clearvars
```

```
clc;

%% Model simulation (MIMO state space model)
A_sys = [-2,-1;6,-4];
B_sys = [4,-2;6,3];
C_sys = [1,0;0,1];
D_sys = [-1,-2;-1,1];

sys = ss(A_sys,B_sys,C_sys,D_sys);
step(sys, 0:0.1: 50*0.1)
grid on
grid minor

%% Initialise parameters
Ts = 0.1; % Sampling time
P = 20; % Prediction horizon
M = 5; % Control horizon
N = 50; % Model length

[a_step, t_step] = step(sys, 0:Ts:50 - Ts); % Step
    response of the system

nIn = size(a_step, 3); % number of inputs
nOut = size(a_step, 2); % number of outputs

a11 = a_step(1:50,1,1);
a12 = a_step(1:50,1,2); % Step response from in(2) to
    out(1)
a21 = a_step(1:50,2,1); % Step response from in(1) to
    out(2)
a22 = a_step(1:50,2,2);
% plot(t_step(1:50),a_step(1:50,2,1))

A11 = zeros(P, M);
A12 = zeros(P, M);
A21 = zeros(P, M);
A22 = zeros(P, M);

```

```
% coloum matrix for YN & YP
YN = zeros(nOut*N, 1);
YP = zeros(nOut*P, 1);

% Elements used to compute D matrix
L = zeros(nIn, nIn*M);
S = zeros(nOut*N, nOut*N);
Q = zeros(nOut*P, nOut*P);
R = zeros(nIn*M, nIn*M);
H = zeros(nOut*N, nOut);

% Construct A matrices
for i = 1: P
    for j = 1: M
        if(i-j+1 > 0)
            A11(i, j) = a11(i - j + 1);
            A12(i, j) = a12(i - j + 1);
            A21(i, j) = a21(i - j + 1);
            A22(i, j) = a22(i - j + 1);
        end
    end
end
A = [A11, A12; A21, A22];
AN = [a11, a12; a21, a22];

% weight for system error and input (here we penalise
    the error more)
Q_arg = [3 4]; % weight for system error
R_arg = [5 5]; % weight for control input
Out_ref = [5 10]; % reference of the output
wr = zeros(nOut * P, 1);

% Construct wr & Q & S & R & L
for i = 1:nOut
    % Construct wr
    wr((i-1)*P+1: i*P) = Out_ref(i) * ones(P, 1);

    % Construct Q
    Q((i-1)*P+1:i*P, (i-1)*P+1:i*P) = Q_arg(i)*eye(P);

    % Construct S
    S((i-1)*N+1:i*N-1, (i-1)*N+2:i*N) = eye(N-1);
    S(i*N, i*N) = 1;

    % Construct H
    H((i-1)*N + 1:i*N, i) = ones(N, 1);
end
for i = 1:nIn
    % Construct R
    R((i-1)*M+1:i*M, (i-1)*M+1:i*M) = R_arg(i)*eye(M);

    % Construct L
    L(i,(i-1)*M + 1) = 1;
end

% Offline computation of matrix D
D = L / (A'*Q*A+R)*A'*Q;

%% MIMO DMC loop

t_sim = 15.0; % Simulation time
mpc_time = 0.0;
u = [0;0]; % Control signal
X_curr = [0;0]; % initial state

Y_array = C_sys*X_curr + D_sys*u; % Store output for
    plotting
Y1_min = Y_array(1); Y1_max = Y_array(1);
Y2_min = Y_array(2); Y2_max = Y_array(2);
u_array = [0;0];
figure

while(mpc_time < t_sim)
    for i = 1:nOut
        YP((i-1)*P+1:i*P) = YN((i-1)*N+1:(i-1)*N+P);
    end
    du = D * (wr - YP);

    YN1 = YN + AN * du;
    Y1 = [YN1(1);YN1(N+1)]; % The current estimation
        after control input

    Y1_min = min(Y1(1), Y1_min); Y1_max = max(Y1(1),
        Y1_max);
    Y2_min = min(Y1(2), Y2_min); Y2_max = max(Y1(2),
        Y2_max);

    u = u + du;
    [X_curr, Y_curr] = RKsolver(sys, u, X_curr, Ts);
    e1 = X_curr - Y1; % calculate the error with the
        real output
    Y_cor = YN1 + H*e1;
    YN = S * Y_cor;

    Y_array = [Y_array, [YN(1);YN(N+1)]];
    u_array = [u_array, u];
    mpc_time = mpc_time + Ts;

    %% visualise by step
    subplot(2,2,2)
    plot(linspace(0, mpc_time, size(Y_array,2)),
        Y_array(1,:), 'b-');
    ylim([Y1_min - 0.4*(Y1_max - Y1_min), Y1_max +
        0.4*(Y1_max - Y1_min)])
    grid on
    grid minor
    yline(Out_ref(1), 'r--', 'Reference')
    xlabel('{t} (sec)');
    ylabel('Output value {y_1}');
    title('Output {y_1} overtime')

    subplot(2,2,1)
    plot(linspace(0, mpc_time, size(Y_array,2)),
        u_array(1,:),"Color",'#D95319');
    grid on
    grid minor
    xlabel('{t} (sec)');
    ylabel('Input {u_1}');
    title('Input {u_1} overtime')

    subplot(2,2,4)
    plot(linspace(0, mpc_time, size(Y_array,2)), Y_array(2,:), 'b-');
    ylim([Y2_min - 0.4*(Y2_max - Y2_min), Y2_max + 0.4*(Y2_max - Y2_min)])
    grid on
    grid minor
    xlabel('{t} (sec)');
    ylabel('Output value {y_2}');
    yline(Out_ref(2),'r--','Reference')
    title('Output {y_2} overtime')

    subplot(2,2,3)
    plot(linspace(0, mpc_time, size(Y_array,2)), u_array(2,:), "Color",'#D95319');
    grid on
    grid minor
    xlabel('{t} (sec)');
    ylabel('Input {u_2}');
    title('Input {u_2} overtime')

    drawnow
end

%% Runge-Kutta method for simulation
function [x1, y1] = RKsolver(sys, u, x0, Ts)
%--
%-> u: the input of the system
%-> sys: the state space model of the system
%<- Fu: the output differential equition
    A = sys.A; B = sys.B;
    C = sys.C; D = sys.D;
    F = @(t, x) A*x + B*u;
    [~, X_tmp] = ode45(F, [0, Ts], x0);
%       plot(t, X_tmp)
    x1 = X_tmp(end,:)';
    y1 = C*x1 + D*u;
end

%% Appendix codes
% Construct the output reference matrix
% for i = 1:nOut
%       wr((i-1)*P+1: i*P) = Out_ref(i) * ones(P, 1);
% end

% Construct L matrix
% for i = 1:nIn
%       L(i,(i-1)*M + 1) = 1;
% end

% Construct S matrix
% for i = 1:nOut
%       S((i-1)*N+1:i*N-1, (i-1)*N+2:i*N) = eye(N-1);
%       S(i*N, i*N) = 1;
% end

% Construct H matrix
% for i = 1:nOut
%       H((i-1)*N + 1:i*N, i) = ones(N, 1);
% end
```