NATIONAL UNIVERSITY OF SINGAPORE
COLLEGE OF DESIGN AND ENGINEERING

Mini-Project for Linear Systems
(EE5101)

CONTROL SYSTEM DESIGN OF A CONTINUOUS-FLOW STIRRED TANK REACTOR (CSTR)

Student: Xu, Yimian
Matriculation Number: A0295779Y *Email: ee1349917@u.nus.edu*

2024.11.17

CONTENTS

ABSTRACT

In this mini-project, a control system is designed for the Continuous-Flow Stirred Tank Reactor (CSTR). The temperature of the CSTR is controlled by using different control strategies: Pole Placement, LQR, and so on. We will target both the regulation and set point tracking problems. The report illustrates the result of the project tasks and discusses on the effects of the control strategies and changing parameters. There are *6* tasks of the project, shown as following:

1. A state feedback controller using the pole placement method.
2. A state feedback controller using the LQR method.
3. A state observer and the resultant observer-based LQR control system.
4. A decoupling controller with closed-loop stability and the resultant control system.
5. A controller such that the plant can operate around the set point of outputs at steady state.
6. A controller such that the plant can operate around the set point of states at steady state.

1 INTRODUCTION

1.1 Background

we are trying to control the reaction process that takes place in a tank shown in the following Figure 1. The chemical reaction is described by A → B. A and B stand for the reactant and product. The reaction is conducted in the large container located at the central of the tank, meanwhile there is water flow inside the surrounded container wall (we call it cooling jacket) to control the reaction temperature.

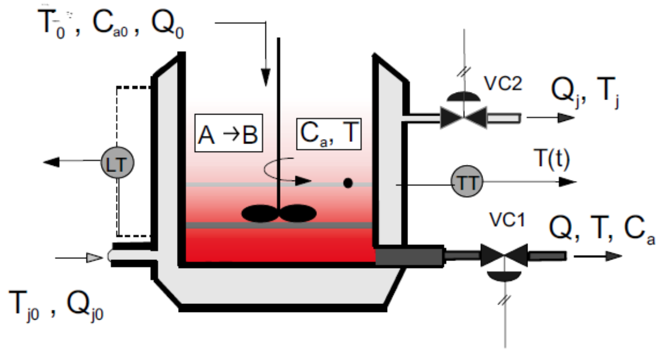


Figure 1 — CSTR reactor

1.2 Modeling

For model-based control, the first step is to build an effective dynamic model for our target plant, i.e., the continuous-flow stirred tank reactor (CSTR) in this project. The system can be simplified to a 3rd order LTI state-space model with 2 inputs and 3 outputs, as is presented in [1].

In this project, the most important factors that we want to control is $\mathbf{C_a}$ - the component concentration of the reactant during reaction and the reaction temperature. Only temperature sensors are available, so we can detect $\mathbf{T}$ - the reaction temperature in the reaction container and $\mathbf{T_j}$ - the outflow water temperature in the cooling jacket outlet pipe.

Our objectives is to use 2 variables, $\mathbf{F}$ - the outlet flow rate of the reaction and $\mathbf{F_j}$ - the flow rate of the water coming out from the cooling jacket to maintain the whole reaction on the given operation point.

Define the state vector $x = [C_a \quad T \quad T_j]^T$,the control signal $u = [F \quad F_j]^T$, the measurement vector $y = [T \quad T_j]^T$. The system is described by

$$\begin{aligned} \dot{x} &= Ax + Bu + Bw, \\ y &= Cx \end{aligned} \tag{1}$$

where

$$\begin{aligned} A &= \begin{bmatrix} -1.7 & -0.25 & 0 \\ 23 & -30 & 20 \\ 0 & -200 - ab0 & -220 - ba0 \end{bmatrix}, \\ B &= \begin{bmatrix} 3 + a & 0 \\ -30 - dc & 0 \\ 0 & -420 - cd0 \end{bmatrix}, \\ C &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2}$$

where the vector w describes some possible load disturbances. The initial condition of the system is assumed to be

$$x_0 = [1 \quad 100 \quad 200]^T. \tag{3}$$

My matriculation number is A0295779Y, so $a = 5, b = 7, c = 7, d = 9$, then matrices is defined as:

$$A = \begin{bmatrix} -1.7 & -0.25 & 0 \\ 23 & -30 & 20 \\ 0 & -770 & -950 \end{bmatrix}, B = \begin{bmatrix} 8 & 0 \\ -127 & 0 \\ 0 & -1210 \end{bmatrix} \tag{4}$$

2 POLE PLACEMENT METHOD

2.1 State feedback controller design

To obtain proper poles, the transient response should be considered. In this project, the transient response performance specifications for all the outputs y in state space model 1 are as follows:

1. The overshoot is less than 10%.
2. The 2% settling time is less than 30 seconds.

Consider the standard second order system, we can use the following formula to design the desired poles:

$$M_p = e^{-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}} \leq 0.1 \tag{5}$$

$$t_s = \frac{4}{\zeta\omega_n} \leq 30 \quad (\pm 2\%) \tag{6}$$

The transfer function of the system is:

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{7}$$

The poles are solved as:

$$s = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2} \quad \text{(with } 0 < \zeta < 1\text{)} \tag{8}$$

After computing, the range of ζ and $\zeta\omega_n$ should satisfy:

$$\begin{cases} \zeta \geq 0.5912 \\ \zeta\omega_n \geq 0.133 \end{cases} \tag{9}$$

My choice of them is:

$$\begin{cases} \zeta = 0.8 \\ \zeta\omega_n = 0.25 \end{cases} \tag{10}$$

Then, determine the pair of dominant poles as:

$$s = -0.2 \pm j0.15 \tag{11}$$

The 3rd order system can be approximated by 2nd order system by locating the extra pole to be 2-5 times faster than the dominant ones:

$$s = -0.6 \tag{12}$$

Notice: Here, we have not yet taken account of the plant zeros. After simulation, the position of the zero point can be fine-tuned by observing the transient response and performance metrics (e.g., overshoot, rise time).

1. In this part, I choose the desired poles as:

Table 1 — Designed pole position

s_1	-0.2+j0.15
s_2	-0.2-j0.15
s_3	-0.6

2. Check controllability of the plant by computing the rank of controllability matrix W_C:

$$W_c = [B \quad AB \quad A^2B] \tag{13}$$

The result from MATLAB show that W_c is full rank, meaning this system is controllable and Transformation matrix T can be further computed:

```
Controllability matrix Wc:
   1.0e+09 *

    0.0000         0    0.0000         0   -0.0000    0.0000
   -0.0000         0    0.0000   -0.0000    0.0018    0.0237
         0   -0.0000    0.0001    0.0011   -0.0960   -1.0734

Rank(Wn)=3, the system is controllable, let's perform full-rank pole placement.
```

Figure 2 — Rank of W_c

3. Compute Transformation matrix T:

   a) For MIMO system, we need to select the n independent vectors out of $n \times m$ vectors from the controllability matrix in the strict order from left to right then group them in a square matrix C. After calculating, matrix C is:

$$C = [b_1 \quad Ab_1 \quad b_2] \tag{14}$$

b) Take out 2 rows from C^{-1} corresponding to the 2 inputs, and form T as:

$$T = \begin{bmatrix} q_2^T \\ q_2^T A \\ q_3^T \end{bmatrix} \tag{15}$$

c) Then the matrix $\bar{B} = TB$, $\bar{A} = TAT^{-1}$, we further design the feedback gain matrix for the controllable canonical form:

$$\bar{K} = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \end{bmatrix} \tag{16}$$

d) Compare the non-trivial rows of the closed-loop matrix $\bar{A} - \bar{B}\bar{K}$ with the desired closed-loop matrix A_d to compute $\bar{K}$, finally, we get $K = \bar{K}T$. The results of MATLAB code is:

```
Cmatrix:
   1.0e+04 *

    0.0008    0.0018         0
   -0.0127    0.3994         0
         0    9.7790   -0.1210

T:
    0.0037    0.0002         0
   -0.0009   -0.0079    0.0047
    0.2996    0.0189   -0.0008


B_bar =

         0         0
    1.0000   -5.6514
         0    1.0000

K:
    0.7866   -3.3106    1.9787
    0.2242    0.0117    1.1632
```

Figure 3 — Results of computing K

2.2 Simulation

Using SIMULINK, we can simulate the state responses to non-zero initial state x_0 with zero external inputs. My model is shown as Figure 4, with designed poles shown in Table 1. The state responses are shown in Figure 5 and the control signals are shown in Figure 6.

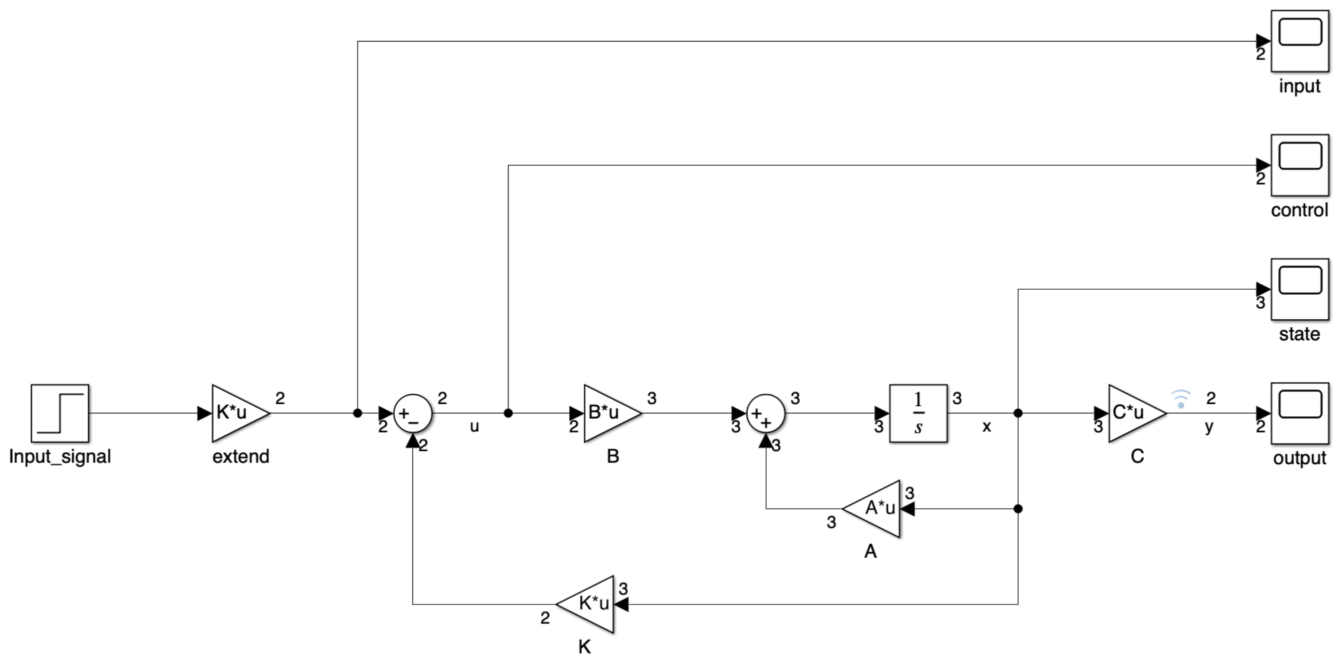


Figure 4 — Pole placement simulation model
(The extend block is a switch to control the input signal, e.g. Gain = [0;0] for zero input)

To check the step responses of the 2 outputs, I apply only one step at a time and keep the other one as zero (r=[1,0], or r=[0,1]), and then observe the behavior of the two outputs. The simulation results are shown in Figure 8 and Figure 9.

The overshoot and settling time are obtained by analysing the output signals with functions implemented from scratch by myself. The results are shown in Figure 7. The step responses of the 2 outputs after pole placement meet the design specifications.

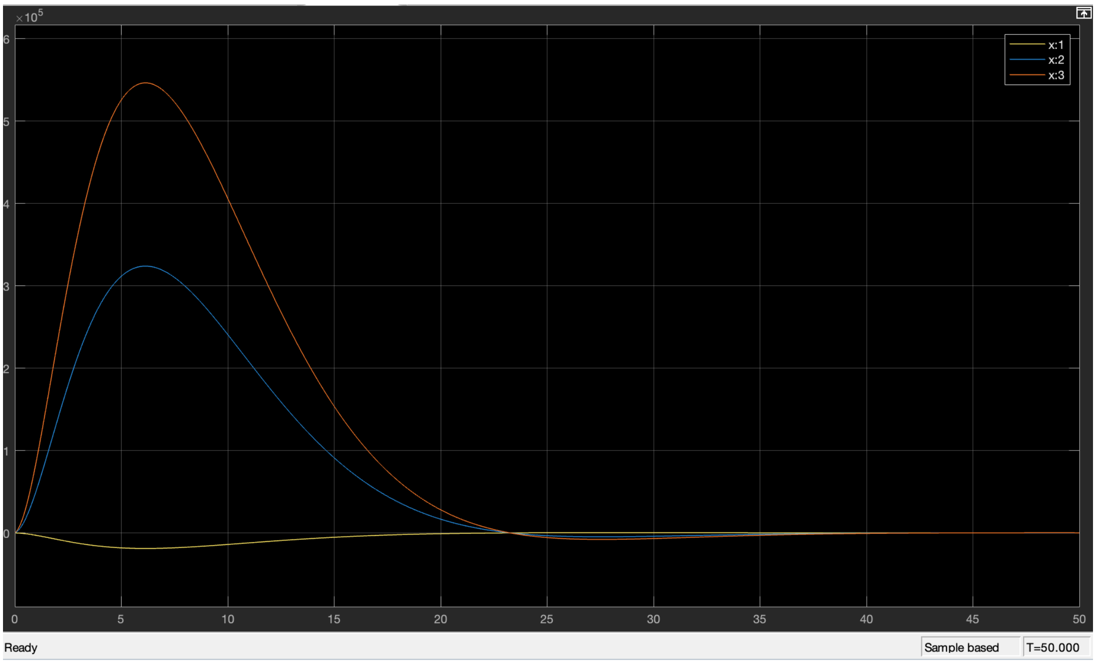


Figure 5 — State responses after pole placement

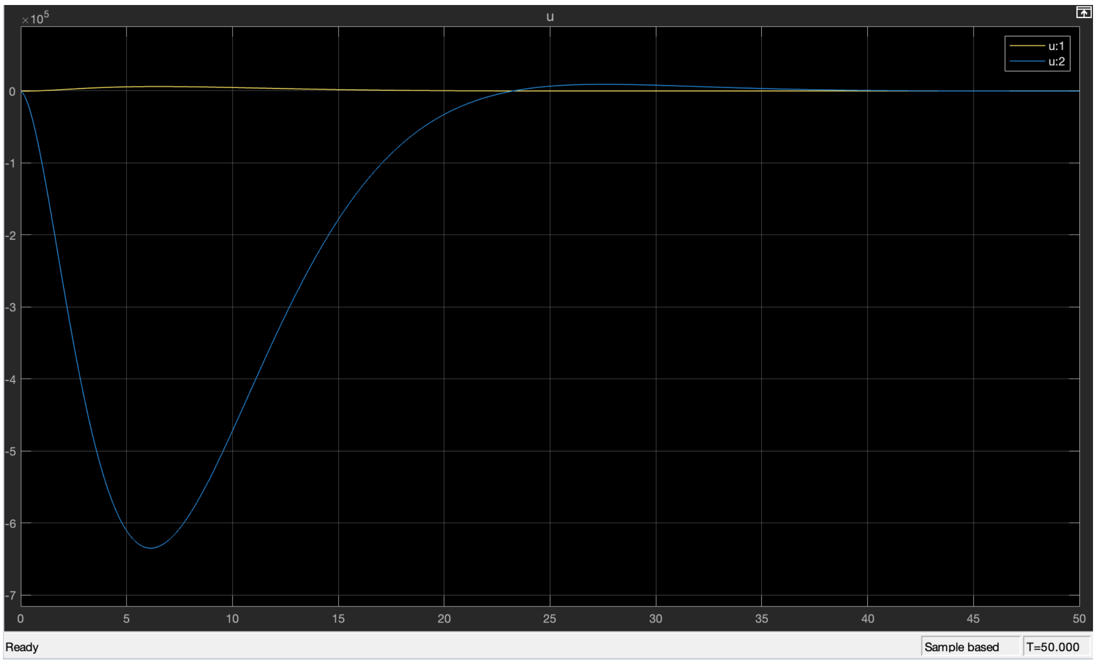


Figure 6 — Control signals after pole placement

```
F=0, F_j=1
y[1] final value:-7194317.022441, settling time:21.760000
y[1] peak value:-7296273.259766, time:21.760000
y[1] overshoot:1.417177%
y[2] final value:-12135414.587385, settling time:21.760000
y[2] peak value:-12307394.443908, time:21.760000
y[2] overshoot:1.417173%
F=1, F_j=0
y[1] final value:1546541.082329, settling time:22.100000
y[1] peak value:1567960.622086, time:22.100000
y[1] overshoot:1.384996%
y[2] final value:2608714.377347, settling time:22.100000
y[2] peak value:2644844.865869, time:22.100000
y[2] overshoot:1.384992%
```

Figure 7 — Transient response performance after pole placement

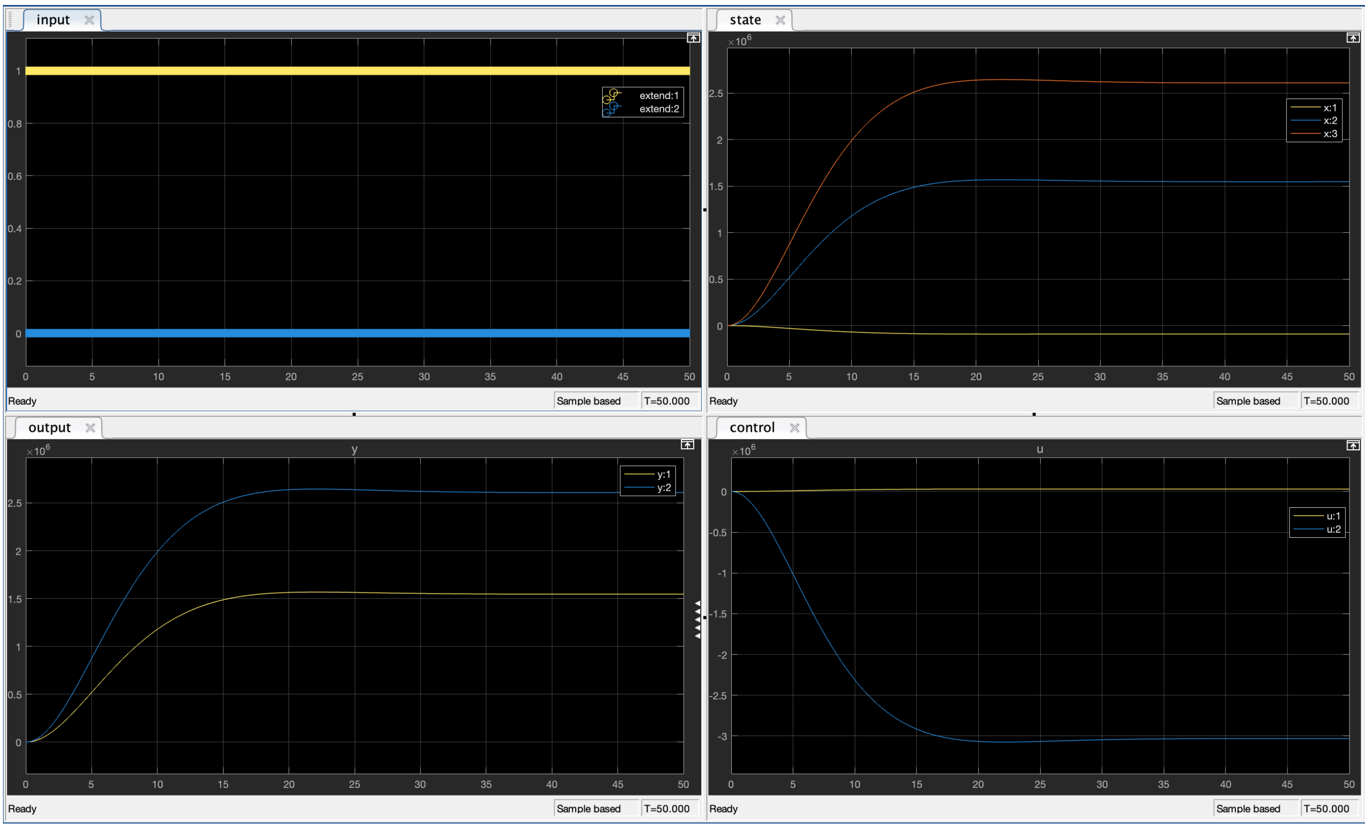


Figure 8 — Pole placement output with non-zero input and $r = [1, 0]$

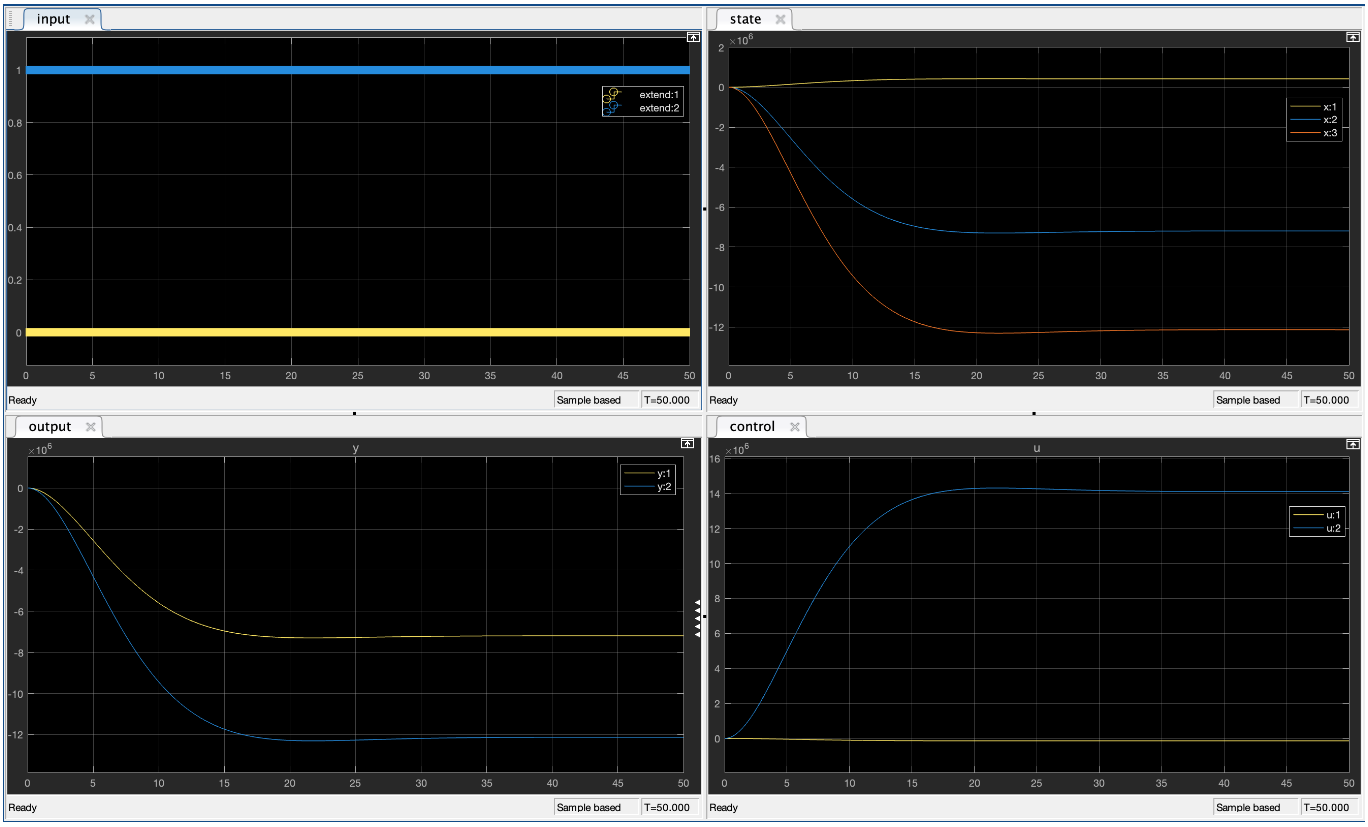


Figure 9 — Pole placement output with non-zero input and $r = [0, 1]$

2.3 Analysis

Change the pole placement as the Table 2 shows, then by simulation, we get the step responses as the Figure 10–13 shows:

Table 2 — Pole placement: changes of the positions of poles

Poles	s_1	s_2	s_3
original	-0.2000 + 0.1500i	-0.2000 + 0.1500i	
test 1	-0.3000 + 0.1500i	-0.3000 + 0.1500i	
test 2	-0.1500 + 0.1500i	-0.1500 + 0.1500i	-6
test 3	-0.2000 + 0.2500i	-0.2000 + 0.2500i	
test 4	-0.2000 + 0.0500i	-0.2000 + 0.0500i	

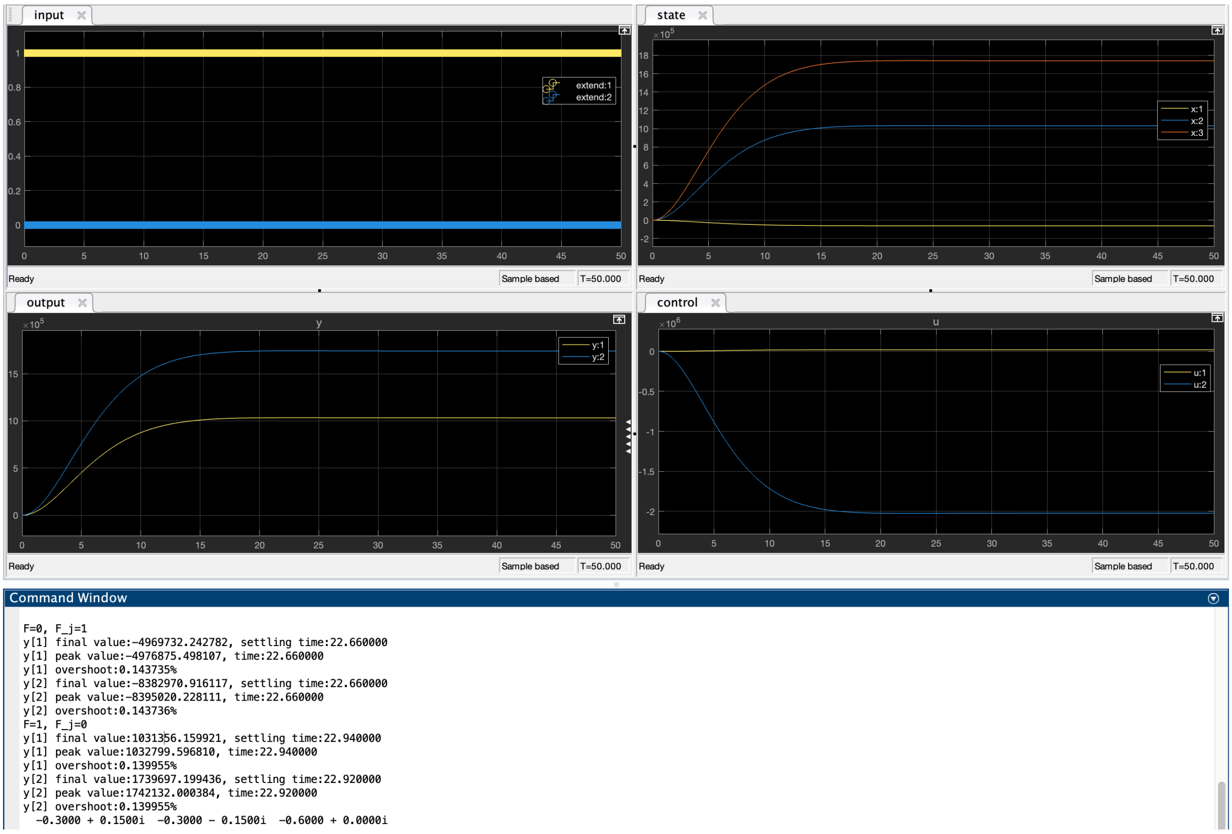


Figure 10 — Step Response for pole change 1

Discussion: In the 2nd order linear systems, the horizontal location of the pole is the reciprocal of the time constant of the exponential decay, so the farther the pole is to the left in the s-plane, the faster the transient response dies out. The vertical location of the pole is the frequency of the oscillations in the response (damped natural frequency ω_d).

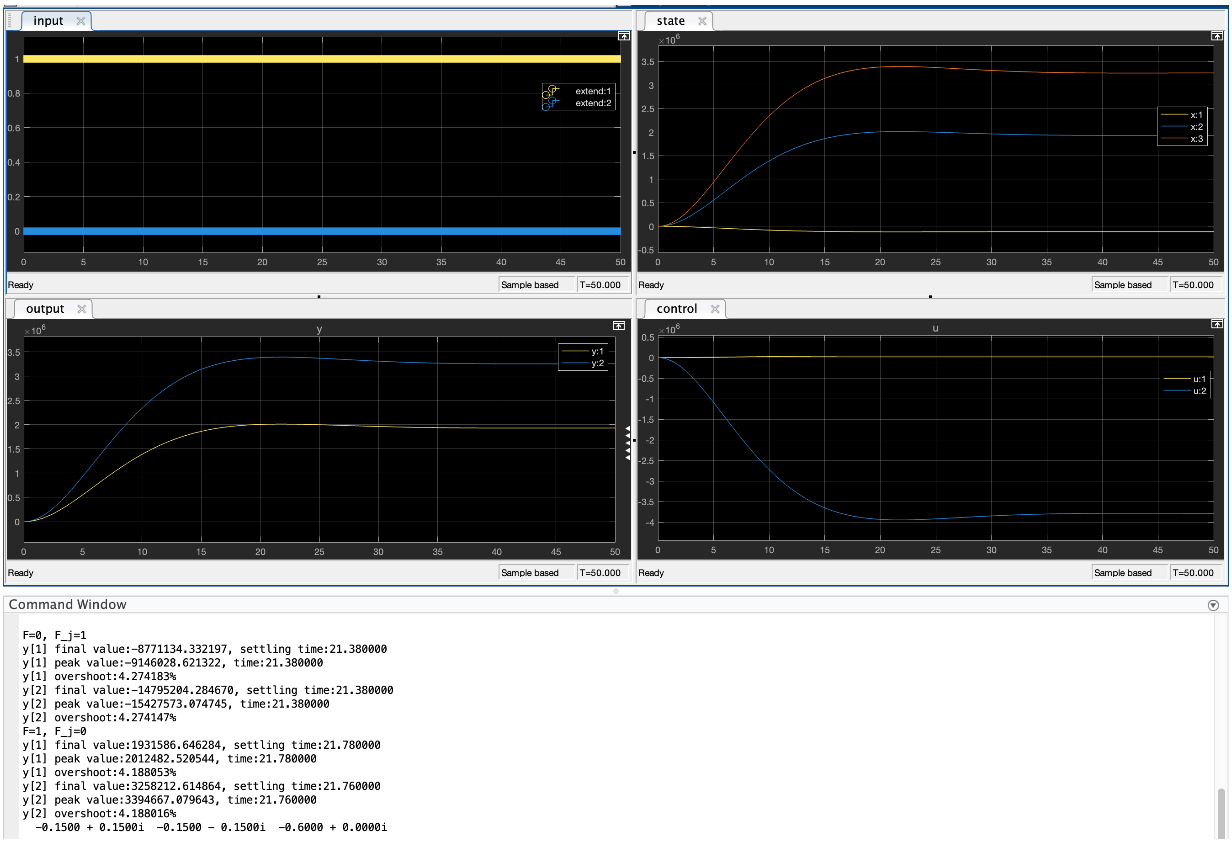


Figure 11 — Step Response for pole change 2

The distance of the pole from the origin in the s-plane is the undamped natural frequency ω_n, the damping ratio is given by $\zeta = \cos\theta$ (θ= Angle of the pole off the horizontal axis)

As the step responses shows, if the absolute values of imaginary part of the poles is larger than those of original poles, the overshoot is larger and settling time is smaller, if smaller, the overshoot becomes smaller and settling time becomes longer. If the absolute values of real part of the poles is larger than those of original poles while the imaginary part unchanged, the damping ratio ζ is smaller, the overshoot is larger and settling time is smaller, if we only decrease the absolute values of real part of the poles, then the damping ratio will become smaller, the overshoot becomes smaller and settling time becomes longer.

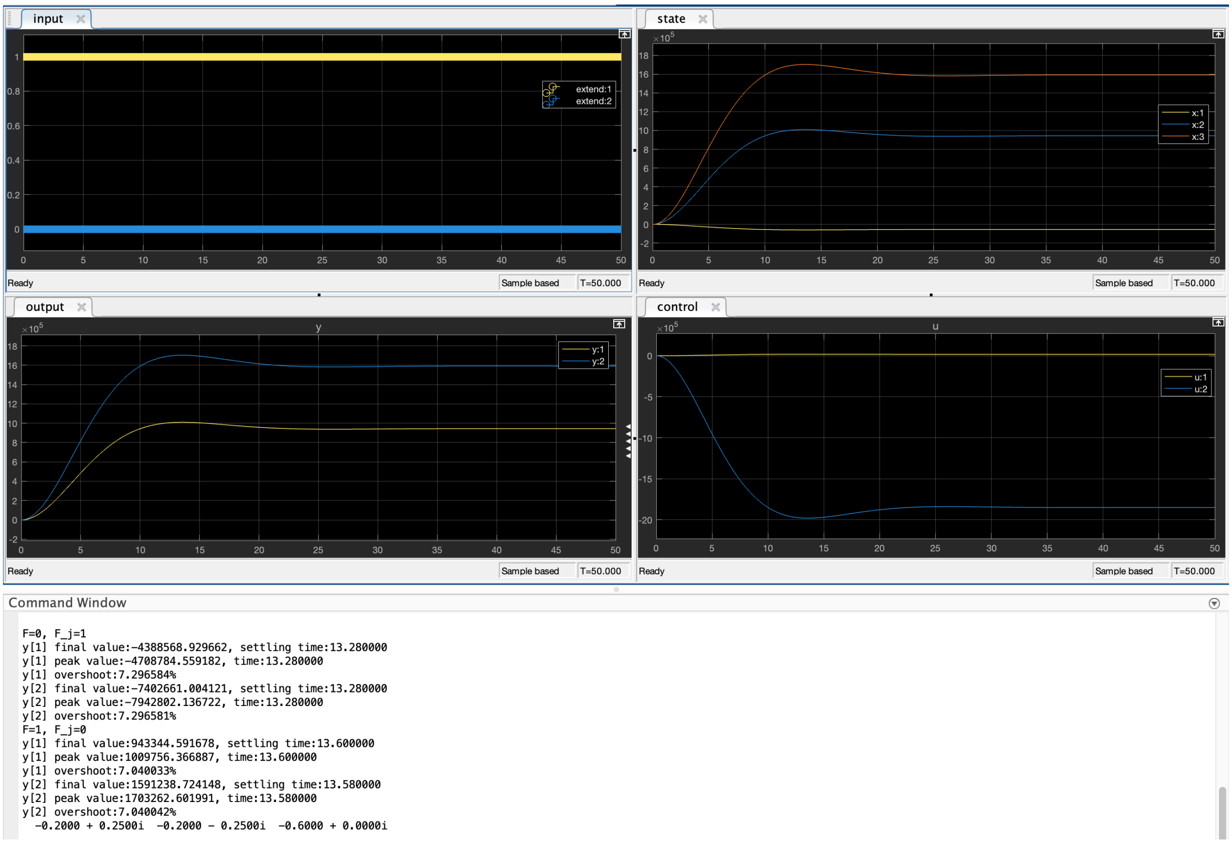


Figure 12 — Step Response for pole change 3

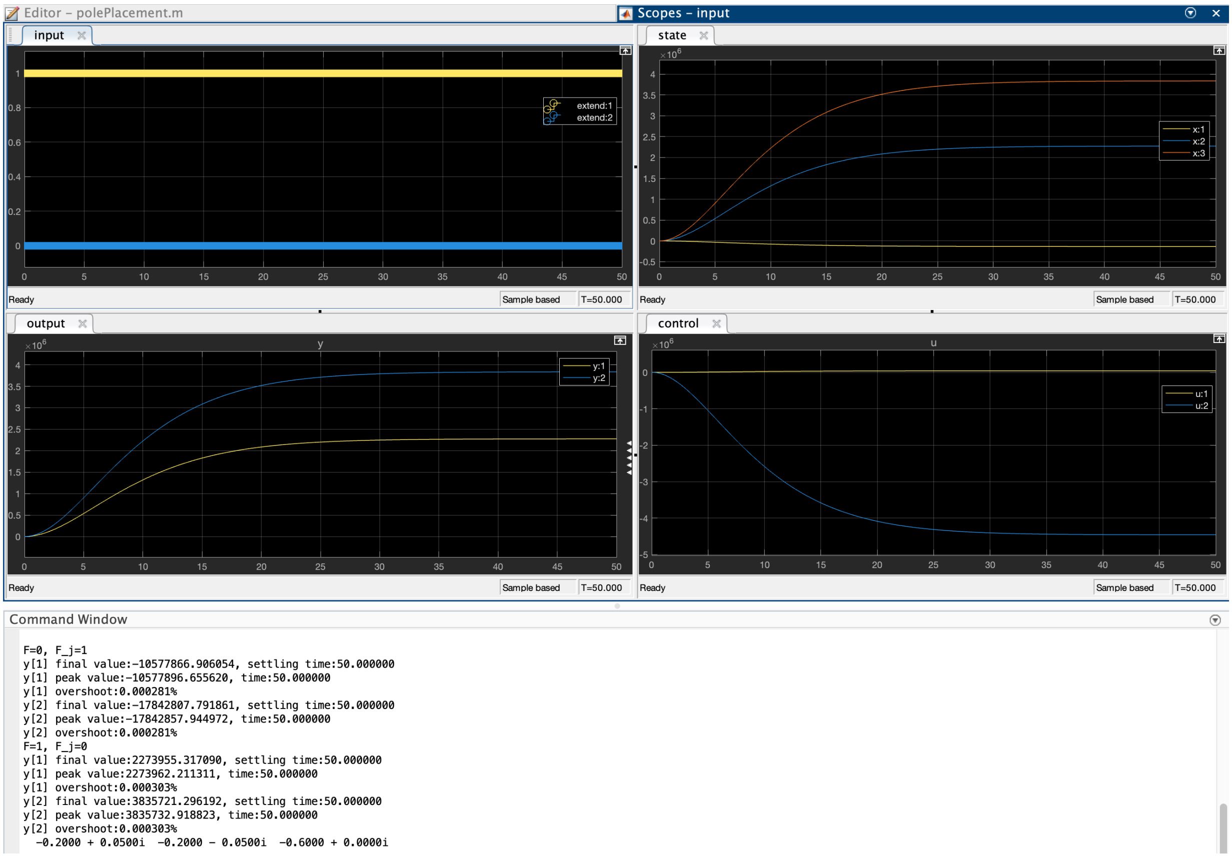


Figure 13 — Step Response for pole change 4

3 LQR METHOD

3.1 State feedback controller design

Assume that I can measure all the state variables, then I can design a state feedback controller using the LQR method. Choose the weight matrices Q and R as:

$$Q = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}, R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{17}$$

The quadratic cost function should be minimized:

$$J = \frac{1}{2}\int_0^\infty \left(x^T Q x + u^T R u\right) dt \tag{18}$$

The optimal control law:

$$u = -Kx + r \tag{19}$$

Try the Lyapunov method:

$$V(x) = x^T P x \tag{20}$$

Where P is a positive definite solution of the Algebraic Riccati Equation (ARE), which is:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{21}$$

To solve the ARE equation, a systematic way using an eigenvalue-eigenvector-based algorithm is usually used. First, define a $2n \times 2n$ matrix Γ:

$$\Gamma = \begin{bmatrix} A & -BR^{-1}B^T \\ -Q & -A^T \end{bmatrix} \tag{22}$$

Find n stable eigenvalues of Γ:

$$\begin{bmatrix} v_i \\ u_i \end{bmatrix}, i = 1, 2, ..., n \tag{23}$$

Then re-organize the eigenvectors of Γ to solve matrix P:

$$P = [u_1, ..., u_n][v_1, ..., v_n]^{-1} \tag{24}$$

K1 | gamma

6x6 double

	1	2	3	4	5	6
1	-1.7000	-0.2500	0	-64	1016	0
2	23	-30	20	1016	-16129	0
3	0	-770	-950	0	0	-1464100
4	-100	0	0	1.7000	-23	0
5	0	-100	0	0.2500	30	770
6	0	0	-100	0	-20	950

Figure 14 — Γ $(Q = diag(100, 100, 100), R = diag(1, 1, 1))$

vueigen

6x3 double

	1	2	3
1	-4.2041e-05	0.0610	-0.0273
2	0.0023	-0.9948	-0.0229
3	-1.0000	0.0271	0.0126
4	6.0623e-07	0.0034	-0.9973
5	5.0270e-04	-0.0765	-0.0628
6	-0.0076	5.2923e-04	3.8744e-06

Figure 15 — Eigenvectors of Γ $(Q = diag(100, 100, 100), R = diag(1, 1, 1))$

P

3x3 double

	1	2	3
1	34.6993	2.1237	0.0035
2	2.1237	0.2071	-1.1117e-04
3	0.0035	-1.1117e-04	0.0076

Figure 16 — P $(Q = diag(100, 100, 100), R = diag(1, 1, 1))$

Finally, the Feedback Gain Matrix K:

$$K = -R^{-1}B^T P \tag{25}$$

The results of computing in MATLAB are shown in Figure 14–17

K

2x3 double

	1	2	3
1	7.8783	-9.3095	0.0419
2	-4.2008	0.1345	-9.2453

Figure 17 — K $(Q = diag(100, 100, 100), R = diag(1, 1, 1))$

3.2 Simulation

Using SIMULINK, we can simulate the state responses to non-zero initial state x_0 with zero external inputs. My model is shown as Figure 18. The control signal and state response of all three non-zero initial state with zero external inputs are shown in Figure 19–20.

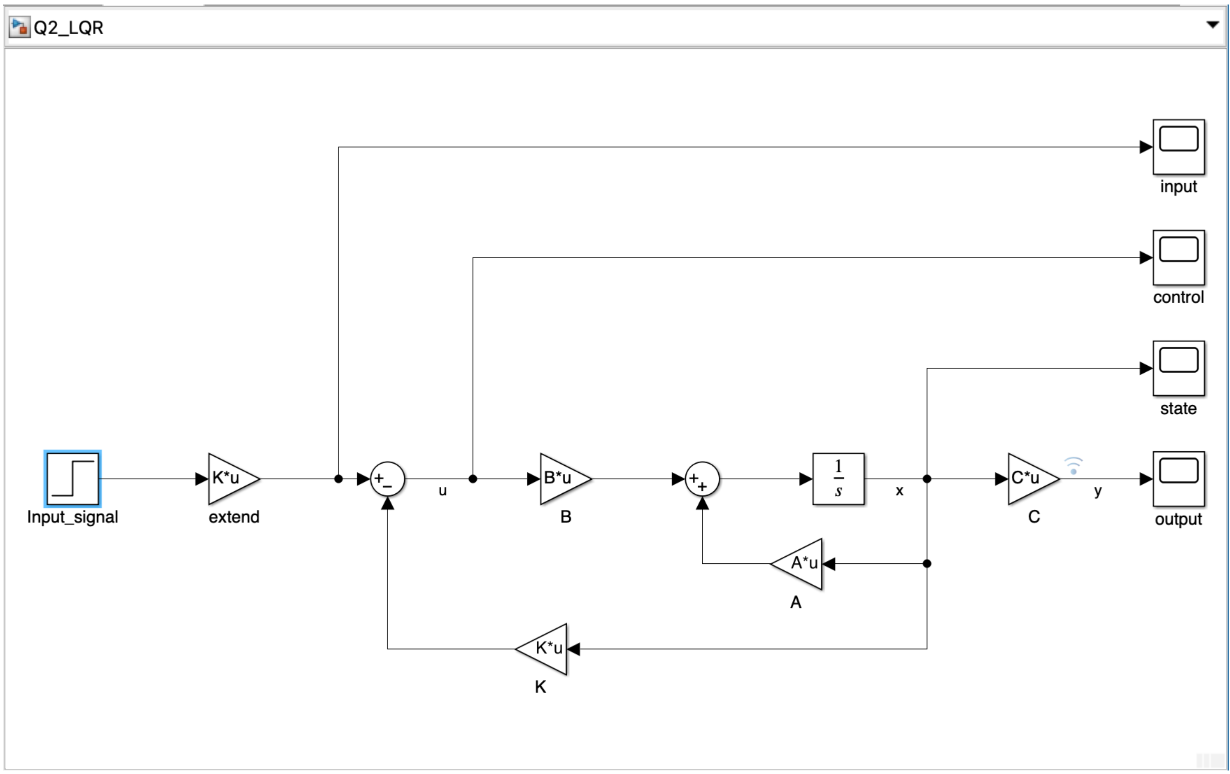


Figure 18 — LQR simulation model

To check the step responses of the 2 outputs, I apply only one step at a time and keep the other one as zero (r=[1,0], or r=[0,1]), and then observe the behavior of the two outputs. The simulation results are shown in Figure 21 and Figure 22.

But the overshoot of step responses do not meet the design specifications. Hence, we need to find proper weight matrices to strike the balance.

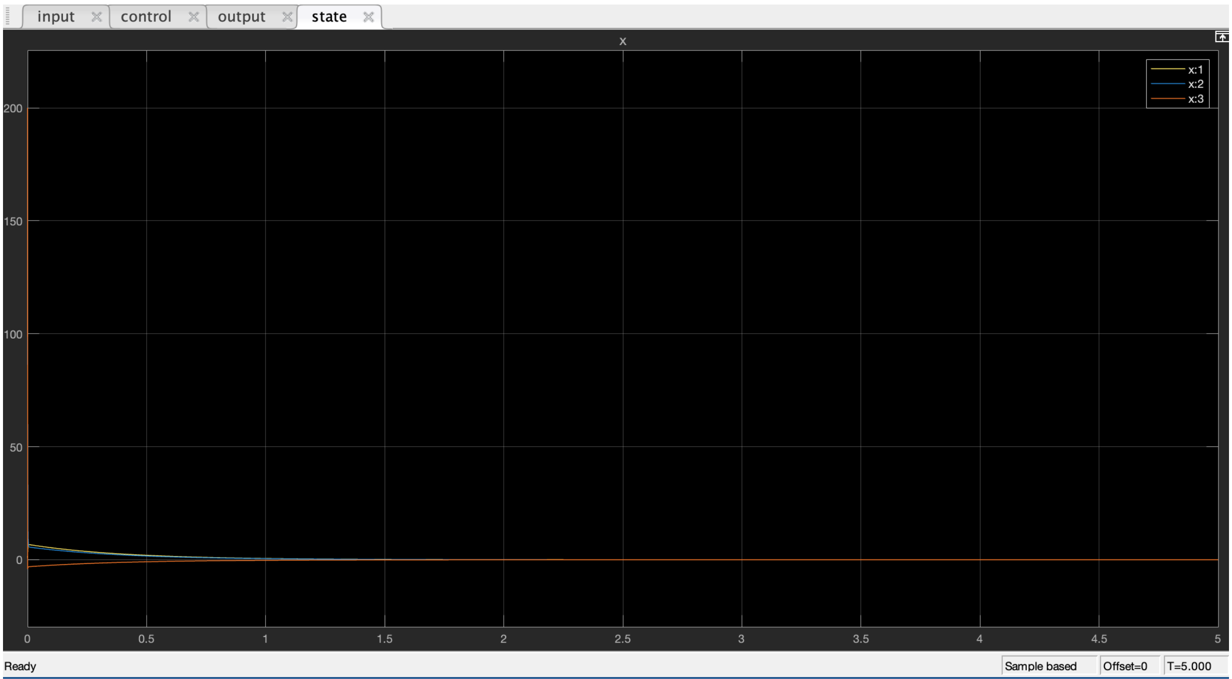


Figure 19 — LQR state responses (original Q and R)

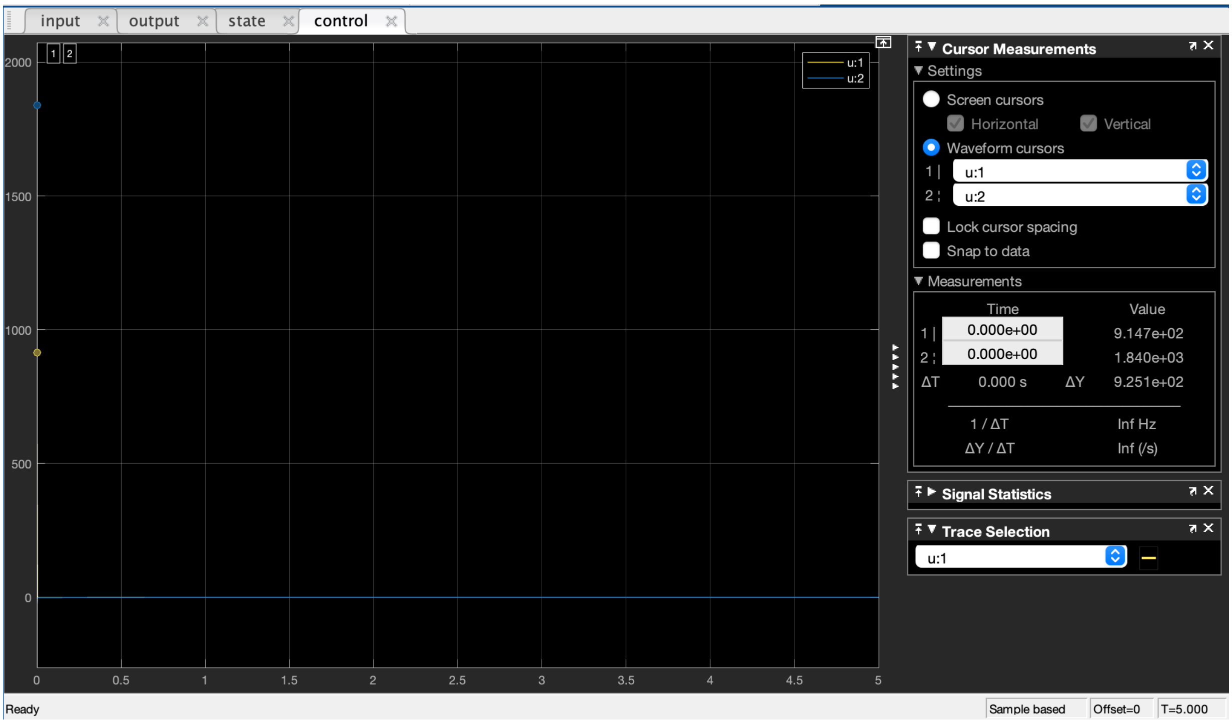


Figure 20 — LQR control signals (original Q and R)

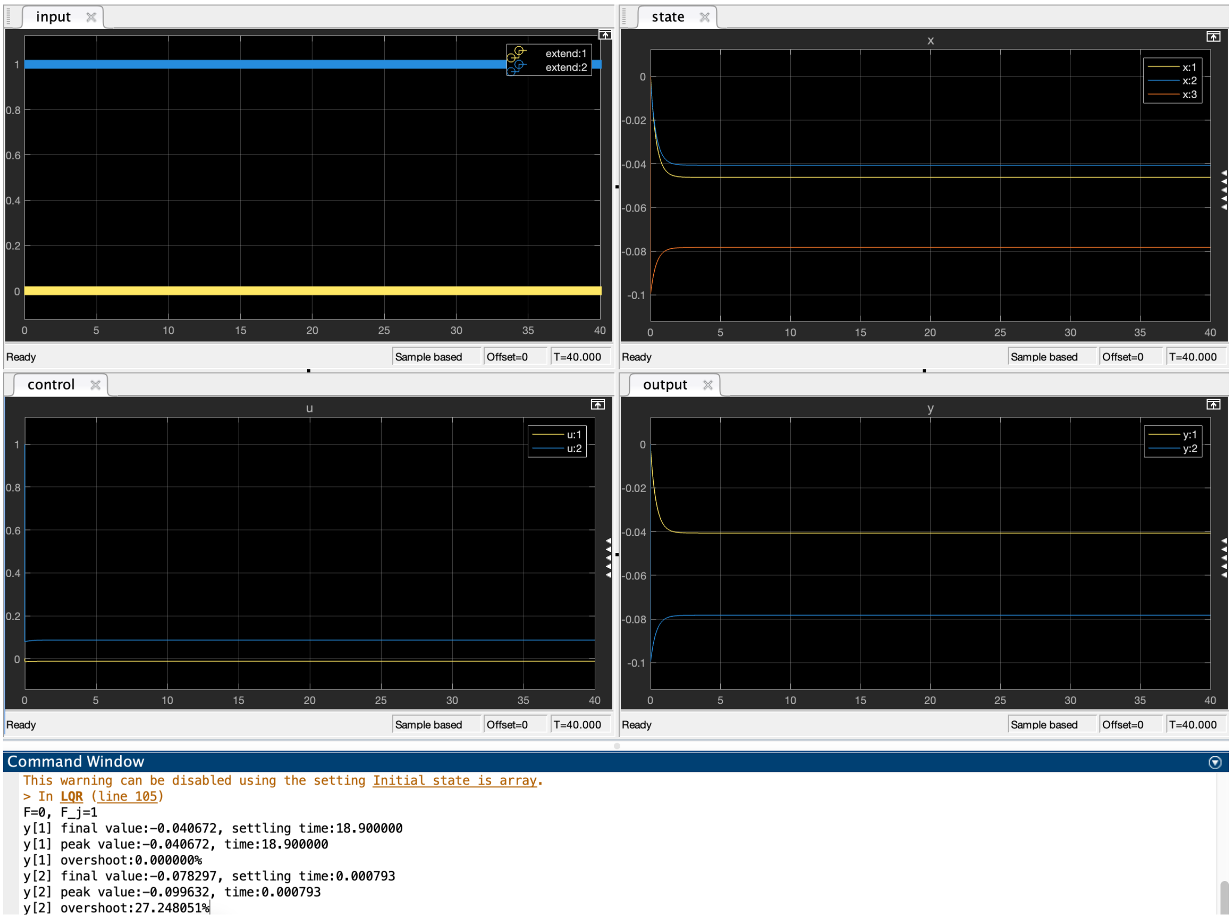


Figure 21 — LQR step responses with non-zero input and r = [0,1]

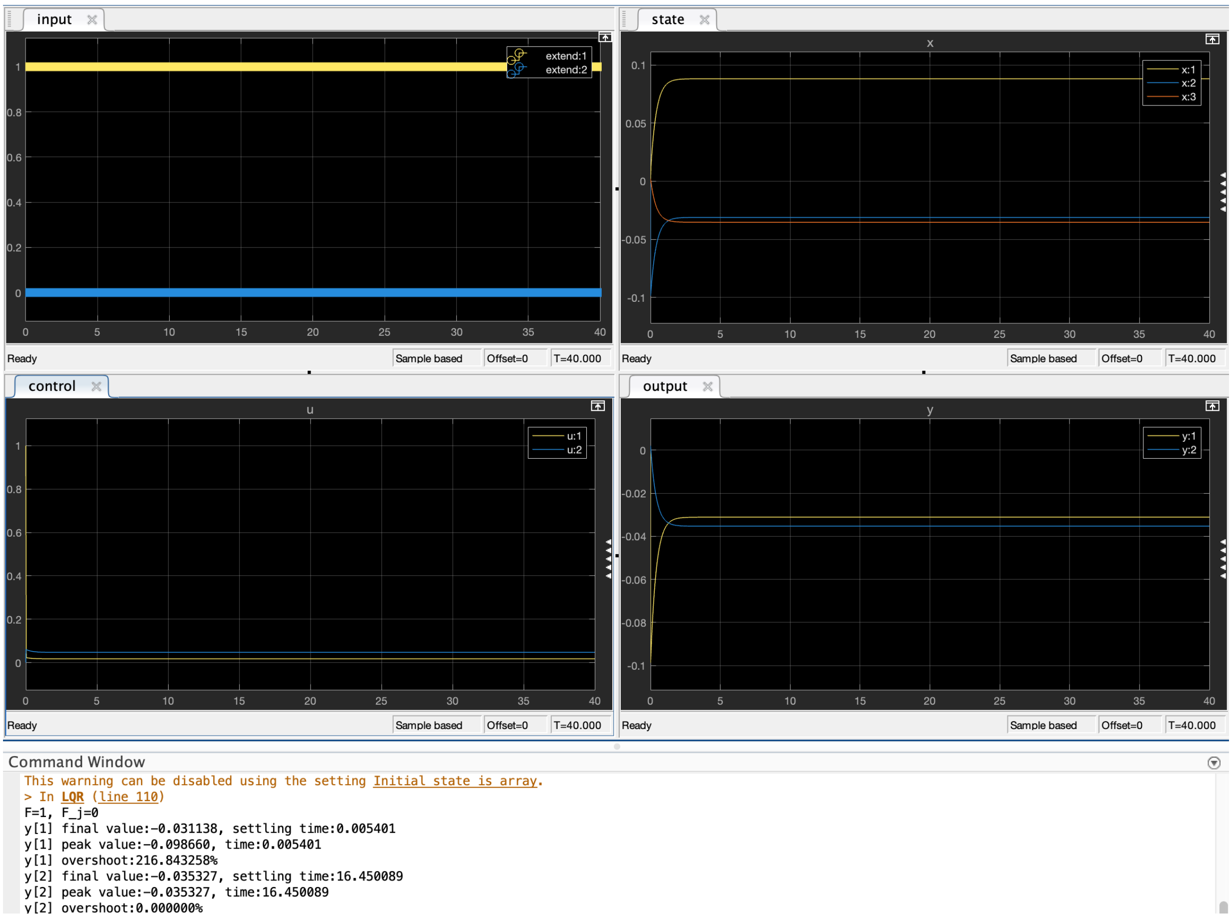


Figure 22 — LQR step responses with non-zero input and r = [1,0]

3.3 Analysis

3.3.1 Adjustment of Q and R

1. Initial design and outcomes
   For Q: We set $Q = diag(100, 100, 100)$, assigning equal importance to all state variables to evaluate baseline system behavior. For R: Initially we set as $R = I$ to observe the effect of equal control energy penalties on transient response.
   - When step input is applied to F_j (cooling flow u_2), with F (flow rate u_1) set to zero, the state T_j (coolant temperature) fails to meet the transient response requirement, while the state T (reactor temperature) exhibits satisfactory transient performance.
   - Then step input is applied to F, with F_j set to zero, the state T exhibited an overshoot of 216.8%, exceeding the acceptable limit of 10%. The state T_j satisfied the overshoot constraint.
2. Adjustment of Q and R
   For Q, we need to adjust the weights corresponding to T and T_j to prioritize their regulation over C_a. For R, we reduces the weight of u_1 to make the transient response faster. Then we observe the step responses. This time I choose $Q = diag(0.1, 80, 50)$ and $R = diag(0.1, 1)$.
3. Final Results and Analysis
   After implementing the adjusted Q and R , the system was re-evaluated under the same step input conditions, results shown in Figure 23–24. The responses both meet the transient specification.
4. **Conclusion:** The increased Q weights improved the regulation of T and T_j, significantly reducing overshoot. Smaller Q values for other states (C_a) avoided unnecessary control actions. Larger R values, particularly for F_j, reduced overshoot by limiting aggressive control actions and ensure control effort remains efficient while avoiding excessive control energy. While smaller R values will increase the control efforts, the control signals after adjustment of R_{11} become larger compared to initial setup.

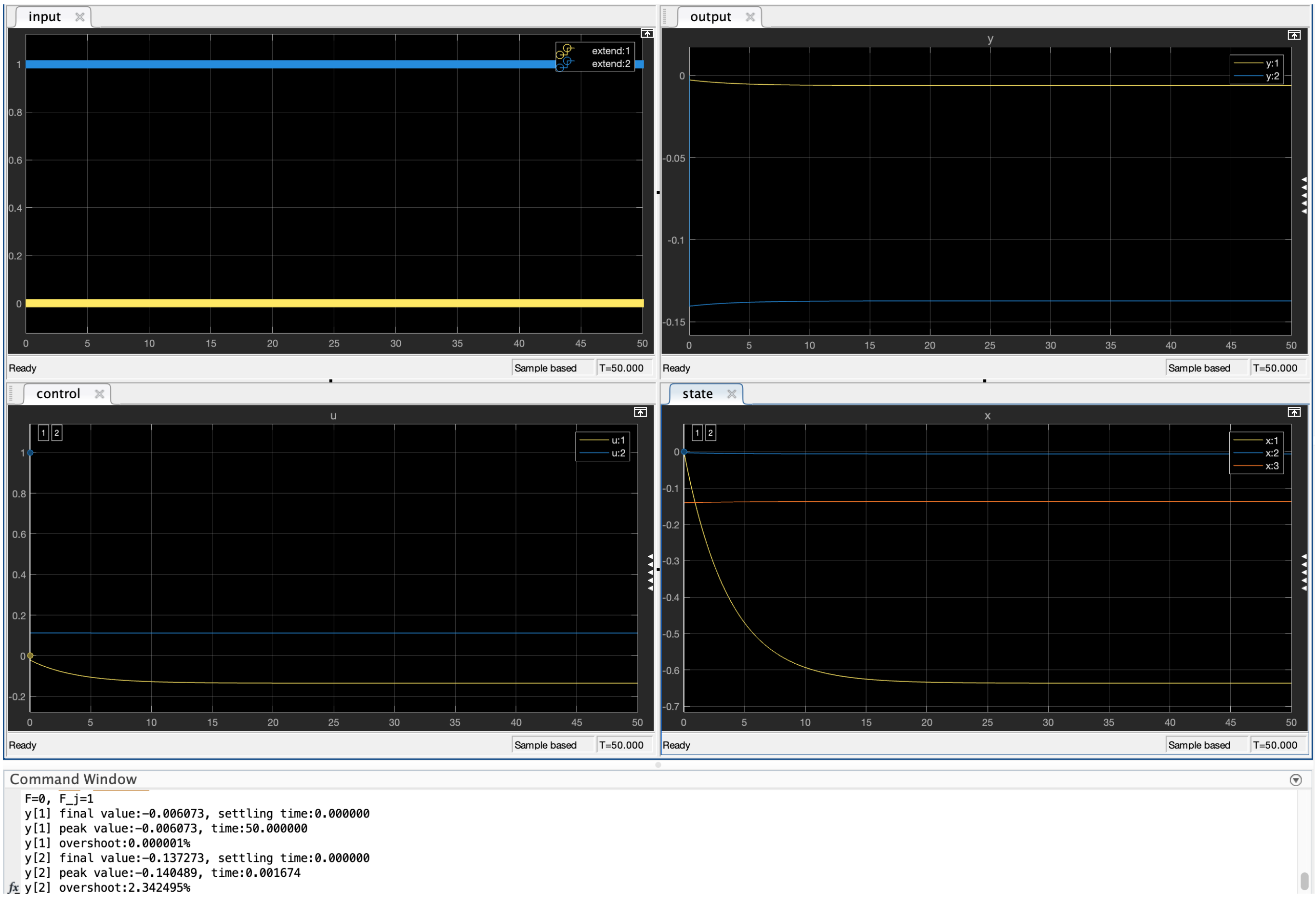


Figure 23 — LQR step responses with non-zero input and r = [0,1] (Adjusted R and Q)

3.3.2 State response to non-zero initial state with zero external inputs

The state responses are shown in Figure 25 and the control signals are shown in Figure 26, which show the results meet the transient specifications.

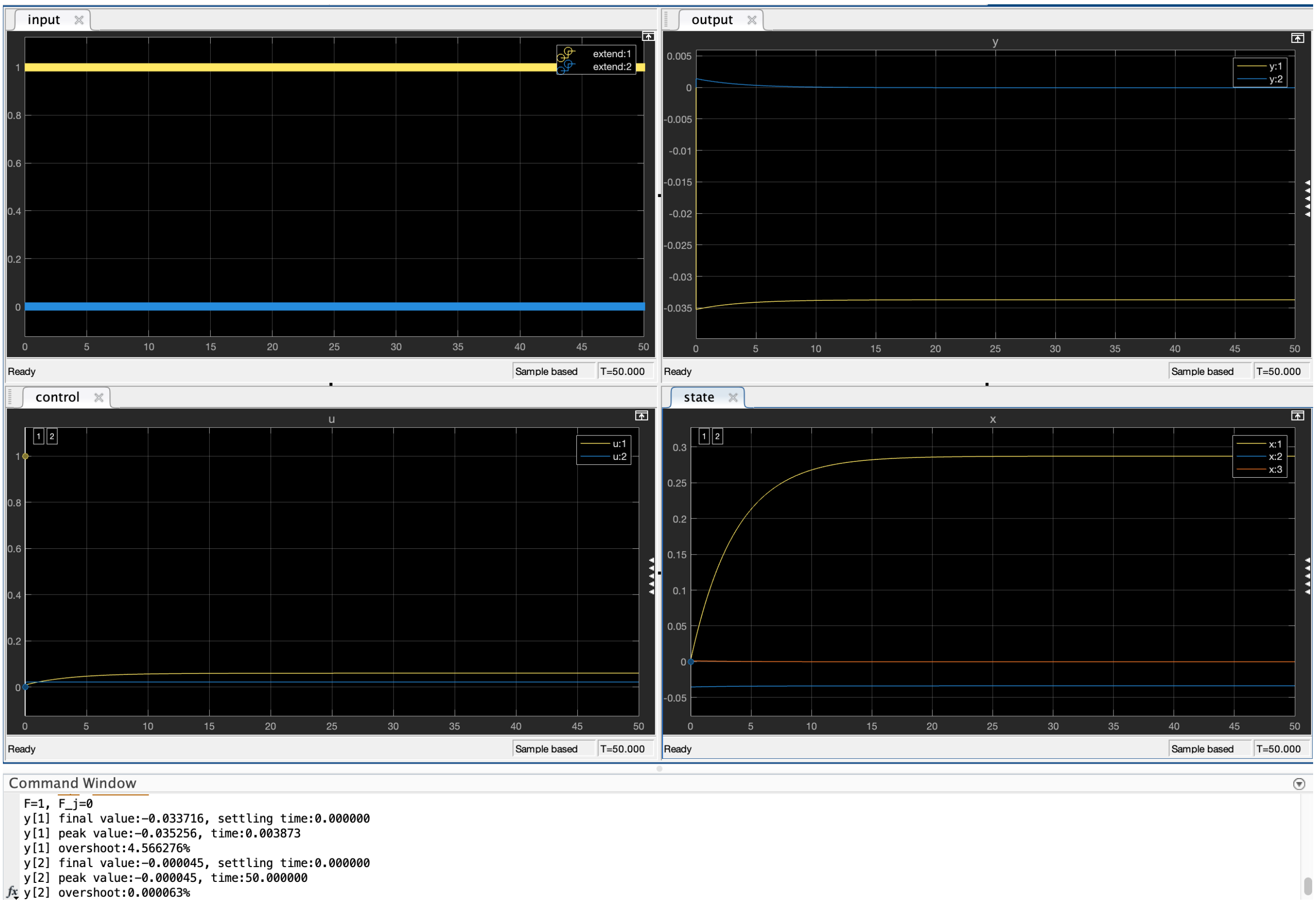


Figure 24 — LQR step responses with non-zero input and r = [1,0] (Adjusted R and Q)

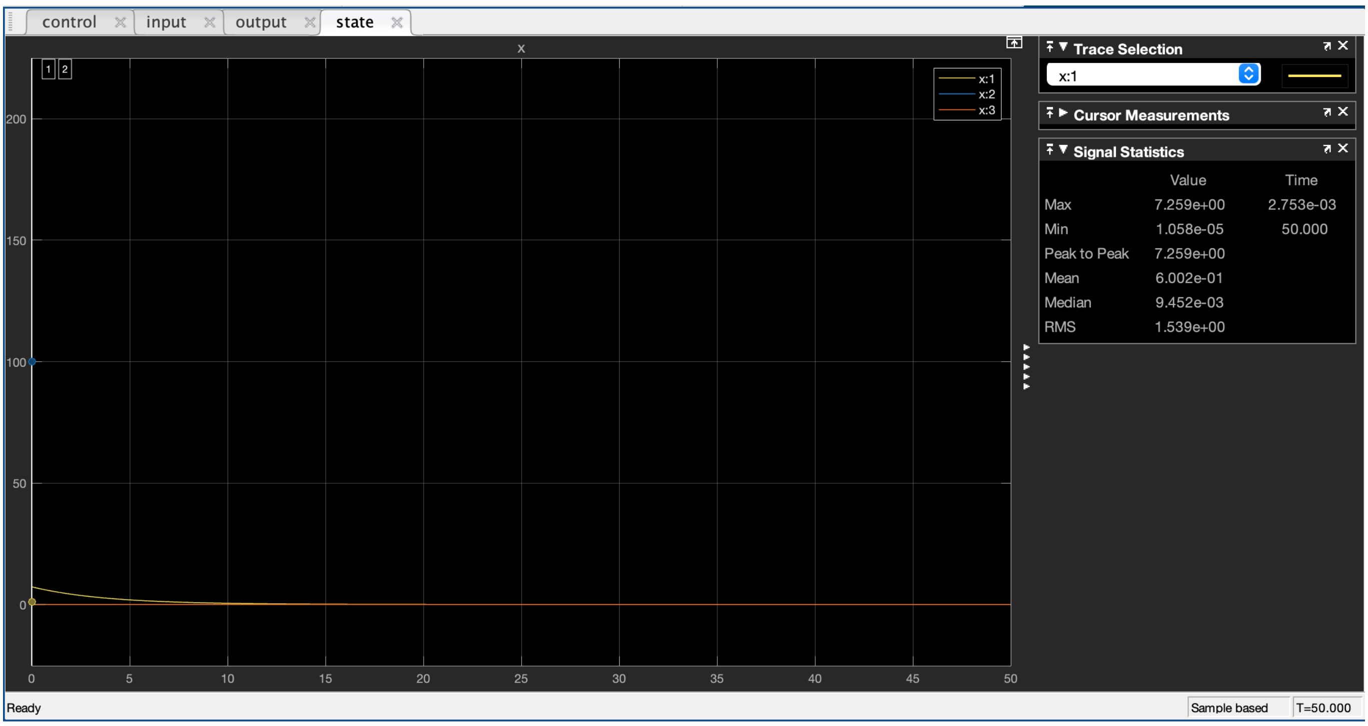


Figure 25 — Pole placement simulation model

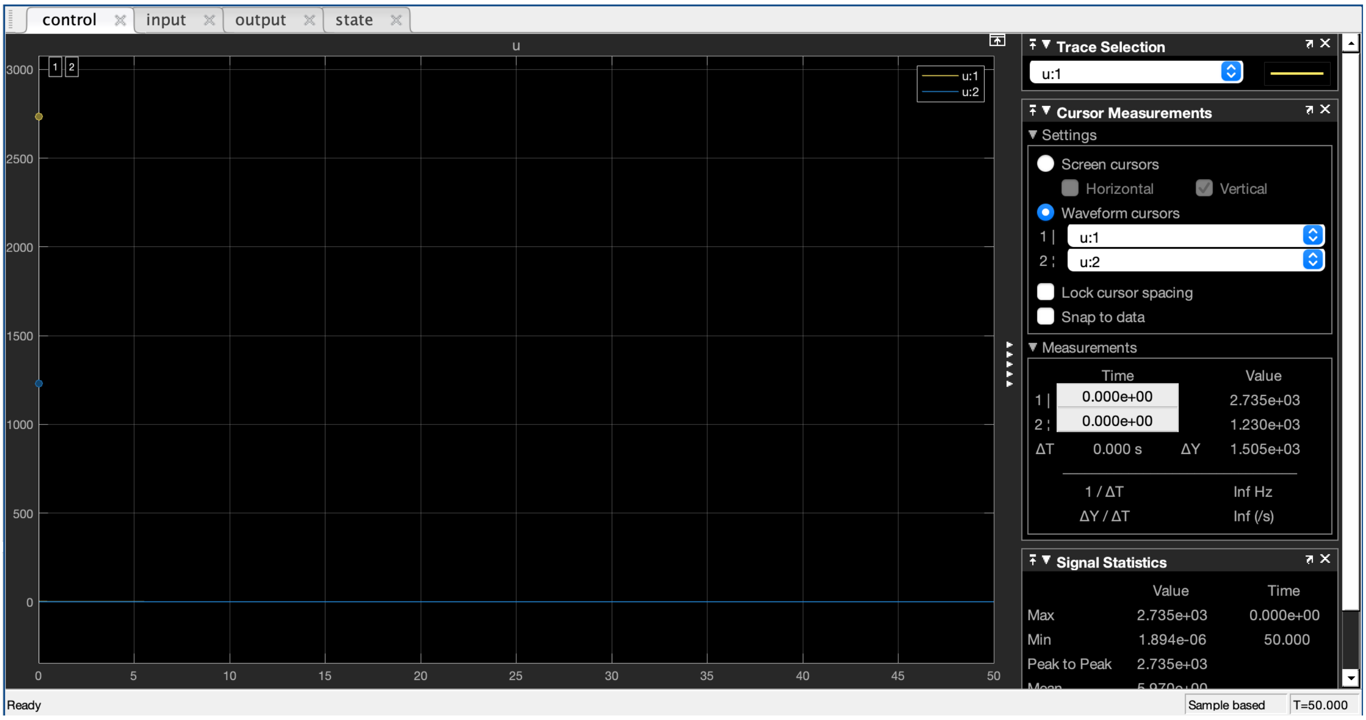


Figure 26 — State responses after pole placement

4 STATE OBSERVER

4.1 Observer design

Since LQR need state feedback, now I can only measure the two states, so an observer is needed to estimate the rest state.

$$Q = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}, R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{26}$$

The full-order observer, also known as a closed-loop estimator, is represented by the following dynamics:

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}), \tag{27}$$

The estimation error is defined as:

$$\tilde{x} = x - \hat{x}, \tag{28}$$

The dynamics of the estimation error are given by:

$$\dot{\tilde{x}} = (A - LC)\tilde{x}. \tag{29}$$

The observer is designed with:

$$\tilde{A} = A^T, \quad \tilde{B} = C^T, \quad \tilde{K} = L^T. \tag{30}$$

The original poles of the closed-loop system generated by LQR method are:

Table 3 — Control poles

s_1	-0.2690
s_2	-8601.8
s_3	-3604.1

A simple guideline is to place observer poles 3-5 times faster than control poles, so I choose the desired observer poles as follows:

Using pole placement method to obtain L:

Table 4 — observer poles

s_1	-2
s_2	-20000
s_3	-15000

L_estimator

3x2 double

	1	2
1	0.7452	2.9959e+07
2	-31.7000	300140000
3	-771	34054

Figure 27 — solve L for estimator

4.2 Simulation

My model is shown as Figure 28. First, I simulate the resultant observer-based LQR control system and monitor the state estimation error, the results are shown as Figure 29 :

The state responses meet the design specifications.

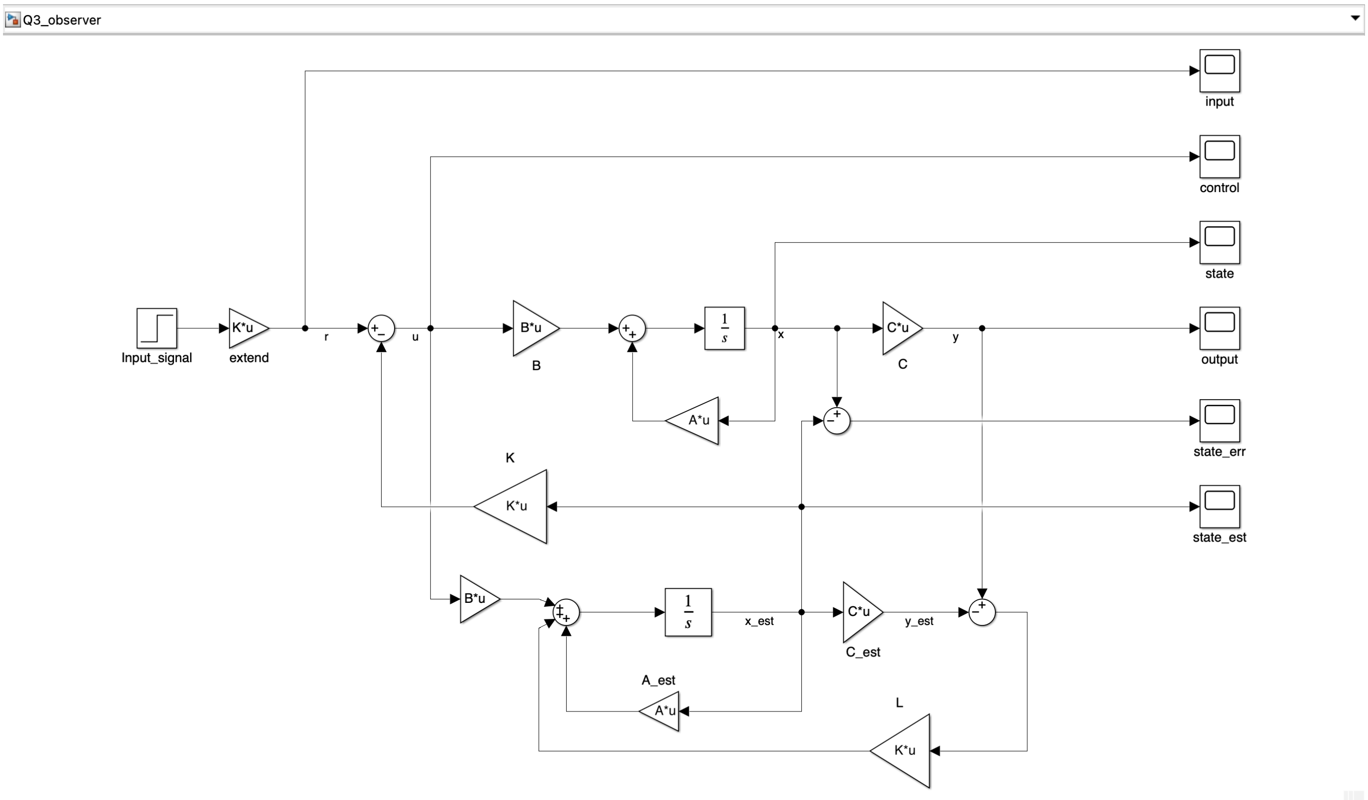


Figure 28 — Observer Simulation Model

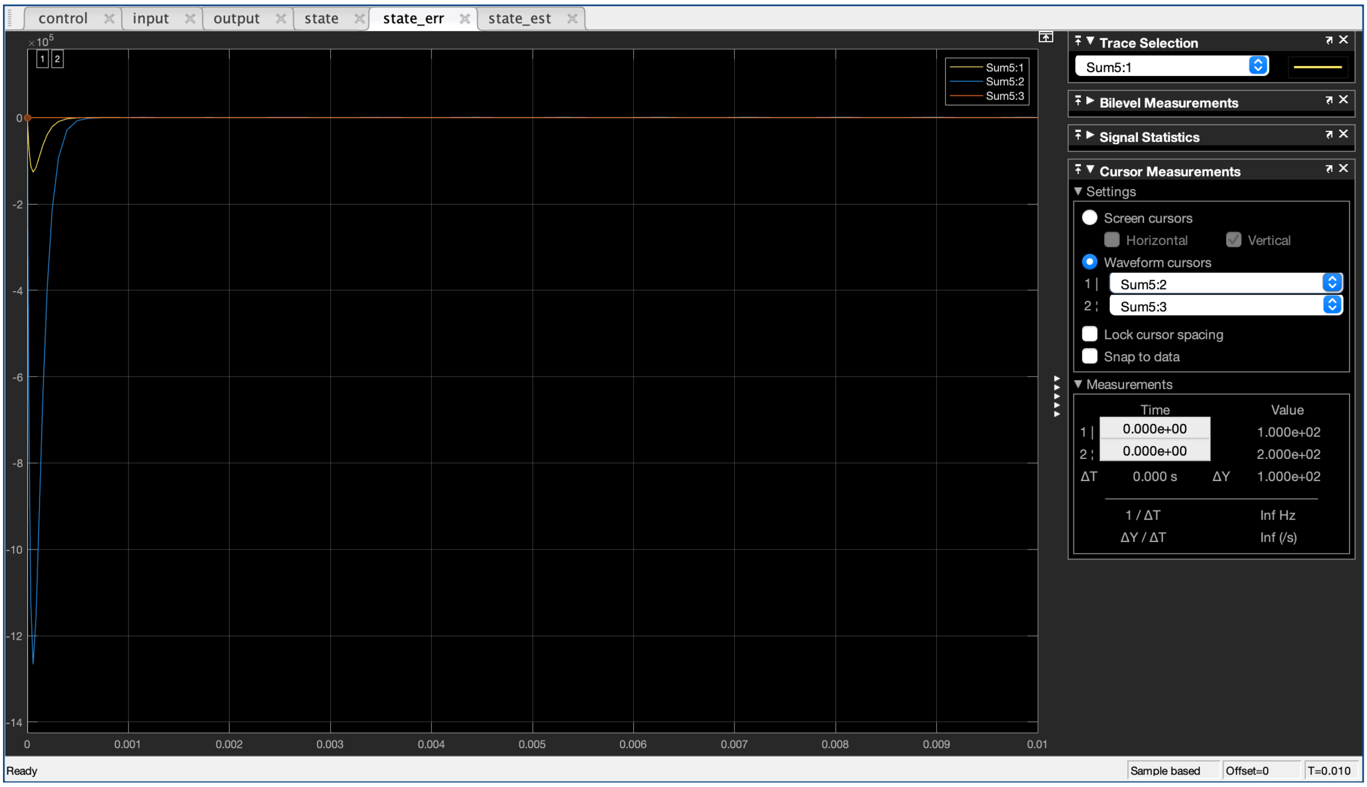


Figure 29 — State estimate error

4.3 Analysis

Now we want to investigate the effects of observer poles on state estimation error and the closed-loop controls performance. Consider change the poles as follows:

Table 5 — change observer poles

Poles	s1	s2	s3
original	-4	-20000	-15000
change1	-1	-17500	-7210
change2	-3	-86100	-36100

Then we simulate the non-zero state response, find out the effects brought by the changes. Results of state responses shown in Figure 30 and Figure 31 :

When we move the original observer poles to the right, meaning closer to the origin point, the state errors become more smooth and change slower, the maximal temporary error of x_2 (blue line) become smaller. On the contrary, when poles are moved to the left, the state errors change faster and maximal temporary error of x_2 becomes bigger.

Figure 30 — State estimate error change-1

Figure 31 — State estimate error change-2

5 DECOUPLING CONTROLLER

5.1 Controller design

Consider the plant 1 can be decoupled by the state feedback control law is defined as:

$$u = -Kx + Fr. \tag{31}$$

We need to solve both K and F. The resultant system is:

$$\begin{aligned} \dot{x} &= (A - BK)x + BFr, \\ y &= Cx. \end{aligned} \tag{32}$$

Open-loop Transfer Function:

$$G(s) = C(sI - A)^{-1}B. \tag{33}$$

Closed-loop Transfer Function:

$$H(s) = C(sI - A + BK)^{-1}BF. \tag{34}$$

The transfer function of the closed-loop system $H(s)$ is related to the open-loop transfer function $G(s)$, and can be re-written as:

$$H(s) = G(s)\left[I + K(sI - A)^{-1}B\right]^{-1} F. \tag{35}$$

Condition for Decoupling:

If $H(s)$ is diagonal, then $G(s)$ must be non-singular. This is the only condition for the system to be decoupled.

$$G(s) = \begin{bmatrix} s^{-\sigma_1} & 0 \\ 0 & s^{-\sigma_2} \end{bmatrix} \left[B^* + C^*(sI - A)^{-1}B\right] \tag{36}$$

When $\sigma_1 = \sigma_2 = 1$, the place poles are set as: $\phi(s) = s + 10 = 0$. Then the matrices F and K can be calculated as:

$$F = (B^*)^{-1} = \begin{bmatrix} -0.0079 & 0 \\ 0 & -8.2645 \times 10^{-4} \end{bmatrix}, \tag{37}$$

$$K = (B^*)^{-1}C^* = \begin{bmatrix} -0.1811 & 0.1575 & -0.1575 \\ 0.6364 & 0.5952 & 0.7769 \end{bmatrix}. \tag{38}$$

The eigen value of the system is shown below, the decoupled system is internally stable, the step responses can be computed by MATLAB, results shown in Figure 33 . And then we will verify the stability in SIMULINK.

```
eigenvalue =

   -0.2512         0         0
         0  -10.0000         0
         0         0  -10.0000
```

Figure 32 — Poles of decoupled system

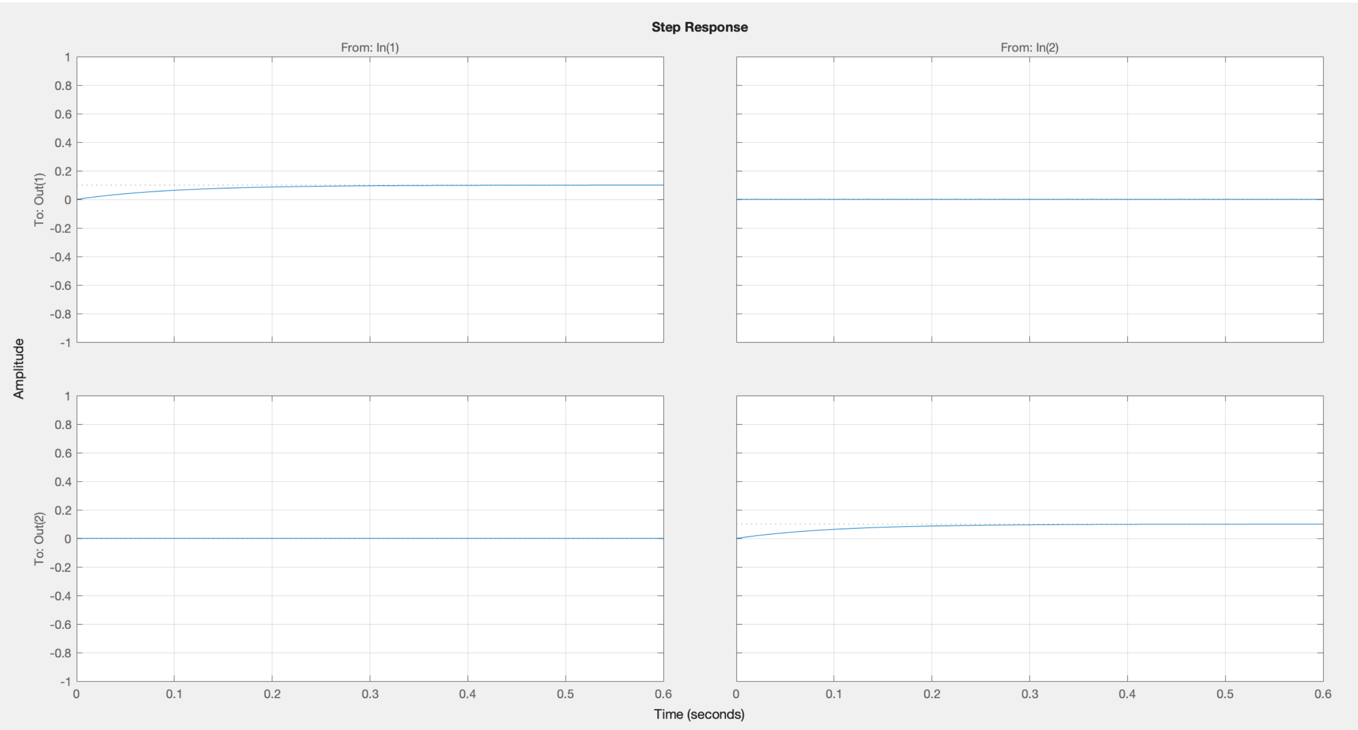


Figure 33 — step responses of decoupled system

5.2 Simulation

Using SIMULINK, we can simulate the state responses to non-zero initial state x_0 with zero external inputs. My model is shown as Figure 34. The state response of all three non-zero initial state with zero external inputs are shown in Figure 35. The step responses with zero initial states are shown in Figure 36–37.

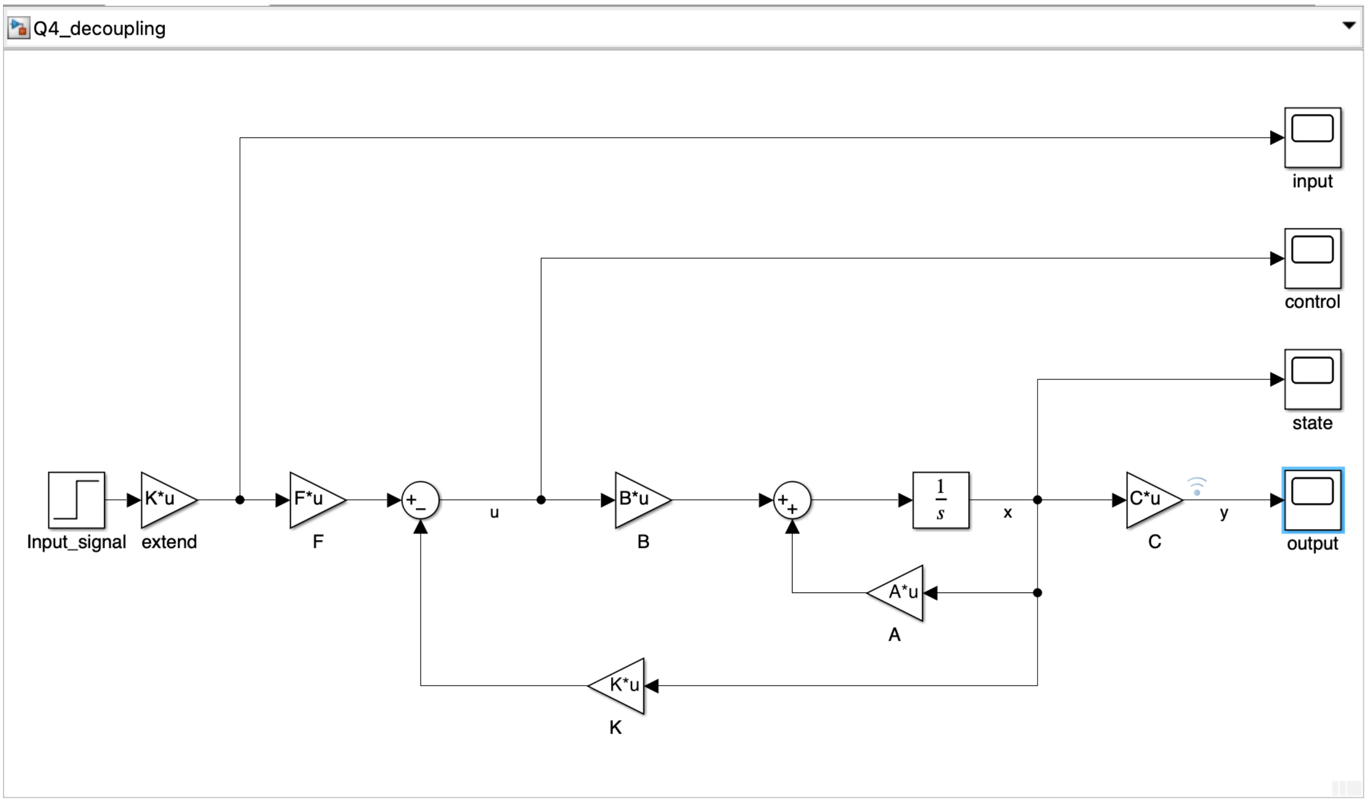


Figure 34 — Decouling Simulation Model

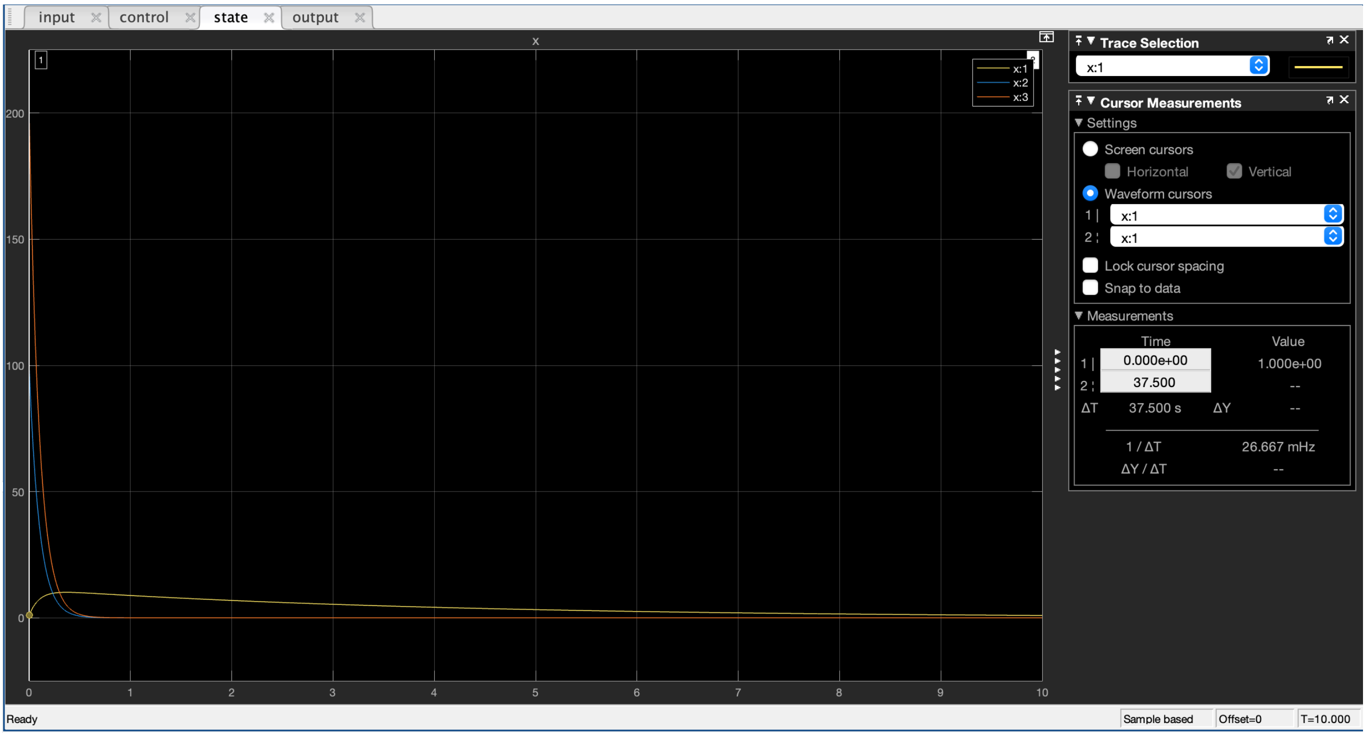


Figure 35 — initial response of the resultant control system

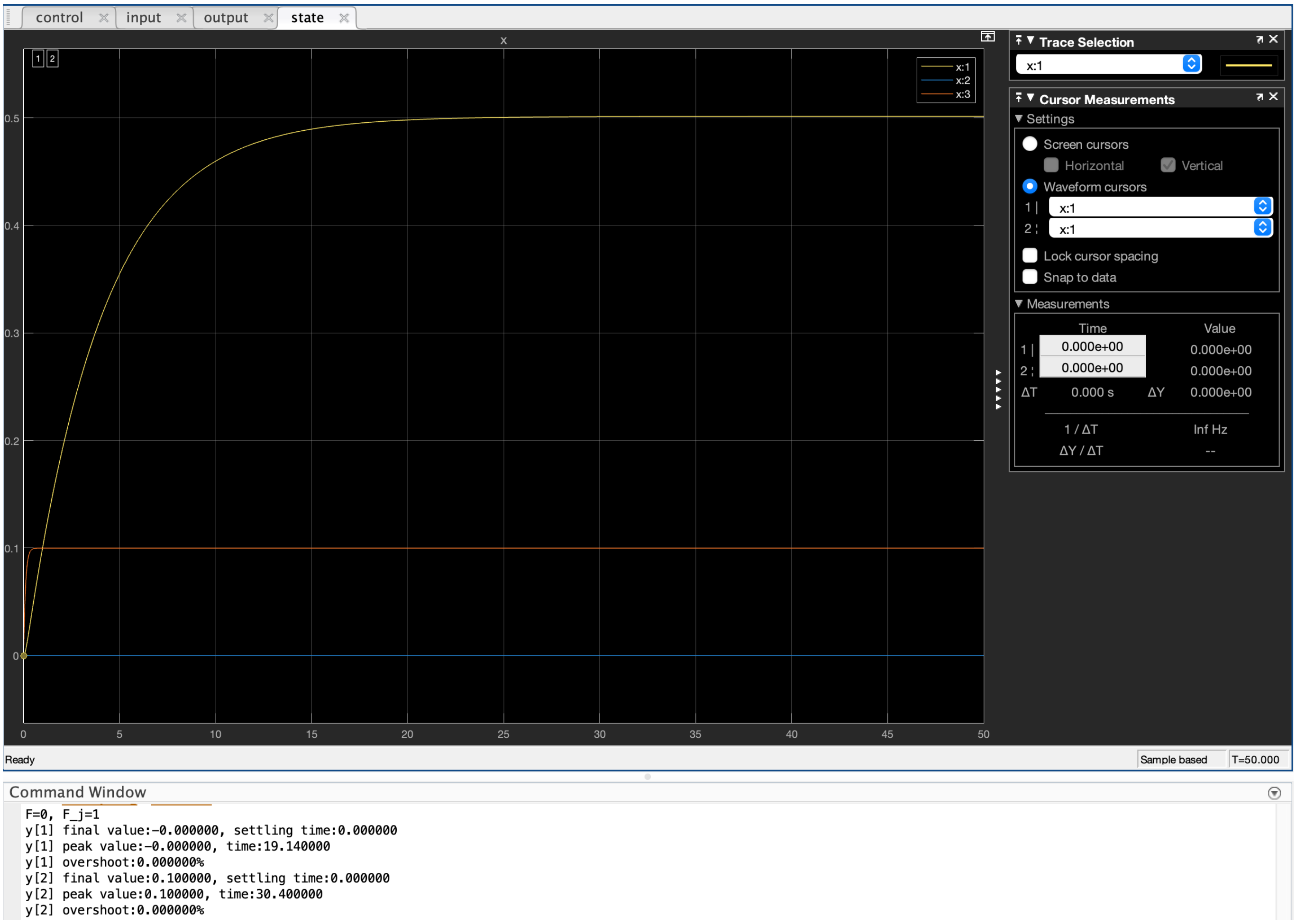


Figure 36 — step responses with F=0, F_j =1

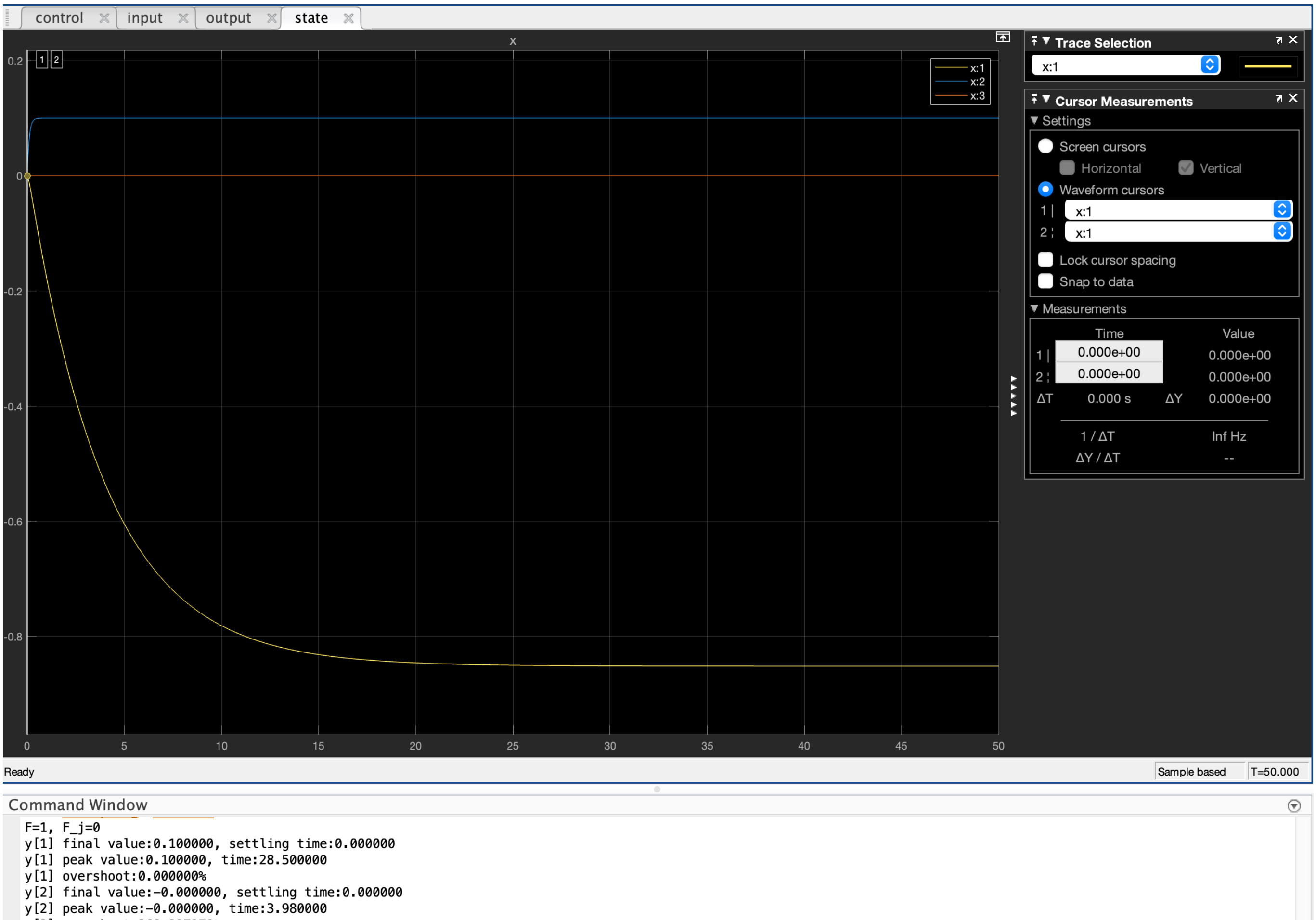


Figure 37 — step responses with F=1, F_j =0

6 SERVO CONTROLLER

6.1 Controller design

The operating set point for the output is $y_{sp} = [100, 150]^T$, and I have only 2 cheap sensors to measure the output. The objective is to design a controller such that the plant can operate around the set point as close as possible at steady state even when disturbances are present at the plant input, assuming the step disturbance $w = [-2, 5]^T$ takes effect from time $t_d = 10s$ afterwards.

For this MIMO case, consider a $m \times m$ plant:

$$\begin{aligned} \dot{x} &= Ax + Bu + B_w w, \\ y &= Cx \end{aligned} \tag{39}$$

where the error is defined as:

$$e = r - y \tag{40}$$

Suppose that the disturbance w and reference r are both of step type. To achieve zero steady-state:

$$v(t) = \int_0^t e(\tau) d\tau. \tag{41}$$

Then, it follows that:

$$\dot{v}(t) = e(t) = r - y(t) = r - Cx(t) \tag{42}$$

$$\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} A & O \\ -C & O \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} + \begin{bmatrix} B \\ O \end{bmatrix} u + \begin{bmatrix} B_w \\ O \end{bmatrix} w + \begin{bmatrix} 0 \\ I \end{bmatrix} r \tag{43}$$

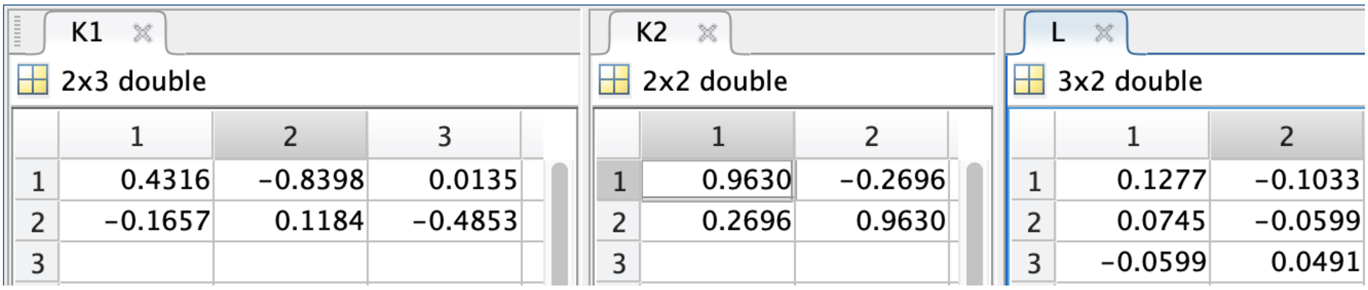
The augmented system can be represented as:

$$\begin{aligned} \dot{\bar{x}} &= \bar{A}\bar{x} + \bar{B}u + \bar{B}_w w + \bar{B}_r r, \\ y &= \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} = \bar{C}\bar{x} \end{aligned} \tag{44}$$

The controllability matrix Q_c:

$$Q_c = \begin{pmatrix} B & AB & A^2B & \cdots \\ 0 & -CB & -CAB & \cdots \end{pmatrix} = \begin{pmatrix} A & B \\ -C & 0 \end{pmatrix} \begin{pmatrix} 0 & B & AB & \cdots \\ I & 0 & 0 & \cdots \end{pmatrix} \tag{45}$$

The augmented system is controllable if and only if:



K1 — 2x3 double

	1	2	3
1	0.4316	-0.8398	0.0135
2	-0.1657	0.1184	-0.4853
3			

K2 — 2x2 double

	1	2
1	0.9630	-0.2696
2	0.2696	0.9630
3		

L — 3x2 double

	1	2
1	0.1277	-0.1033
2	0.0745	-0.0599
3	-0.0599	0.0491

Figure 38 — Feedback gain matrices

(i) The plant is controllable, and

(ii)

$$\operatorname{rank}\begin{pmatrix} A & B \\ -C & 0 \end{pmatrix} = n + m.$$

or equivalently:

$$\operatorname{rank}\begin{pmatrix} A & B \\ C & 0 \end{pmatrix} = n + m.$$

This implies that the plant does not have any zero at the origin. If the augmented system is controllable, it can be stabilized by the state feedback control law, the state feedback gain matrices K_1 and K_2 can be obtained by LQR method:

$$u = -K\bar{x} = -\begin{bmatrix} K_1 & K_2 \end{bmatrix}\begin{bmatrix} x \\ v \end{bmatrix}. \tag{46}$$

Since we can only measure 2 state, an observer is needed to estimate another state. I designed the system based on LQR . The observer can be expressed as:

$$\begin{aligned} \dot{x} &= Ax + Bu + B_w w + L[y - \hat{y}], \\ \hat{y} &= C\hat{x} \end{aligned} \tag{47}$$

After computing, we get the matrices K_1, K_2, and L as follows:

6.2 Simulation

The servo control model is shown in Figure 39. Then we set the operating set point y_{sp} and step disturbance w to observe the output and control signals, which are shown in Figure 40–43.

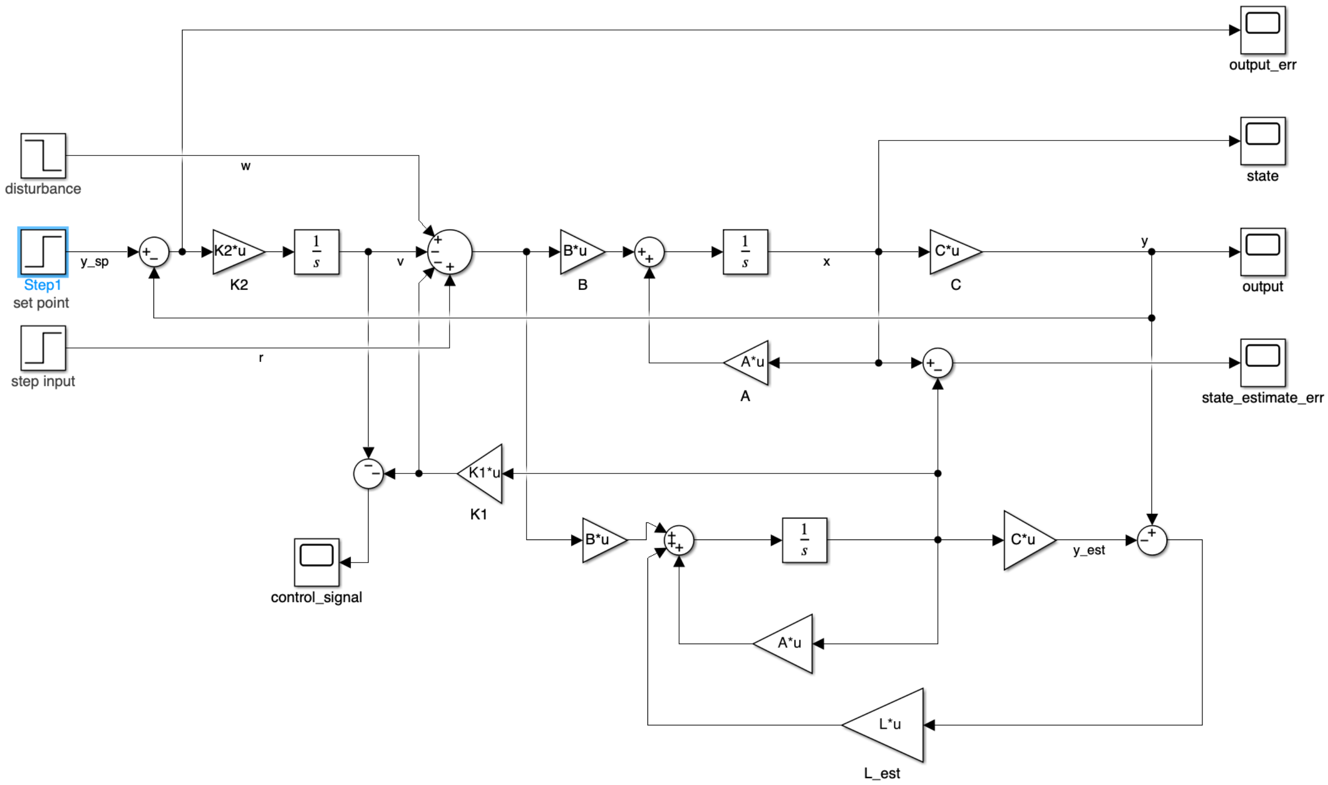


Figure 39 — servo control model

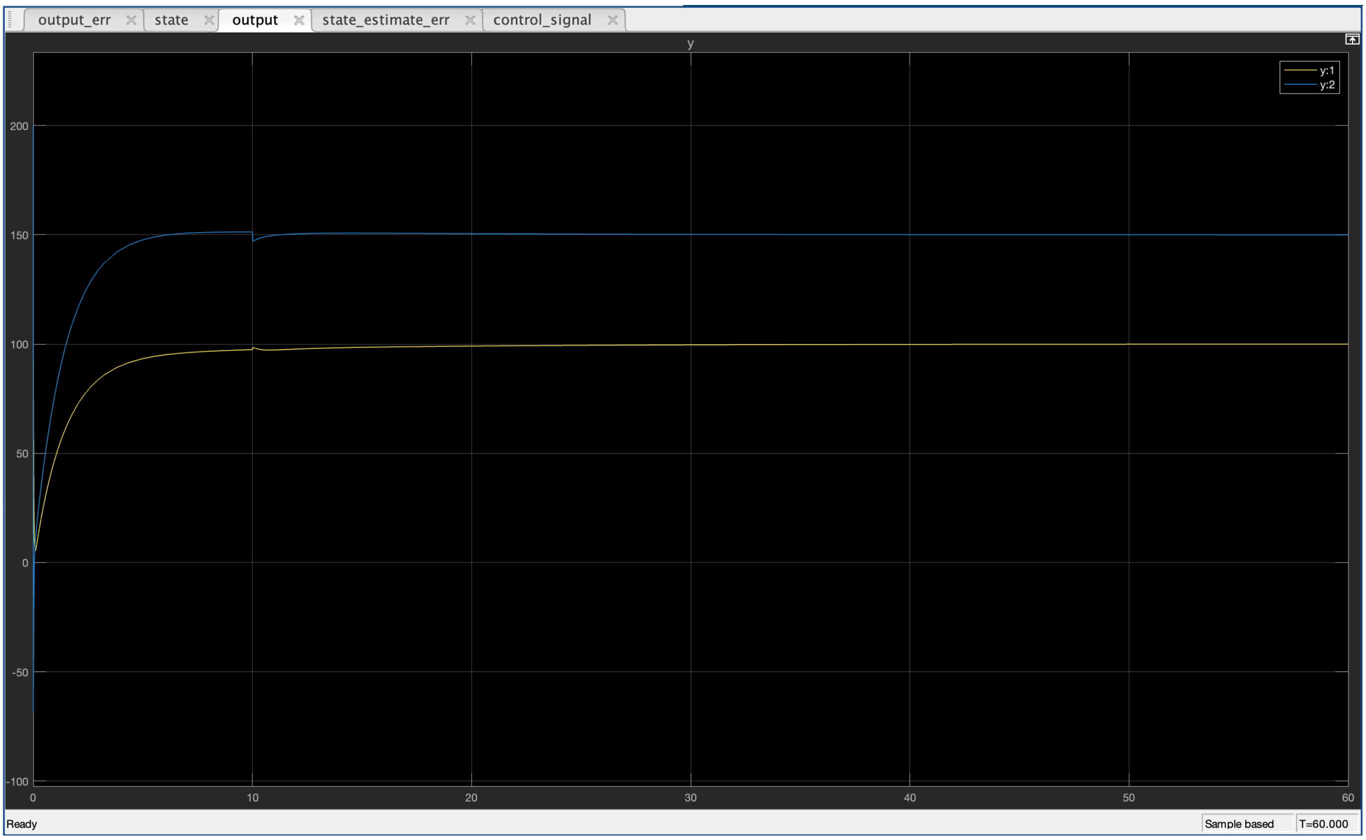


Figure 40 — output signals

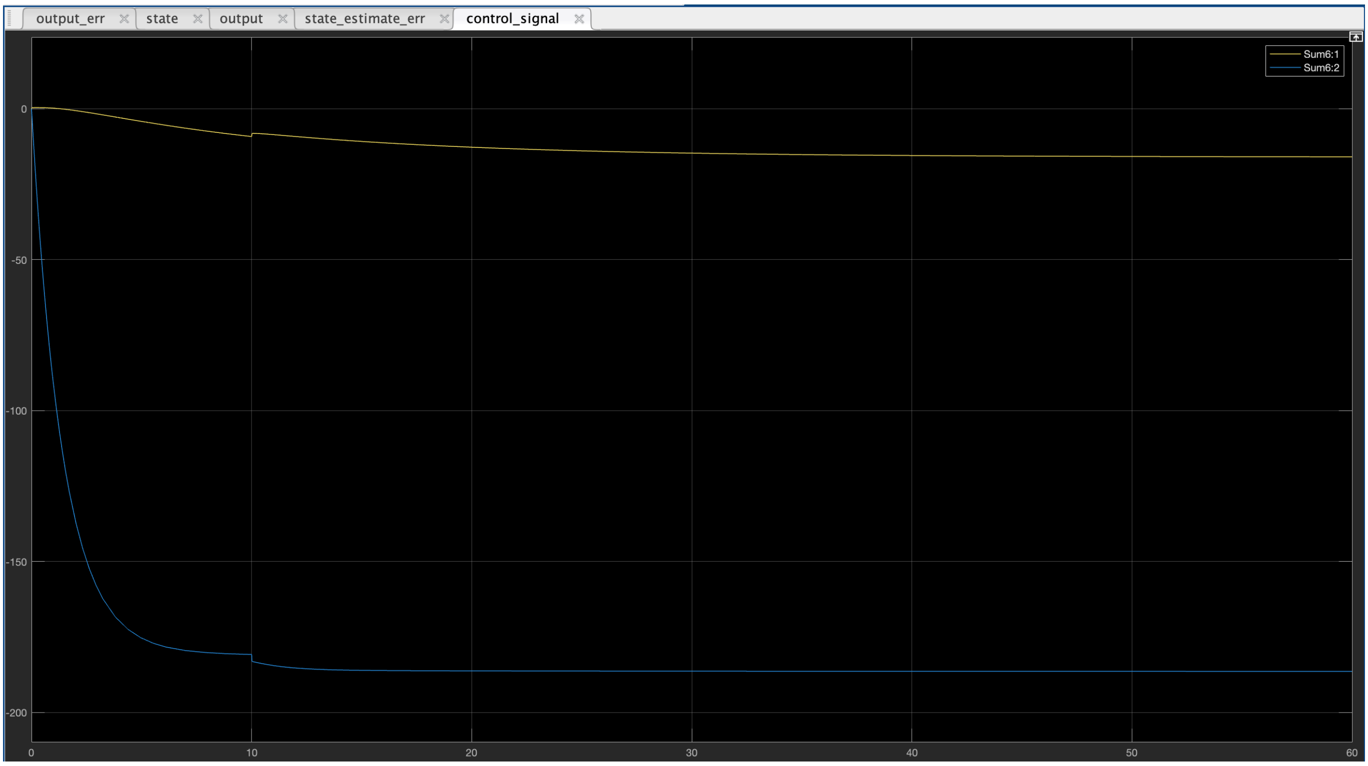


Figure 41 — Control signals

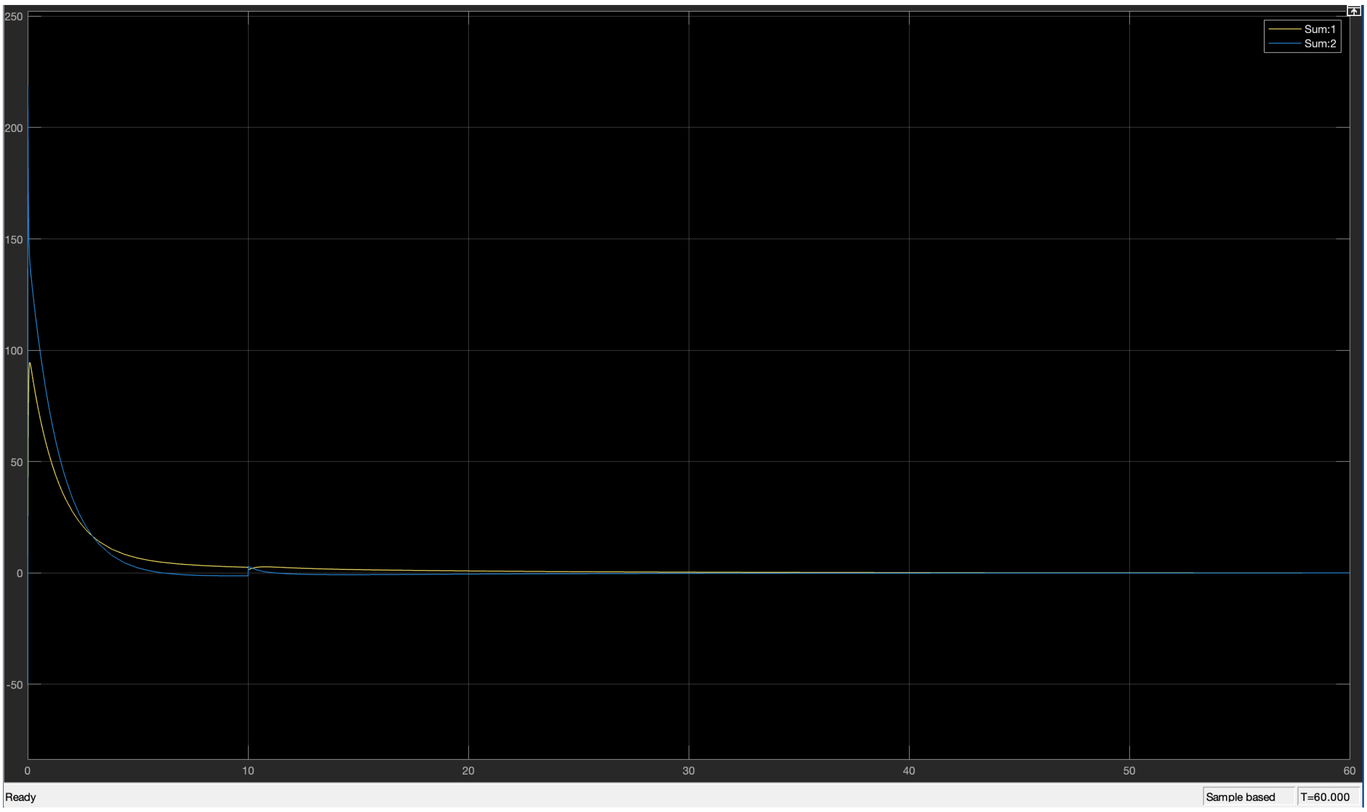


Figure 42 — Steady state error

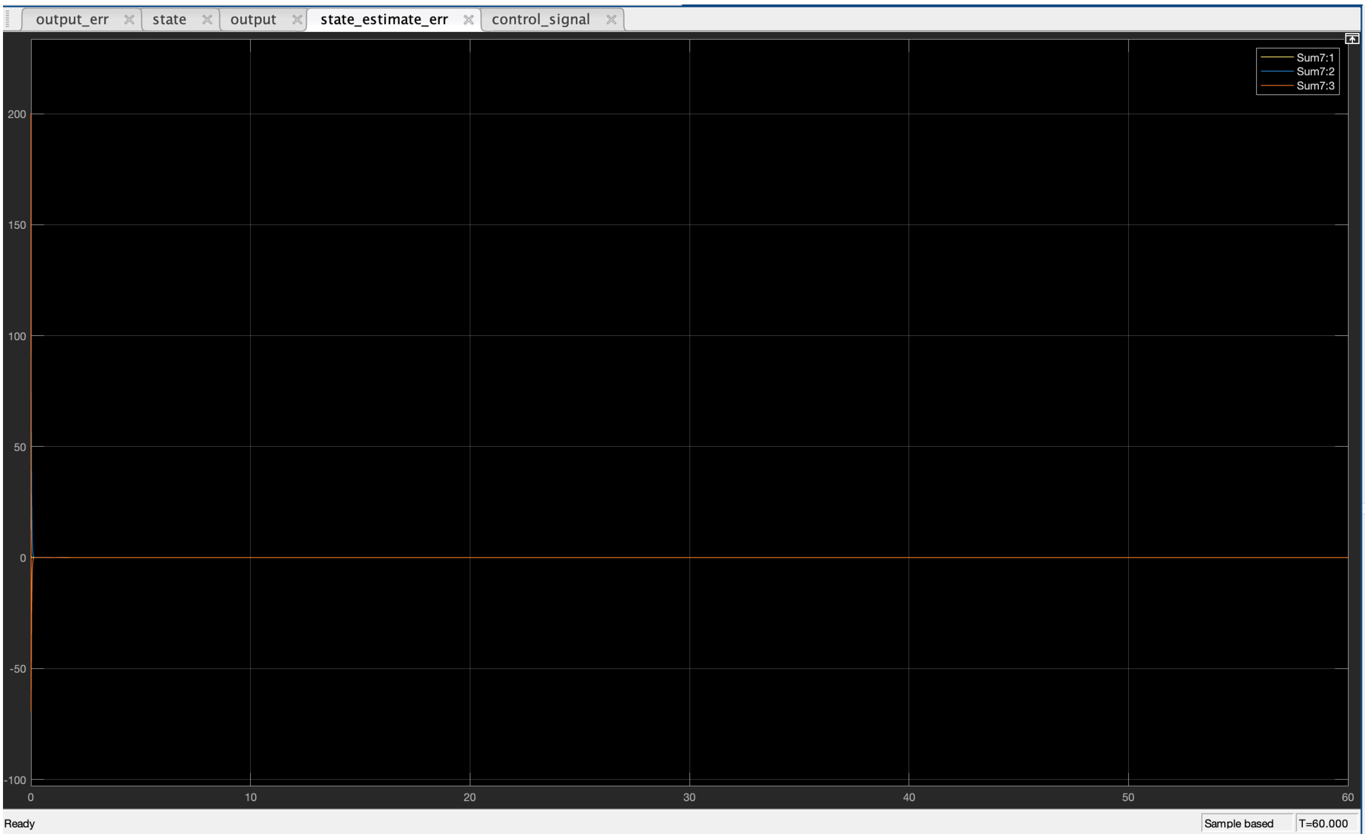


Figure 43 — State estimation error

6.3 Analysis

As shown in the figures, the system achieves zero state estimation error and steady-state error, and then adjusts quickly to maintain following the operating set point when the step disturbance occurs. The state estimation and servo control mechanism ensure the success of the signal tracking and disturbance rejection. if we want to increase the response speed, the weight matrices R and Q can be further fine-tuned to meet the transient specification.

7 DESIRED STATE SETPOINT

The objective is not to maintain the states around a given set points at steady state starting from the initial state x0. The reference state variables $x_sp = [5, 250, 300]^T$.

8 CONCLUSION

In this mini-project, I designed several kinds of controller to achieve the desired system performance for the continuous-flow stirred tank reactor (CSTR) model.

In question 1, pole placement method is used to design a state feedback controller and the effects of the position of poles on system transient performance were discussed. The real part and imaginary part of the poles have physical meaning in the 2nd order system, since higher order system can be approximated by 2nd order system, these principles can also be leveraged in higher order systems.

In question 2, the LQR method is used to design the state feedback controller and the effects of weight matrix Q and R on the system performance are discussed. By manipulating the diagonal matrices related to the penalty of control efforts and speed we can strike the balance to achieve optimal system control.

In question 3, a state observer controller is designed by pole placement method to help estimate the states we can not directly measure. The effect of observer poles on the transient performance is investigated.

In question 4, a decoupling controller with close-loop stability is designed and the step responses are used to verify the decoupled system.

In question 5, a servo controller for set point tracking and disturbance rejection is designed by LQR method. By introducing an observer, we enable the full-order state feedback.

In question 6, I try to figure out but limited by time I can not finish it.

From this project, I get a deeper understanding of the control system and explore the intersting application . From the practical design project for the specific control system, I can leverage the knowledge and theory I learned in the linear system course.

9 APPENDIX

9.1 Matlab code

```
%% Q1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear
close all
set(0, 'DefaultFigureWindowStyle', 'docked');
matriculation_number = 'A0295779Y';
params = getParams(matriculation_number);
A = params{1};
B = params{2};
C = params{3};
x0 = params{4};
D = [0, 0;
    0, 0];
% compute the zeros
z = tzero(A, B, C, D)
p = eig(A)

%% pole placement
syms s;
ts=30;  % settling time 30s
mp=0.1; % overshoot 10%

real_abs_min = 4/ts;
real_abs = ceil(real_abs_min*10)/10;

ep_min = abs(log(mp))/sqrt((log(mp))^2+pi^2);
ep = ceil(ep_min*10)/10+0.2; % set ep=0.8
wn = real_abs/ep;


% verify the ts condition:
ts_designed = 4/(ep*wn);
if ts_designed < ts
    fprintf("Designed poles satisfy: ts=%f smaller then %f.\n", 4/(ep*wn), ts);
    fprintf("Choose ep=%f, wn=%f\n", ep, wn);
end

% Use wn and ep to generate poles: s^2+2*ep*wn+wn^2 = 0
lamda1 = -ep*wn+wn*sqrt(1-ep^2)*1i
lamda2 = -ep*wn-wn*sqrt(1-ep^2)*1i
% locate other extra poles to be 2-5 times faster than the dominate ones:
lamda3 = real(lamda1)*3;

```

```
% Desired poles
original_poles = [lamda1, lamda2, lamda3];           % original
changePoles_1 = [lamda1-0.1,  lamda2-0.1,  lamda3]; % abs(real) increase
changePoles_2 = [lamda1+0.05, lamda2+0.05, lamda3]; % abs(real) decrease
changePoles_3 = [lamda1+0.1i, lamda2-0.1i, lamda3]; % abs(img) increase
changePoles_4 = [lamda1-0.1i, lamda2+0.1i, lamda3]; % abs(img) decrease

desired_poles = changePoles_4; % test different poles
% Compute characteristic polynomial coefficients (target closed-loop polynomials)
desired_char_poly = poly(desired_poles); % the highest order to the lowest order
desired_char_poly = flip(desired_char_poly);


%% Full Rank Pole Placement
syms k11 k12 k13 k21 k22 k23
K_bar = [k11 k12 k13;
        k21 k22 k23];

% For MIMO system:
% First compute the controllability matrix and check if it's full-rank
n = size(A,1);
m = size(B,2);
Wc = [];
for i = 1:n
    Wc = [Wc, A^(i-1) * B];
end
disp('Controllability matrix Wc:');
disp(Wc);
if rank(Wc) < size(A, 1)
    error('The system is not controllable we cannot conduct pole placement.');
else
    fprintf("Rank(Wn)=%d, the system is controllable, let's perform full-rank pole placement.\n", rank(Wc)
    % Extract 3 independent vectors and form square matric C:
    [R, pivot_columns] = rref(Wc);
    pivot_columns = pivot_columns(1:n);
    b1_column = pivot_columns(mod(pivot_columns, m) == 1);
    b2_column = pivot_columns(mod(pivot_columns, m) == 0);
    vec_order = [b1_column,b2_column];
    Cmatrix = Wc(:, vec_order(1:n));
    disp('Cmatrix:');
    disp(Cmatrix);
    C_inv = inv(Cmatrix);
    % Take the d1_th and d2_th row out of C_inv, and compute T
    d1 = length(b1_column);
    d2 = length(b2_column);

    T=[];
    for i = 1:d1
        T = [T;C_inv(d1, :)*A^(i-1)];
    end
```

```
    dd = d1+d2;
    for i = 1:d2
        T = [T;C_inv(dd, :)*A^(i-1)];
    end
    T(abs(T)<10^(-10))=0;
    disp('T:');
    disp(T);
    A_bar = T*A/(T);%*inv(T);
    B_bar=T*B
    A_bar(abs(A_bar)<10^(-10))=0;
    B_bar(abs(B_bar)<10^(-10))=0;

    % Compare Ad and A_bar-B_bar*K
    Acl=A_bar-B_bar*K_bar;
    Ad = [0    1    0;
          0    0    1;
          -desired_char_poly(1:n)];
    % Solve K
    rotation=Acl==Ad;
    K_num=solve(rotation);
    K_ans=struct2array(K_num);
    K_ans=double(K_ans);
    K_ba=[K_ans(1:3);
          K_ans(4:6)];
    K = K_ba*T;
    disp('K:');
    disp(K);
end



%% Plot the performance

Af=A-B*K;

% compute the zeros, they didn't change (s=-0.2512)
z_pp = tzero(Af, B, C, D);

% state-space model form
sys=ss(Af,B,C,0);

t=0:0.02:50;
len = size(t,2);
u0=zeros(len,2);

% zero inputs and x0 initial state
[y,tout,x]=lsim(sys,u0,t,x0);

% plot zero input response
figure()
```

```
plot(t,x)
grid on
legend('Ca','T','T_j')
xlabel('time')
ylabel('state')
title('State (zero input)')

figure()
plot(t,y)
grid on
legend('T','T_j')
xlabel('time')
ylabel('output')
title('Output (zero input)')

for i = 1:length(t)
    u_fb(i,:) = -K*x(i,:)';
end
figure()
plot(t,u_fb)
grid on
legend('F','F_j')
xlabel('time')
ylabel('control signal')
title('Control signal (zero input)')

% zero state, step response
figure()
step(sys);
grid on


disp(desired_poles)
%% Simulink
% open model
addpath('../models');
open_system('Q1stateFeedback')
set_param('Q1stateFeedback/Input_signal', 'Time', '0', 'SampleTime', '0.02');

% zero inputs and x0 initial state
set_param('Q1stateFeedback', 'LoadInitialState', 'on')
set_param('Q1stateFeedback', 'InitialState', 'x0')
set_param('Q1stateFeedback/extend', 'Gain', '[0;0]'); % F=0, F_j=0
simOut_x0 = sim('Q1stateFeedback');

% zero-state step response
x0_zero=[0,0,0];
set_param('Q1stateFeedback', 'InitialState', 'x0_zero')

set_param('Q1stateFeedback/extend', 'Gain', '[0;1]'); % F=0, F_j=1
```

```
simOut_01 = sim('Q1stateFeedback');
disp('F=0, F_j=1');
transient_rc(simOut_01);

set_param('Q1stateFeedback/extend', 'Gain', '[1;0]'); % F=1, F_j=0
simOut_10 = sim('Q1stateFeedback');
disp('F=1, F_j=0');
transient_rc(simOut_10);

% close model
close_system('model_name', 0);

%% Q2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear;
close all
addpath('../')
set(0, 'DefaultFigureWindowStyle', 'docked');
matriculation_number = 'A0295779Y';
params = getParams(matriculation_number);
A = params{1};
B = params{2};
C = params{3};
x0 = params{4};
D = [0, 0;
    0, 0];

%% system design by LQR method

% verify controllability
W=[B A*B A^2*B];
disp(W);
assert(rank(W)==3);

% original Q and R
% Q = [100 0  0;
%      0  100  0;
%      0  0 100];
% R = [1   0;
%      0   1];

% change to meet the transient performance specifications.
Q = [0.1   0  0;
     0    80   0;
     0   0    50];
R = [0.1   0;
     0    1];

K = solve_lqr(A, B, Q, R);
%% LQR function
```

```
[K1,~,P]=lqr(A,B,Q,R);

epsilon = 1e-3;
assert( norm(K - K1) < epsilon, 'all');

function K = solve_lqr(A, B, Q, R)
    gamma=[A -B/R*B';-Q -A'];
    [eig_vector,eig_value]=eig(gamma);
    eig_value_sum=sum(eig_value);
    vueigen=eig_vector(:,real(eig_value_sum)<0);
    P=vueigen(4:6,:)/vueigen(1:3,:);
    K=real(inv(R)*B'*P);

    disp('LQR method, K:');
    disp(K);
end
%% PLOT figure
Af=A-B*K;
sys=ss(Af,B,C,0);


if 1
t=0:0.01:10;


len = size(t,2);
u0=zeros(len,2);

% zero inputs and x0 initial state
[y,tout,x]=lsim(sys,u0,t,x0);

figure()
plot(t,x)
legend('Ca','T','T_j')
xlabel('time')
ylabel('state')
title('zero inputs and x0 initial state')

figure()
plot(t,y)
legend('T','T_j')
xlabel('time')
ylabel('output')
title('zero inputs and x0 initial state')

for i = 1:length(t)
    u_in(i,:) = -K*x(i,:)';
end
figure()
plot(t,u_in)
```

```
legend('F','F_j')
xlabel('time')
ylabel('control signal')
title('zero inputs and x0 initial state')

% zero state, step response
figure()
step(sys)
grid on
end

%% Simulink
% open model
addpath('../models');
open_system('Q2_LQR');
set_param('Q2_LQR', 'StopTime', '50');
set_param('Q2_LQR/Input_signal', 'Time', '0','SampleTime', '0.02');


% zero inputs and x0 initial state
set_param('Q2_LQR', 'LoadInitialState', 'on')
set_param('Q2_LQR', 'InitialState', 'x0')
set_param('Q2_LQR/extend', 'Gain', '[0;0]'); % F=0, F_j=0
simOut_x0 = sim('Q2_LQR');

% zero-state step response
x0_zero=[0,0,0];
set_param('Q2_LQR', 'InitialState', 'x0_zero')

set_param('Q2_LQR/extend', 'Gain', '[0;1]'); % F=0, F_j=1
simOut_01 = sim('Q2_LQR');
disp('F=0, F_j=1');
transient_rc(simOut_01);

set_param('Q2_LQR/extend', 'Gain', '[1;0]'); % F=1, F_j=0
simOut_10 = sim('Q2_LQR');
disp('F=1, F_j=0');
transient_rc(simOut_10);

% close model
close_system('Q2_LQR', 0);


%% Q3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear;
close all
addpath('../')
set(0, 'DefaultFigureWindowStyle', 'docked');
matriculation_number = 'A0295779Y';
```

```
params = getParams(matriculation_number);
A = params{1};
B = params{2};
C = params{3};
x0 = params{4};
D = [0, 0;
    0, 0];

Q = [0.1   0  0;
     0    80   0;
     0   0    50];
R = [0.1   0;
     0    1];

K = solve_lqr(A, B, Q, R);
%% LQR function
[K1,~,P]=lqr(A,B,Q,R)

epsilon = 1e-3;
assert( norm(K - K1) < epsilon, 'all');

function K = solve_lqr(A, B, Q, R)
    gamma=[A -B/R*B';-Q -A'];
    [eig_vector,eig_value]=eig(gamma);
    eig_value_sum=sum(eig_value);
    vueigen=eig_vector(:,real(eig_value_sum)<0);
    P=vueigen(4:6,:)/vueigen(1:3,:);
    K=real(inv(R)*B'*P);

    disp('LQR method, K:');
    disp(K);
end

Af=A-B*K;
sys=ss(Af,B,C,0);
orig_pole=pole(sys)

%% full order pole placement,
% A_ba = A', B_ba = C', K_ba = L'
syms s
desired_poles = [-4,-20000,-15000];
change1 = [-1,-17500,-7210];
change2 = [-4,-86100,-36100];
desired_char_poly = poly(change2);
desired_char_poly = flip(desired_char_poly);

syms l11 l12 l13 l21 l22 l23
L = [l11 l12 l13;
        l21 l22 l23];

```

```
n = size(A,1);
m = size(B,2);
W_bar=[C' A'*C' (A')^2*C'];
assert(rank(W_bar)==3);
% Extract 3 independent vectors and form square matric C:
[R, pivot_columns] = rref(W_bar);
pivot_columns = pivot_columns(1:n);
b1_column = pivot_columns(mod(pivot_columns, m) == 1);
b2_column = pivot_columns(mod(pivot_columns, m) == 0);
vec_order = [b1_column,b2_column];
Cmatrix = W_bar(:, vec_order(1:n));
disp('Cmatrix:');
disp(Cmatrix);

C_inv = inv(Cmatrix);
% Take the d1_th and d2_th row out of C_inv, and compute T
d1 = length(b1_column);
d2 = length(b2_column);

T=[];
for i = 1:d1
    T = [T;C_inv(d1, :)*(A')^(i-1)];
end
dd = d1+d2;
for i = 1:d2
    T = [T;C_inv(dd, :)*(A')^(i-1)];
end
T(abs(T)<10^(-10))=0;
disp('T:');
disp(T);

A_bar = T*(A')/(T); %*inv(T);
B_bar=T*(C');
A_bar(abs(A_bar)<10^(-10))=0;
B_bar(abs(B_bar)<10^(-10))=0;

% Compare Ad and A_bar-B_bar*L
Acl=A_bar-B_bar*L;
Ad = [0    1    0;
      0    0    1;
      -desired_char_poly(1:n)];
% Solve K_bar
rotation=Acl==Ad;
K_num=solve(rotation);
K_ans=struct2array(K_num);
K_ans=double(K_ans);
K_bar=[K_ans(1:3);
      K_ans(4:6)];
K_estimator = K_bar*T;
L_estimator = K_estimator';
```

```
% L_estimator=real(L_estimator);
disp('L_estimator:');
disp(L_estimator);

%% Simulink
% open model

addpath('../models');
open_system('Q3_observer');
set_param('Q3_observer', 'StopTime', '0.01');
set_param('Q3_observer', 'LoadInitialState', 'off');

% zero inputs and x0 initial state
set_param('Q3_observer/extend', 'Gain', '[0;0]'); % F=0, F_j=0
simOut_x0 = sim('Q3_observer');

% zero-state step response
set_param('/extend', 'Gain', '[0;1]'); % F=0, F_j=1
simOut_01 = sim('Q2_LQR');
disp('F=0, F_j=1');
transient_rc(simOut_01);

set_param('Q3_observer/extend', 'Gain', '[1;0]'); % F=1, F_j=0
simOut_10 = sim('Q2_LQR');
disp('F=1, F_j=0');
transient_rc(simOut_10);

% close model
close_system('Q3_observer', 0);

%% Q4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear;
close all
addpath('../')
set(0, 'DefaultFigureWindowStyle', 'docked');
matriculation_number = 'A0295779Y';
params = getParams(matriculation_number);
A = params{1};
B = params{2};
C = params{3};
x0 = params{4};
D = [0, 0;
    0, 0];

%% decoupling performance
syms s
epsilon = 1e-9;
for i = 1:3
    if norm((C(1,:)*A^(i-1)*B)-zeros(1,2))>epsilon
```

```
        degree1 = i;
        break
    end
end
for i = 1:3
    if norm((C(2,:)*A^(i-1)*B)-zeros(1,2))>epsilon
        degree2 = i;
        break
    end
end
B_star = [C(1,:)*A^(degree1-1)*B;
          C(2,:)*A^(degree2-1)*B];

Phi_A1=A+10*eye(3);
Phi_A2=A+10*eye(3);

C_doublestar = [C(1,:)*Phi_A1;
                C(2,:)*Phi_A2];

F = inv(B_star);
K = F*C_doublestar;
Bf=B*F;
Af = A-B*K;
decouple_model=ss(Af,Bf,C,0);

W=[Bf Af*Bf Af^2*Bf];
assert(rank(W)==3);
p=pole(decouple_model);
z = tzero(decouple_model)
H=C*inv(s*eye(3)-Af)*Bf;

H2 = ss(A-B*K, B*F, C, D);
tf_H = tf(H2)

[~,eigenvalue] = eig(Af)
%% Plot
% non-zero inputs and zero initial state
figure()
step(H2);
grid on

% zero inputs and x0 initial state
t=0:0.01:5;
len = size(t,2);
u0=zeros(len,2);

[y,tout,x]=lsim(decouple_model,u0,t,x0);

figure()
plot(t,x)
```

```
grid on
legend('x1','x2','x3')
xlabel('time')
ylabel('state')
title('zero inputs and x0 initial state')

figure()
plot(t,y)
grid on
legend('y1','y2')
xlabel('time')
ylabel('output')
title('zero inputs and x0 initial state')

%% Simulink
% open model
addpath('../models');
open_system('Q4_decoupling');
set_param('Q4_decoupling', 'StopTime', '50');

set_param('Q4_decoupling', 'LoadInitialState', 'on')
set_param('Q4_decoupling', 'InitialState', 'x0')

% zero inputs and x0 initial state
set_param('Q4_decoupling/extend', 'Gain', '[0;0]'); % F=0, F_j=0
simOut_x0 = sim('Q4_decoupling');

% zero-state step response
x0_zero=[0,0,0];
set_param('Q4_decoupling', 'InitialState', 'x0_zero')

set_param('Q4_decoupling/extend', 'Gain', '[0;1]'); % F=0, F_j=1
simOut_01 = sim('Q4_decoupling');
disp('F=0, F_j=1');
transient_rc(simOut_01);

set_param('Q4_decoupling/extend', 'Gain', '[1;0]'); % F=1, F_j=0
simOut_10 = sim('Q4_decoupling');
disp('F=1, F_j=0');
transient_rc(simOut_10);

% close model
close_system('Q4_decoupling', 0);

%% Q5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear;
close all
addpath('../')
close_system('Q2_LQR', 0);
```

```
set(0, 'DefaultFigureWindowStyle', 'docked');
matriculation_number = 'A0295779Y';
params = getParams(matriculation_number);
A = params{1};
B = params{2};
C = params{3};
x0 = params{4};
ysp = params{5};
w = params{6};
D = [0, 0;
    0, 0];

%% Servo controller
% verify controllability
Qc = [A B; C zeros(2,2)];
assert(rank(Qc)==5);
% augmented state space
A_bar=[A zeros(3,2);
      -C zeros(2,2)];

B_bar=[B;zeros(2,2)];

B_w_bar=[B;zeros(2,2)];

B_r_bar=[zeros(3,2);eye(2)];

C_bar=[C,zeros(2,2)];

Q = eye(5);
R = eye(2);

gamma=[A_bar -B_bar/R*(B_bar');-Q -A_bar'];
[eig_vector,eig_value]=eig(gamma);
eig_value_sum=sum(eig_value);
vueogen=eig_vector(:,real(eig_value_sum)<0)
P=vueogen(6:10,:)/vueogen(1:5,:);
K_calculated=real(inv(R)*(B_bar')*P);

K1=K_calculated(:,1:3);
K2=K_calculated(:,4:5);

%% full order Observer LQR method
% A_ba = A', B_ba = C', K_ba = L'
Qbar = eye(3);
Rbar = eye(2);

Phi1 = [A',-C'/Rbar*C;-Qbar,-A];

[eig_vector_observed,eig_value_observed]=eig(Phi1);
eig_value_observed_sum=sum(eig_value_observed);
```

```
vueigen_observered=eig_vector_observed(:,real(eig_value_observed_sum)<0);
P_observed = vueigen_observered(4:6,:)/vueigen_observered(1:3,:);
Kbar = real(inv(Rbar)*C*P_observed);
L=Kbar';
L=real(L);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% function for getting parameters
function outputArgs = getParams(matriculation_number)
    if nargin < 1
        matriculation_number = 'A0295779Y';
    end

    a = str2num(matriculation_number(5));
    b = str2num(matriculation_number(6));
    c = str2num(matriculation_number(7));
    d = str2num(matriculation_number(8));

    ab0 = a*100+b*10;
    ba0 = b*100+a*10;
    dc = d*10+c;
    cd0 = c*100+d*10;


    A = [-1.7   -0.25   0;
        23      -30     20;
        0   -200-ab0    -200-ba0];
    B = [3+a    0;
        -30-dc  0;
        0   -420-cd0];
    C = [0  1   0;
         0  0   1];
    x0 = [1  100   200];


    % Q5
    y_sp = [100; 150];
    w = [-2; 5];

    % Q6
    x_sp = [5; 250; 300];
    % objective function:  J = 0.5*(xs-xsp).T*W*(xs-xsp)
    W = diag([a+b+1   c+4   d+5]);

    outputArgs = {A,B,C,x0,y_sp,w,x_sp,W};
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% function for analyse transient response characteristics
function transient_rc(simOut)
    signal = simOut.logsout.get('y'); % check the output signal
```

```
    time = signal.Values.Time;
    data = signal.Values.Data;
    % Use stepinfo to calculate the transient response characteristics
    info = stepinfo(data, time, 'SettlingTimeThreshold', 0.02);

    y_final_value = [];
    y_peak = []; %for compute overshoot
    y_os = []; % overshoot
    y_ts = []; % settling time
    for i =1:size(data,2)
        dt = data(:,i);
        % check the steady state and return settling time (if available)
        [res, pk_idx, final_value, ts]= check_steady(dt, time, 1e-6);
        if res == 1
            fprintf('y[%d] final value:%f, settling time:%f\n', i, final_value, ts);
            fprintf('y[%d] peak value:%f, time:%f\n', i, dt(pk_idx), time(pk_idx));
        else
            disp('Error: The simulation time should be extended.');
            break;
        end
        y_final_value = [y_final_value, final_value];
        y_peak = [y_peak, dt(pk_idx)];
        % compute overshoot
        if(y_final_value<1e-6)
            os = 0;
        else
            os = abs((y_peak(i)-y_final_value(i))/y_final_value(i))*100;
        end
        y_os = [y_os, os];
        fprintf('y[%d] overshoot:%f%%\n', i, os);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% function to check steady state, if achieved, find settling time
function [res,peak_idx,final_value,ts]= check_steady(data, time, thresh)
    % initial values:
    res = 0;
    peak_idx = 0;
    final_value = 0;
    ts = 0;

    % Set the differential window size and thresholds for change rate
    diff_window = 10;    % window size: 10 data
    tolerance = thresh;  % thresholds for changing rate
    % Calculate the differential changing rate of the segment
    diff_data = abs(diff(data(end-diff_window:end)));
    mean_data = abs(mean(data(end-diff_window:end)));
    diff_data_ratio = diff_data/mean_data;
    % Determine whether a steady state has been reached
    if (max(diff_data_ratio) < tolerance) || (abs(max(diff_data))<0.01)
```

```
        res = 1;
        final_value = mean(data(end-diff_window:end));
        % search for peak value
        if final_value>0
            % data(data<0) = 0;
            [peak_val, peak_idx] = max(data);
        else
            % data(data>0) = 0;
            [peak_val, peak_idx] = min(data);
        end
        % find settling time (+/-2% around the final value)
        start_index = peak_idx; % start from the max/min value
        index= find(abs(data(start_index:end)-data(peak_idx))<abs(data(peak_idx)*0.02), 1, 'first');
        if ~isempty(index)
            if(index>start_index)
                global_index = index + start_index - 1;
                ts = time(global_index);
            else
                global_index = start_index;
                ts = 0;
            end
        else
            disp('Cannot compute the settling time');
        end
    else
        disp('The signal has not yet reached steady state.');
    end
end
```

BIBLIOGRAPHY

1. *Albertos P.*, *Antonio S.* Multivariable control systems: an engineering approach. — Springer Science & Business Media, 2006. — P. 19–27.