ENSC 483 – Modern Control Systems

Spring 2016

Project 1

Name: Lestley A. Gabo

Student Number: 301170055

Date: April 12, 2016

Research Paper used: "MIMO PID Controller Design Based on Integral-Type Optimal Servomechanism and Its Extension to Model-Following-Type" Hiroyuki KONDO, Yoshimasa OCHI, 2011

2. (Present and explain the mathematical model (state space equations and transfer functions are necessary, differential equation is optional) of the system.)

We consider the lateral-directional motion of the F-16 at sea level and the airspeed of 153.0[m/sec][11]. With the definition of the elements of x are the side-slip angle β [rad], the roll angle ϕ [rad], the roll rate p [rad/sec] and the yaw rate [rad/sec], those of u are the aileron angle δa [deg] and the rudder angle δr [deg], and those of y are the roll angle ϕd [deg] and the side-slip angle βd [deg].

The paper provided explains this example more thoroughly.

```
>> A = [-10 -6.25 0 0; 4 0 1 0; 0 0 -10 -6.25; 0 0 4 0]
```

$$A = \begin{bmatrix} -10 & -6.25 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 0 & 0 & -10 & -6.25 \\ 0 & 0 & 4 & 0 \end{bmatrix}$$

```
>> B = [2 0; 0 0 ; 0 2; 0 0]
```

$$B = \begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix}$$

```
>> C = [0 3.125 0 0; 0 0  0 3.125]
```

$$C = \begin{bmatrix} 0 & 3.125 & 0 & 0 \\ 0 & 0 & 0 & 3.125 \end{bmatrix}$$

```
>> D =0
```

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Inputting the matrixes below into MATLAB. Then using *sys* function and the *tf* function to figure out the transfer function of the system. The answer MATLAB gives is shown below.

```
>> sys = ss(A,B,C,D)
>> originaltf = tf(sys)
originaltf =
  From input 1 to output...
           25
   1:  ---------------
       s^2 + 10 s + 25
   2:  0
  From input 2 to output...
              6.25 s^2 + 62.5 s
   1:  -----------------------------------
       s^4 + 20 s^3 + 150 s^2 + 500 s + 625
            25
   2:  ---------------
       s^2 + 10 s + 25
 Continuous-time transfer function.
```

3. (Simulate the model and present its impulse, step, and ramp responses. Pay attention that you have a separate transfer function from each input to each output.)

Impulse response using command *impulse(sys):*

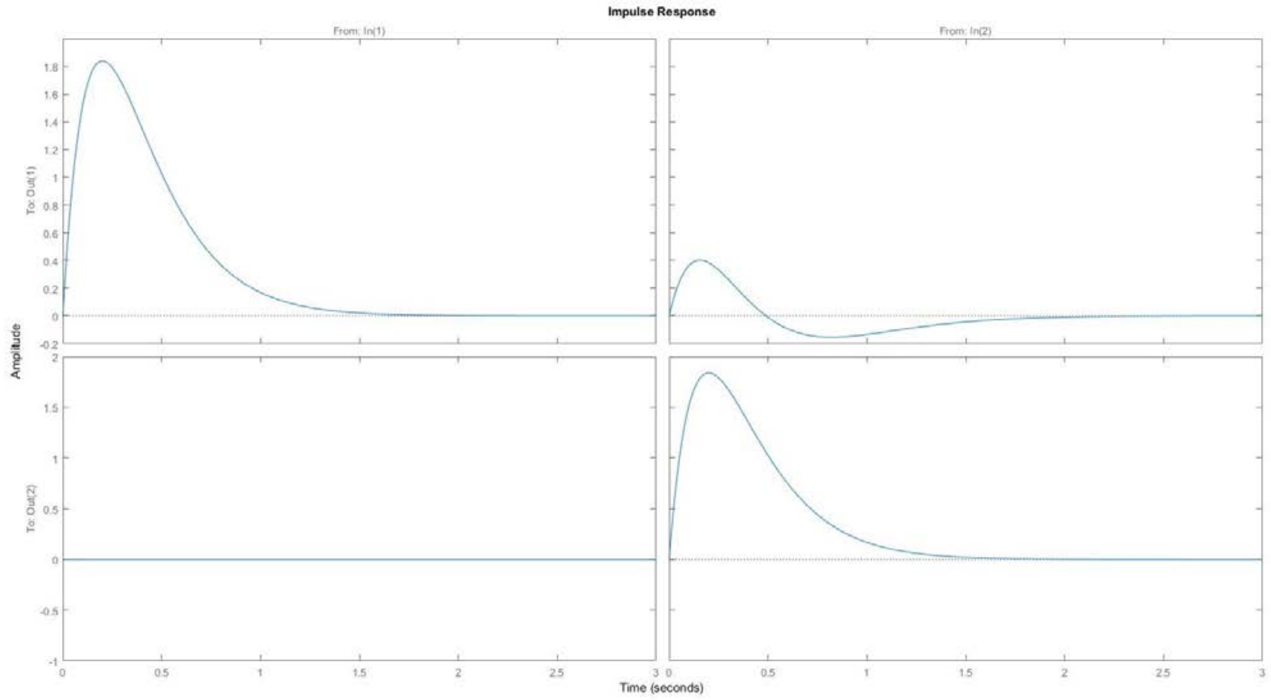

Figure of impulse response

Step response using command *step(sys)*:

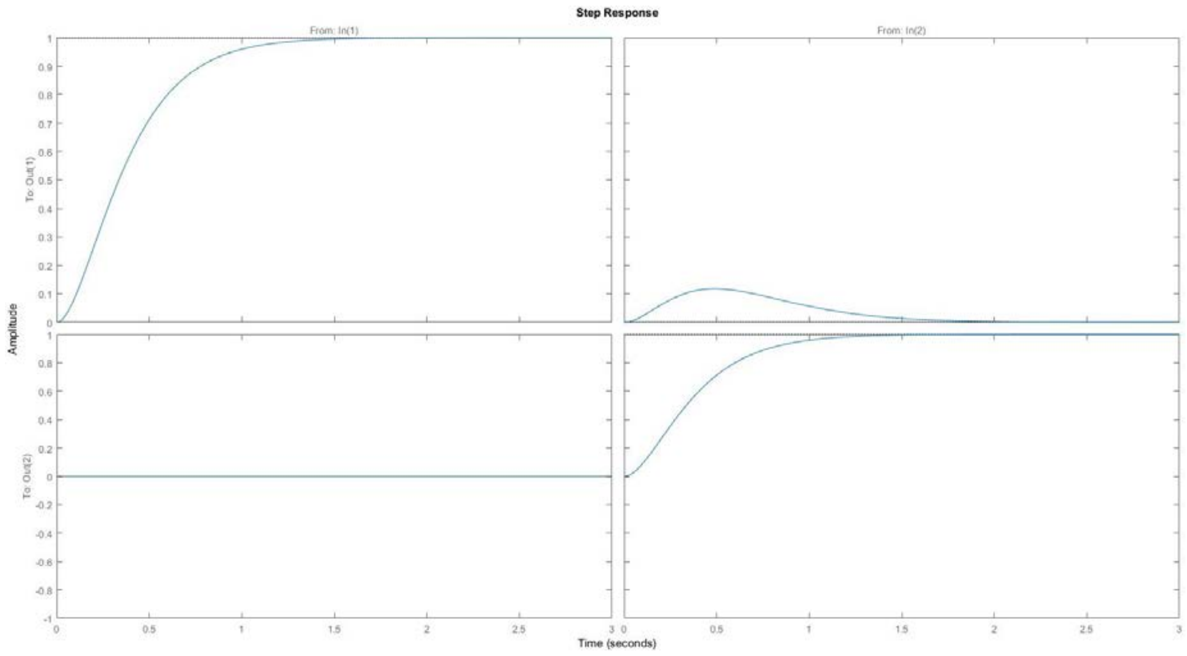

Figure of step response

Ramp response using the codes below, the difference being Gi, CLi, ti, and the tf. The i goes from 1 to 3 because of the 3 different transfer functions (1 of them transfer functions is a zero). The codes below used to find the ramp responses of the transfer functions:

```
G1 = tf([25], [1 10 25])
CL1 = feedback(G1, 1);
step(CL1) % Step response
t1 = 0:.01:5;
lsim(CL1,t1,t1) % Ramp response
```
```
G2 = tf([6.25 62.5 0],[1 20 150 500 625])
CL2 = feedback(G2, 1);
step(CL2) % Step response
t2 = 0:.01:5;
lsim(CL2,t2,t2) % Ramp response
```
```
G3 = tf([25],[1 10 25])
CL3 = feedback(G3, 1);
step(CL3) % Step response
t3 = 0:.01:5;
lsim(CL3,t3,t3) % Ramp response
```

Transfer function 1 ramp response:

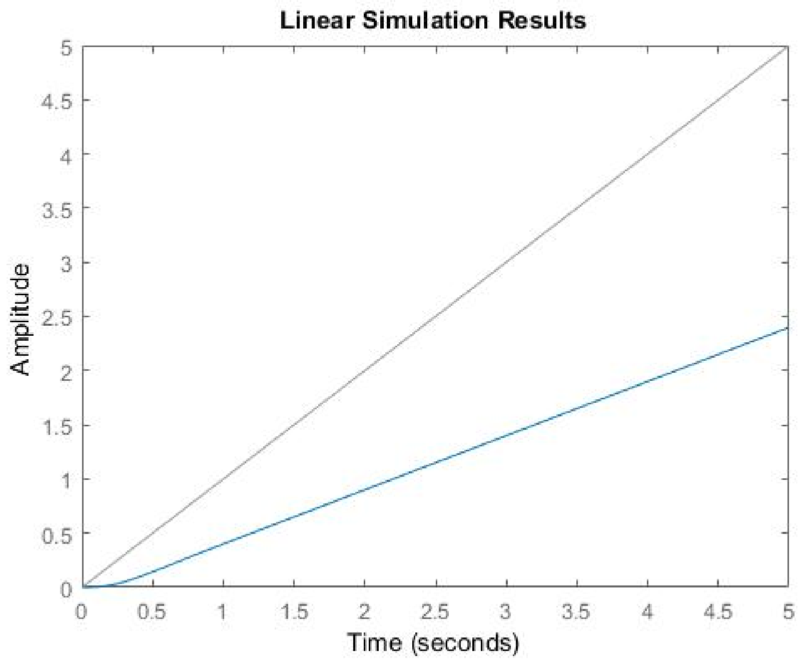

Figure of ramp response `tf([25], [1 10 25])`

Transfer function 2 ramp response:

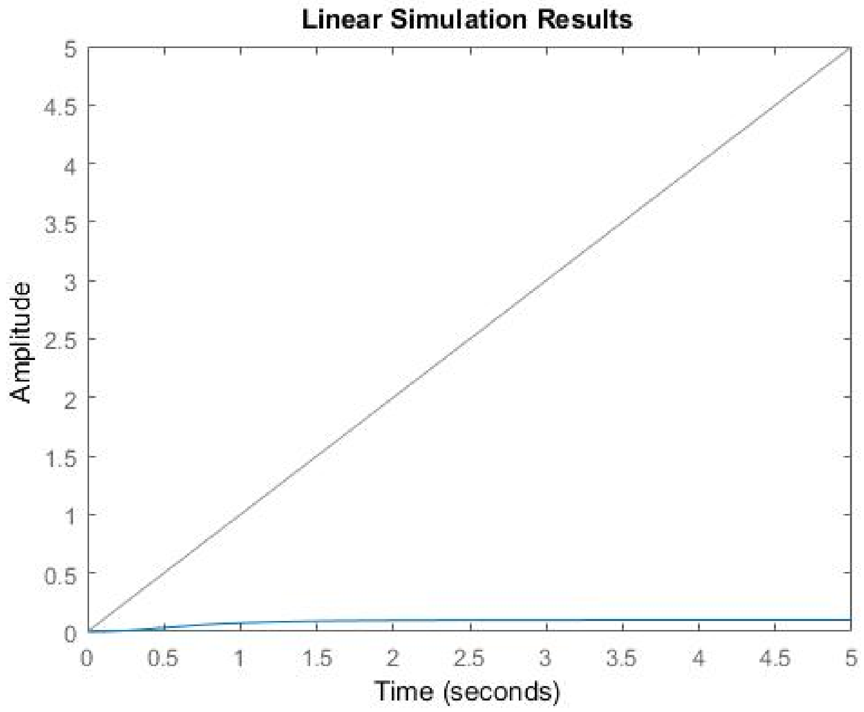

Figure of ramp response `tf([6.25 62.5 0],[1 20 150 500 625])`

Transfer function 3 ramp response:

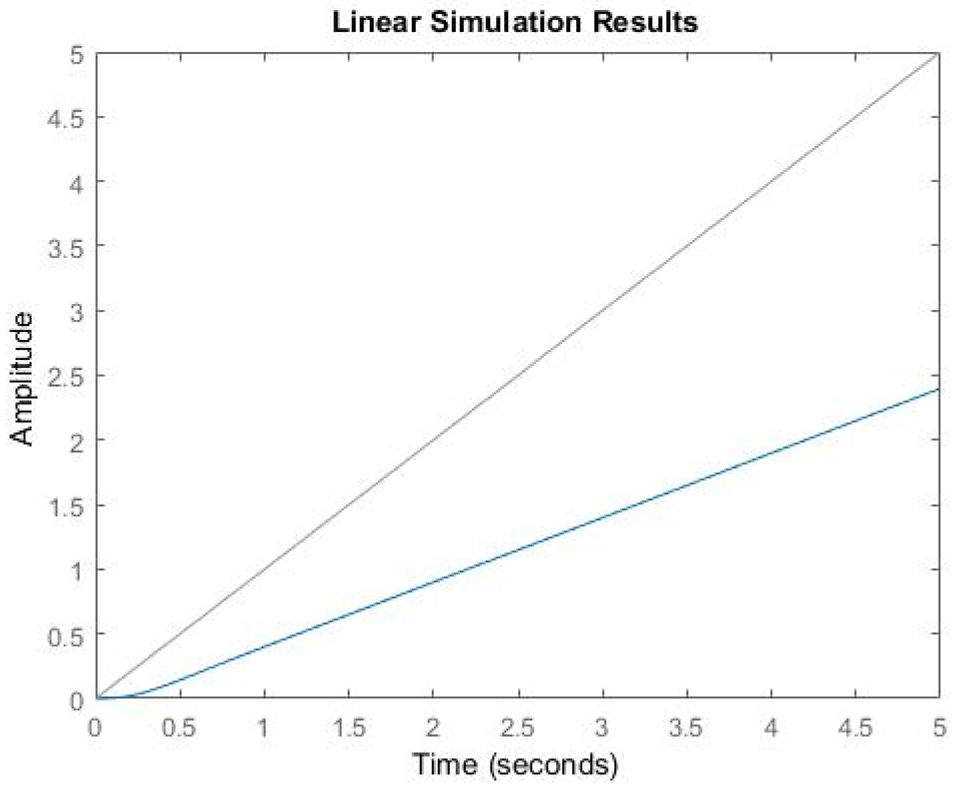

Figure of ramp response `tf([25], [1 10 25])`

4. (Find the eigenvalues and eigenvectors of the system.)
Eigenvalues for the system using *eig(A)* function:

```
>> eigenvalues = eig(A)
```

$$\text{eigenvalues} = \begin{bmatrix} -5 \\ -5 \\ -5 \\ -5 \end{bmatrix}$$

```
>> [eigenvectorV,eigenvectorD] = eig(A)
```

$$\text{eigenvector} = \begin{bmatrix} -5 & 0 & 0 & 0 \\ 0 & -5 & 0 & 0 \\ 0 & 0 & -5 & 0 \\ 0 & 0 & 0 & -5 \end{bmatrix}$$

5. (Find the Jordan form representation of the system.)
Jordan form of the system using *[jordanV, jordanD] = jordan(A)*:

```
>> [jordanV,jordanD] = jordan(A)
```

$$\text{jordanD} = \begin{bmatrix} -5 & 1 & 0 & 0 \\ 0 & -5 & 1 & 0 \\ 0 & 0 & -5 & 1 \\ 0 & 0 & 0 & -5 \end{bmatrix}$$

```
>> [aa bb cc dd P] = canon(A,B,C,D,'modal')
```

```
aa =    -5.0000  -10.2500  -0.4878   -0.3902      bb =    1.5617      0
              0   -5.0000   0.6098    0.4878              1.2494      0
              0         0  -5.0000  -10.2500                   0  1.5617
              0         0        0   -5.0000                   0  1.2494
cc =    -1.9522    2.4402        0         0      dd =         0      0
              0         0  -1.9522    2.4402                   0      0
```

```
P =      0.7809   -0.6247        0         0
         0.6247    0.7809        0         0
              0         0   0.7809   -0.6247
              0         0   0.6247    0.7809
```

6. (Investigate the controllability of the system using two different methods.)

Method1: Using the function *rank(ctrb(sys))* the answer is 4. This is a full column rank and therefore the **system is controllable**.

```
>> cm = rank(ctrb(sys))
cm =
    4
```

Method2: Find rank of [A- λ_nI B] then see if that matrix has a full column rank for every λ_n. Code used:

```
function [rankofeachcolumn] =
controlrankmethodtwo(eigenvalues, matrixA, matrixB)
%UNTITLED Summary of this function goes here
%   controlrankmethod2

 [eig1,eig2,eig3,eig4] = matsplit(eigenvalues,2)

column1 = [matrixA-(eig1*eye(4)) matrixB];
column2 = [matrixA-(eig2*eye(4)) matrixB];
column3 = [matrixA-(eig3*eye(4)) matrixB];
column4 = [matrixA-(eig4*eye(4)) matrixB];
```

```
rankofeachcolumn = [rank(column1) rank(column2)
rank(column3) rank(column4)]
end

>> controlrankmethodtwo(eigenvalues,A,B)

ans =

   4    4    4    4
```

The answer is a full column rank, therefore the **system is controllable**.

7. (Investigate the observability of the system using two different methods.)
Method1: Using the function *rank(obsv(sys))* the answer is 4. This is a full row rank and therefore the **system is observable**.

```
>> om = rank(obsv(sys))
om =
    4
```

Method2: Find rank of [A- λ_nI ; C] then see if that matrix has a full row rank for every λ_n. Code used:

```
function [rankofeachrow] =
observerankmethodtwo(eigenvalues, matrixA, matrixC)
%UNTITLED Summary of this function goes here
%   observerankmethod2
[eig1,eig2,eig3,eig4] = matsplit(eigenvalues,2);

row1 = [matrixA-(eig1*eye(4)) ;matrixC];
row2 = [matrixA-(eig2*eye(4)) ;matrixC];
row3 = [matrixA-(eig3*eye(4)) ;matrixC];
row4 = [matrixA-(eig4*eye(4)) ;matrixC];

rankofeachrow = [rank(row1) ;rank(row2);
rank(row3); rank(row4)]
end

>> observerankmethodtwo(eigenvalues,A,C)

rankofeachrow =

   4

   4

   4

   4
```

The answer a full row rank, therefore the **system is observable**.

8. (Considering your result in the two previous parts (6 and 7), decompose the system into controllable, observable, and minimal realization forms, if necessary.)

The system is both controllable and observable. There is no need to decompose the system.

9. (Is the system BIBO stable? Asymptotically stable? Marginally stable?)

The system is BIBO stable if the eigenvalues are to the left of the s-plane. This means that the eigenvalues should have negative and real parts. Looking at part 4 above we see that all four eigenvalues are in the left hand side of the s-plane. Therefore the **system is BIBO stable**.

For a system to be asymptotically stable all eigenvalues of A must have negative real parts. Therefore the **system is asymptotically stable.**

A system needs to be stable but not asymptotically stable to be marginally stable. This system BIBO stable and it is also asymptotically stable, therefore this **system is not marginally stable**.

10. (Place all the poles of this system at -1 using two methods (the main algorithm and Ackermann's formula))

Main algorithm:

Step 1. Find the characteristic polynomial of A

```
>> cp = charpoly(A)
cp =
   1   20   150   500   625
```

The characteristic polynomial is then:
$\Delta(\lambda) = (1)\lambda^4 + (20)\lambda^3 + (150)\lambda^2 + (500)\lambda + (625)$

The α values are:
$\alpha_1 = 20$; $\alpha_2 = 150$; $\alpha_3 = 500$; $\alpha_4 = 625$

Step 2. Choose a desired characteristic polynomial

Putting all poles into -1. Therefore the desired characteristic polynomial is:
$\widehat{\Delta}_d(\lambda) = (\lambda_d+1)^4 = \lambda_d^4 + 4\lambda_d^3 + 6\lambda_d^2 + 4\lambda_d + 1$

The $\widehat{\alpha}$ values are:
$\widehat{\alpha}_1 = 4$; $\widehat{\alpha}_2 = 6$; $\widehat{\alpha}_3 = 4$; $\widehat{\alpha}_4 = 1$

Step 3. Find kbar.

```
>> kbar = [(20 -4)   (150 - 6)   (500 -4)    (625 -1)]
kbar =
   16   144   496   624
```

$\bar{k} = [\ 16\ \ 144\ \ 496\ \ 624\ ]$

We know that b = Bv, we are choosing v = [0;1], therefore our b:

$$b = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$

Controllability matrix (CM) with the b is found using *ctrb(A,b)*:

```
>> CM = ctrb(A,b)
```

$$CM = \begin{bmatrix} 0 & 0 & -12.5 & 250 \\ 0 & 2 & -20 & 100 \\ 2 & -20 & 150 & -1000 \\ 0 & 8 & -80 & 600 \end{bmatrix}$$

Big lambda is shown below:

$$\Lambda = \begin{bmatrix} 1 & 20 & 150 & 500 \\ 0 & 1 & 20 & 150 \\ 0 & 0 & 1 & 20 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 4. Get the Q and inverse it to get P. k = $\bar{k}$P. Where Q = P^{-1}. The Q equals the controllability matrix multiplied the big lambda Q = CM*Λ

```
>> Pinverse = CM*biglambda
Q = Pinverse;
```

$$Q = \begin{bmatrix} 0 & 0 & -12.5 & 0 \\ 0 & 2 & 20 & 0 \\ 2 & 20 & 50 & 0 \\ 0 & 8 & 80 & 200 \end{bmatrix}$$

Inversing Q to get the P, because Q= P^{-1} by using the code *inv(Q)*:

```
>> P = inv(Q)
```

$$P = \begin{bmatrix} -6 & -5 & 0.5 & 0 \\ 0.8 & 0.5 & 0 & 0 \\ -0.8 & 0 & 0 & 0 \\ 0 & -0.02 & 0 & 0.005 \end{bmatrix}$$

Step 5. Get the k from kbar and the K from the k.

Finally from step 3 above we already have the kbar. The formula for k is kbar multiplied with P:

```
>> k = kbar*P
k =
  -20.4800  -20.4800    8.0000    3.1200
```

k = [-20.4800 -20.4800   8.0000   3.1200]

The big K is v*k

```
>> K = v*k
K =
         0         0         0         0
  -20.4800  -20.4800    8.0000    3.1200
```

K= $\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{-20.4800} & \mathbf{-20.4800} & \mathbf{8.0000} & \mathbf{3.1200} \end{bmatrix}$

Ackermann's formula

Simply use the formula:

$k = -e_n^T C^{-1} \Delta_d(A)$; where en = [0;0;0;…;1]

Or we can use the function *acker* in MATLAB. We already the v and b values from above. From the notes MATLAB always gives negative configuration of ackerman so put a negative sign to compensate and have the same k and K as above.

```
>> ackerk = -acker(A,b,[-1, -1, -1, -1])
ackerk =
  -20.4800  -20.4800    8.0000    3.1200
```

ackerk = [-20.4800 -20.4800   8.0000   3.1200]

The big ackerk is v*ackerk

```
>> ackerK = v*ackerk
ackerK =
         0         0         0         0
  -20.4800  -20.4800    8.0000    3.1200
```

ackerK = $\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{-20.4800} & \mathbf{-20.4800} & \mathbf{8.0000} & \mathbf{3.1200} \end{bmatrix}$

11. (Simulate the open-loop (without controller) and the controlled system using Simulink.)

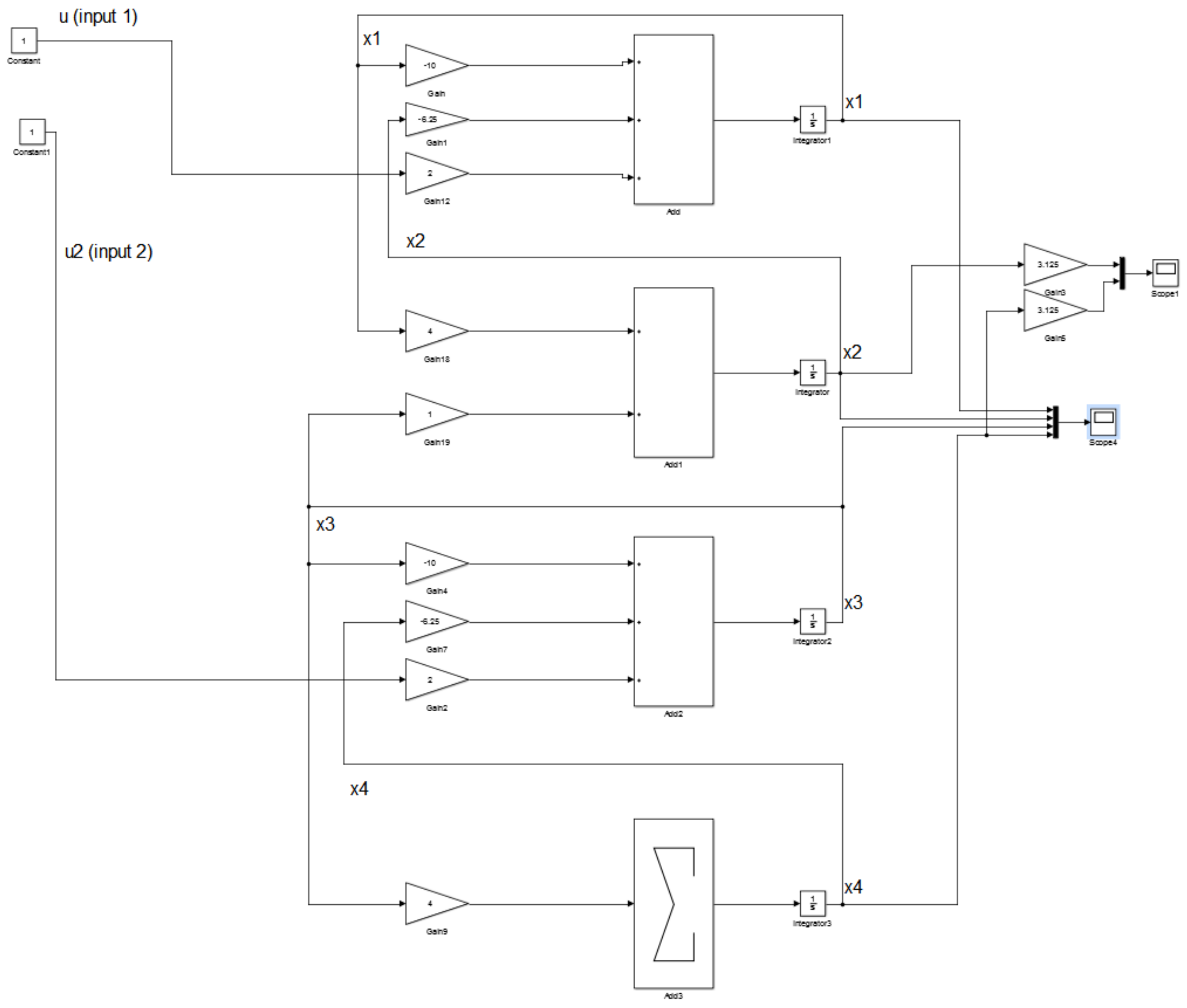

Figure of open-loop without controller for part 11

11.

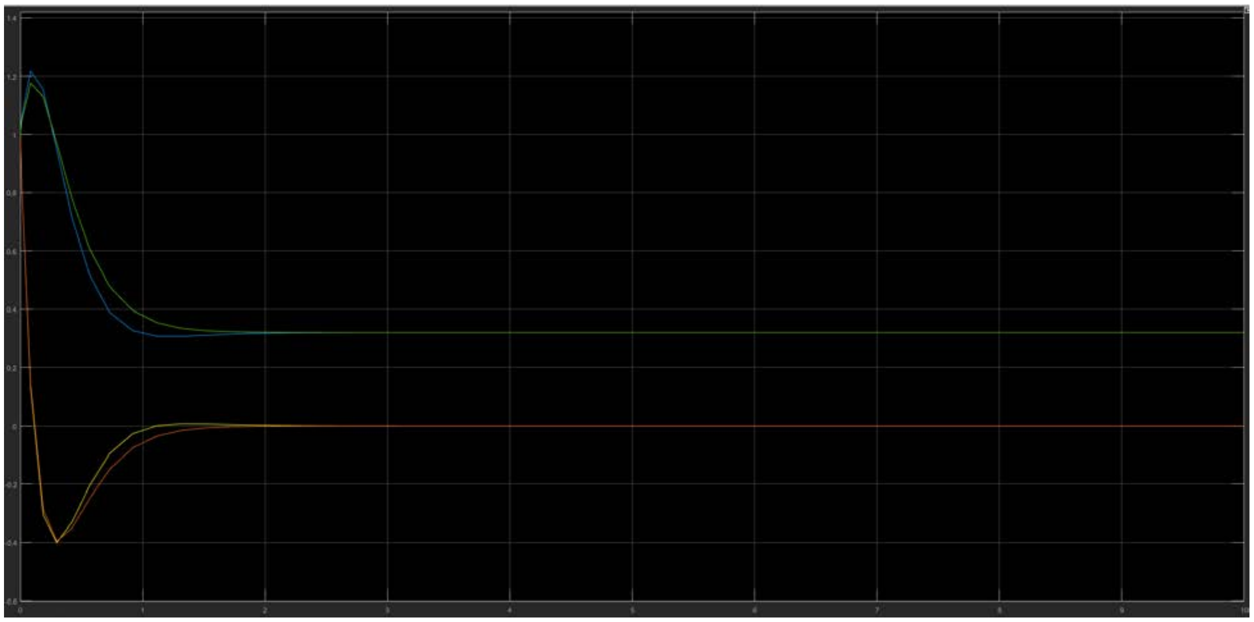

Figure of open-loop output **x without controller**

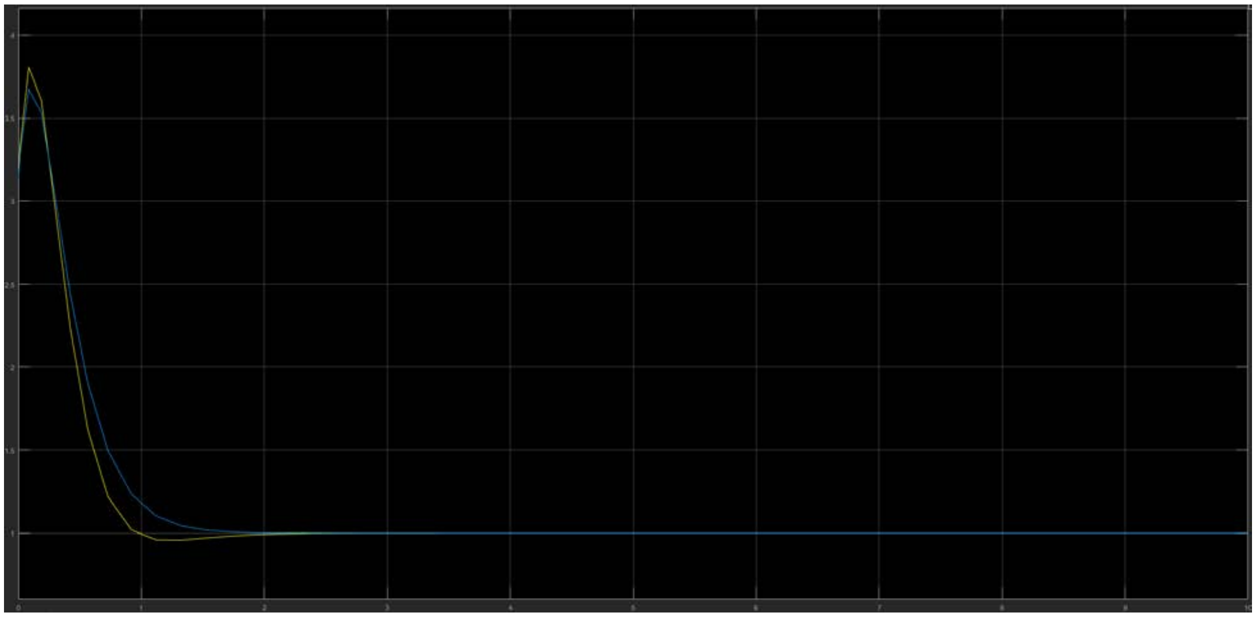

Figure of open-loop output **y without controller**

11.

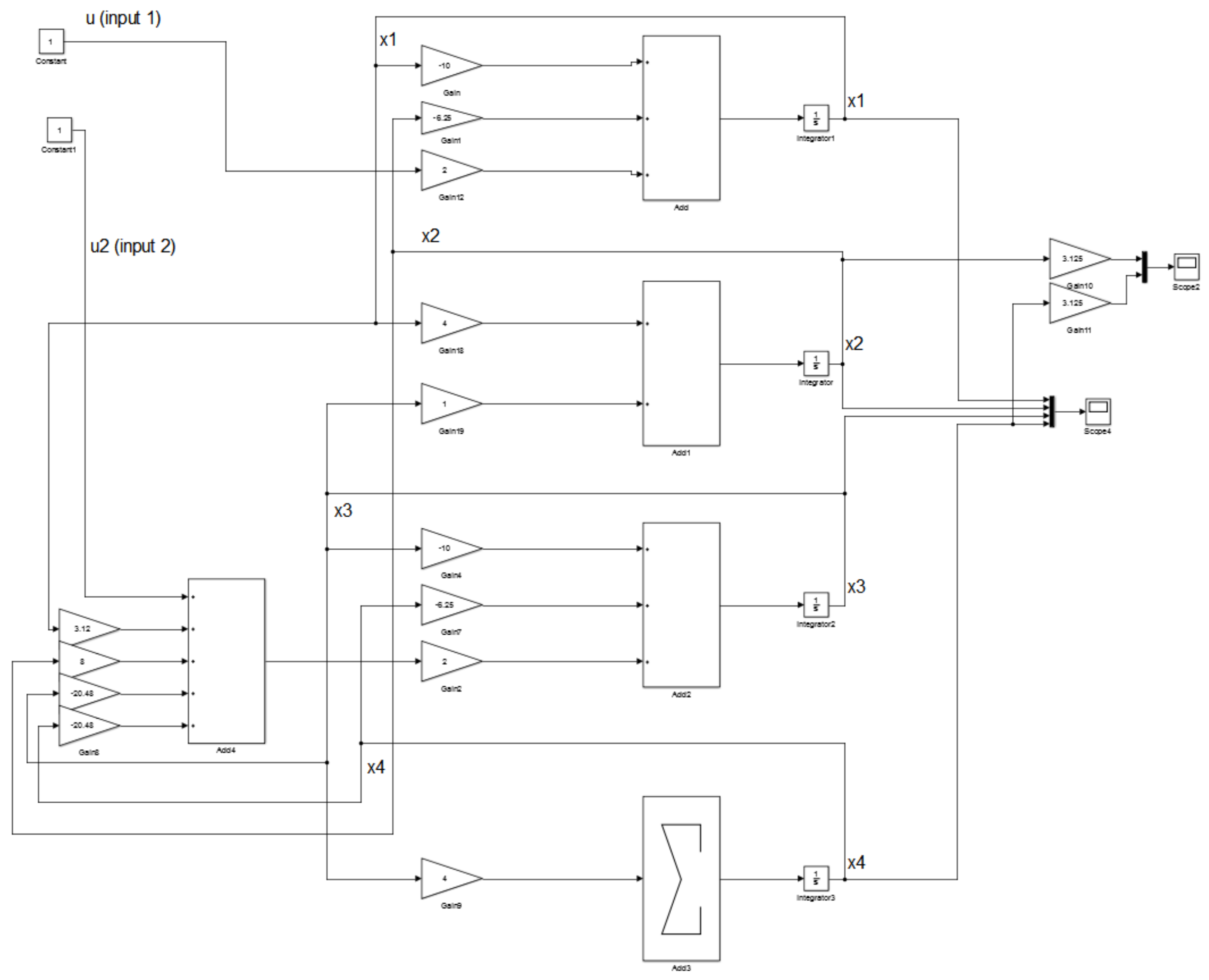

Figure of circuit of open-loop with controller for part 11

11.

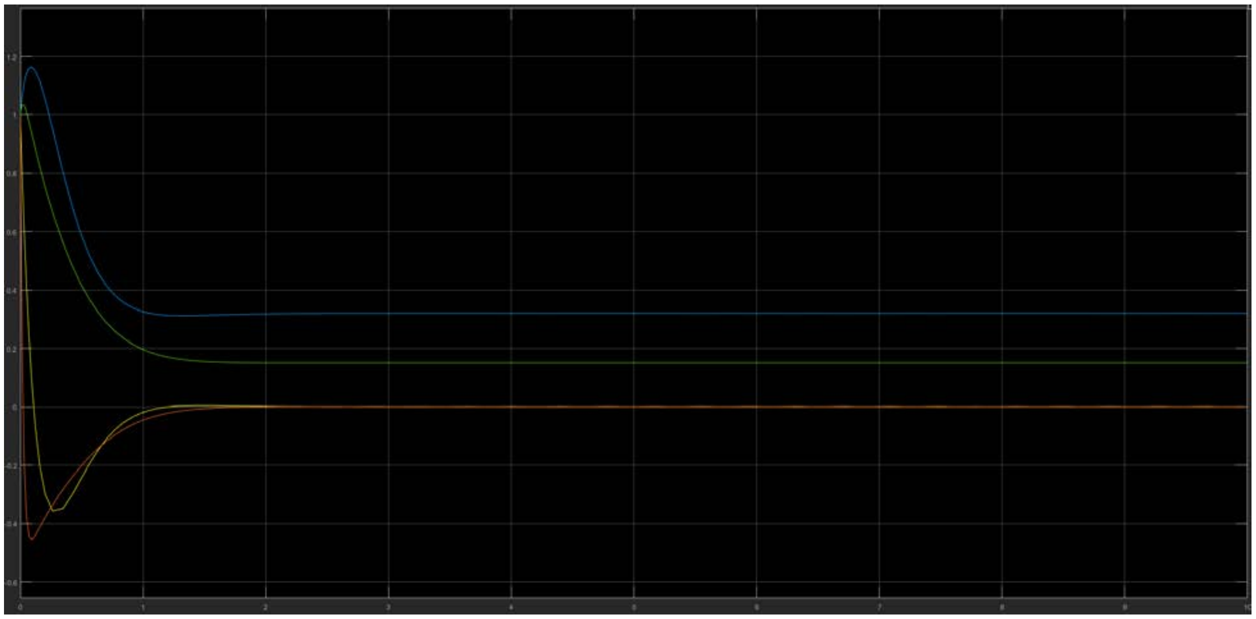

Figure of open-loop output **x with controller**

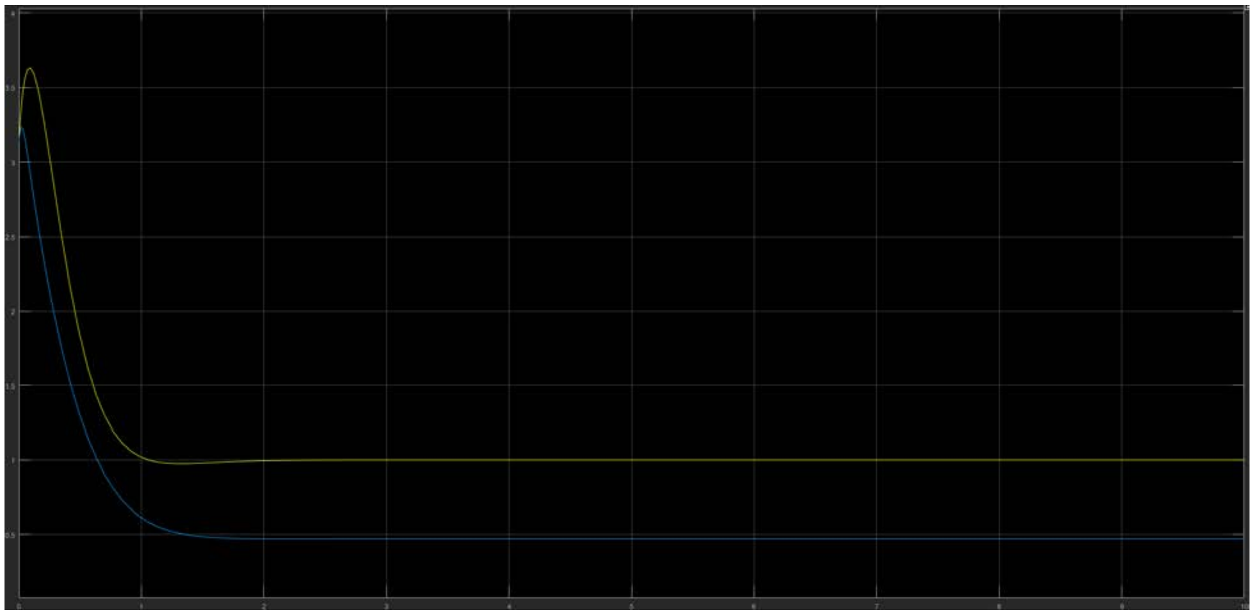

Figure of open-loop output **y with controller**

12. (Choose another set of eigenvalues (poles) and design state feedback controller based on that such that the control criteria IAE, ISE, ITAE, and ITSE (for output) are all better compared to both cases in part 11.)

The reference K values in the research paper:

$$K = \begin{bmatrix} -26.75 & -83.54 & 12.92 & 36.50 \\ 4.189 & 13.31 & 76.77 & 219.0 \end{bmatrix}$$

For some reason, when I use the reference K, I always get a limit error. The Simulink run always gives an error on the integrators of the control criteria.

So I chose the K values similarly proportional to the K I got in step 11.

I played around with the K values for U1 and U2 and found out that proportionally increasing the values of K, I got a better looking graph compared to 11.

$$\mathbf{K=}\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{-20.4800} & \mathbf{-20.4800} & \mathbf{8.0000} & \mathbf{3.1200} \end{bmatrix}$$

For both the K values of U1 and U2, I basically times the negative value by a big value while I divided the positive values by a big value as shown by the values below.

$$\mathbf{K=}\begin{bmatrix} \mathbf{-20.4800 * 999} & \mathbf{-20.4800 * 999} & \mathbf{8.0000/999} & \mathbf{3.1200/999} \\ \mathbf{-20.4800 * 999} & \mathbf{-20.4800 * 999} & \mathbf{8.0000/999} & \mathbf{3.1200/999} \end{bmatrix}$$

See the next pages for the circuit and figures for the x, y, IAE, ISE, ITAE, and ITSE.

12.

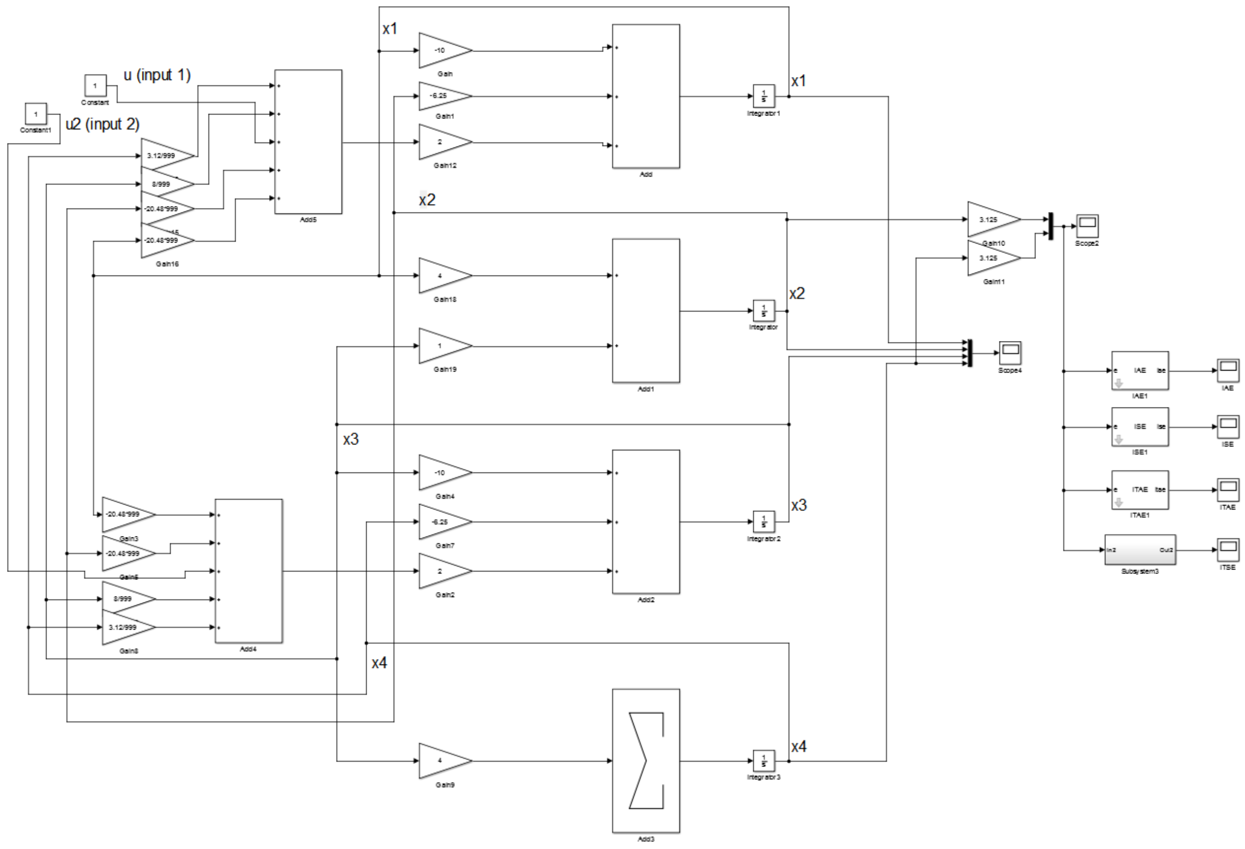

Figure of circuit of state feedback controller for part 12

12.

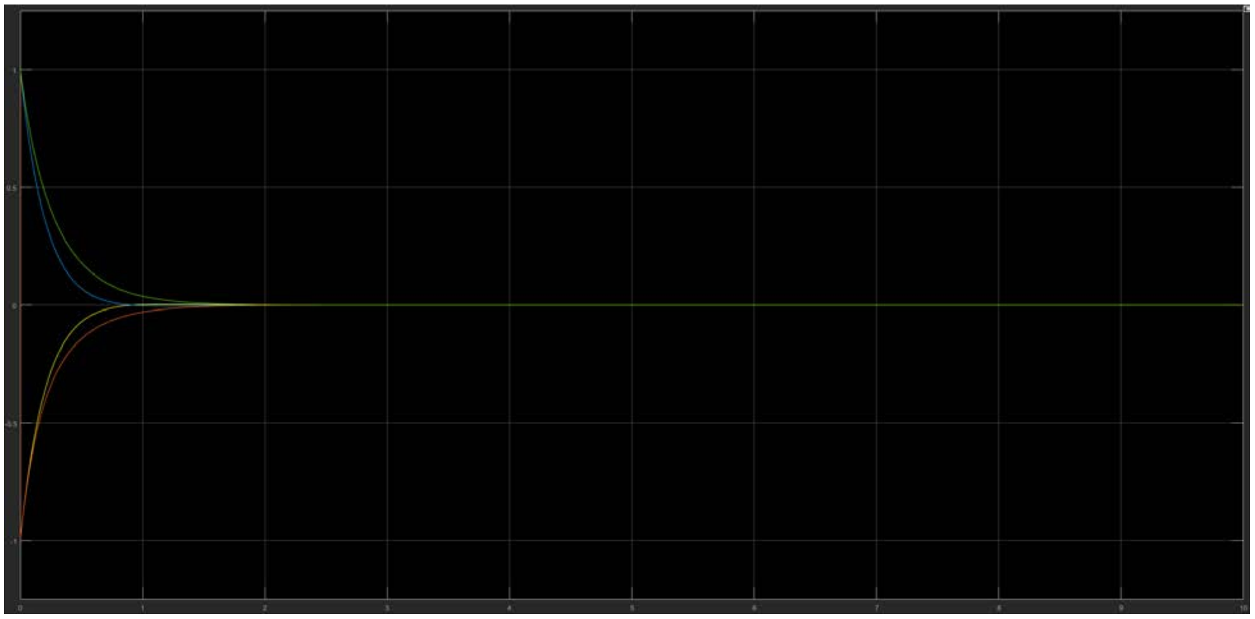

Figure of state feedback controller output **x**

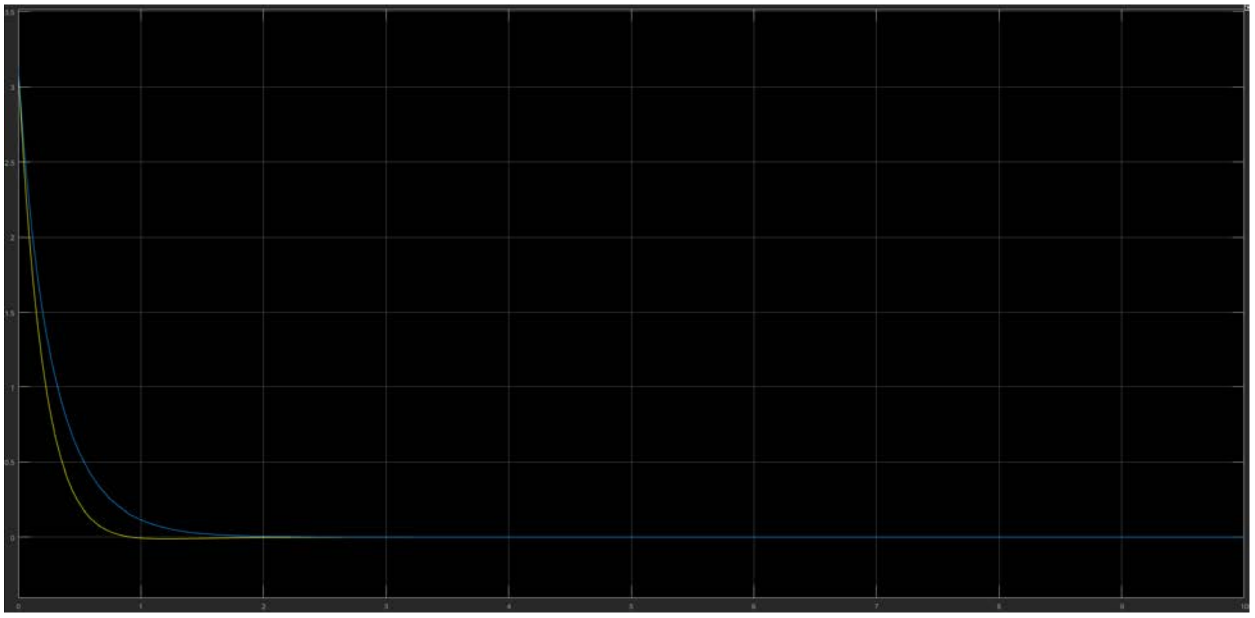

Figure of state feedback controller output **y**

12.

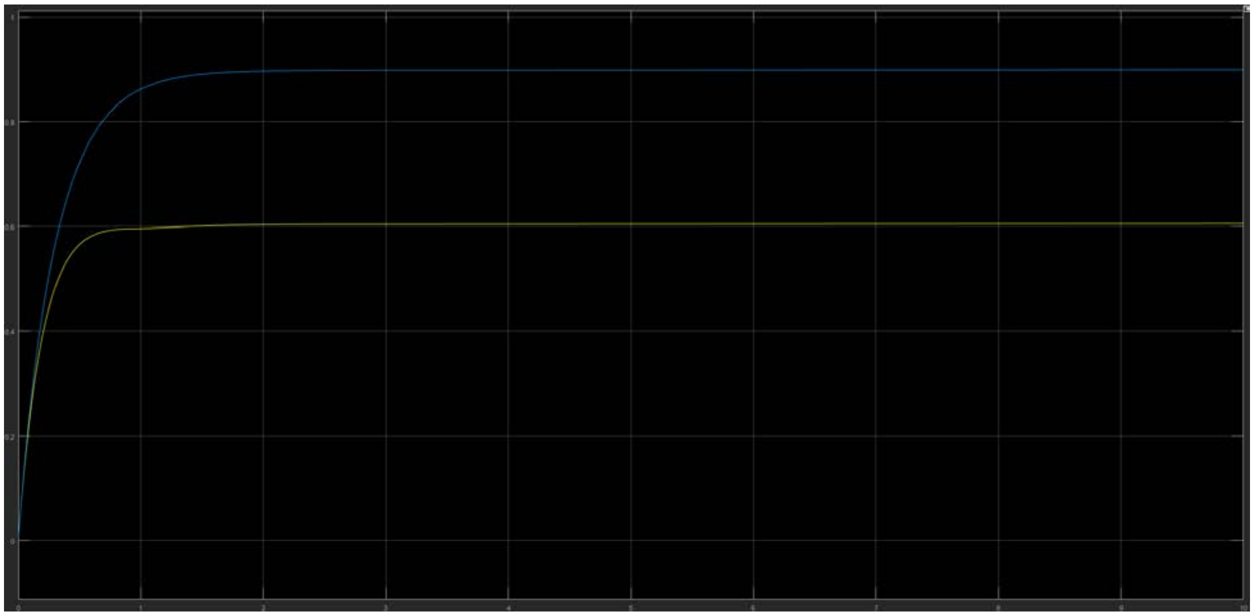

Figure of state feedback controller output **IAE**

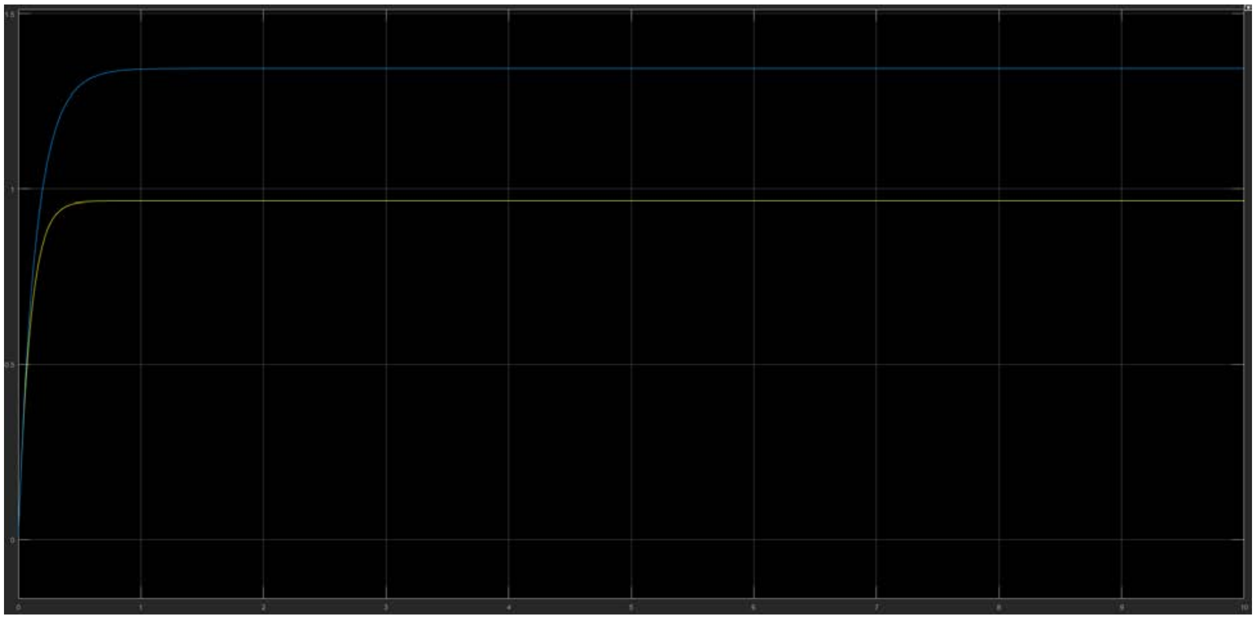

Figure of state feedback controller output **ISE**

12.

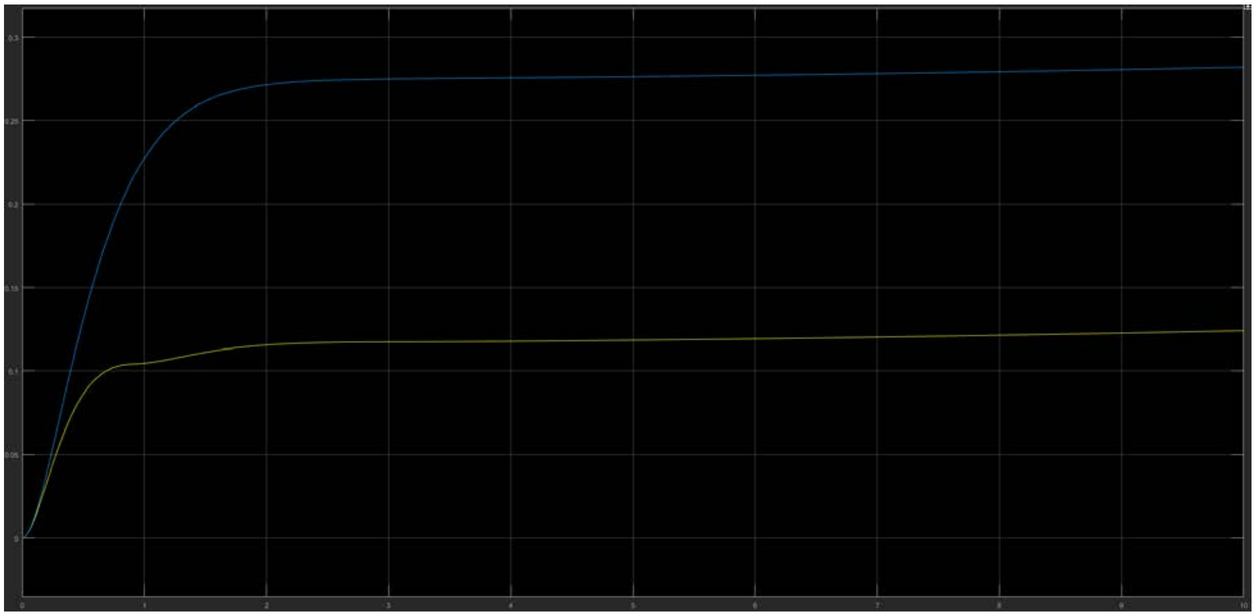

Figure of state feedback controller output **ITAE**

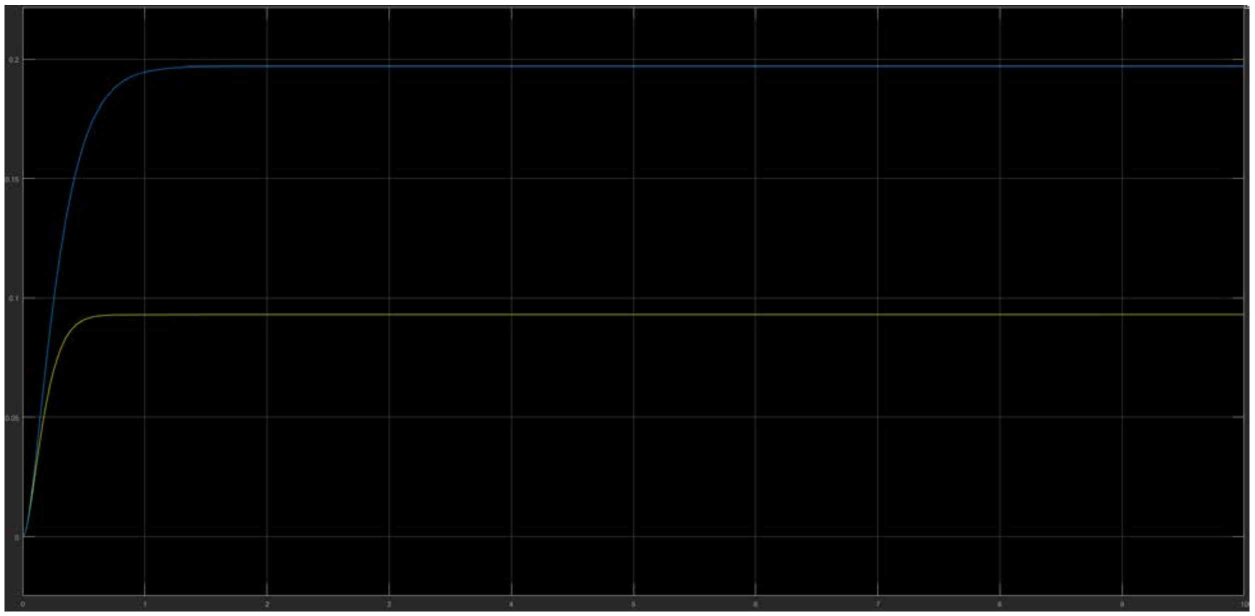

Figure of state feedback controller output **ITSE**