# *MC Project - Phase ii*

Professor: Hamed Kebriaei

January 23, 2025

1-

First, we redefine the system and then analyze its poles:

## Current Pole Analysis

- **-250.000:** This pole is very fast because its real part is highly negative. It causes the system response to decay rapidly in this mode. However, achieving control may require a large gain.
- **15.6011:** This pole is unstable since its real part is positive. It makes the system unstable. In control systems, unstable poles must be shifted to the left of the real axis to ensure stability.
- **-15.6951:** This pole is relatively fast, though much slower than -250.0000. While it can affect the system's performance, the need to modify it depends on the design objectives.

## Guidelines for Adjusting Poles

- **Fast Poles:** These should be shifted further to the left of the real axis to improve system stability and responsiveness. For example:

$$\text{New fast poles: } -100, -120, -150.$$

- **Slow Poles:** These should be positioned closer to the imaginary axis to optimize system performance. For example:

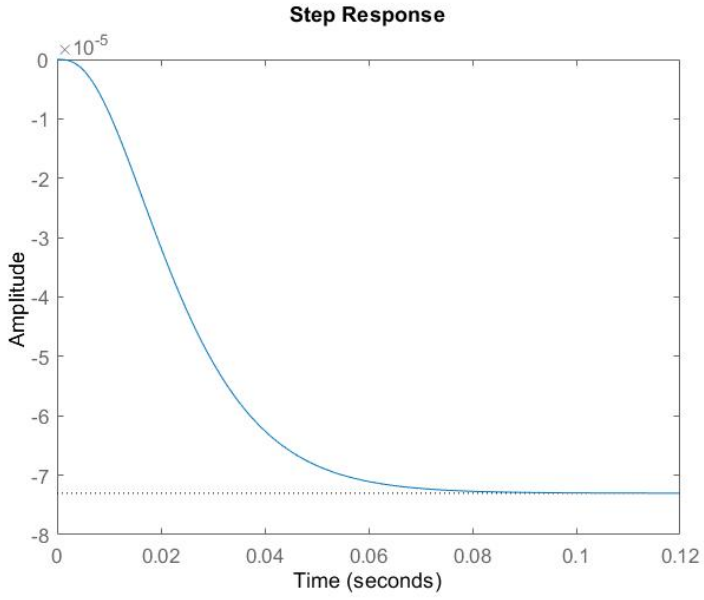
$$\text{New slow poles: } -2, -3, -4.$$

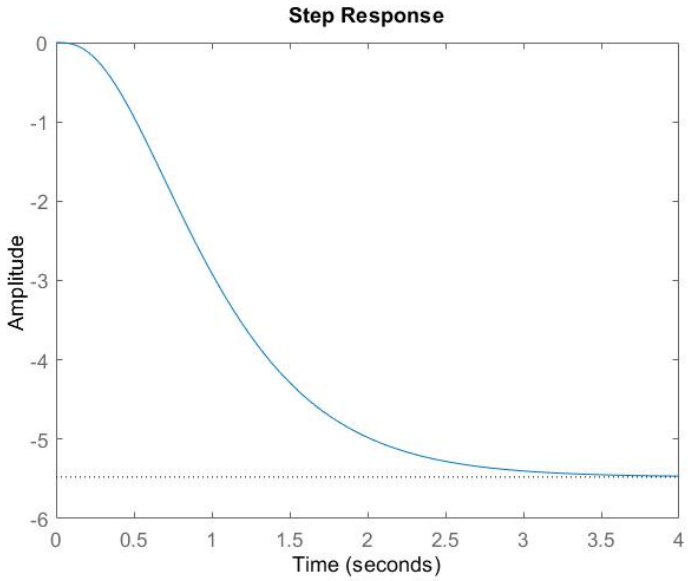
To examine the step response of each designed state feedback, a step-by-step approach is implemented in MATLAB as follows:

```
clc;
clear;

A = [0 1 0; 244.86 -0.094 -26.29; 0 0 -250];
B = [0; 0; 5];
C = [1 0 0];
D = 0;

sys = ss(A, B, C, D);

[num, denum] = ss2tf(A, B, C, D);

disp('Poles of the system:');
poles = roots(denum);
disp(poles);

p_fast = [-120 -100 -150];
disp('Gain matrix for fast poles:');
```

```
K1 = place(A, B, p_fast)
Ac_fast = (A - B*K1);
sys1 = ss(Ac_fast, B, C, D);
step(sys1);
figure;

p_slow = [-2 -3 -4];
disp('Gain matrix for slow poles:');
K2 = place(A, B, p_slow)
Ac_slow = (A - B*K2);
sys2 = ss(Ac_slow, B, C, D);
step(sys2);

>> >> >>
Poles of the system:
 -250.0000
   15.6011
  -15.6951

Gain matrix for fast poles:
K1 =
   1.0e+04 *
   -1.4382   -0.0344    0.0024

Gain matrix for slow poles:
K2 =
  -16.7723   -2.0542  -48.2188
```

**MATLAB Code 1: Step Response Analysis**






It is clear that the system has become significantly slower. However, the gain values are acceptable and reasonable, and the final value is easily reached by a proportional controller with a low gain relative to the input value. On the other hand, in the case of fast poles, the gain values are very high and unreasonable.

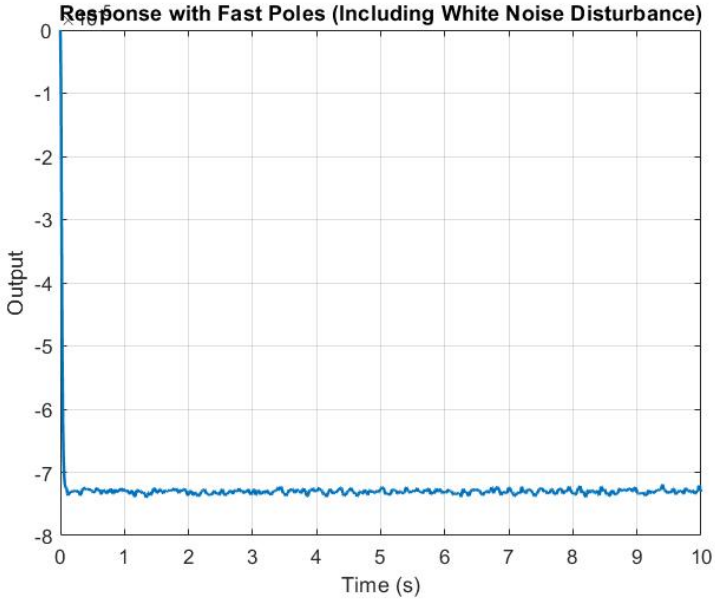
2.

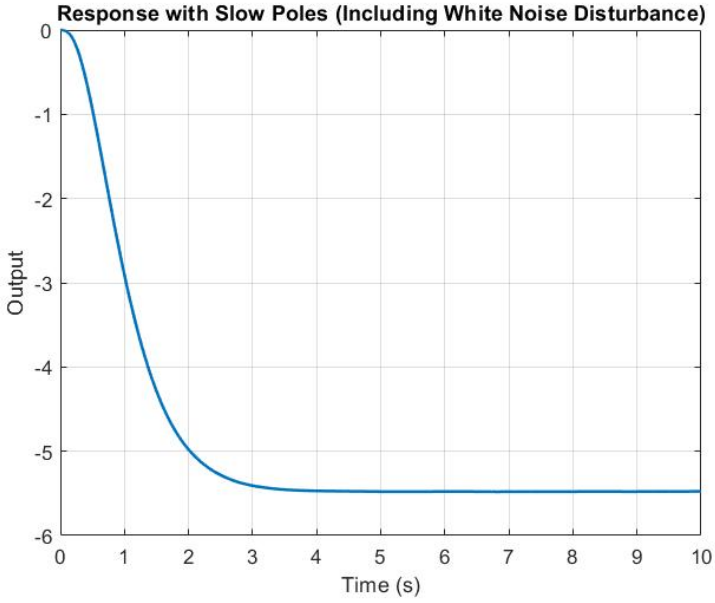
In this section, a cumulative disturbance is incorporated to analyze the system's response under different conditions. To achieve this, a step-by-step approach is implemented in MATLAB as follows:

```
clc;
clear all;
A = [0 1 0; 244.86 -0.094 -26.29; 0 0 -250];
B = [0; 0; 5];
C = [1 0 0];
D = 0;

sys = ss(A, B, C, D);

[num, denum] = ss2tf(A, B, C, D);

disp('Poles of the system:');
poles = roots(denum);
disp(poles);

p_fast = [-120 -100 -150];
K1 = place(A, B, p_fast);
disp('Gain matrix for fast poles:');
disp(K1);

```

```
p_slow = [-2 -3 -4];
K2 = place(A, B, p_slow);
disp('Gain matrix for slow poles:');
disp(K2);

t = 0:0.01:10;
disturbance = 0.05 * randn(size(t));

Ac_fast = A - B*K1;
Bc_fast = [B, [0; 0; 1]];
Cc_fast = C;
Dc_fast = [0 0];
sys_fast = ss(Ac_fast, Bc_fast, Cc_fast, Dc_fast);

u_fast = [ones(size(t)); disturbance];
[y_fast, t_fast, x_fast] = lsim(sys_fast, u_fast, t);

Ac_slow = A - B*K2;
Bc_slow = [B, [0; 0; 1]];
Cc_slow = C;
Dc_slow = [0 0];
sys_slow = ss(Ac_slow, Bc_slow, Cc_slow, Dc_slow);

u_slow = [ones(size(t)); disturbance];
[y_slow, t_slow, x_slow] = lsim(sys_slow, u_slow, t);

figure;
plot(t_fast, y_fast, 'LineWidth', 1.5);
title('Response with Fast Poles (Including White Noise
    Disturbance)');
xlabel('Time (s)');
ylabel('Output');
grid on;

figure;
plot(t_slow, y_slow, 'LineWidth', 1.5);
title('Response with Slow Poles (Including White Noise
    Disturbance)');
xlabel('Time (s)');
ylabel('Output');
grid on;
>>>>>>
Poles of the system:
 -250.0000
   15.6011
  -15.6951

Gain matrix for fast poles:
   1.0e+04 *
   -1.4382   -0.0344    0.0024

Gain matrix for slow poles:
  -16.7723   -2.0542  -48.2188
```

**MATLAB Code 2: Step Response Analysis in the Presence of a Cumulative Disturbance**


Response with Fast Poles (Including White Noise Disturbance)
x10^5
0
-1
-2
-3
-4
-5
-6
-7
-8
Output
0
1
2
3
4
5
6
7
8
9
10
Time (s)


Response with Slow Poles (Including White Noise Disturbance)
0
-1
-2
-3
-4
-5
-6
Output
0
1
2
3
4
5
6
7
8
9
10
Time (s)

**Response with Fast Poles**

- The disturbance has minimal impact on the system's stability, indicating strong disturbance rejection.
- However, the system output does not converge to 0, suggesting poor tracking performance.

**Response with Slow Poles**

- The system response starts at 0 and quickly converges to a steady-state value of approximately $-5$.
- The presence of white noise does not significantly affect the overall system behavior.
- The system demonstrates better tracking capabilities compared to the fast poles.

3.

Firstly, to verify the controllability of the tracker system associated with state-feedback and integral control, a MATLAB code is executed as shown below:

```
clc;
clear all;
format long g

A = [0 1 0; 244.86 -0.094 -26.29; 0 0 -250];
B = [0; 0; 5];
C = [1 0 0];
D = 0;
sys = ss(A, B, C, D);

p_fast = [-120 -100 -150];
K1 = place(A, B, p_fast);
p_slow = [-2 -3 -4];
K2 = place(A, B, p_slow);

% controllabilty of the tracker system associated with state
    feedback
Co_state = [B , A*B ,A^2*B];
disp(Co_state)
rank(Co_state)
% controllabilty of the tracker system associated with
    integral-control
A_bar = [0 1 0 0; 244.86 -0.094 -26.29 0; 0 0 -250 0;-1 0 0 0];
B_bar = [0; 0; 5; 0];
Co_integral = [B_bar , A_bar*B_bar
    ,(A_bar)^2*B_bar,(A_bar)^3*B_bar];
disp(Co_integral)
rank(Co_integral)
K = place(A_bar, B_bar, [-10, -20, -30,-40])
```

**MATLAB Code 3: Controllability of The Designed Systems**

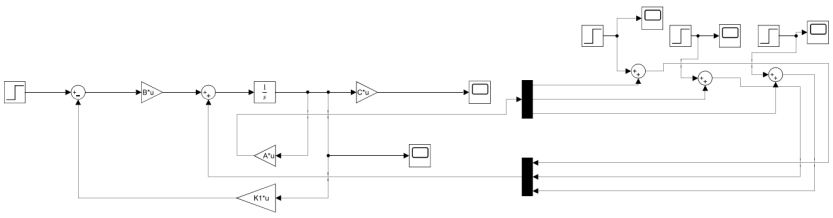
**Results of the Implemented Code :**

$$\text{Co} - \text{state} = \begin{bmatrix} 0 & 0 & -131.45 \\ 0 & -131.45 & 32874.85 \\ 5 & -1250 & 312500 \end{bmatrix} \rightarrow Rank = 3$$

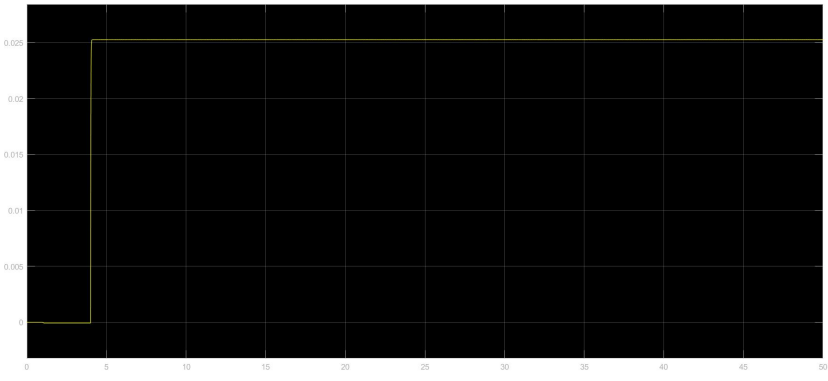
$$\text{Co} - \text{integral} = \begin{bmatrix} 0 & 0 & -131.45 & 32874 \\ 0 & -131.45 & 32874 & -8250902 \\ 5 & -1250 & 312500 & -78125000 \\ 0 & 0 & 0 & 131.45 \end{bmatrix} \rightarrow Rank = 4$$

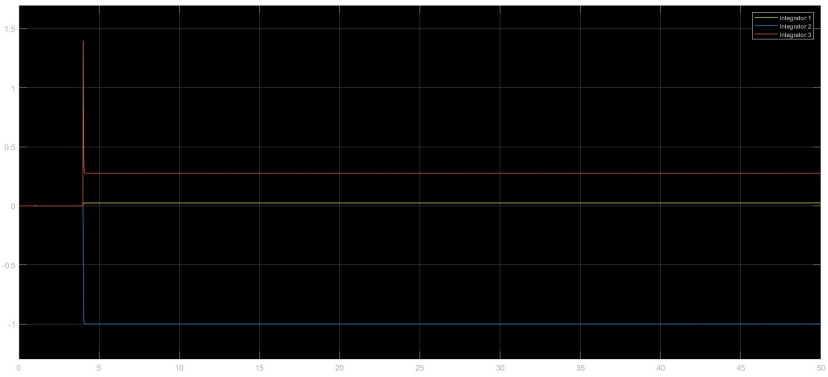
$$K = \begin{bmatrix} -566.47 & -28.42 & -30.02 & 1825.79 \end{bmatrix}$$

As observed, both designed trackers are controllable. Therefore, the tracker associated with state-feedback is modeled in Simulink, as shown below:



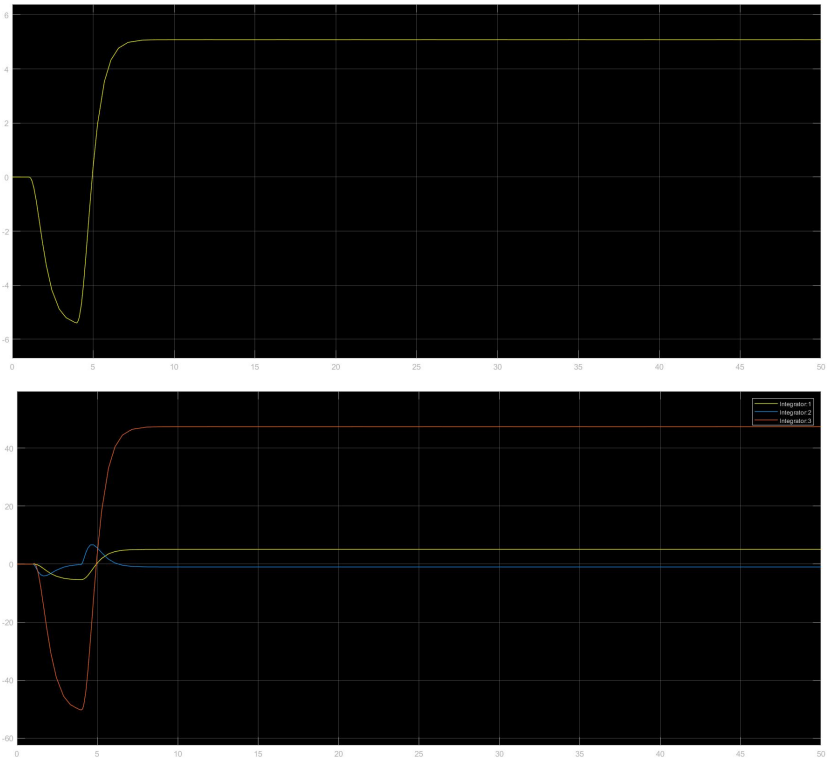

**Performance Results of the Tracker Associated with State-Feedback for the Fast Pole Set:**





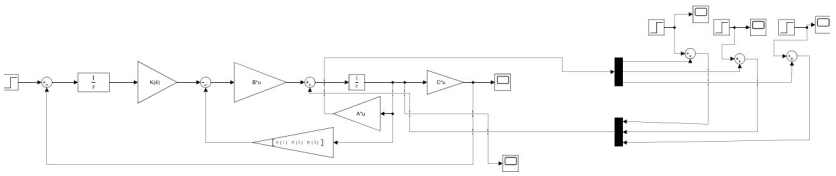

As observed, the designed tracker, incorporating the fast pole set, failed to track the step input. However, it demonstrated solid resilience to the posed disturbance.

**Performance Results of the Tracker Associated with State-Feedback for the Slow Pole Set:**



As observed, the designed tracker, incorporating the slow pole set, failed to track the step input. However, it demonstrated solid resilience to the posed disturbance.

The tracker associated with integral-control is modeled in Simulink, as shown below:



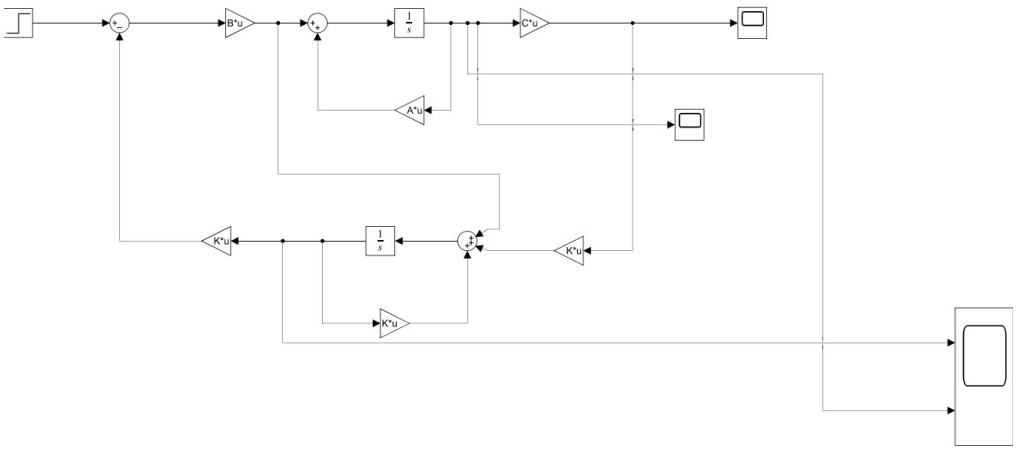
**Performance Results of the Tracker Associated with integral-control:**


On the flip side, the designed tracker associated with integral-control not only successfully tracked the step input but also demonstrated complete resilience to the posed disturbance.

**Comparison Between the Designed PID Controller & the State-Feedback and Integral-Control Trackers:**

- The PID controller and integral-control tracker successfully tracked the step input, whereas the state-feedback tracker failed to do so.
- Both the PID controller's and the integral-control tracker's step response plots converge to 1 at approximately $t = 1$.

4.
A controller associated with a full-order Lionberger estimator is modeled and implemented in Simulink as follows:

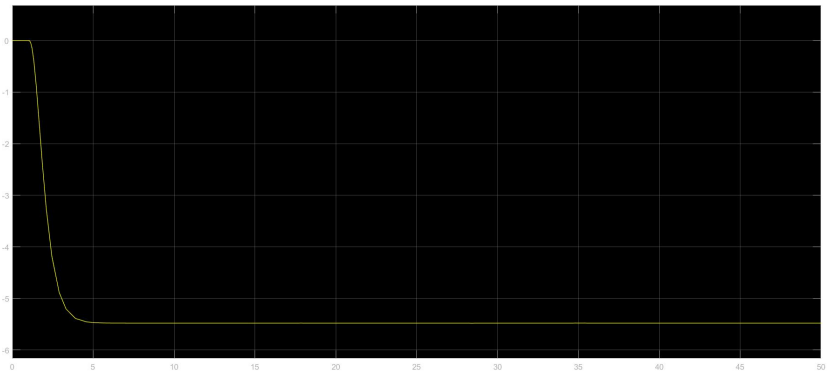


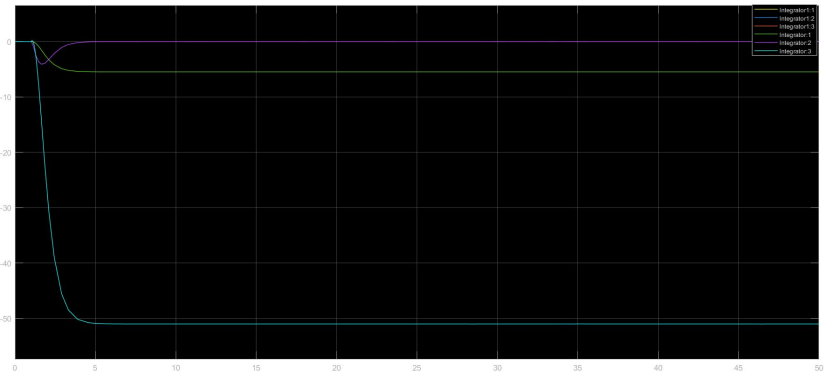

```
clc;
clear all;

A = [0 1 0; 244.86 -0.094 -26.29; 0 0 -250];
B = [0; 0; 5];
C = [1 0 0];
D = 0;

sys = ss(A, B, C, D);


A_n = transpose([0 1 0; 244.86 -0.094 -26.29; 0 0 -250]);
B_n = transpose([1 0 0]);

%a(s) = s^3 + 250.094 s^2 -221.36 s - 61215
% alpha(s) = (s+3)(s+2)(s+4)   slow pole
% alpha(s)=(s+100)(s+120)(s+150)
sub1=[-241.094,247.36,61239];
sub2=[370-250.094,45000+221.36,1800000+61215];
```

```
20
21 sigh = [1 250.094 -221.36;0 1 250.094; 0 0 1];
22
23 sigh_inverse = inv(sigh)
24
25 Controlable=ctrb(A_n,B_n)
26
27 rank(Controlable)
28
29 Controlable_inverse = inv(Controlable)
30
31 K1 = place(A, B, [-4 -2 -3])
32 K2 = place(A, B, [-100 -120 -150])
33
34 K_slow = sub1 * sigh_inverse * Controlable_inverse
35 K_fast = sub2 * sigh_inverse * Controlable_inverse
36
37
38 L_slow = transpose(K_slow)
39 Ahat_slow = A - L_slow * C
40
41 L_fast = transpose(K_fast)
42 Ahat_fast = A - L_fast * C
```

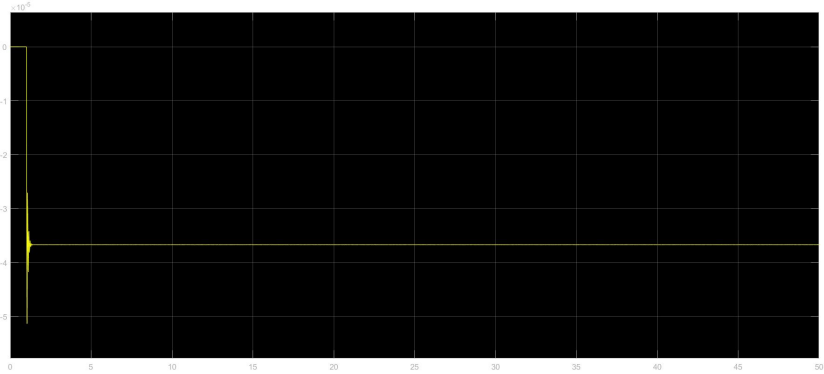
**MATLAB Code 4: Controller with a Lionberger estimator**

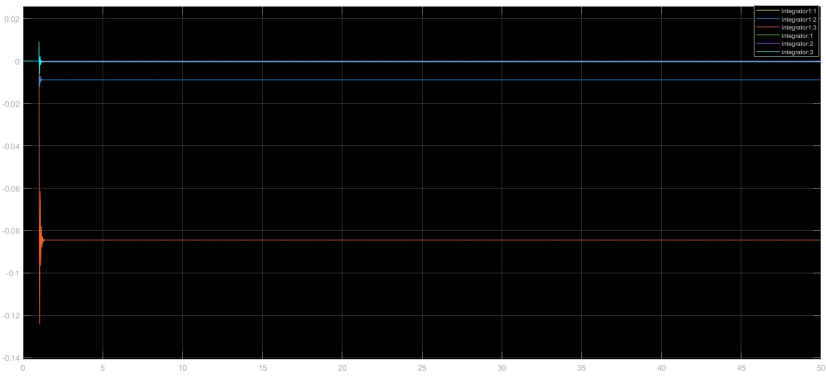
**Performance Results for the Slow Pole Set:**





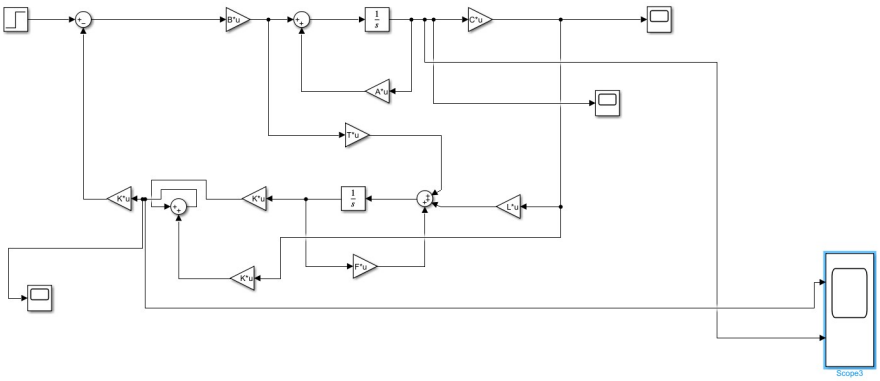

**Performance Results for the Fast Pole Set:**



5.

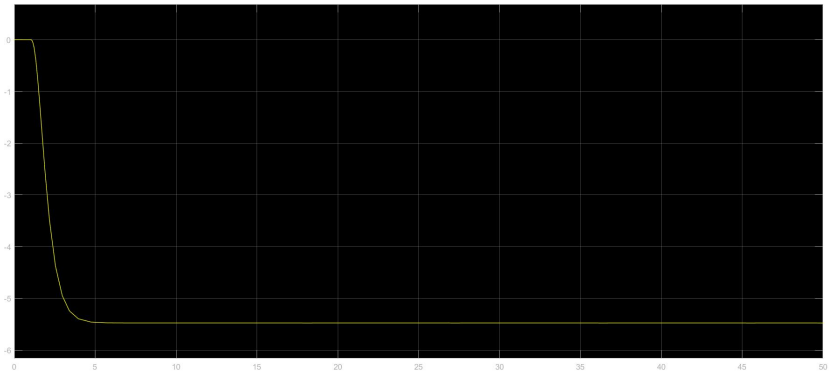
A controller associated with a reduced-order state observer is modeled and implemented in Simulink as follows:

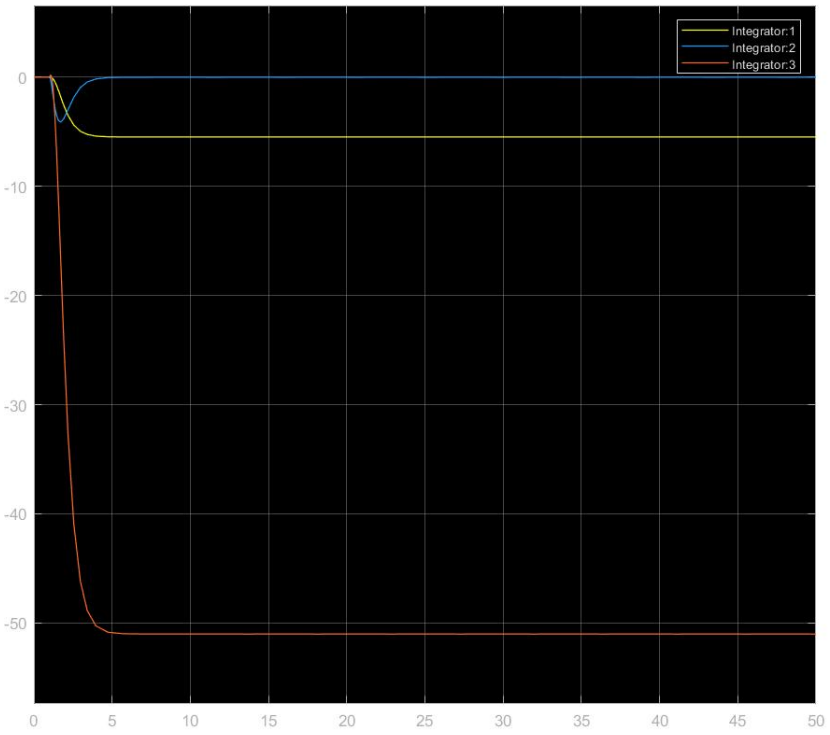


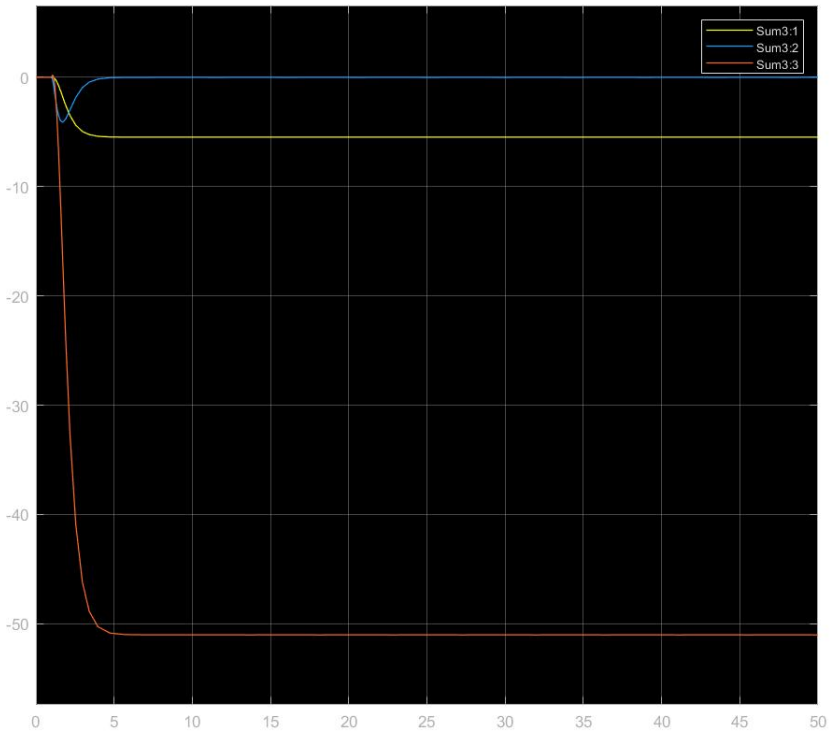

```
clc;
clear all ;

A = [0 1 0; 244.86 -0.094 -26.29; 0 0 -250];
B = [0; 0; 5];
C = [1 0 0];
D = 0;

F = [-2 0; 0 -3];
L = [1;1];
T = lyap(-F,A,-L*C)
P = inv([C;T])
 F1 = P(:, 1)
 F2 = P(:, 2:3)
R = T * B

K1 = place(A, B, [-4 -2 -3])
K2 = place(A, B, [-100 -120 -150])
```

**MATLAB Code 5: Controller with a Reduced-Order State Observer**

**Performance Results:**






As observed, the designed observer successfully estimated the state variables. However, it failed to track the step input.