Control System Design

Lab - Assignment

Names:
Liam Glennon 20344313

Date:
23/11/2023

1 State Space Model

```
% Define the transfer function

num = 1.3 * [1 12];

den = conv(conv([1 3], [1 5]), [1 2]); % (s+3)(s+5)(s+2)

G = tf(num, den);
```

```
% Convert to state-space (observer canonical form)

[A_obs, B_obs, C_obs, D_obs] = tf2ss(num, den);
```

```
% Convert to controller canonical form

% For a 3rd-order system, the A matrix in controller canonical form
is:

A_ctrl = [0 1 0; 0 0 1; -den(4) -den(3) -den(2)];

B_ctrl = [0; 0; 1];

C_ctrl = [num(2) - den(2)*num(1), num(1), 0]; % Adjust based on the
order of the numerator

D_ctrl = 0; % Assuming no direct transmission path

% Display the matrices

A_ctrl

B_ctrl

C_ctrl

D_ctrl

sys = ss(A_ctrl, B_ctrl, C_ctrl, D_ctrl)

play the state-space model

Sys
```

Code 1.1

```
A_ctrl = 3×3
   0    1    0
   0    0    1
 -30  -31  -10

B_ctrl = 3×1
   0
   0
   1

C_ctrl = 1×3
  2.6000   1.3000       0

D_ctrl = 0
sys =

 A =
     x1  x2  x3
 x1   0   1   0
 x2   0   0   1
 x3 -30 -31 -10

 B =
    u1
 x1  0
 x2  0
 x3  1

 C =
     x1  x2  x3
 y1 2.6 1.3   0

 D =
    u1
 y1  0

Continuous-time state-space model.
```

Fig 1.1 (Matlab output to code 1.1)

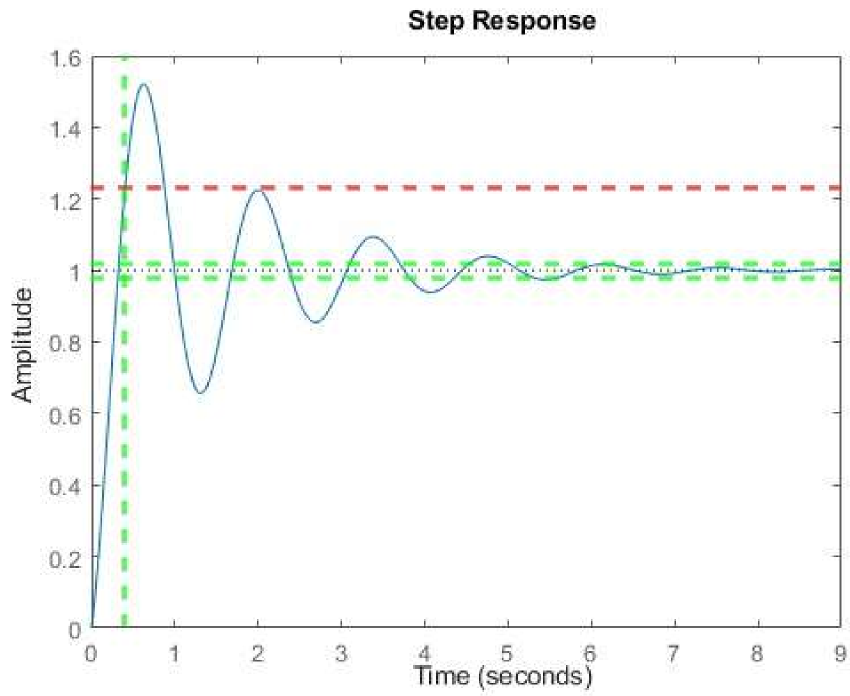
2 Controller Design

2.1 Open-loop Ziegler-Nichols tuning method

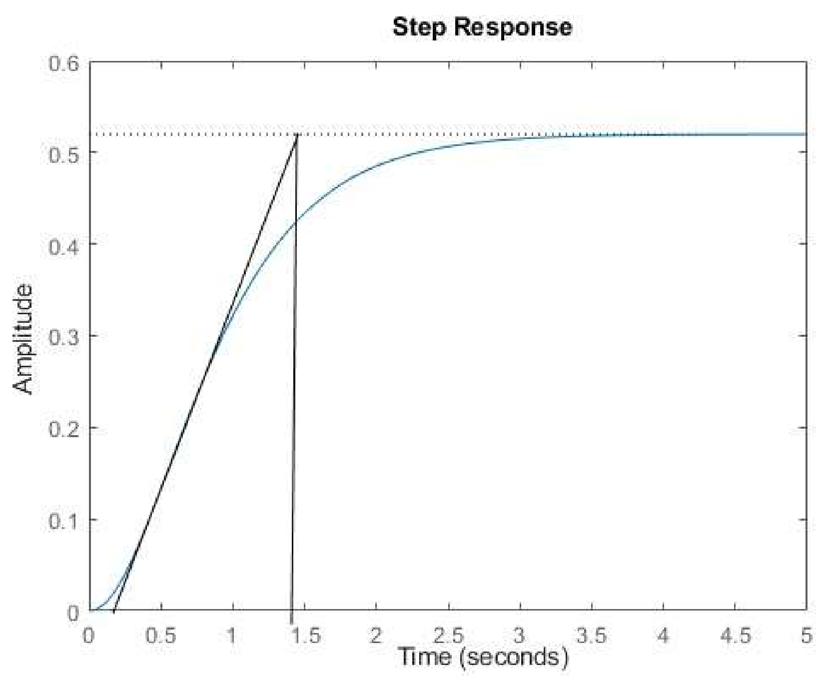
```
s = tf('s');
sys = (1.3*(s+12))/((s+3)*(s+5)*(s+2))
step(sys);
k = 0.53;  %
l = 0.25;
t = 1.5 - l;
a = k*l/t;
Ki = 2*l;
Td = l/2;
Kp = 1.2/a;
Ki = Kp/Ti;
Kd = Kp*Td;
cont= pid(Kp,Ki,Kd)
step(feedback(cont*sys,1))
yline(1.23, 'r--', 'LineWidth', 2); % Red dashed line
yline(0.98, 'g--', 'LineWidth', 2); % Red dashed line
yline(1.02, 'g--', 'LineWidth', 2); % Red dashed line
xline(0.4, 'g--', 'LineWidth', 2); % Red dashed line
```

Code 2.1

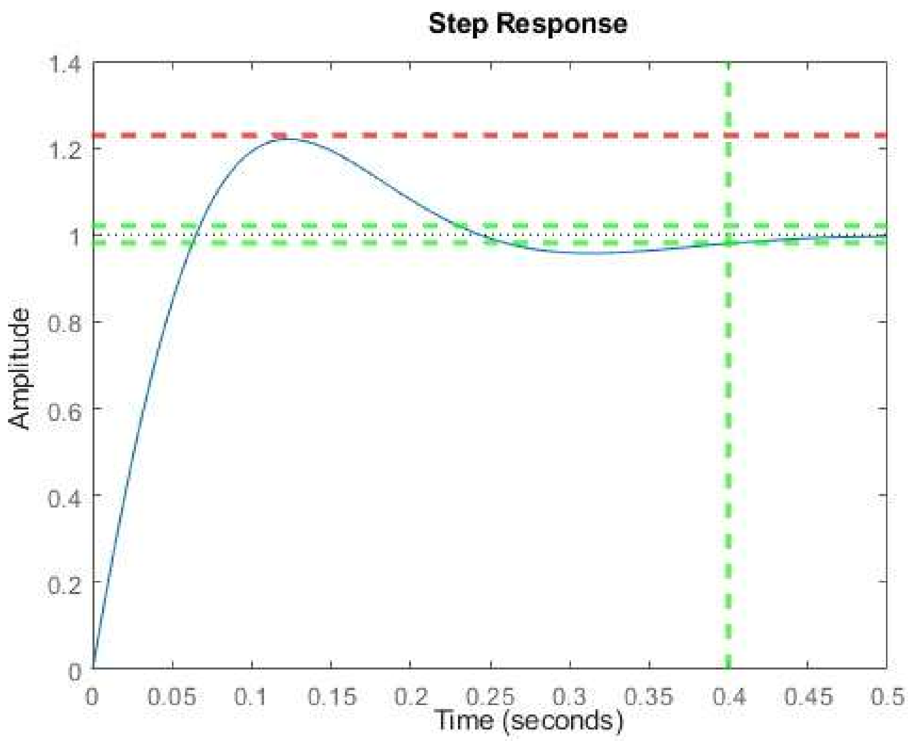
Values k,l are taken from plot 2.1.2



Plot 2.1.1 Open-loop Ziegler-Nichols Step Response Initial Values



Plot 2.1.2 Step Response of System



Plot 2.1.3 Open-loop Ziegler-Nichols Step Response

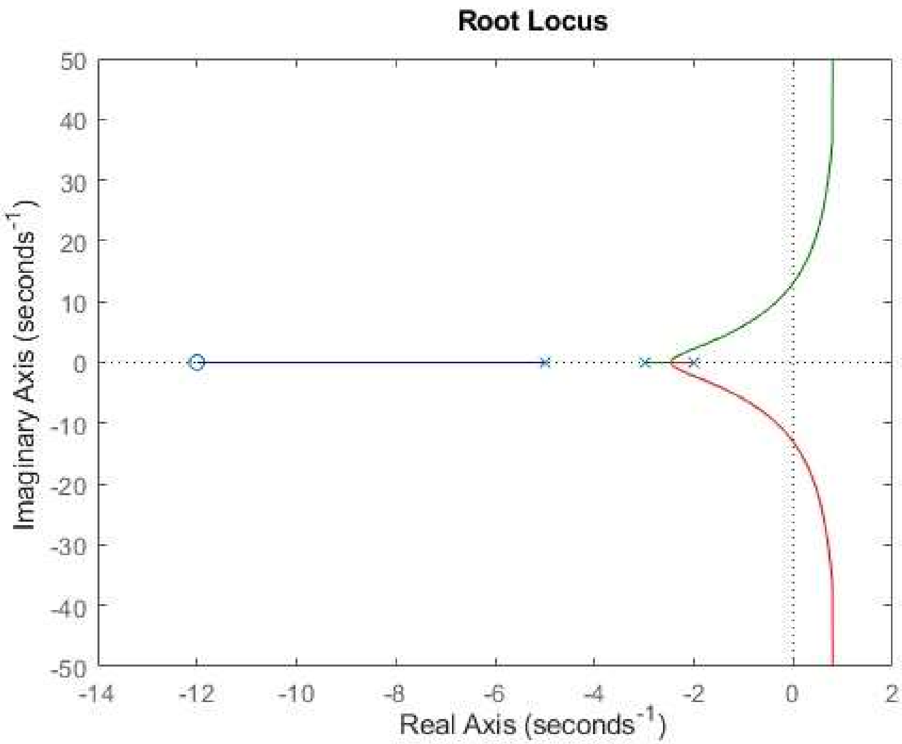
2.2 Closed-loop Ziegler-Nichols tuning method

```
% Define the system
s = tf('s');
sys = (1.3*(s+12))/((s+3)*(s+5)*(s+2));
% Plot the root locus
figure;
rlocus(sys);
```

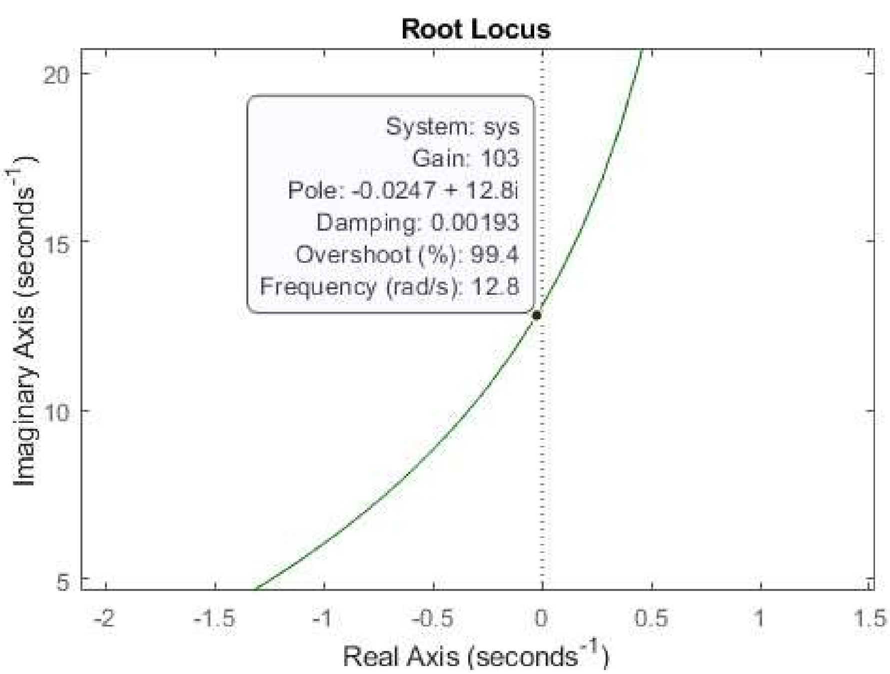
Code 2.2.1 Plot the root locus

```
% Define the system
s = tf('s');
sys = (1.3*(s+12))/((s+3)*(s+5)*(s+2));
% Ku and Pu from the Root Locus
% For demonstration, let's assume Ku and Pu are given
Ku = 103; % Replace with your calculated value
Pu = 0.49; % Replace with your calculated value
% Calculate PID parameters using Ziegler-Nichols closed-loop method
Kp = 0.6 * Ku; Ti = 0.5 * Pu; Td = 0.125 * Pu;
Ki = Kp / Ti; Kd = Kp * Td;
% Create the PID controller cont = pid((Kp*4),(Ki),(Kd*4.5));
cont = pid((Kp),(Ki),(Kd));
% Closed-loop system response
step(feedback(cont*sys, 1));
yline(1.23, 'r--', 'LineWidth', 2); % Red dashed line
yline(0.98, 'g--', 'LineWidth', 2); % Green dashed line
yline(1.02, 'g--', 'LineWidth', 2); % Green dashed line
xline(0.4, 'g--', 'LineWidth', 2); % Green dashed line
```

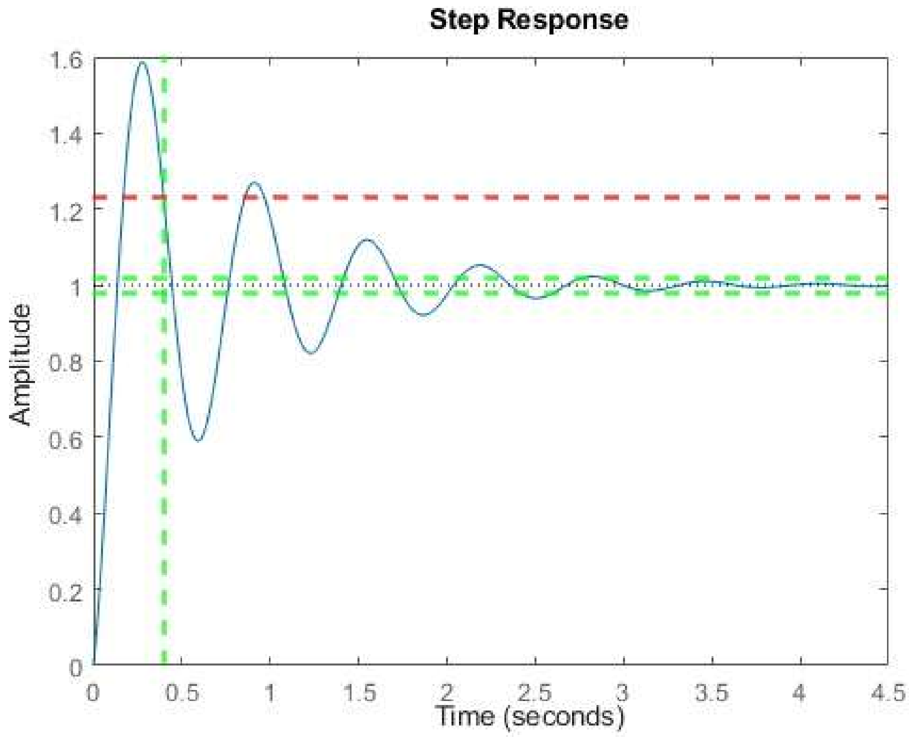
Code 2.2.2 Plot the root locus



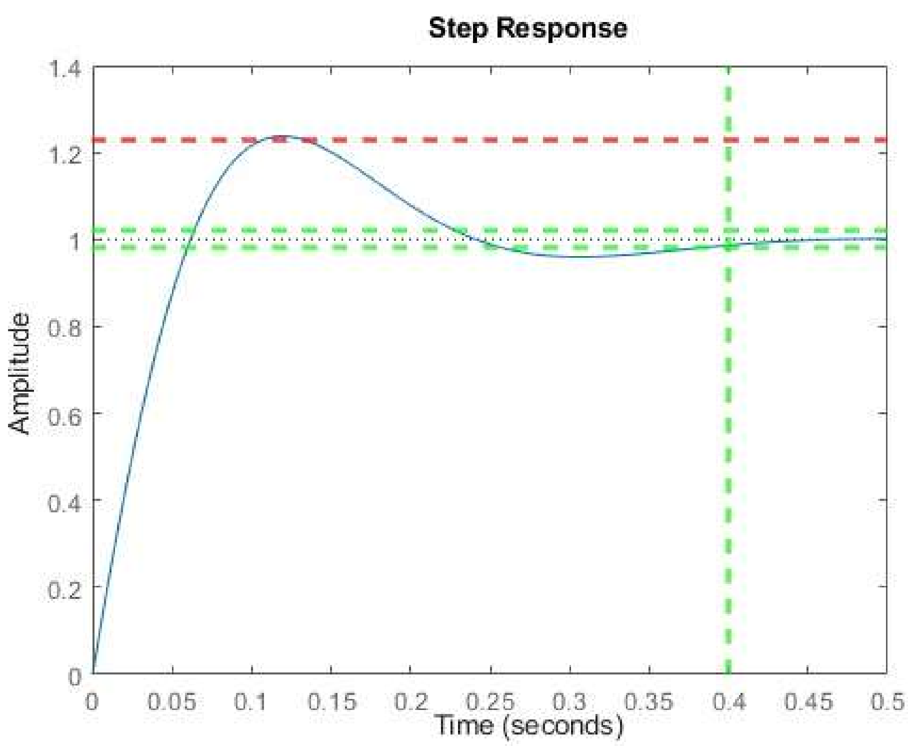
Plot 2.2.1 Root Locus Plot



Plot 2.2.2 Root Locus Plot (zoomed on key figure)



Plot 2.2.3 Closed-loop Ziegler-Nichols Step Response Root Locus Values



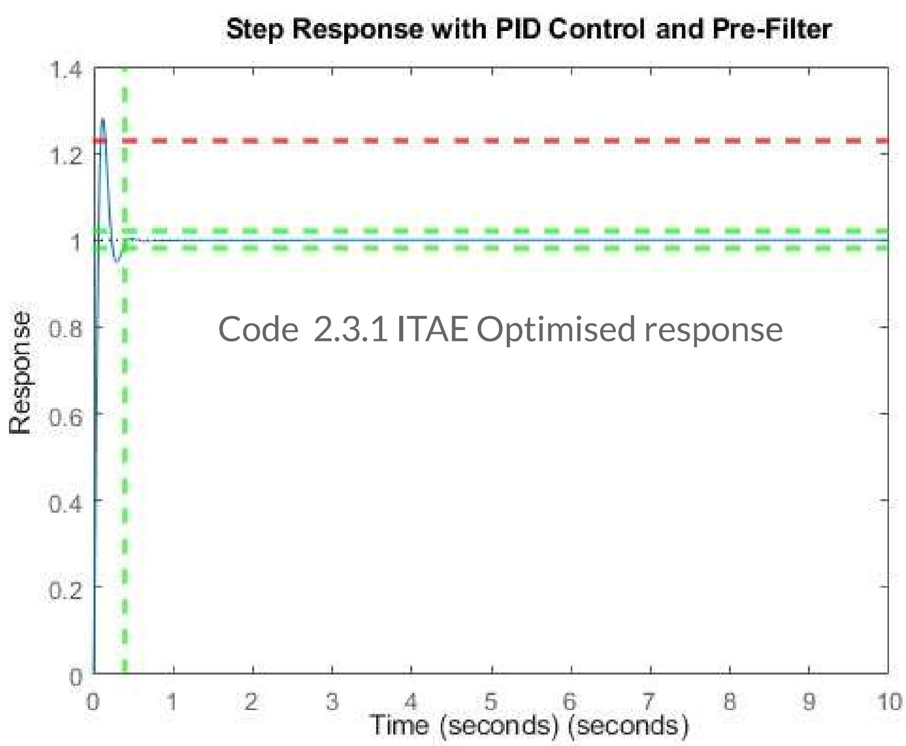
Plot 2.2.4 Closed-loop Ziegler-Nichols Step Response Root Tuned Values

2.3 ITAE performance index (With pre-filter)

```
% Define the system
s = tf('s');
sys = (1.3*(s+12))/((s+3)*(s+5)*(s+2));
% Find Ku and Pu
Ku = 103; % Replace with your calculated value
Pu = 0.49; % Replace with your calculated value
% Calculate PID parameters using Ziegler-Nichols closed-loop method
Kp = 0.6 * Ku;
Ti = 0.5 * Pu;
Td = 0.125 * Pu;
Ki = Kp / Ti;
Kd = Kp * Td;
% Create the PID controller
cont = pid((Kp*4),(Ki),(Kd*4.5));
% Define the pre-filter
tau_f = 0.01 * Pu; % Example time constant, adjust based on your system
Kf = 1; % Filter gain
filter = Kf / (tau_f * s + 1);
% Closed-loop system with pre-filter
Tfinal = 10; % Define a suitable final time for simulation
[t, y] = step(feedback(filter * cont * sys, 1), Tfinal);
% Calculate the error signal (assuming a unit step response)
e = 1 - y; % Error = Desired - Actual
% Calculate ITAE
itae = trapz(t, t .* abs(e));
```

```
% Display ITAE
disp(['ITAE value: ', num2str(itae)]);
% Plot the step response
figure;
step(feedback(filter * cont * sys, 1), Tfinal);
title('Step Response with PID Control and Pre-Filter');
xlabel('Time (seconds)');
ylabel('Response');
% Optional: Add lines to visualize specific values
yline(1.23, 'r--', 'LineWidth', 2); % Red dashed line
yline(0.98, 'g--', 'LineWidth', 2); % Green dashed line
yline(1.02, 'g--', 'LineWidth', 2); % Green dashed line
xline(0.4, 'g--', 'LineWidth', 2); % Green dashed line
```

Code 2.3.1 ITAE Optimised response



Code 2.3.1 ITAE Optimised response

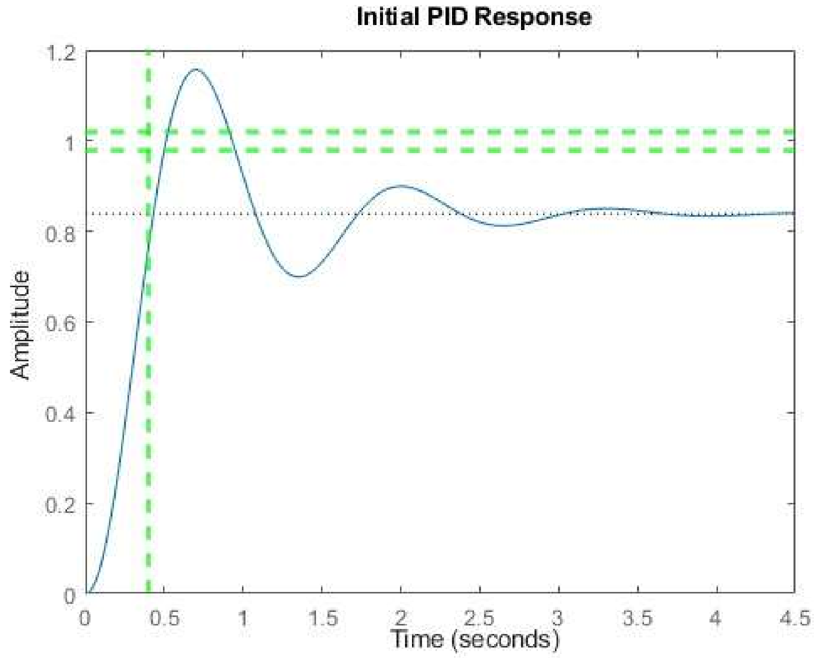
Plot 2.3.1 ITAE Optimised response

2.4.1 PID controller based on manual tuning

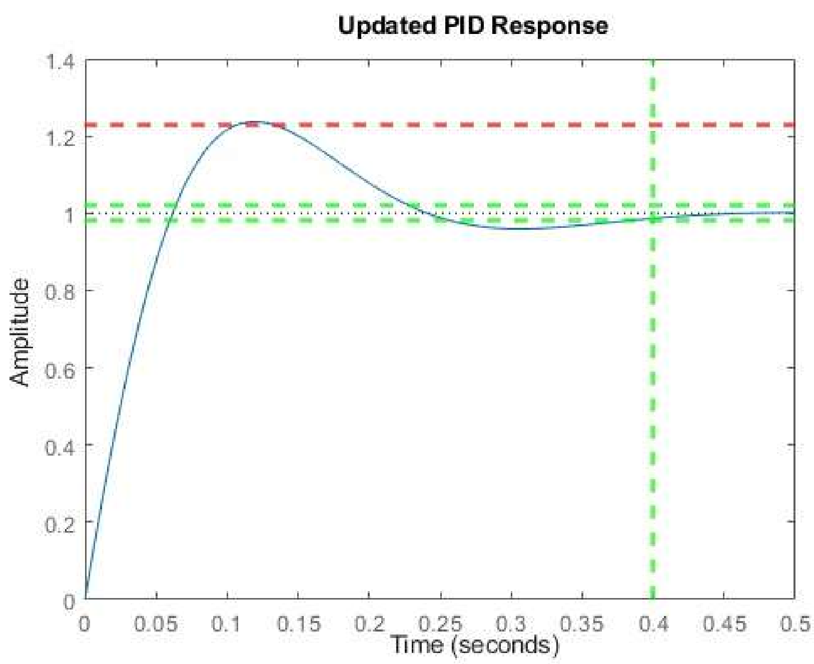
```
% Define the system
s = tf('s');
sys = (1.3*(s+12))/((s+3)*(s+5)*(s+2));

% Initialize PID parameters for manual tuning
Kp = 10; % Start with a guess and adjust based on system response
Ki = 0;  % Initially set to zero
Kd = 0;  % Initially set to zero

% Create the PID controller
cont = pid(Kp, Ki, Kd);

% Closed-loop system response
figure;
step(feedback(cont*sys, 1));
title('Initial PID Response');

% Optional: Add lines to visualize specific values
yline(1.23, 'r--', 'LineWidth', 2); % Red dashed line
yline(0.98, 'g--', 'LineWidth', 2); % Green dashed line
yline(1.02, 'g--', 'LineWidth', 2); % Green dashed line
xline(0.4, 'g--', 'LineWidth', 2); % Green dashed line
```

```
% Tune the parameters manually and iteratively
% Adjust Kp, Ki, Kd based on the observed response
% You can use the following lines to update the controller and plot new responses
Kp = [247.5];
Ki = [252.5];
Kd = [17.0334];
cont = pid(Kp, Ki, Kd);
step(feedback(cont*sys, 1));
title('Updated PID Response');
% Optional: Add lines to visualize specific values
yline(1.23, 'r--', 'LineWidth', 2); % Red dashed line
yline(0.98, 'g--', 'LineWidth', 2); % Green dashed line
yline(1.02, 'g--', 'LineWidth', 2); % Green dashed line
xline(0.4, 'g--', 'LineWidth', 2); % Green dashed line
```

Code 2.4.1 Manual PID response



Plot 2.4.1 Manual PID response



Plot 2.4.1 Manual PID response

State-feedback controller & State-feedback optimal controller

Sections e & f were combined as f (LQR) is just an add on to the end of e's(State-feedback controller)

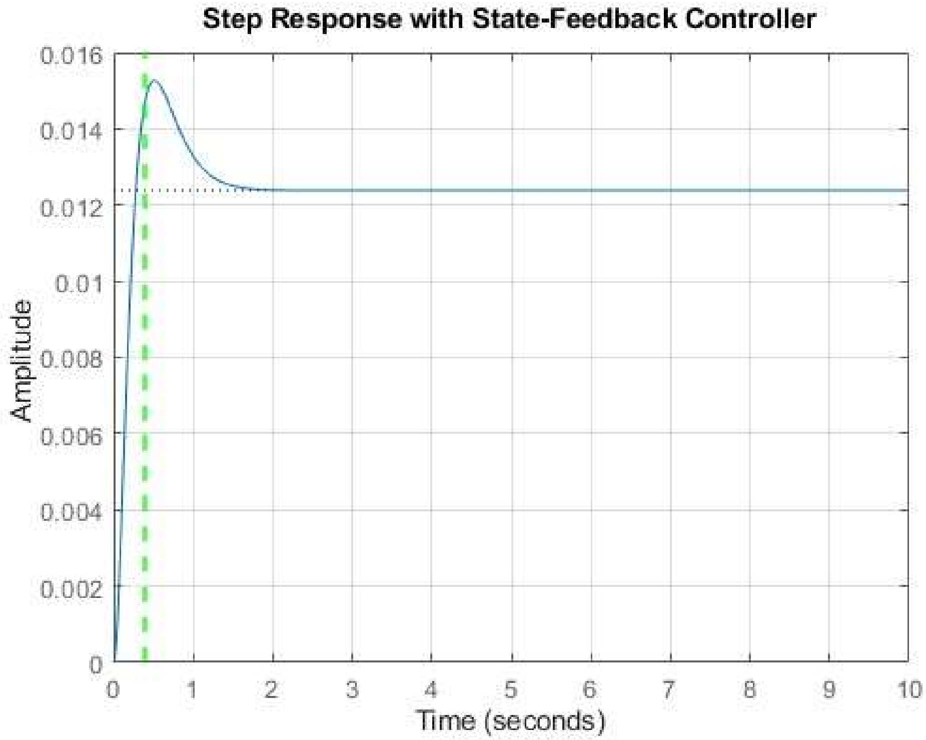
```
% Define the transfer function
num = 1.3 * [1 12];
den = conv(conv([1 3], [1 5]), [1 2]); % (s+3)(s+5)(s+2)
G = tf(num, den);

% Convert to state-space (observer canonical form)
[A_obs, B_obs, C_obs, D_obs] = tf2ss(num, den);

% Convert to controller canonical form
A_ctrl = [0 1 0; 0 0 1; -den(4) -den(3) -den(2)];
B_ctrl = [0; 0; 1];
C_ctrl = [num(2) - den(2)*num(1), num(1), 0]; % Adjust based on the
order of the numerator
D_ctrl = 0; % Assuming no direct transmission path

% Define the state-space model
sys = ss(A_ctrl, B_ctrl, C_ctrl, D_ctrl);

% Desired closed-loop poles for the controller
P_controller = [-5 -6 -7]; % Example poles

% Calculate the state-feedback gain using pole placement
K = place(A_ctrl, B_ctrl, P_controller);
```

```
% Alternatively, you can use the acker function
% K = acker(A_ctrl, B_ctrl, P_controller);

% Desired poles for the observer (twice as fast as controller poles)
P_observer = 2 * P_controller;

% Observer gain calculation
L = place(A_obs', C_obs', P_observer)';

% Define the weighting matrices Q and R for LQR
Q = eye(size(A_ctrl)); % Example Q matrix
R = 2.468; % Example R value

% Calculate the optimal feedback gain using LQR
[K_optimal, P, Poles] = lqr(A_ctrl, B_ctrl, Q, R);

% Display the results
disp('State-feedback gain K:');
disp(K);
disp('Observer gain L:');
disp(L);
disp('Optimal feedback gain K (LQR):');
disp(K_optimal);


% Closed-loop system with state-feedback controller
A_cl = A_ctrl - B_ctrl * K;
sys_cl = ss(A_cl, B_ctrl, C_ctrl, D_ctrl);

% Closed-loop system with LQR controller
A_cl_lqr = A_ctrl - B_ctrl * K_optimal;
```

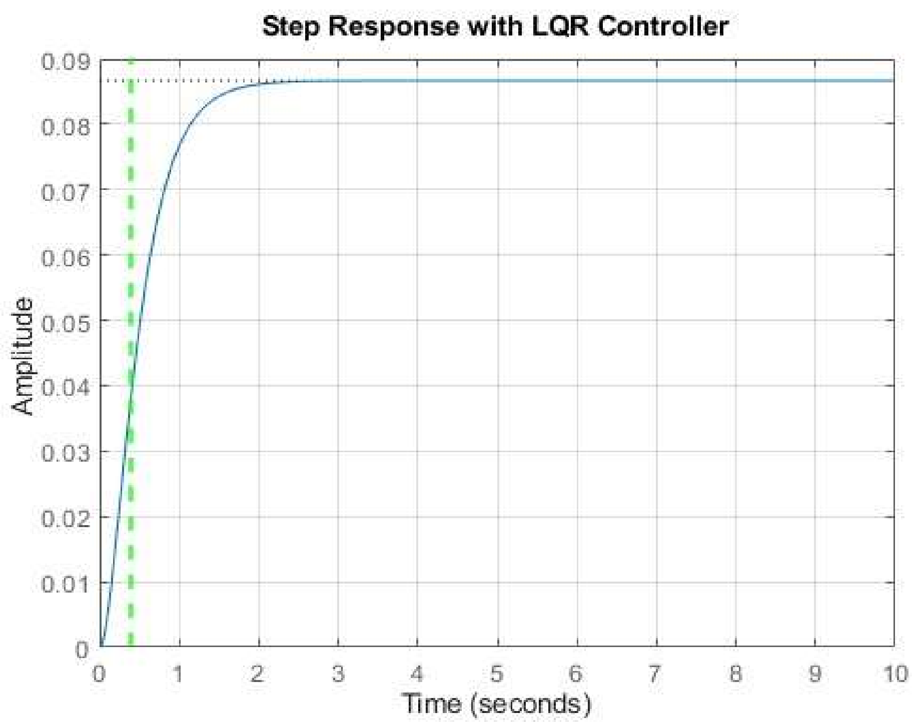
```
sys_cl_lqr = ss(A_cl_lqr, B_ctrl, C_ctrl, D_ctrl);

% Time vector for simulation
t = 0:0.01:10;

% Step response of the original system
figure;
step(sys, t);
title('Step Response of Original System');
yline(1.23, 'r--', 'LineWidth', 2); % Red dashed line
yline(0.98, 'g--', 'LineWidth', 2); % Green dashed line
yline(1.02, 'g--', 'LineWidth', 2); % Green dashed line
xline(0.4, 'g--', 'LineWidth', 2); % Green dashed line
grid on;

% Step response with state-feedback controller
figure;
step(sys_cl, t);
title('Step Response with State-Feedback Controller');
yline(1.23, 'r--', 'LineWidth', 2); % Red dashed line
yline(0.98, 'g--', 'LineWidth', 2); % Green dashed line
yline(1.02, 'g--', 'LineWidth', 2); % Green dashed line
xline(0.4, 'g--', 'LineWidth', 2); % Green dashed line
grid on;

% Step response with LQR controller
figure;
step(sys_cl_lqr, t);
title('Step Response with LQR Controller');
yline(1.23, 'r--', 'LineWidth', 2); % Red dashed line
```

```
yline(0.98, 'g--', 'LineWidth', 2); % Green dashed line
yline(1.02, 'g--', 'LineWidth', 2); % Green dashed line
xline(0.4, 'g--', 'LineWidth', 2); % Green dashed line
grid on;

% Pole-Zero Map of the original system
figure;
pzmap(sys);
title('Pole-Zero Map of Original System');
grid on;

% Pole-Zero Map with state-feedback controller
figure;
pzmap(sys_cl);
title('Pole-Zero Map with State-Feedback Controller');
grid on;

% Pole-Zero Map with LQR controller
figure;
pzmap(sys_cl_lqr);
title('Pole-Zero Map with LQR Controller');
grid on;
```

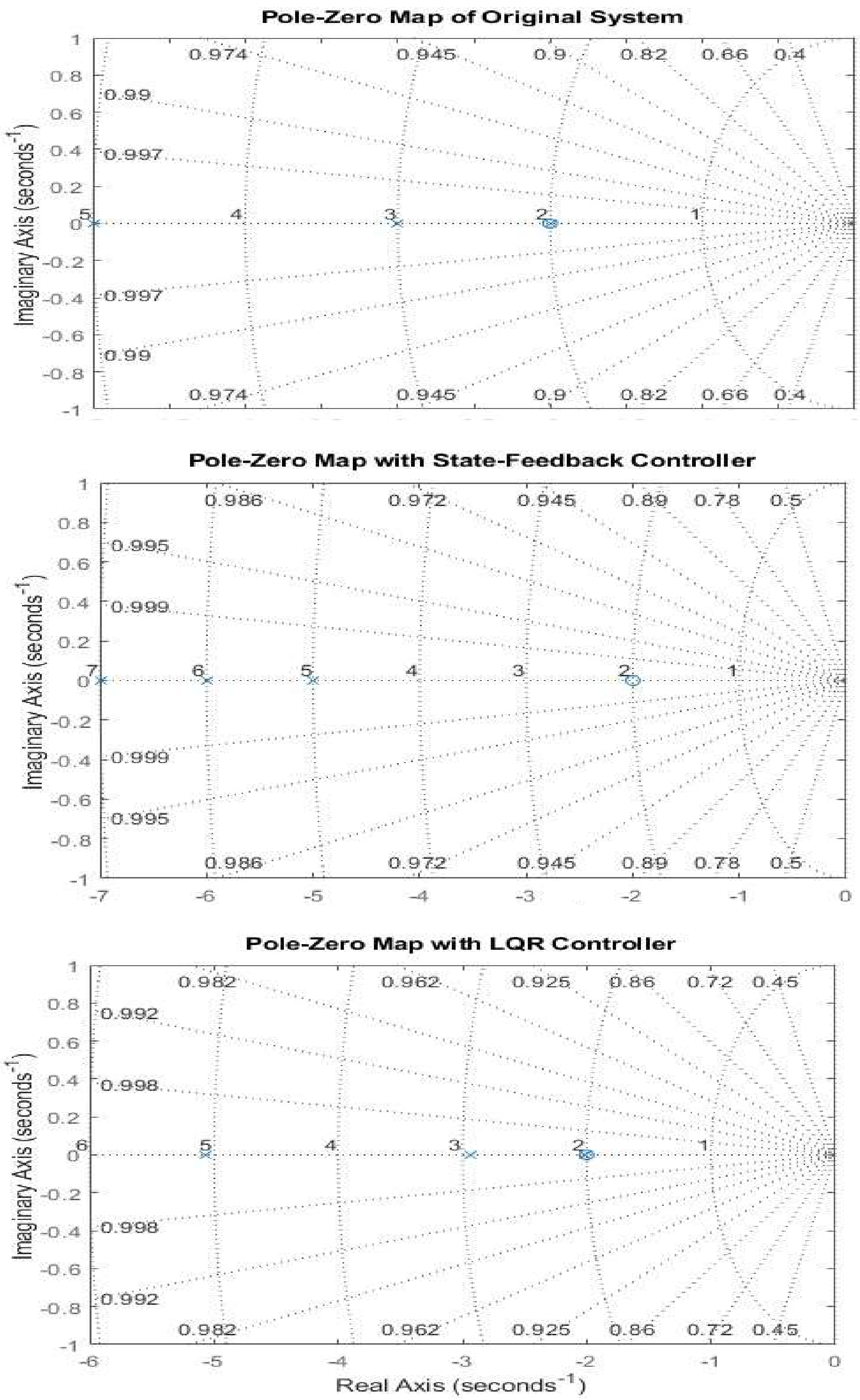
Code 2.5.1 State-feedback controller & LQR



Plot 2.5.1 State-feedback controller



Plot 2.5.2 State-feedback optimal controller



Plot 2.5.2 Pole-Zero Map of Original, State control and LQR

3 Simulink simulation

Controller	Settling time (s)	PO(%)	Steady-state error	Max value of control signal
PID (open-loop ZN) Initial*	5	55	0	10
PID (open-loop ZN)	0.4	23	0	-18
PID (closed-loop ZN) Initial**	2.5	60	0	60
PID (closed-loop ZN)	0.4	23	0	-19
PID (ITAE)	0.4	23	0	15
PID (Manual)	0.4	23	0	-17.5
State-feedback	1.2	23	0,99	9.12
Optimal (LQR)	1	0	0.91	12.7

PID (open-loop ZN) Initial* Uses unoptimized values from fig.
PID (closed-loop ZN) Initial** Uses unoptimized values from fig.

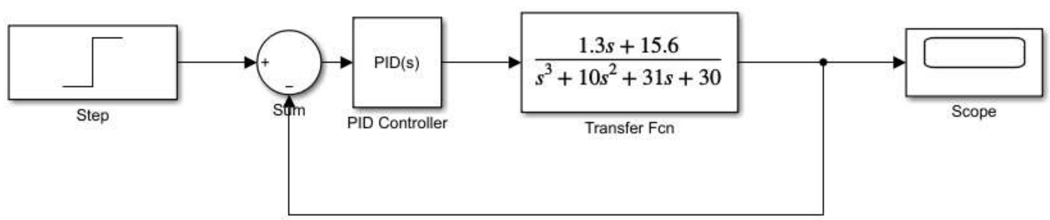
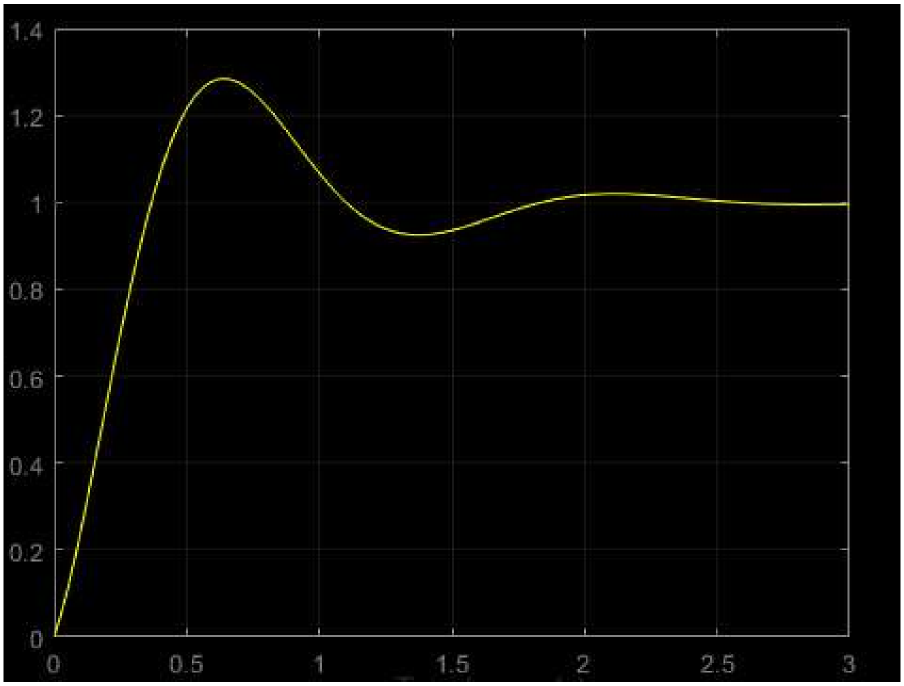
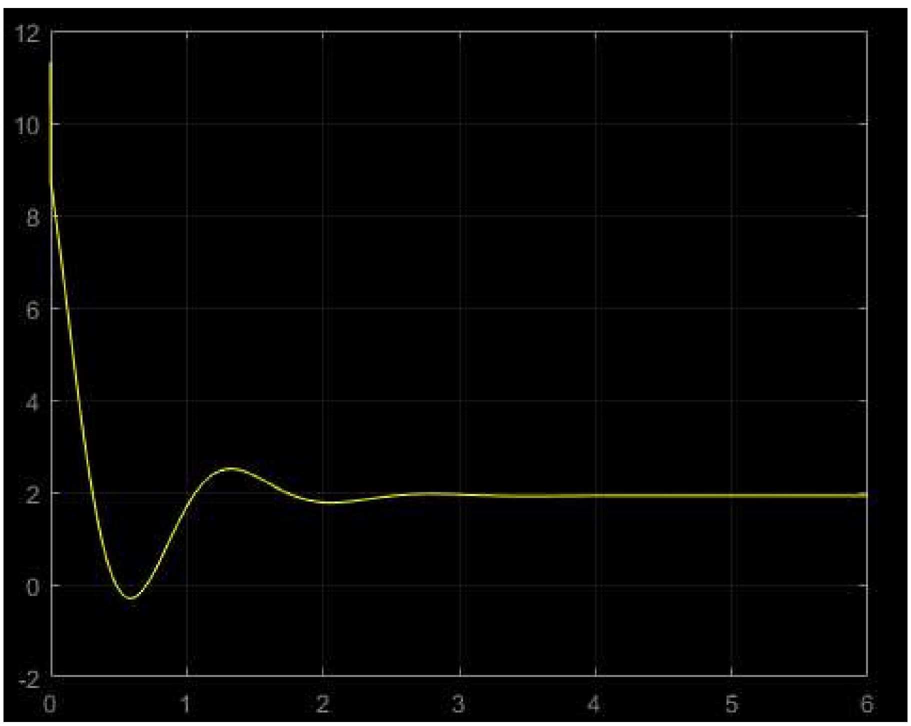


Fig 3.1.1 SImulink Model

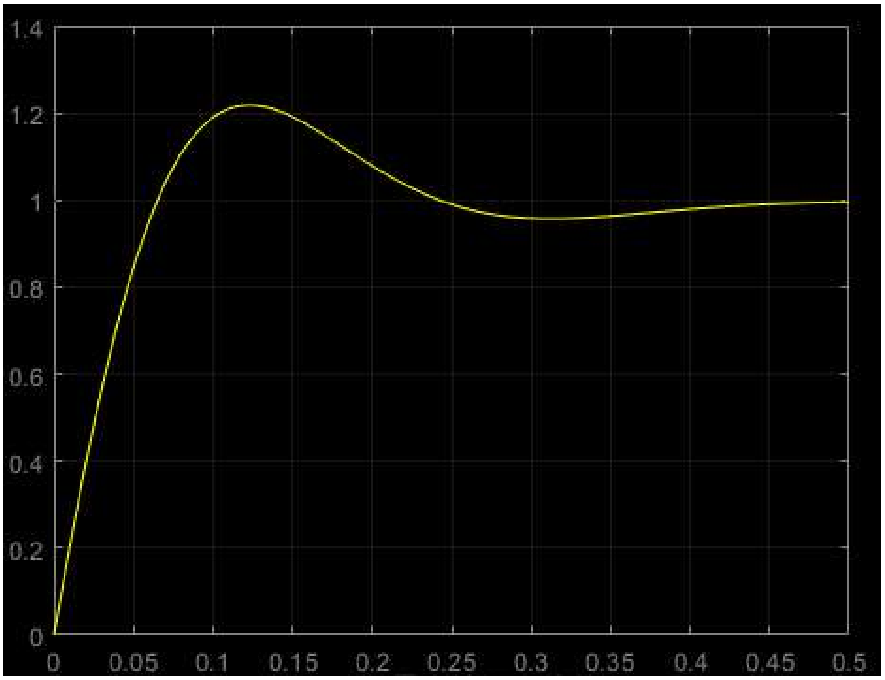
OpenLoop Ziegler signal



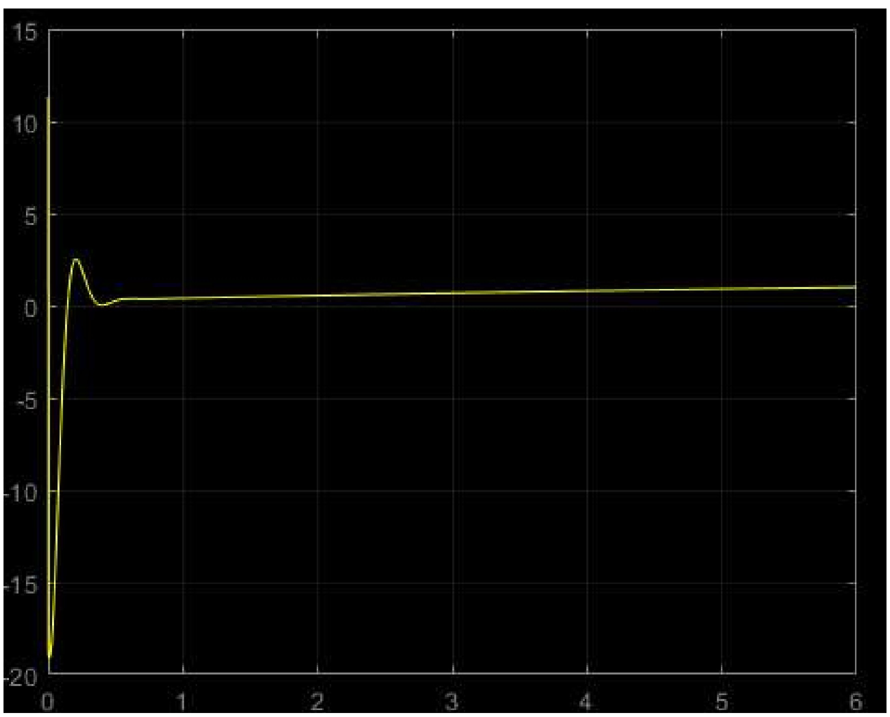
OpenLoop Ziegler control signal



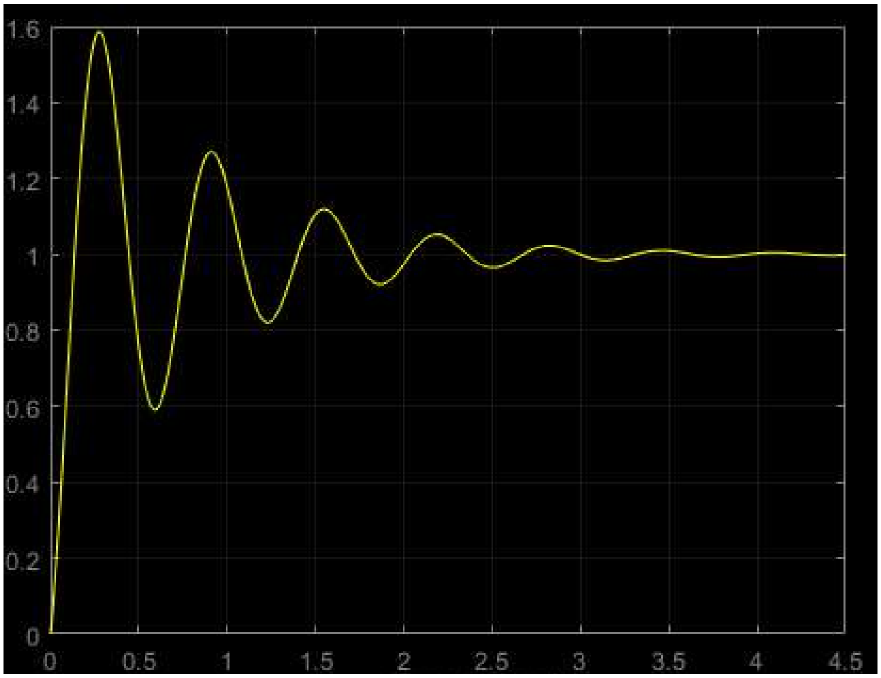
OpenLoop Ziegler tuned signal



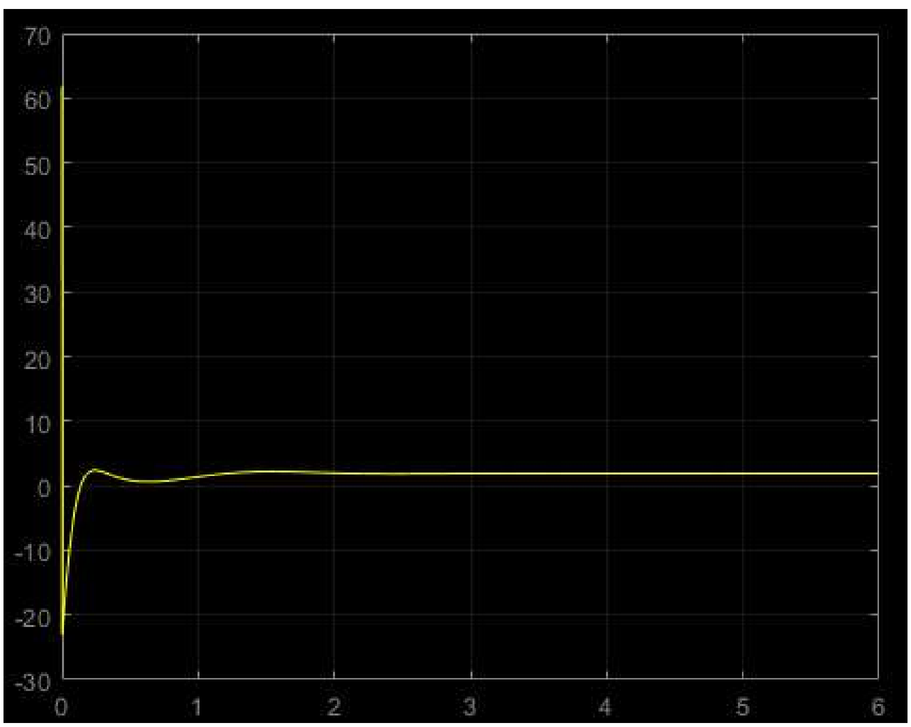
OpenLoop Ziegler tuned signal



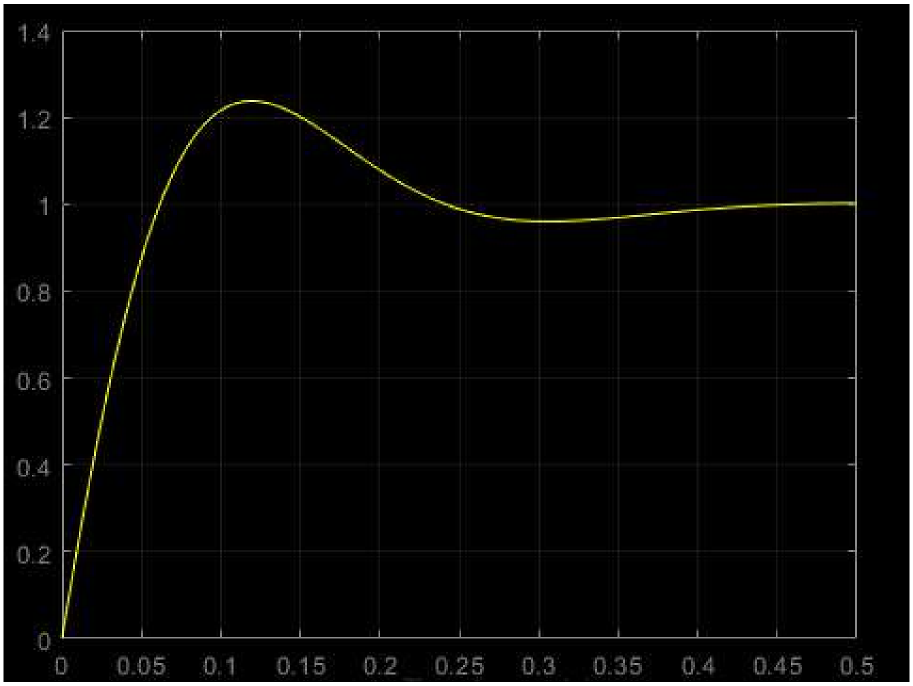
CloesdLoop Ziegler



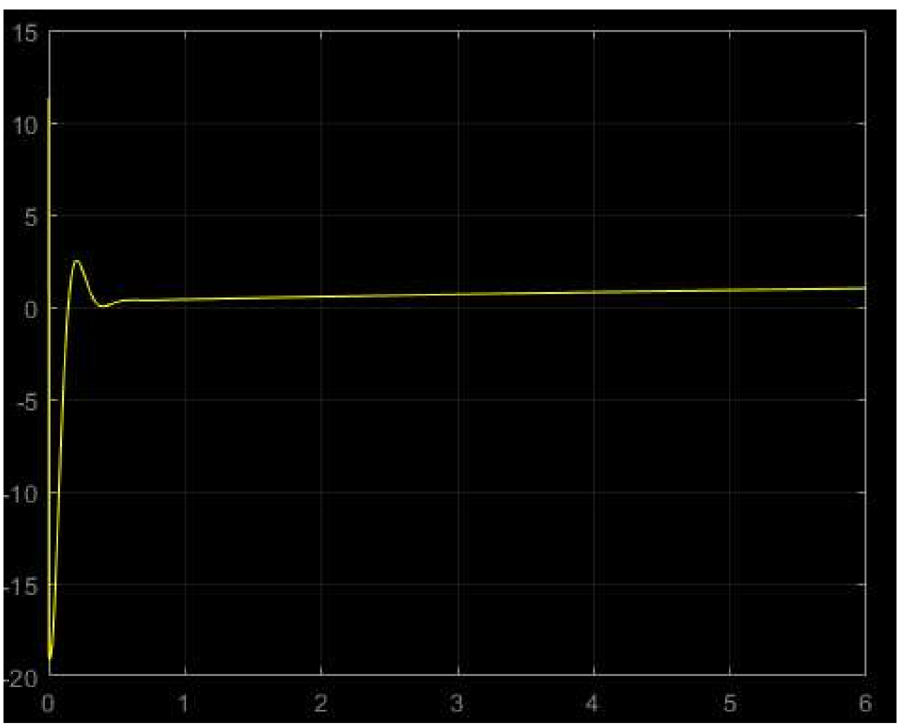
CloesdLoop Ziegler control signal



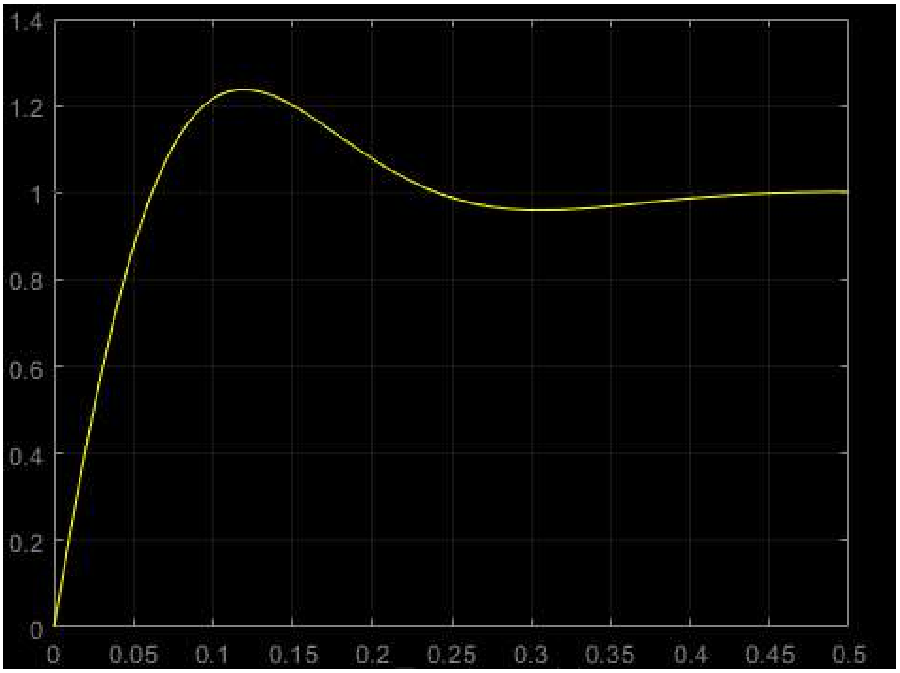
CloesdLoop Ziegler tuned



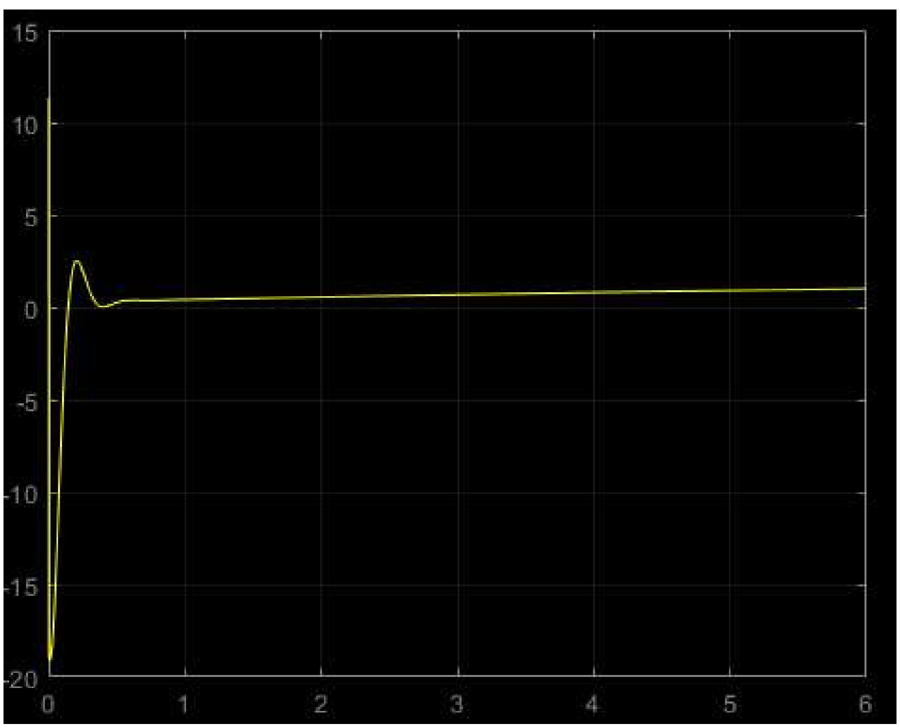
CloesdLoop Ziegler tuned control signal



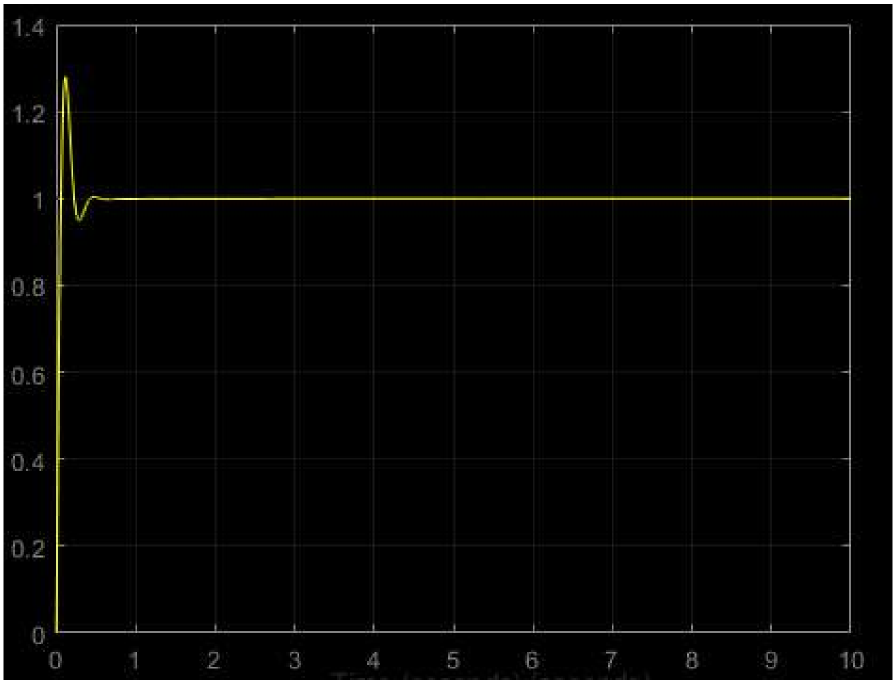
Manual



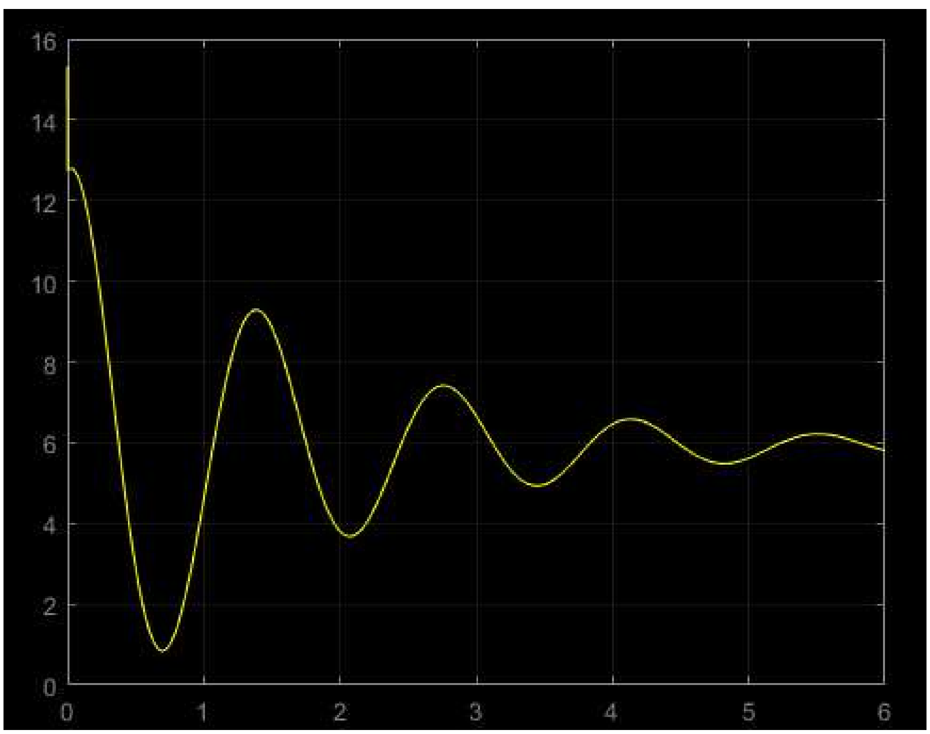
Manual control signal



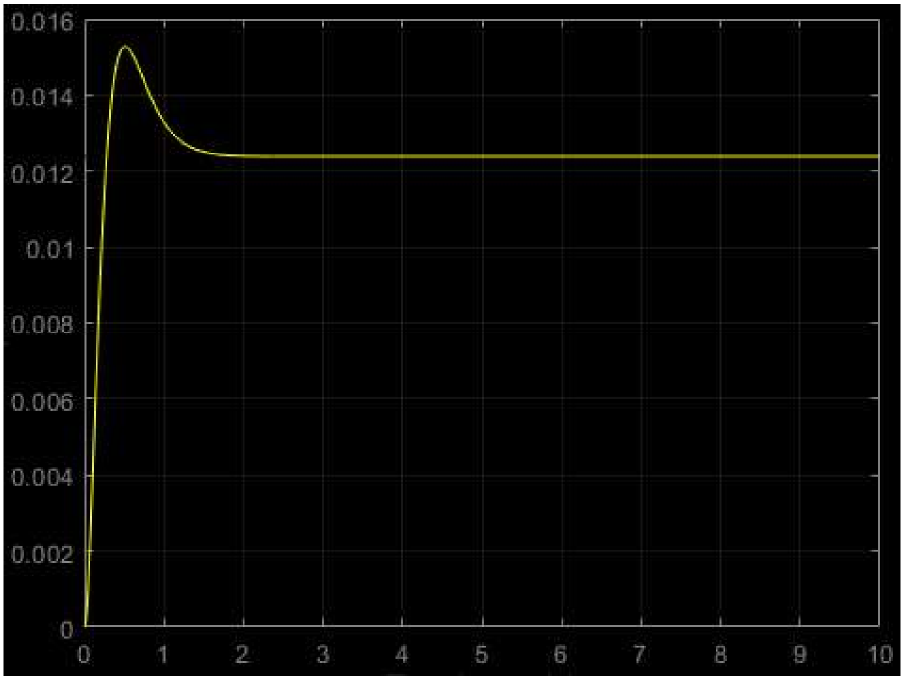
ITAE



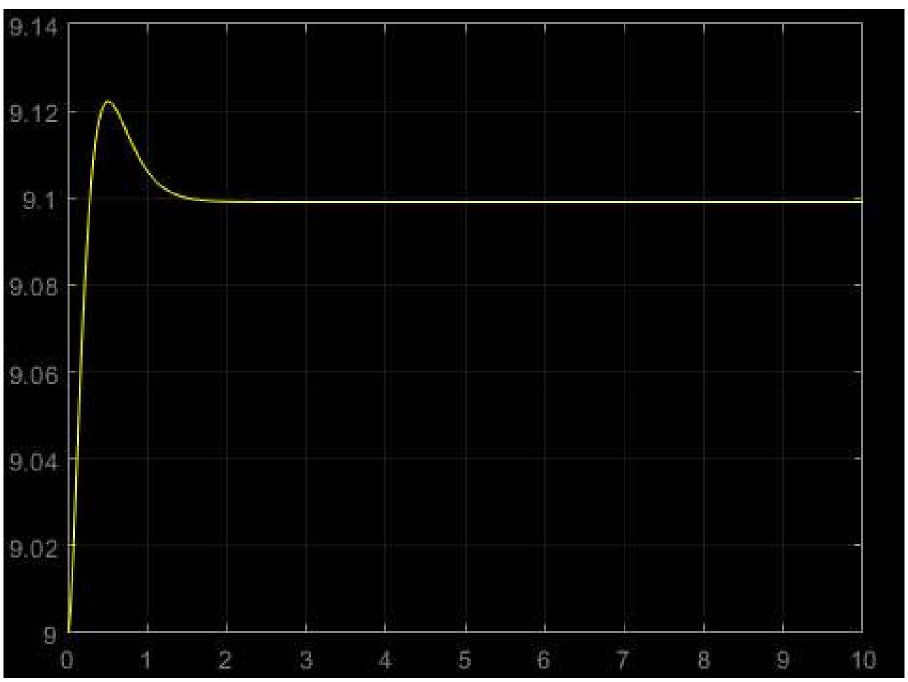
ITAE control signal



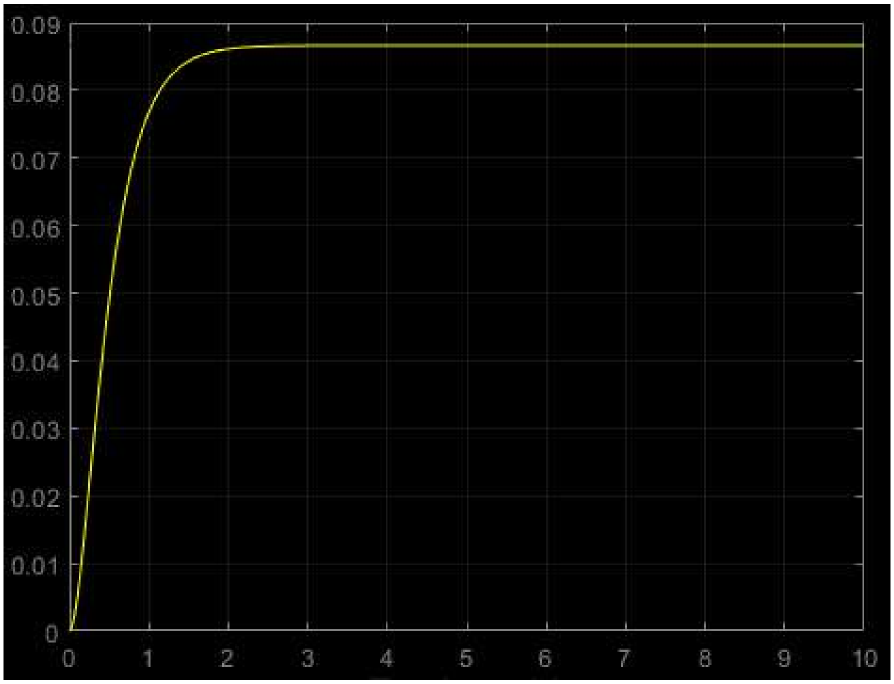
State Space



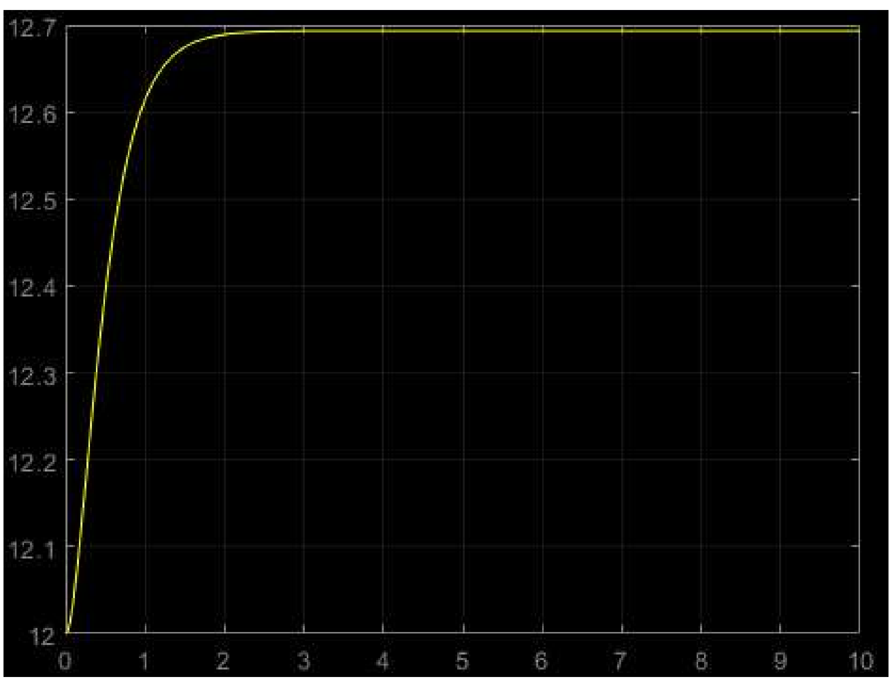
State Space Control signal



LQR



LQR Control Signal



4 Conclusion

4.1. PID (Open-loop ZN)

Settling Time: 0.4 s - Indicates a fast response.
PO (%): 23 - High overshoot, suggesting a potentially aggressive response.
Steady-State Error: 0 - Excellent steady-state tracking.
Max Control Signal: -18 - Moderate control effort.
Comment: This controller provides a quick response with excellent steady-state performance but at the cost of a high overshoot, indicating a potentially less stable or more oscillatory response.

4.2. PID (Closed-loop ZN)

Settling Time: 0.4 s - Similarly fast response as the open-loop ZN.
PO (%): 23 - Same high overshoot, indicating aggressive control.
Steady-State Error: 0 - Maintains excellent steady-state tracking.
Max Control Signal: -19 - Slightly higher control effort than open-loop ZN.
Comment: Very similar performance to the open-loop ZN PID, with a slightly higher control effort. It maintains fast response and good steady-state performance but with high overshoot.

4.3. PID (ITAE)

Settling Time: 0.4 s - Fast response.
PO (%): 23 - High overshoot, similar to other PID configurations.
Steady-State Error: 0 - Excellent steady-state tracking.
Max Control Signal: 15 - Notably higher control signal, indicating higher energy/resource usage.
Comment: This controller shows a balance between fast response and steady-state accuracy but requires a significantly higher control effort, which could be a concern in energy-sensitive applications.

4.4. PID (Manual)

Settling Time: 0.4 s - Quick response.
PO (%): 23 - High overshoot, indicating aggressive behaviour.
Steady-State Error: 0 - Perfect steady-state tracking.
Max Control Signal: -17.5 - Comparable control effort to other PID methods.
Comment: This manually tuned PID controller achieves a performance similar to the ZN methods, balancing fast response and steady-state accuracy but with high overshoot.

4.5. State-feedback

Settling Time: 1.2 s - Slower response compared to PID controllers.
PO (%): 23 - High overshoot, similar to PID controllers.
Steady-State Error: 0.99 - Noticeable steady-state error, which might be unacceptable in precision-critical applications.
Max Control Signal: 9.12 - Moderate control effort.
Comment: This controller is slower and has a significant steady-state error, which might limit its use in applications requiring high precision. However, it uses a moderate control effort.

4.6. Optimal (LQR)

Settling Time: 1 s - Moderately fast response, slower than PID but faster than state-feedback.
PO (%): 0 - No overshoot, indicating a very stable or well-damped response.
Steady-State Error: 0.91 - Some steady-state error, but less than state-feedback.
Max Control Signal: 12.7 - Highest control effort among the controllers.