Control Spring - Mass System Project

Ahmed Osama Helmy

Aliaa Abdelaziz

Abdallah Ahmed Ali

Omar Mahmoud Mohamed

ELC 3253-G2 Control Systems

28-4-2024

Req[1]: System Analysis

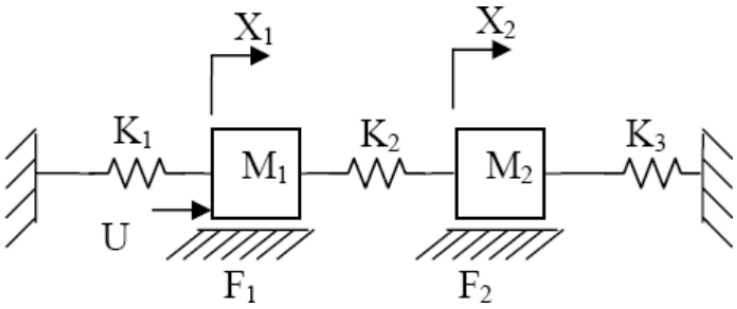


Figure 1: Spring-mass dashpot system

M1:

$$u - k_1x_1 - k_2(x_1 - x_2) - F_1\dot{x}_1 = M_1\ddot{x}_1$$

$$U(S) - K_1 * X_1(S) - K_2 * \big(X_1(S) - X_2(S)\big) - F_1 * S * X_1(S) = M_1 * S^2 * X_1(S)$$

$$U(S) + K_2 * X_2(S) = X_1(S) * (M_1S^2 + F_1S + K_2 + K_1) \rightarrow \boldsymbol{eq[1]}$$

M2:

$$-k_3x_2 - k_2(x_2 - x_1) - F_2\dot{x}_2 = M_2\ddot{x}_2$$

$$X_2(S) * (M_2S^2 + F_2S + K_2 + K_3) = K_2 * X_1(S) \rightarrow \boldsymbol{eq[2]}$$

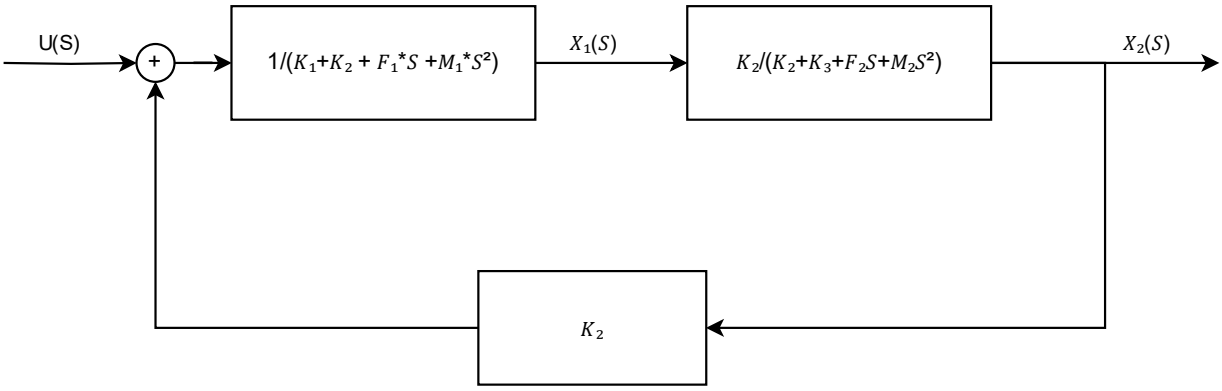


Figure 2: Block diagram of the system

NOTE:

A none simplified version of the block diagram is in the appendix at the end of the report.

Req[2]: Obtaining TFs using MATLAB

System Parameters:

$M_1 = M_2 = 100\ kg\ , K_1 = K_3 = 5\ N/m\ ,\quad K_2 = 50\ N/m\ ,\quad F_1 = F_2 = 100\ kg/sec$

$eq[1]:\ U(S) + 50 * X_2(S) = X_1(S) * (100 * S^2 + 100 * S + 55)$

$eq[2]:\ X_2(S) * (100 * S^2 + 100 * S + 55) = 50 * X_1(S)$

Code:

```
% Define transfer functions for individual components of the system
g1 = tf(1,[100,100,55]);  % Transfer function for X1
g2 = tf(50,[100,100,55]); % Transfer function for X2
g3 = tf(50,[-1]);         % Transfer function for g3 (feedback loop)

% Combine g1 and g2 in series to form sys1
sys1 = series(g1, g2);

% Create the complete control system by connecting sys1 and g3 in a feedback loop
sys_total = feedback(sys1, g3);

% transfer function X1/U & X2/U
X1_over_U = sys_total / g2;
X2_over_U = sys_total
```

Running the previous piece of code, we get the transfer function of the system

$$\frac{X_2(S)}{U(S)} = \frac{K_2}{(M_1S^2 + F_1S + K_2 + K_1) * (M_2S^2 + F_2S + K_3 + K_2) - {K_2}^2}$$

We also have from $eq[2]$

$$\frac{X_1(S)}{X_2(S)} = \frac{(M_2S^2 + F_2S + K_3 + K_2)}{K_2}$$

So, we can easily get

$$\frac{X_1(S)}{U(S)} = \frac{X_2(S)}{U(S)} * \frac{X_1(S)}{X_2(S)} = \frac{(M_2S^2 + F_2S + K_3 + K_2)}{(M_1S^2 + F_1S + K_2 + K_1) * (M_2S^2 + F_2S + K_3 + K_2) - {K_2}^2}$$

Req[3]: System Stability

Studying the stability of the system by analyzing the denominator of the transfer function:

$$(M_1S^2 + F_1S + K_2 + K_1) * (M_2S^2 + F_2S + K_3 + K_2) - {K_2}^2$$

We find that the system has 4 poles:

$$P_1 = -0.5000 + 0.8944i, P_2 = -0.5000 - 0.8944i,$$

$$P_3 = -0.9472 + 0.0000i\ , P_4 = -0.0528 + 0.0000i$$

We can see that all of them lie on the left half of the plane which means that the system is stable we also checked it using MATLAB isstable() function.

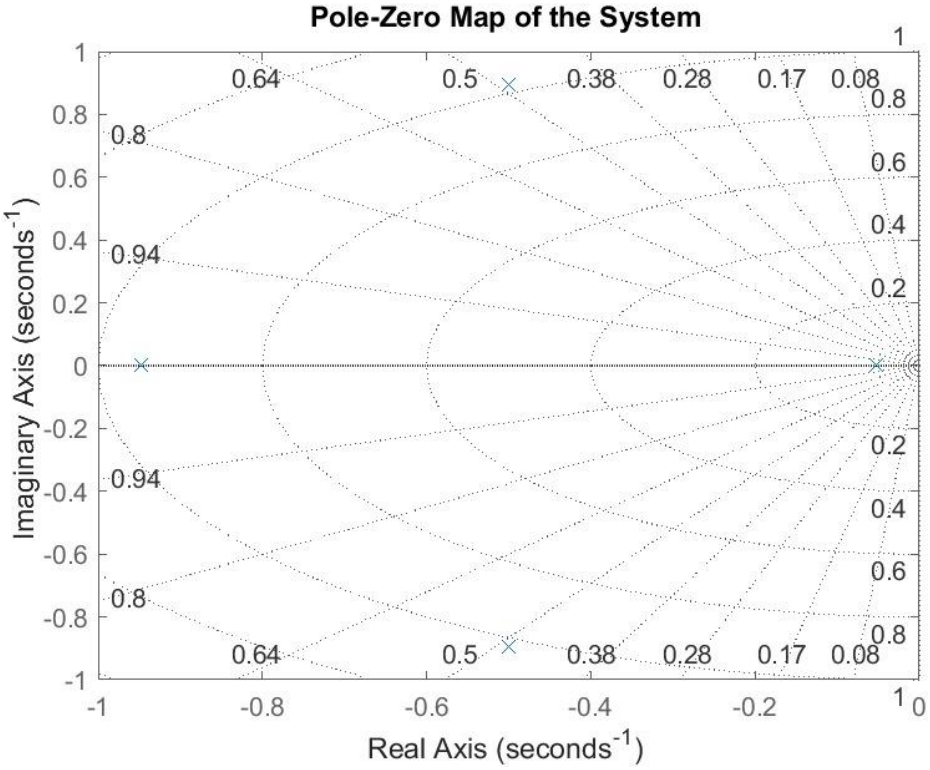


Figure 3: Pole-Zero map of the system

Req[4]: Simulating Input Force Of 1N.

Here we plot step responses for two transfer functions and then we calculate the steady state values of these signals.

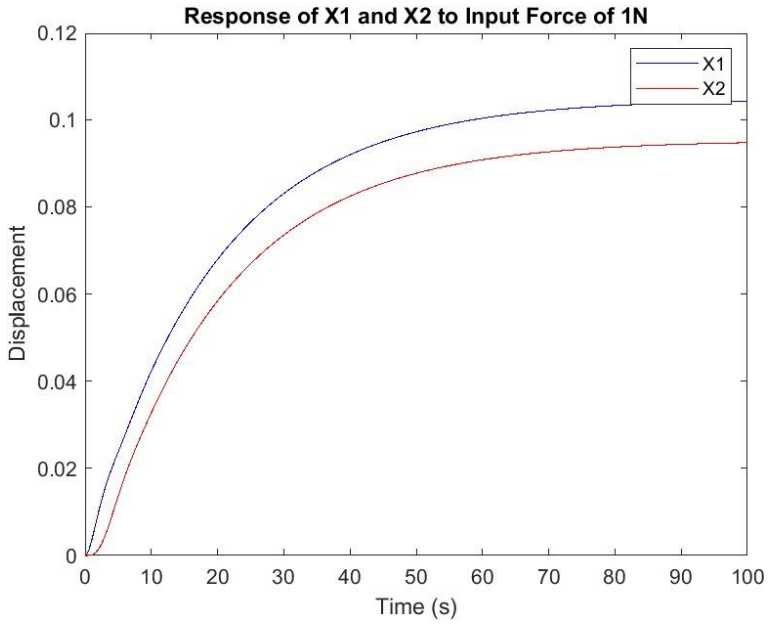


Figure 4: Response of X1 & X2 to U = 1N

Steady State Calculations:

- Steady-state value of X_1: 0.1044
- Steady-state value of X_2: 0.0949

Req[5]: Modifying System Input to Xd(S)

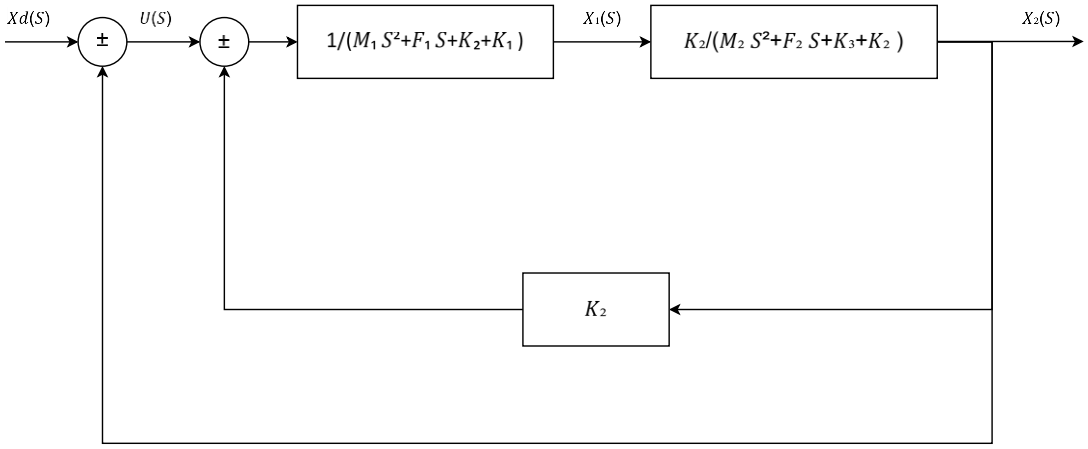


Figure 5: Block Diagram of the system after modification

Req[6]: Simulating the Modified System to Xd = 2m.

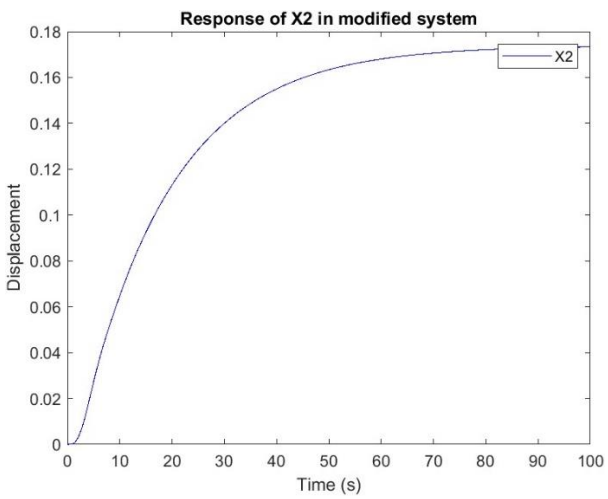


Figure 6: Response of X2 in modified system

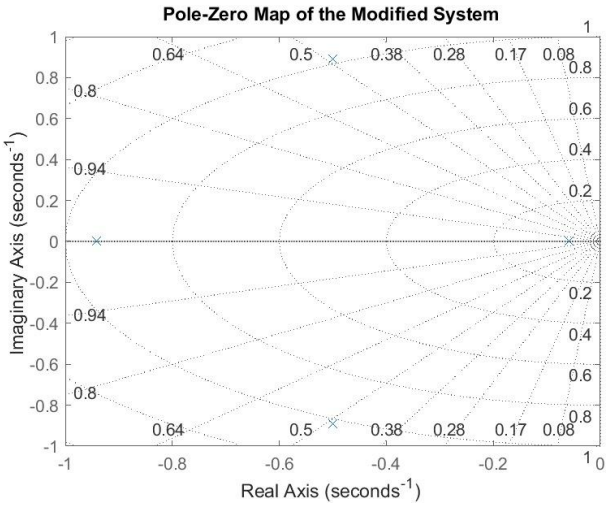


Figure 7: PZ map of the modified system

Req[7]: Modified System Calculations

For the response of X_2 we calculate the value of the rise time, peak time, max peak, and settling time.

Also calculate the value of e_{ss}.

Calculations:

- $X_{2-ss} = 0.1733$
- Rise Time: 36.9553
- Peak Time: 99.2892
- Maximum Peak: 0.1733
- Settling Time: 66.3290
- Steady-State Error: 1.8267

Req[8]: Controller Modifications & Adjustments

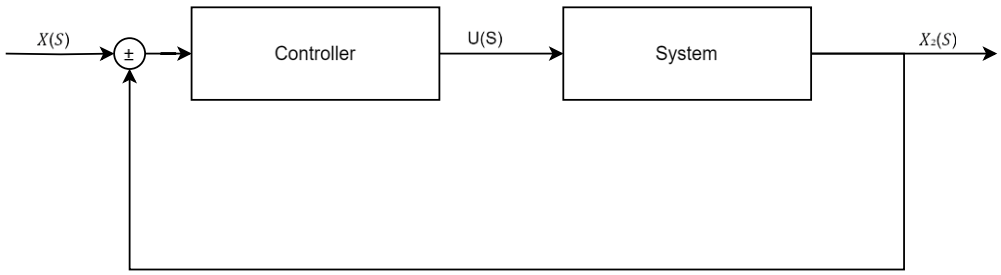
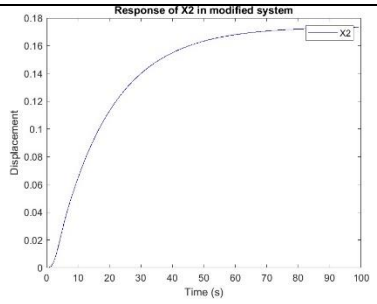
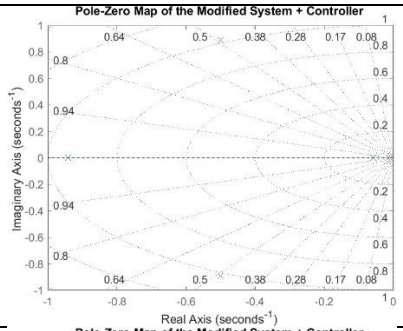
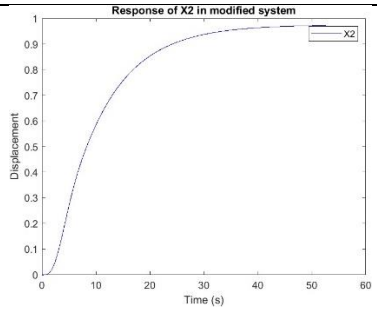
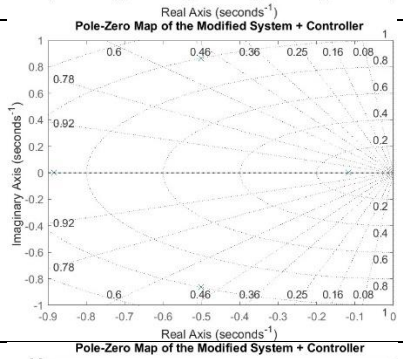
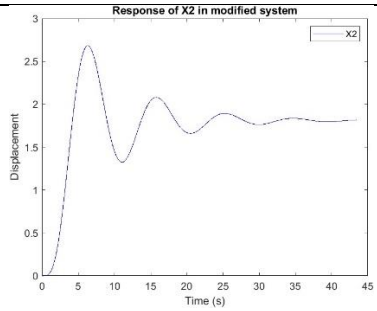
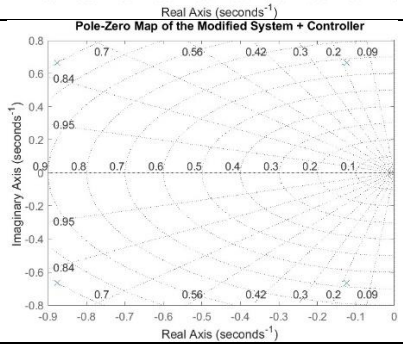
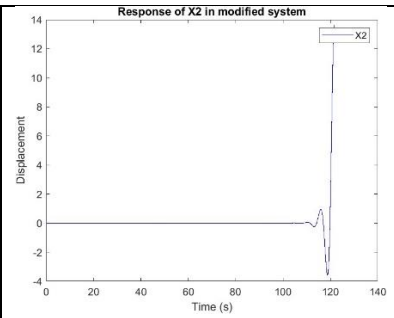
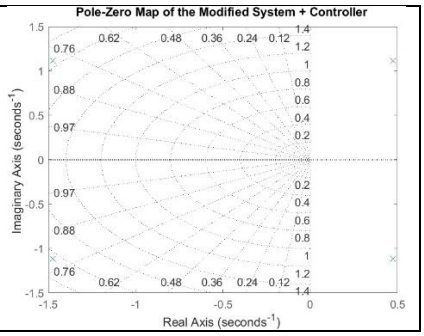


Figure 8: Adding proportional controller to the system

	Steady State Values	Response	Pole-Zero Map
$K_p = 1$	$X_{2-ss} =$ 0.1733 Rise Time: 36.9553 Peak Time: 99.2892 Maximum Peak: 0.1733 Settling Time: 66.3290 Steady-State Error: 1.8267		
$K_p = 10$	$X_{2-ss} =$ 0.9729 Rise Time: 18.6348 Peak Time: 52.6344 Maximum Peak: 0.9729 Settling Time: 34.6701 Steady-State Error: 1.0271		
$K_p = 100$	$X_{2-ss} =$ 1.8170 Rise Time: 2.2240 Peak Time: 6.3068 Maximum Peak: 2.6833 Settling Time: 31.3585 Steady-State Error: 0.1830		

$K_p = 1000$	The system is unstable $X_{2-ss} = 13619050112 * e^{15}$		

Comments:

The system is stable for all cases except for the last case, the system will be unstable.
Which is why we ignore this case in our comments and focus on the values of the controller where the system is stable.

- ***Rise time*:**
  As the value of the proportional controller increases the rise time which is the time required for a pulse to rise from 10 per cent to 90 per cent of its steady value decreases.

- ***Peak time*:**
  As the value of the proportional controller increases the peak time decreases for the stable cases.

- ***Settling time*:**
  As the value of the proportional controller increases the settling time which is the time required for the output to settle around a certain value with some tolerance decreases.

- ***Steady state error (ess):***
  As the value of the proportional controller increases the steady state error decreases, so the system becomes more accurate in reaching the desired output value.

Req[9]: Finding K_p value for $e_{ss} \leq 0.01$

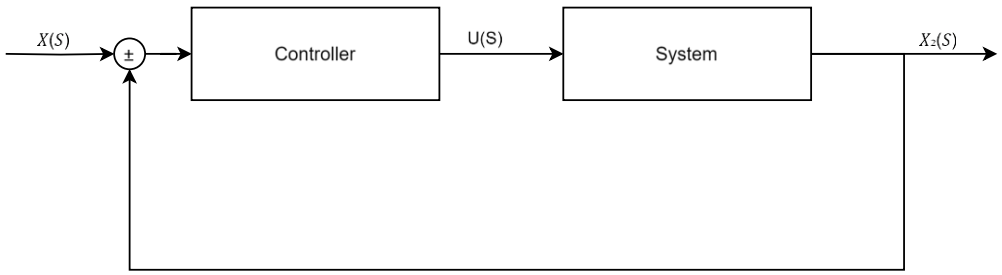


Figure 9: Finding Kp value to minimize the ess

Theoretical Analysis:

$$TF_{sys} = \frac{X_2(S)}{U(S)} = \frac{50}{10^4 * S^4 + 2 * 10^4 * S^3 + 21 * 10^3 * S^2 + 11 * 10^3 * S + 525}$$

$$E(S) = R(S) - C(S) * H(S)\,, \qquad C(S) = G(S) * E(S), \qquad H(S) = 1, \qquad G(S) = K_p * TF_{sys}$$

$$R(S) = \frac{4}{S}\,, \qquad E(S) = \frac{R(S)}{1 + G(S)}$$

$$e_{ss} = \lim_{s\to 0} S * E(S) = \lim_{s\to 0} \frac{S * (4/S)}{1 + G(S)} = \lim_{s\to 0} \frac{4}{1 + G(S)}$$

$$\lim_{s\to 0} G(S) = \lim_{s\to 0} K_p * TF_{sys} = \lim_{s\to 0} \frac{50 * K_p}{10^4 * S^4 + 2 * 10^4 * S^3 + 21 * 10^3 * S^2 + 11 * 10^3 * S + 525}$$

$$= \frac{50 * K_p}{525} = \frac{2}{21} * K_p$$

For $\boldsymbol{e_{ss} \leq 0.01}$**:**

$$\frac{4}{1 + \frac{2}{21} * K_p} \leq 0.01 \;\rightarrow K_p \geq 4189.5$$

Simulation:

We will try to simulate this result but we know this is not feasible as this will cause the system to be unstable as this is higher than $K_p = 1000$ from the previous point but we will try and take a look at the results.

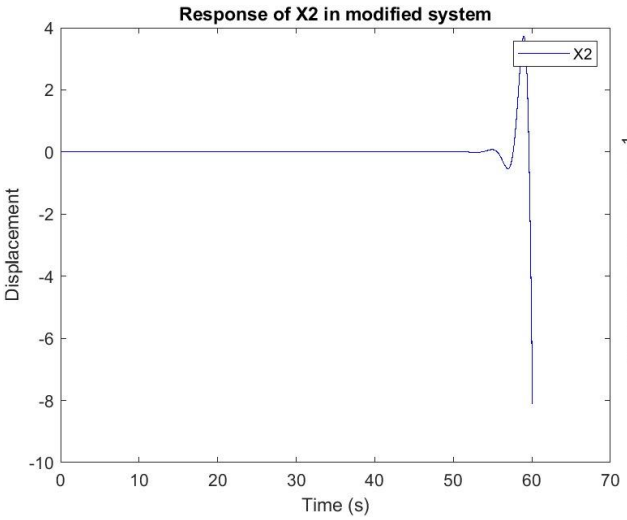


Figure 10: Response of system with new controller

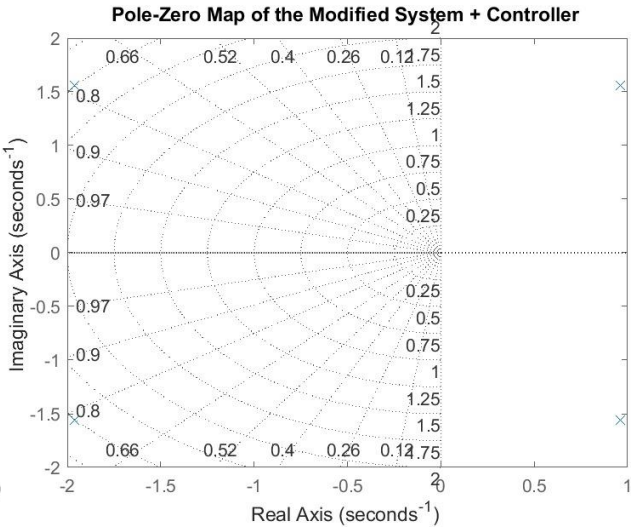


Figure 11: The system is unstable

We got exactly what we expected the Pole-Zero Map shows that the system is unstable so the requirement is not achievable using only a proportional controller.

Req[10]: Using a PI Controller

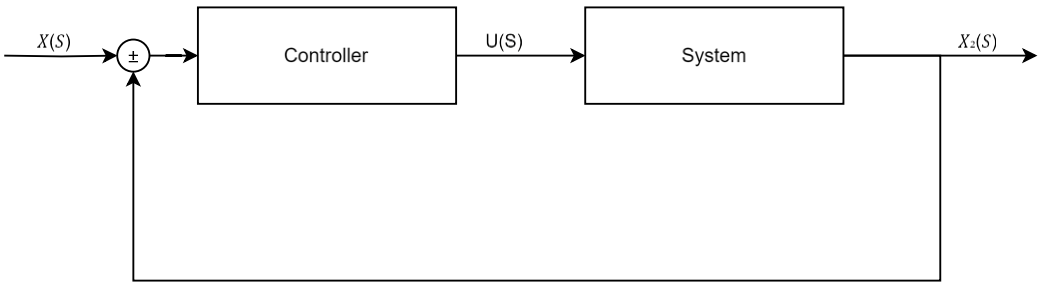
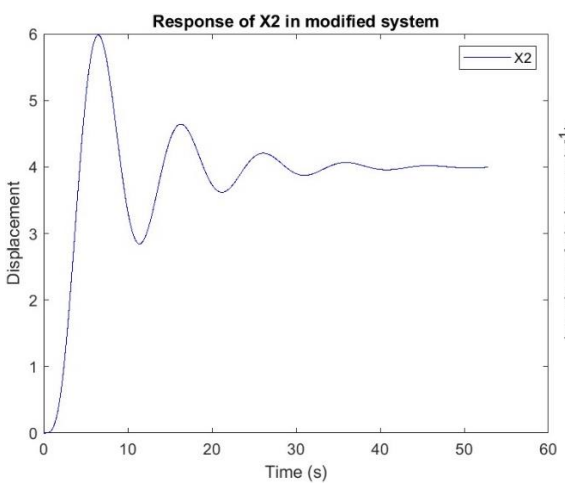


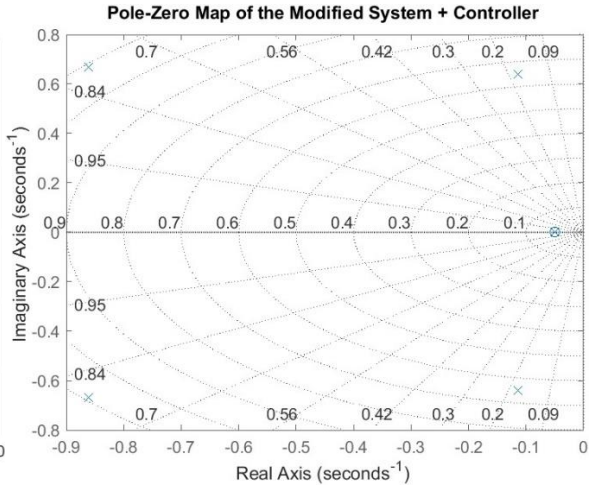
Figure 12: System W/ PI Controller

Here we used a PI controller to achieve $\boldsymbol{e_{ss} \leq 0.01}$ and through trial and error we found that these values satisfied the requirements $K_p = 100$ & $K_i = 4\ or\ 5$ both gave similar acceptable results which result in a stable system as shown below.

Results:

- $X_{2-ss} = 4$
- Rise Time: 2.2657
- Peak Time: 6.5247
- Maximum Peak: 5.9835
- Settling Time: 32.3876
- Steady-State Error: 0.0086





[Appendix]: Non-Simplified Block Diagram

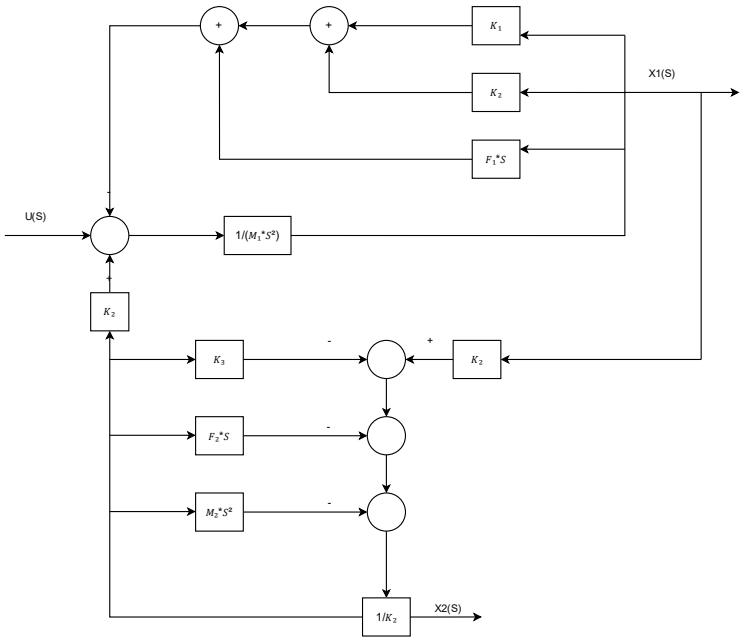


Figure 13: non-simplified block diagram

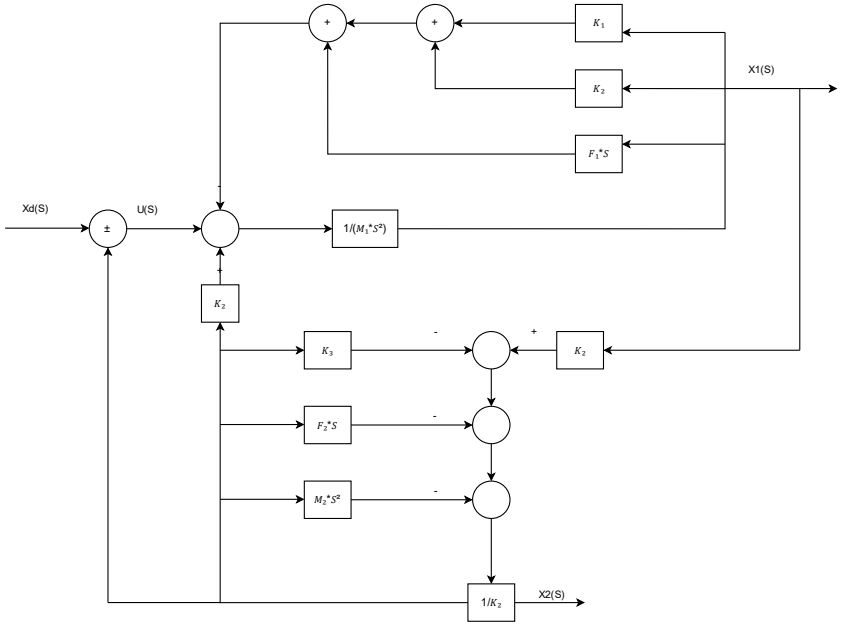


Figure 14: suggested modification to non-simplified block diagram

MATLAB Code

```
close all;
% Define transfer functions for individual components of the system
g1 = tf(1,[100,100,55]);  % Transfer function for X1
g2 = tf(50,[100,100,55]); % Transfer function for X2
g3 = tf(50,[-1]);         % Transfer function for g3 (feedback loop)

% Combine g1 and g2 in series to form sys1
sys1 = series(g1, g2);

% Create the complete control system by connecting sys1 and g3 in a feedback loop
sys_total = feedback(sys1, g3);

% transfer function X1/U & X2/U
X1_over_U = sys_total / g2
X2_over_U = sys_total

% Get the poles of the system sys_total
P = pole(sys_total);

P

% Plot the pole-zero map for the system sys_total
figure;
pzmap(sys_total);
title('Pole-Zero Map of the System');
grid on;

% Check stability of the system sys_total
is_stable = isstable(sys_total);

% Define the input force of 1N
input_force = 1;

% Simulate the response of the system to the input force
% Simulation
force_input = input_force * ones(size(t));
[y1, t1] = step(X1_over_U);
[y2, t2] = step(X2_over_U);

% Get the responses
X1_response = y1(:, 1); % Response of X1
X2_response = y2(:, 1); % Response of X2
```

```
% Calculate steady-state values
X1_steady_state = y1(end); % Mean value of X1 in the last 1 second
X2_steady_state = y2(end); % Mean value of X2 in the last 1 second

% Plots of X1 & X2
figure;
plot(t1, X1_response, 'b', t2, X2_response, 'r');
xlabel('Time (s)');
ylabel('Displacement');
legend('X1', 'X2');
title('Response of X1 and X2 to Input Force of 1N');

% Print the result in the command window
fprintf('Steady-state value of X1: %.4f\n', X1_steady_state);
fprintf('Steady-state value of X2: %.4f\n', X2_steady_state);

% Req[5] : Suggested modification

Xd = 2;
mod_system = feedback(sys_total,1)
Xd_input = Xd * ones(size(t));

[y, t] = step(2*mod_system);

X2_response = y(:, 1); % Response of X2
X2_steady_state = y(end); % Mean value of X2 in the last 1 second

% Plot the pole-zero map for the modified system
figure;
pzmap(mod_system);
title('Pole-Zero Map of the Modified System');
grid on;

figure;
plot(t, X2_response, 'b');
xlabel('Time (s)');
ylabel('Displacement');
legend('X2');
title('Response of X2 in modified system');
fprintf('Steady-state value of X2: %.4f\n', X2_steady_state);

% Calculate step response of X2
[y, t] = step(2*mod_system);

% Calculate step response characteristics
```

```
step_info = stepinfo(y, t);

% Rise Time
rise_time = step_info.RiseTime;

% Peak Time
peak_time = step_info.PeakTime;

% Maximum Peak
max_peak = step_info.Peak;

% Settling Time
settling_time = step_info.SettlingTime;

% Steady-State Error
% For step input, steady-state error is the difference between the final value
and the desired value
final_value = y(end);
desired_value = 2; % Desired value of Xd for the modified system
ess = desired_value - final_value;

% Display the calculated values
fprintf('Rise Time: %.4f\n', rise_time);
fprintf('Peak Time: %.4f\n', peak_time);
fprintf('Maximum Peak: %.4f\n', max_peak);
fprintf('Settling Time: %.4f\n', settling_time);
fprintf('Steady-State Error: %.4f\n', ess);

% Proportional controller values to test
Kp_values = [1, 10, 100, 1000, 4189.5];

% Initialize arrays to store transient response parameters for each case
rise_times = zeros(size(Kp_values));
peak_times = zeros(size(Kp_values));
max_peaks = zeros(size(Kp_values));
settling_times = zeros(size(Kp_values));
ess_values = zeros(size(Kp_values));

% Simulate the system with different values of Kp
for i = 1:length(Kp_values)
    % Define the proportional controller
    Kp = Kp_values(i);
    controller = tf(Kp);

    % Connect the controller in series with the system
```

```
    sys_with_controller = series(controller, sys_total);
    mod_sys_with_controller = feedback(sys_with_controller, 1);

    % Simulate the response of the system to a step input
    if Kp == 4189.5
        [y, t] = step(4*mod_sys_with_controller);
    else
        [y, t] = step(2*mod_sys_with_controller);
    end

    % Calculate step response characteristics
    step_info = stepinfo(y, t);

    % Store the transient response parameters
    rise_times(i) = step_info.RiseTime;
    peak_times(i) = step_info.PeakTime;
    max_peaks(i) = step_info.Peak;
    settling_times(i) = step_info.SettlingTime;

    % Calculate steady-state error
    final_value = y(end);
    ess_values(i) = desired_value - final_value;
    figure;
    plot(t, y, 'b');
    xlabel('Time (s)');
    ylabel('Displacement');
    legend('X2');
    title('Response of X2 in modified system');
    fprintf('Steady-state value of X2: %.4f\n', X2_steady_state);
    figure;
    pzmap(mod_sys_with_controller);
    title('Pole-Zero Map of the Modified System + Controller');
    grid on;

end

% Display the results
disp('Proportional Controller (Kp) | Rise Time | Peak Time | Max Peak | Settling
Time | Steady-State Error');
disp('-------------------------------------------------------------------------
------------------');
for i = 1:length(Kp_values)
    fprintf('%12d | %9.4f | %9.4f | %8.4f | %13.4f | %19.4f\n', Kp_values(i),
rise_times(i), peak_times(i), max_peaks(i), settling_times(i), ess_values(i));
end
```

```
% Proportional controller values to test
Kp_values = [1, 10, 100, 1000];

% Initialize arrays to store transient response parameters for each case
steady_state_values = zeros(size(Kp_values));

% Simulate the system with different values of Kp
for i = 1:length(Kp_values)
    % Define the proportional controller
    Kp = Kp_values(i);
    controller = tf(Kp);

    % Connect the controller in series with the system
    sys_with_controller = series(controller, sys_total);
    mod_sys_with_controller = feedback(sys_with_controller, 1);

    % Simulate the response of the system to a step input
    [y, t] = step(2*mod_sys_with_controller);

    % Calculate steady-state value of X2
    steady_state_values(i) = y(end);
end

% Display the steady-state values of X2 for each case
disp('Proportional Controller (Kp) | Steady-State Value of X2');
disp('-----------------------------------------------------');
for i = 1:length(Kp_values)
    fprintf('%12d | %22.4f\n', Kp_values(i), steady_state_values(i));
end

% PI Controller
    Ki = 5;
    Kp = 100;
    controller = pid(Kp, Ki);

    % Connect the controller in series with the system
    sys_with_controller = series(controller, sys_total);
    mod_sys_with_PI_controller = feedback(sys_with_controller, 1);

    % Simulate the response of the system to a step input
    [y, t] = step(4*mod_sys_with_PI_controller);

    % Calculate step response characteristics
    step_info = stepinfo(y, t);
```

```
    % Store the transient response parameters
    rise_times_PI = step_info.RiseTime;
    peak_times_PI = step_info.PeakTime;
    max_peaks_PI = step_info.Peak;
    settling_times_PI = step_info.SettlingTime;

    % Calculate steady-state error
    final_value = y(end);
    ess_values_PI = desired_value - final_value;
    figure;
    plot(t, y, 'b');
    xlabel('Time (s)');
    ylabel('Displacement');
    legend('X2');
    title('Response of X2 in modified system');
    fprintf('Steady-state value of X2: %.4f\n', X2_steady_state);
    figure;
    pzmap(mod_sys_with_PI_controller);
    title('Pole-Zero Map of the Modified System + Controller');
    grid on;

    fprintf('%12d | %9.4f | %9.4f | %8.4f | %13.4f | %19.4f\n', Kp,
rise_times_PI, peak_times_PI, max_peaks_PI, settling_times_PI, ess_values_PI);
```