Alperen ISIK
Monyrotanak Sambath UY
Hasan Emre AYDIN

Control Theory Laboratory Report

Introduction

The aim of this laboratory study is to delve into the complexities of a DC motor, which is an essential component in electromechanical systems engineering. This system comprises both electrical and mechanical parts and operates based on principles of electro-mechanical energy conversion. Our investigation will focus on understanding and optimizing the DC motor's behavior through the use of various control strategies implemented in MATLAB/SIMULINK. We will examine the DC motor's block diagram and scrutinize its performance across different controller configurations including Proportional (P), Integral (I), Proportional Integral (PI), Proportional Derivative (PD), and Proportional Integral Derivative (PID) controllers.

The primary objectives of this study are to control the DC armature current and the angular velocity of the motor and to assess the effectiveness and limitations of each controller in terms of parameters such as rise time, overshoot, settling time, steady state error, and overall system stability. This comprehensive approach will allow us to explore the intricacies of each control mechanism and their impact on the motor's operation, thereby enhancing our ability to optimize and refine its performance in practical applications.

TASK 1:

Question 1: A)

Electrical equation

$$\mathcal{L} \quad \text{Initial condition} = 0 \qquad l\,s\,I(s) + r\,I(s) = U(s) - E(s)$$

$$I(s)\left[ls + r\right] = U(s) - E(s) \Rightarrow I(s) = \frac{1}{ls + r}\left[U(s) - E(s)\right]$$

Mechanical equation

$$j\Omega(s) + f\Omega(s) = C(s) - C_s$$

$$\Omega(s)(j + f) = C(s) - C_s$$

$$\Omega(s) = \frac{C(s)-C_s}{js+f}$$

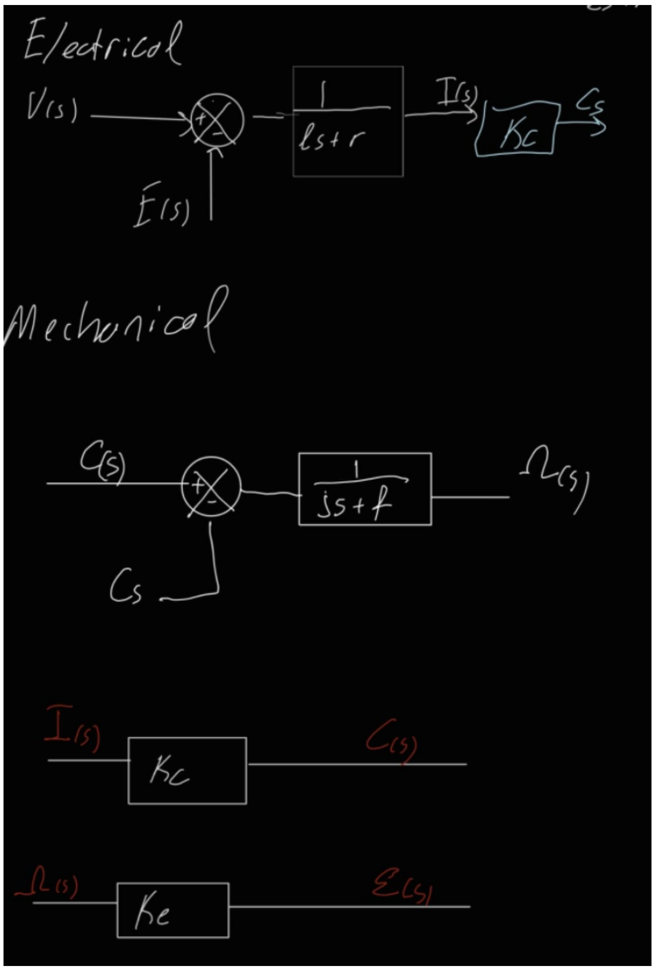
Electro-mechanical equation

$$C(s) = K_c I(s)$$

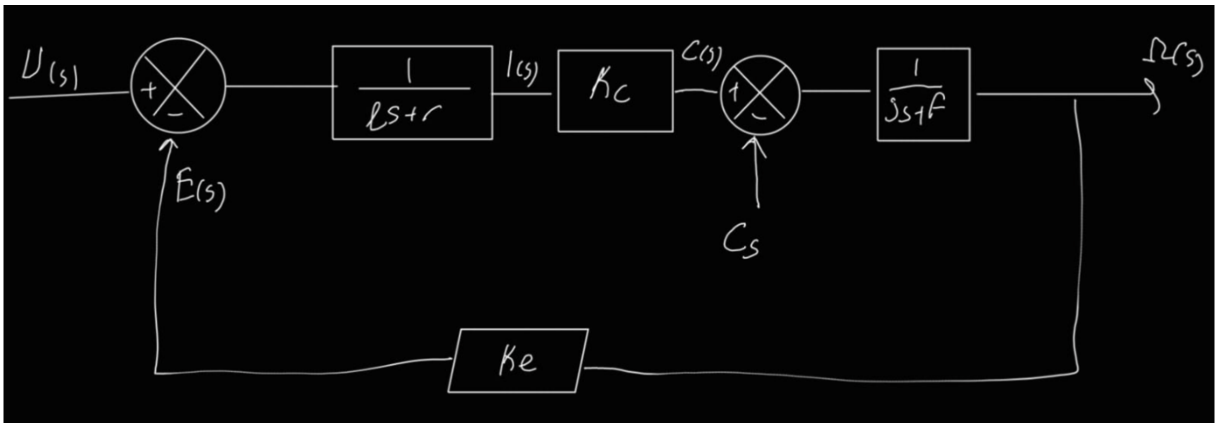
$$E(s) = K_e \Omega(s)$$

b) Block diagram of each part

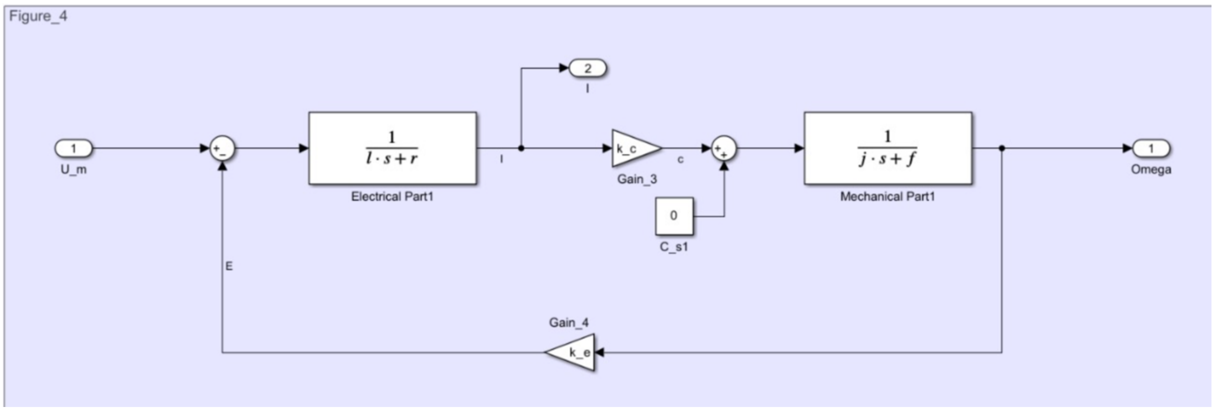
:



c) If we combine these block diagrams according to the inputs and outputs we wrote in the block diagram, we get the same visual as figure 4.



d) Block diagram of the DC motor



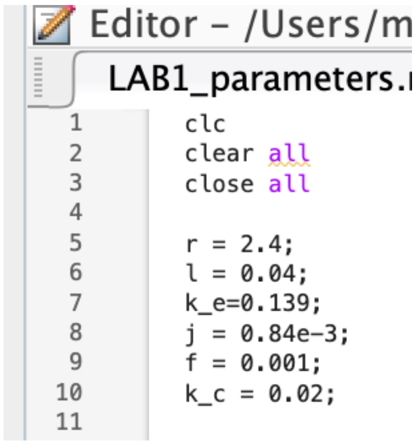
QUESTION 2:

- We define the following parameters value

$$r = 2.4\,\Omega\,; l = 0.04\,H; k_e = 0.139\,V/(rad\,/s);$$

$$j = 0{,}84 \times 10^{-3} kg\,m^2; f = 0.001; k_c = 0.02\,N\,m\,A^{-1}$$

We took information's from given datasheet

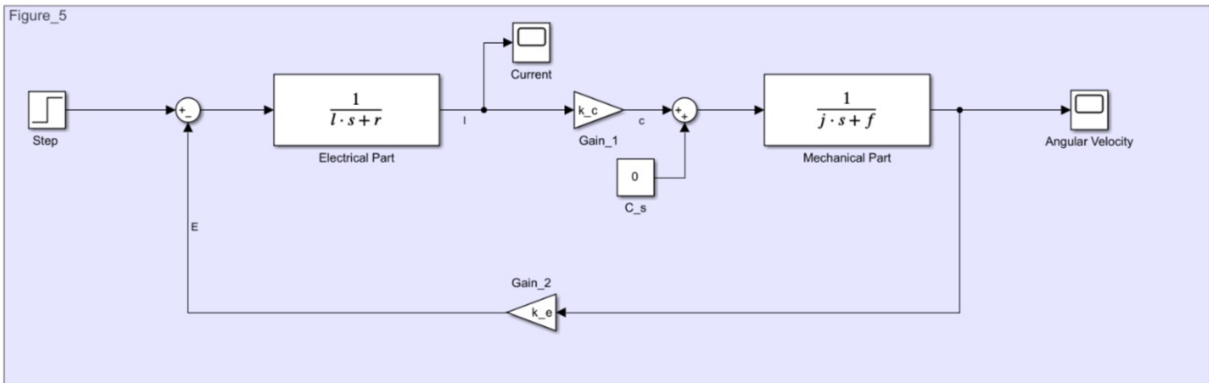


Editor – /Users/m

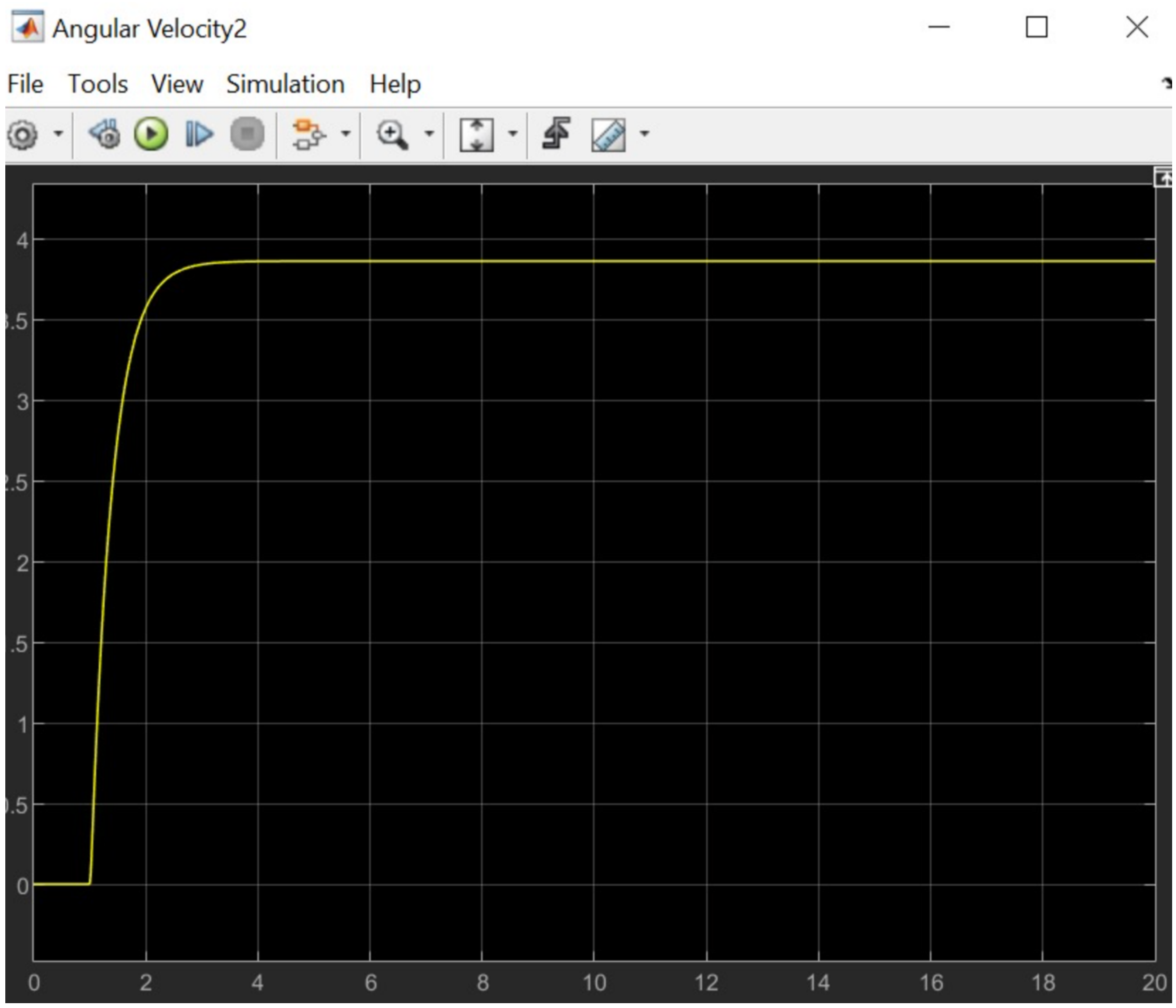
LAB1_parameters.

```
clc
clear all
close all

r = 2.4;
l = 0.04;
k_e=0.139;
j = 0.84e-3;
f = 0.001;
k_c = 0.02;

```

In our MATLAB/Simulink project, we designed and ran a simulation by creating a new block diagram. In this diagram, we replaced the I(s) and Ω(s) components with scopes to effectively display outputs and introduced a unit step as the input to the circuit. This setup allows for a clear visualization of the system's response to changes in input within the Simulink environment.

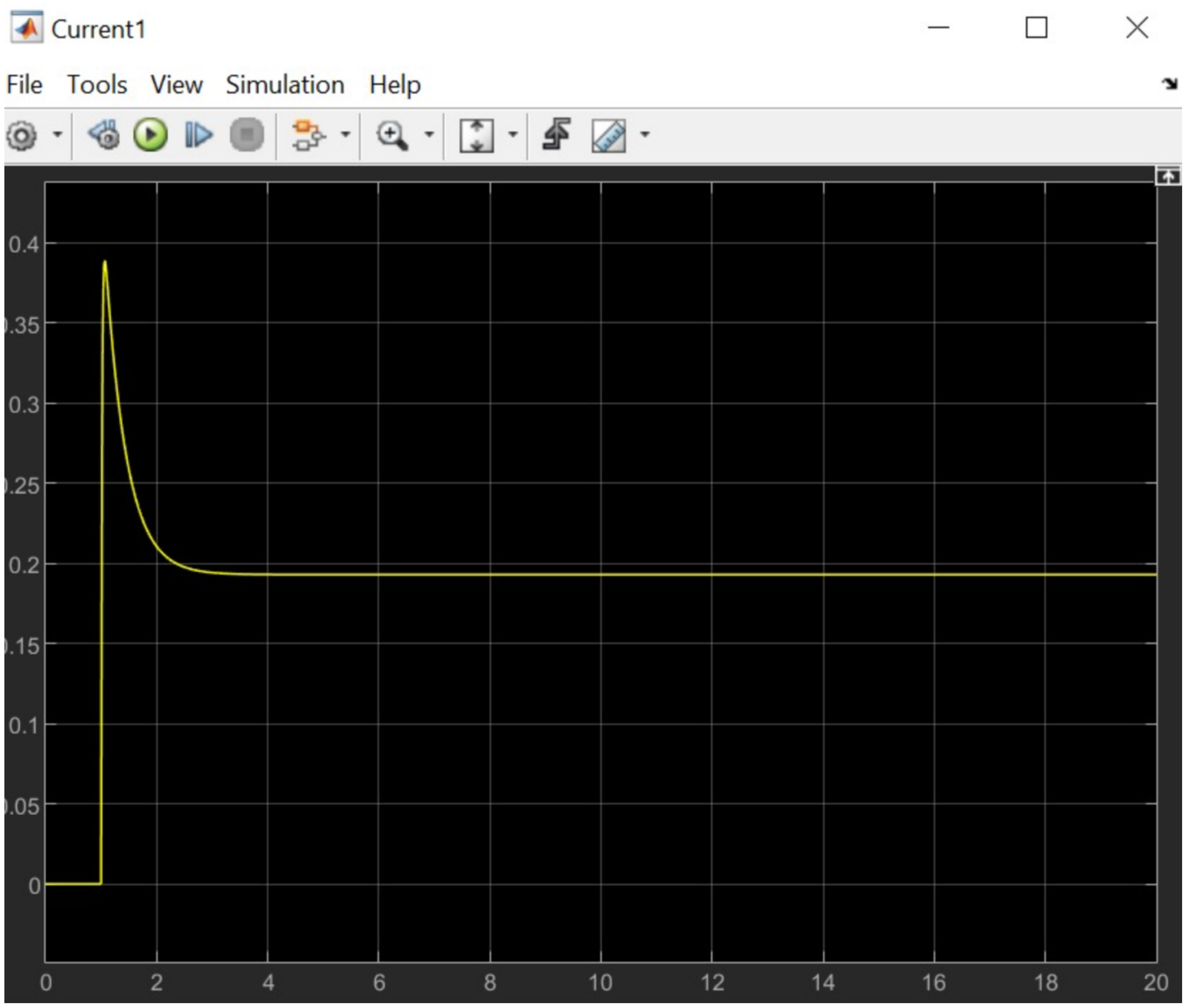


The simulation results are shown in these plots:



In Simulink, we analyzed the comparison between the input and the Angular velocity output, resulting in the plot depicted below:

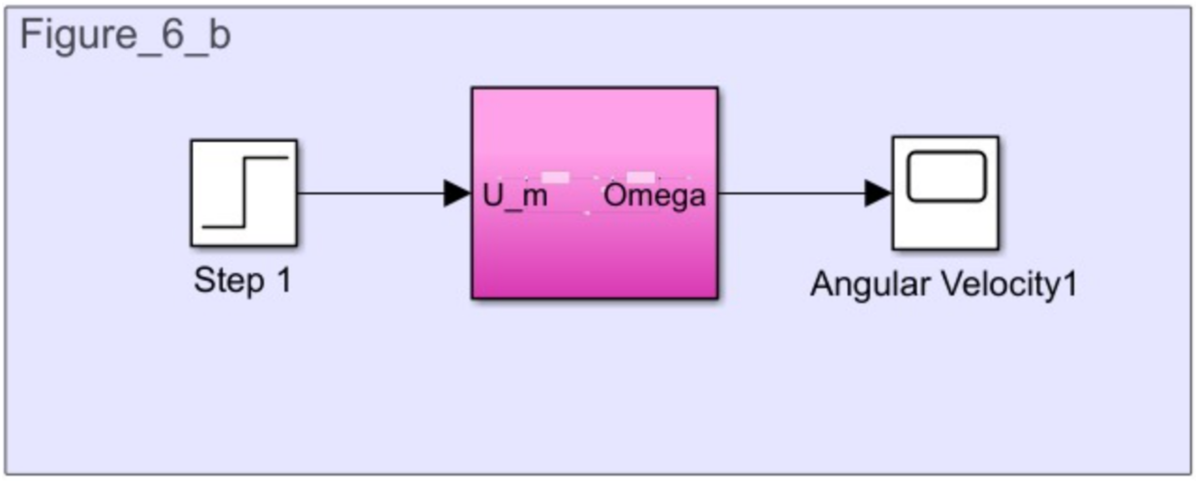
It's noticeable that the signal remains at 0 before any input is applied, which makes sense. When we introduce a unit step signal, the angular velocity Ω(s) responds similarly to the impulse response characteristic of a first-order system, displaying a vertical tangent at the origin and stabilizing near 3.8 rad/s.



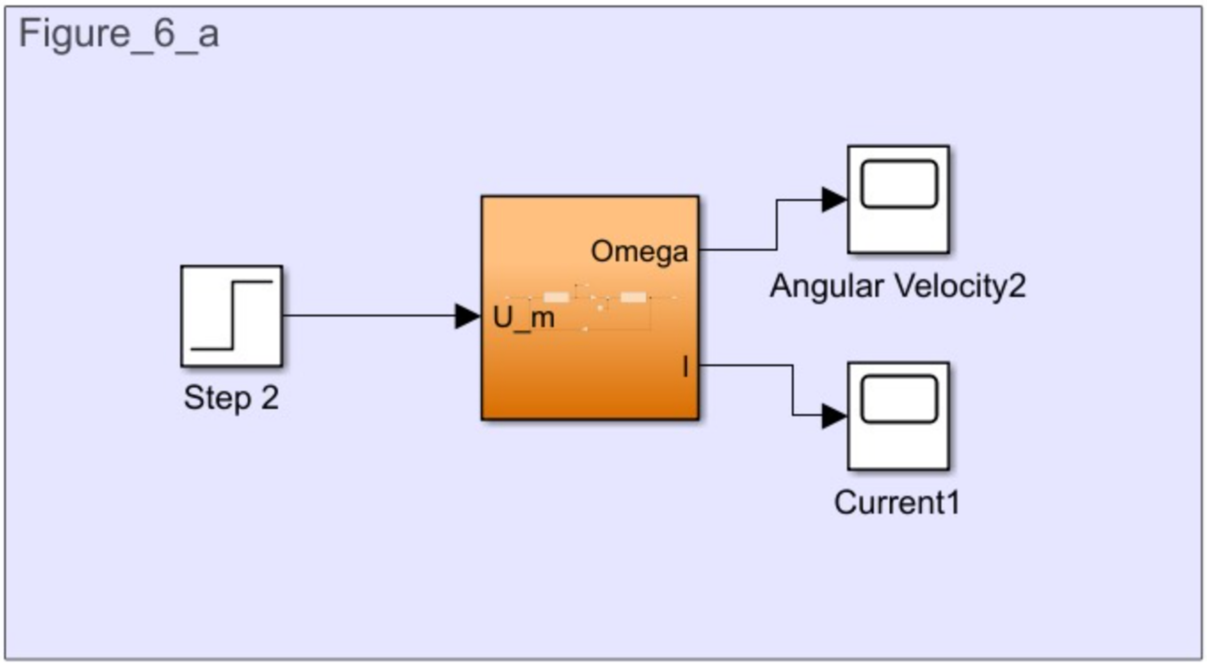
This figure graphically displays the armature current of the motor. It clearly shows a peak with an amplitude of 0.37A around 1 second, indicative of significant overshoot following the unit step input signal. By the third second, the system stabilizes at approximately 0.19A.

QUESTION 3:

In our simulation, we developed two subsystems to analyze the behavior of a motor under a unit step signal. The first subsystem, highlighted in pink, comprises the armature voltage *Um* and angular velocity ω, while the second, colored in orange, includes the armature voltage *Um*, armature current *I*, and angular velocity $\omega\omega$. After running the simulation, we plotted the armature current and angular velocity, observing that these outputs matched those of previous simulations, confirming the system’s consistency as no alterations were made to it.



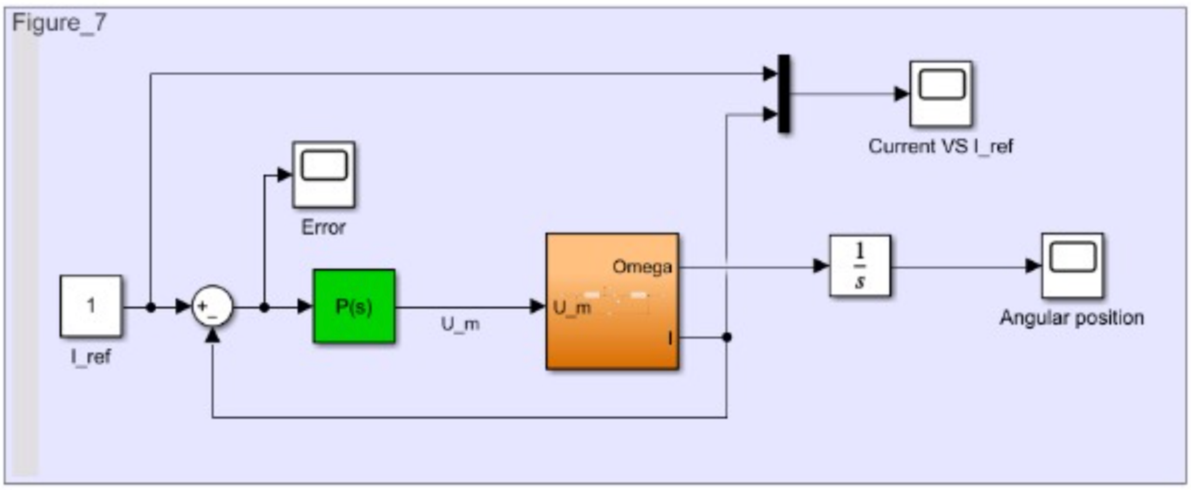
Since the subsystems are a reduced version of the main system, we get the same results in the graphics.



This representation enhances visibility in Simulink, making it easier to manage large systems. You can also access the circuit by double-clicking on the subsystem.

Part 1: Control of the armature current

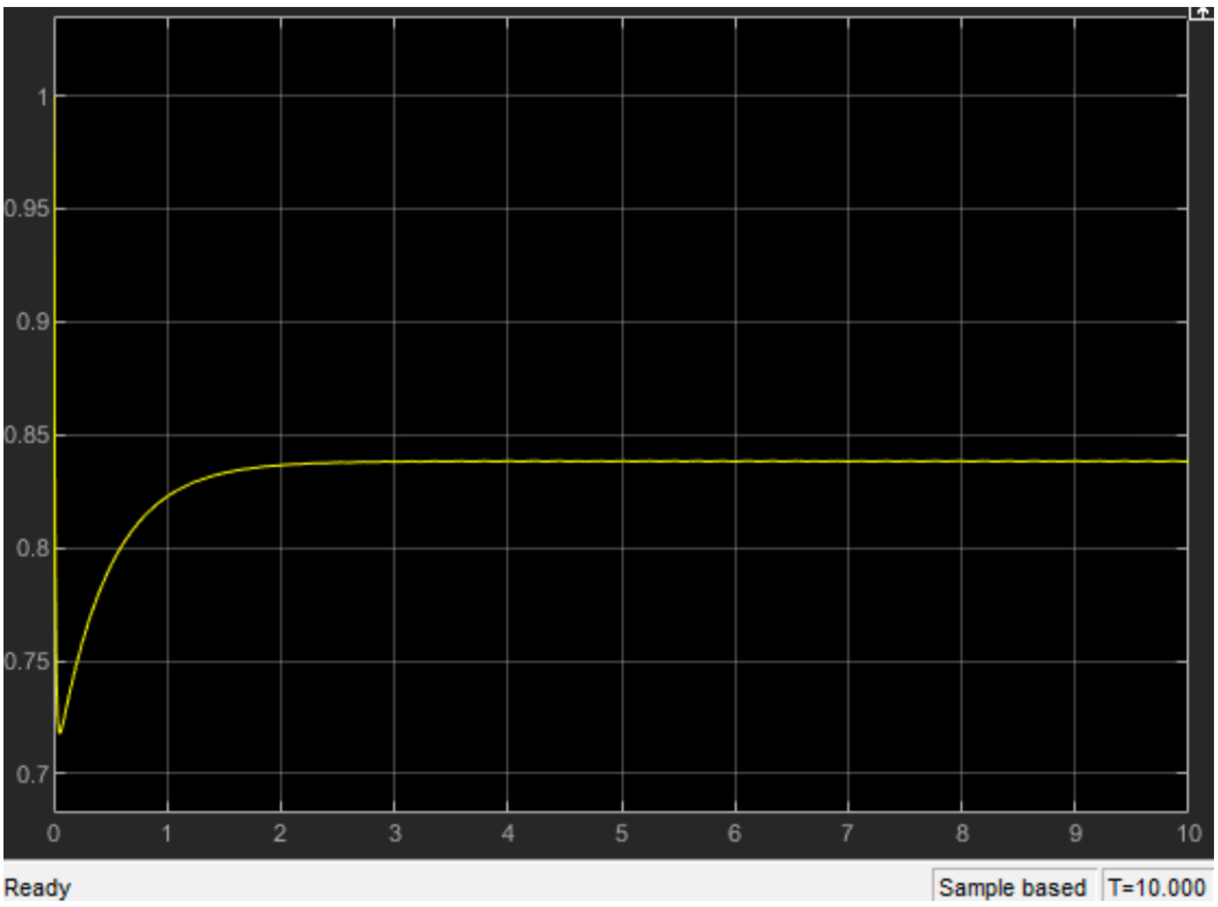
we replicated the specified circuit and focused on controlling the armature current of the DC motor, as outlined in the practical subject.



Block diagram of the DC motor in a closed loop configuration to control the armature current

QUESTION 4:

In our setup, the green block represents a Proportional Controller with a coefficient (K_p) set at 1. We applied a unit step input, where (iref(t) = 1) for (t geq 0) and (iref(t) = 0) for (t < 0), to measure the steady-state error between the current (i) and the reference (iref) graphically.



In the graph shown in the figure, there is a rapid oscillation immediately following the input signal. The steady-state error, ε_s, is depicted as 0.84. However, it spikes to 1 during the transient phase.

The theoretical value of this error, ε_s, slightly differs and is calculated as follows:

$$\epsilon(s) = \lim_{t\to\infty} \epsilon(t)$$

$$\epsilon(s) = \lim_{s\to 0} s\epsilon(s)$$

$$\epsilon(s) = \lim_{s\to 0} s(I_{ref}(s) - I(s))$$

As: $F(s) = \frac{I(s)}{I_{ref}(s)}$ we get:

$$\epsilon(s) = \lim_{s\to 0} s(I_{ref}(s) - I_{ref}(s)F(s))$$

$$\epsilon(s) = \lim_{s\to 0} sI_{ref}(1 - F(s)) \to \lim_{s\to 0} 1 - F(s)$$

$$\epsilon(s) = \lim_{s\to 0} 1 - \frac{I(s)}{I_{ref}(s)}$$

$$\epsilon(s) = \lim_{s\to 0} 1 - sI(s)$$

$$\epsilon(s) = 1$$

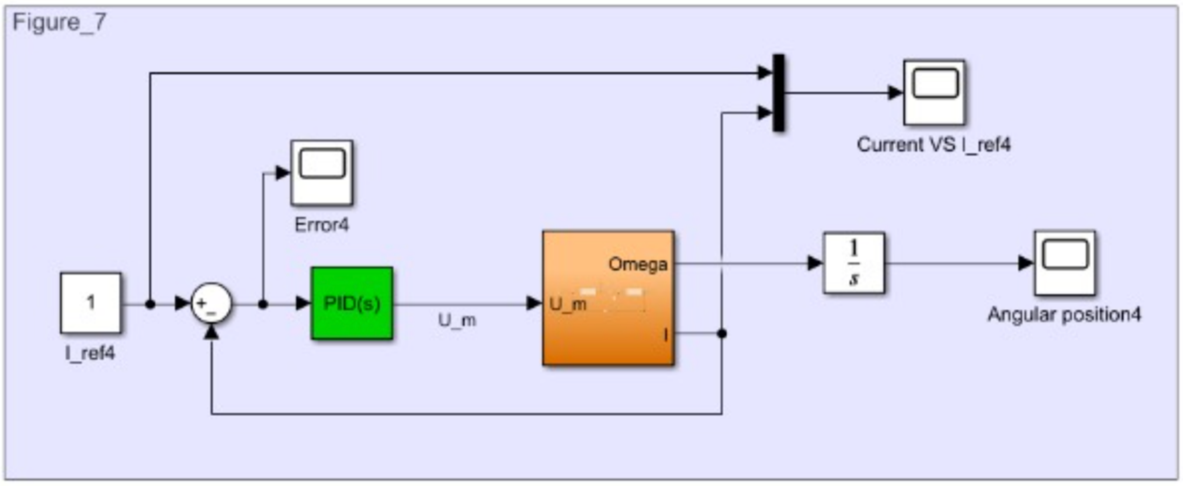
Since we used calculus involving limits, obtaining an exact result is challenging. Nevertheless, 1 is close to 0.84.

We applied a coefficient Kp=18to the proportional controller to enable the current i*i* to closely match the reference iref*iref* with a rapid response time, achieving an output of 0.86Awhich is near the target value of 1A but with a residual error of 0.2 This setup was chosen after considering the limitations of the “tune” option, which was non-functional on our system. Despite a higher Kp value like 100 potentially offering better precision, the cost and practicality of implementation led us to deem 18as sufficient for our needs. We observed that the first output plot reached a steady state around 3 seconds, but with a significantly lower amplitude than expected. However, increasing Kp to 18improved amplitude close to 0.86A, three times what was initially observed without significantly affecting the time to reach steady state.

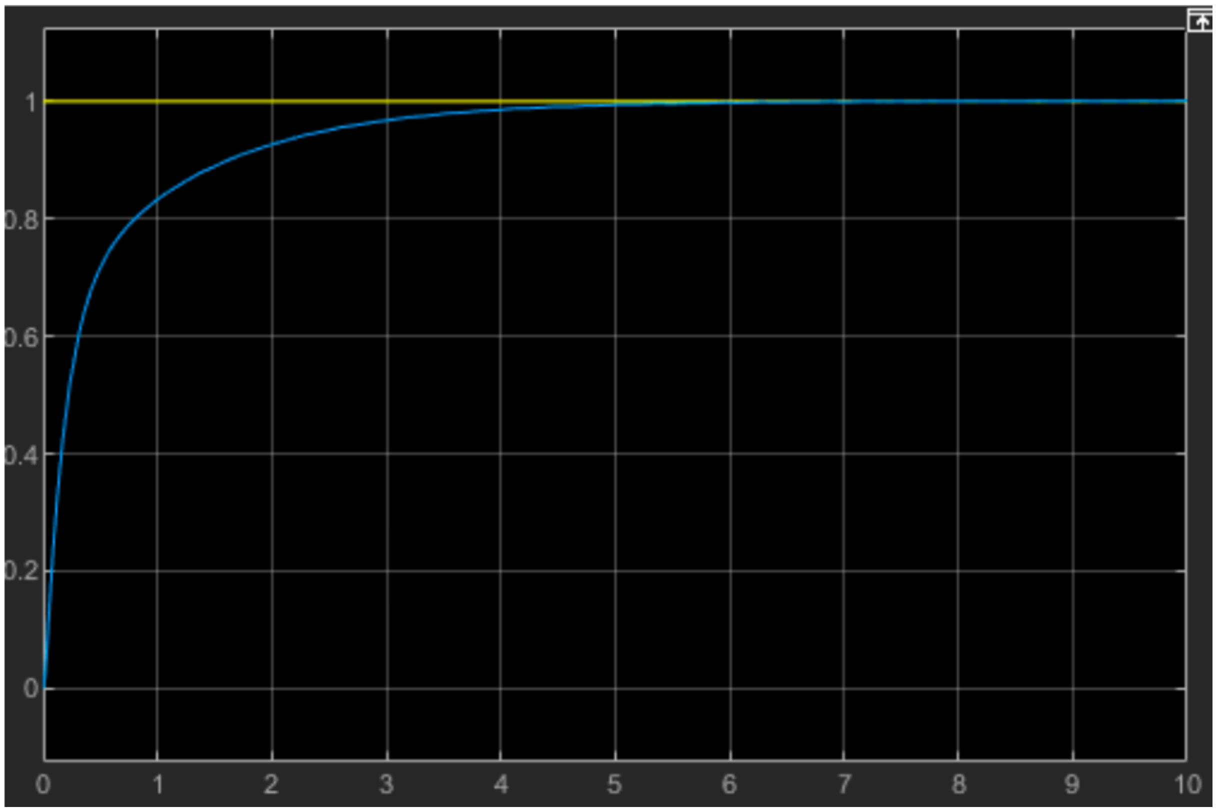
The main limitation of proportional control is its inability to completely eliminate the steady-state error, as illustrated by the residual SP−PV error where SP (set point) is the target value 11 and PV(process variable) is the measured 0.86 To address these deficiencies, alternatives such as integral control, which integrates the error over time, and derivative control, which anticipates future trends, are considered. Moreover, a PID controller, combining proportional, integral, and derivative controls, could further minimize steady-state errors and enhance system stability by adjusting not just Kp but also Ki and Kd to optimize the overall system response according to our specific requirements. This approach balances affordability with the quality of output in our control system configuration.

QUESTION 7:

Our objective is to control the DC armature current using various configurations of the PID controller, as illustrated in the recreated block diagram from Figure 8 of the subject. We systematically substituted the proportional controller with other proposed controllers including I, PI, PD, and PID within the block diagram. Our aim throughout these modifications is to achieve a target armature current I(s)of 1 by adjusting the values of these controllers. This methodical approach allows us to evaluate the effectiveness of each controller configuration in maintaining the desired current level.

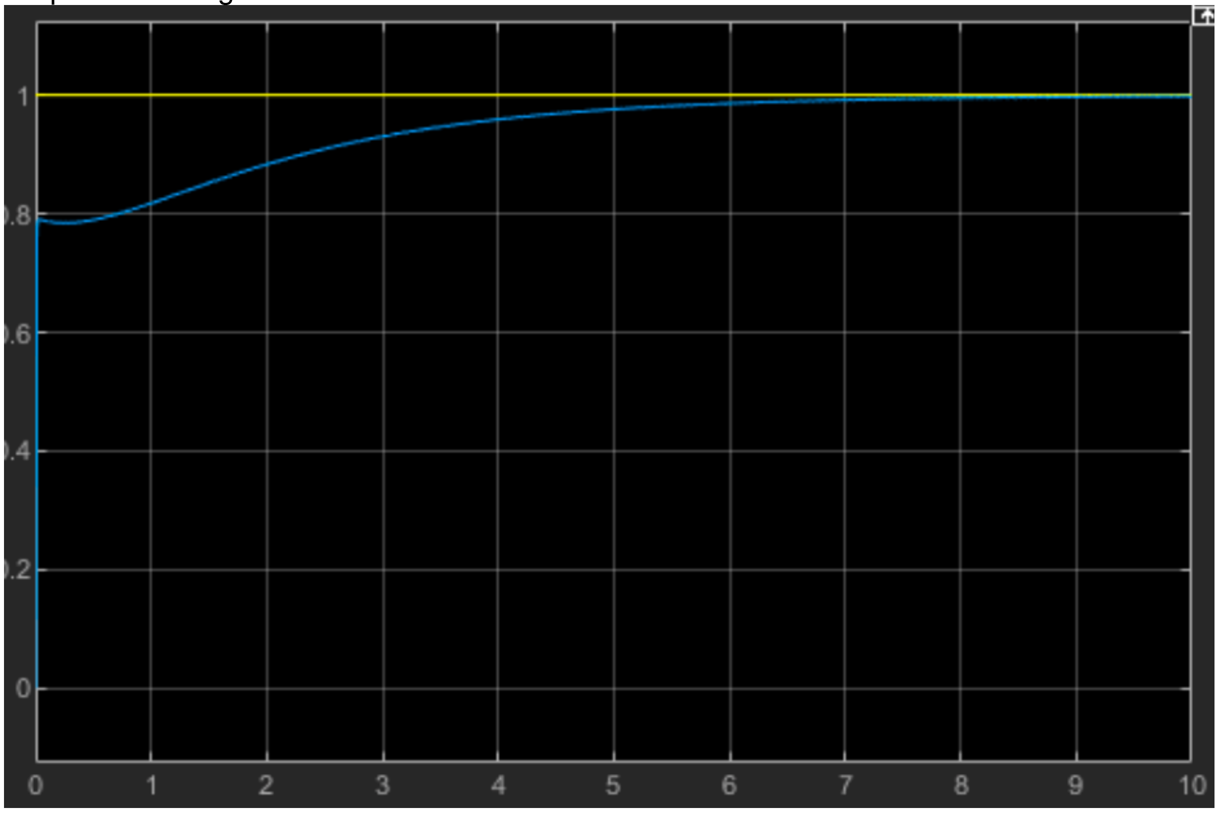


• Integral controller: for I = 9 we get:



The integral controller effectively eliminates steady-state error, ensuring that the measured value matches the target exactly. However, this accuracy comes with a trade-off as it slightly increases overshoot, which may compromise the stability of the system.

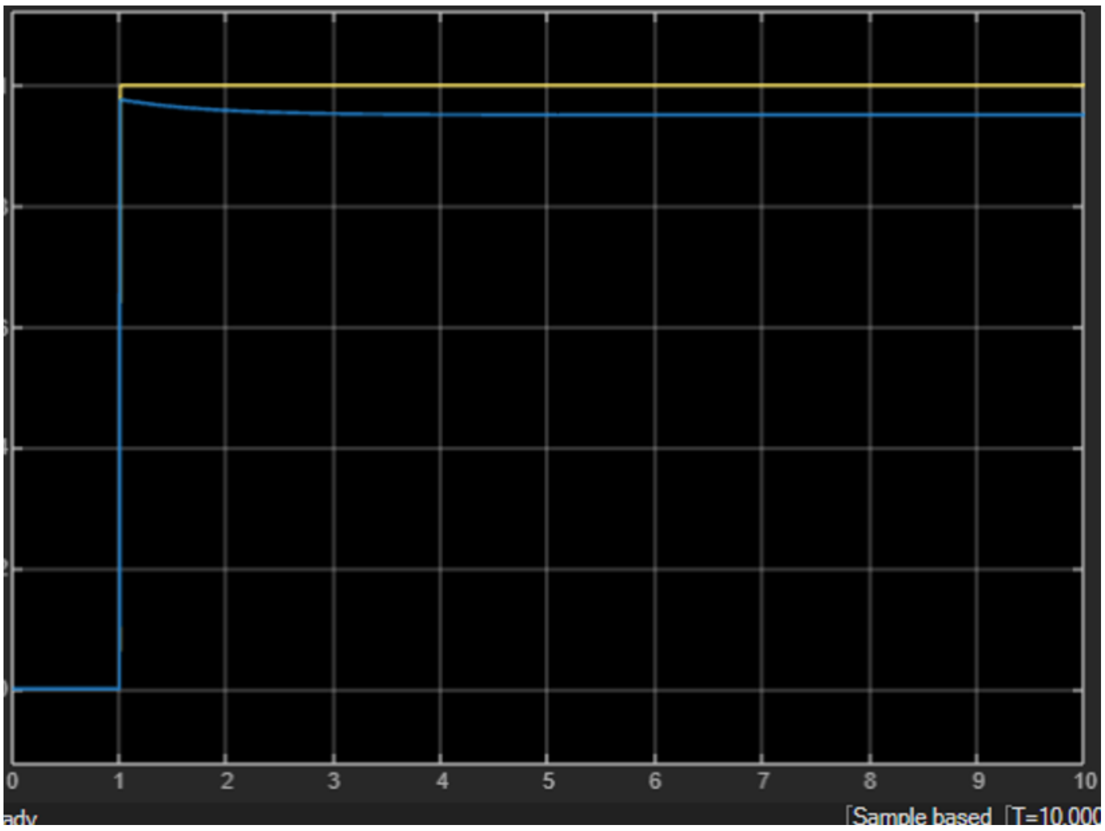
Proportional integral: for P = 9 & I = 9:



The Proportional Integral controller effectively reduces the steady-state error to zero, offering improved stability and less maximum peak overshoot compared to an integral-only controller. However, it exhibits a slow response to disturbances, highlighting a trade-off between precision and reaction speed.

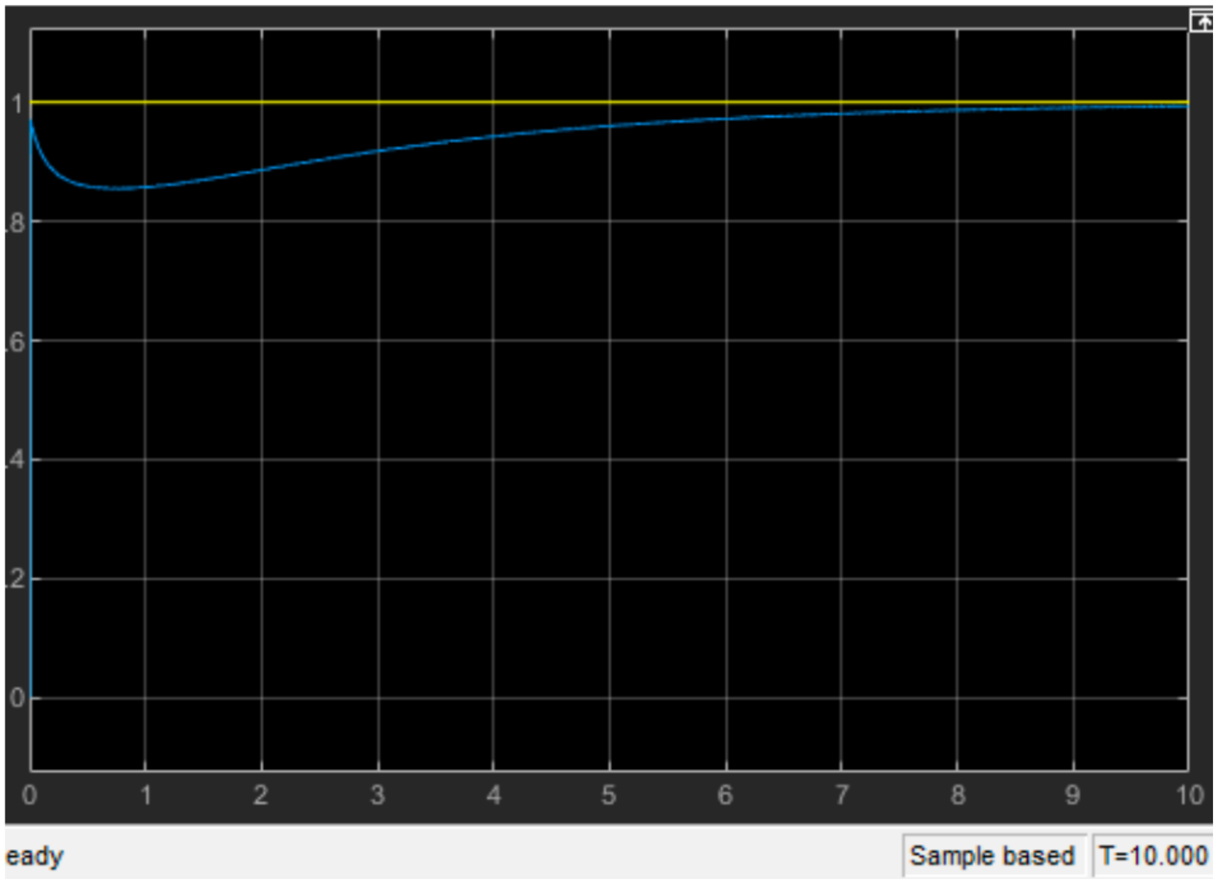
Proportional Derivative (PD) P = 100, D = 0,

-



The Proportional Derivative (PD) controller, known for its ability to stabilize the system and effectively handle processes with time lags, requires a high proportional value (P) around 100 to achieve an ideal response, which is not practical for all applications. Despite these benefits, the derivative component (D) does not influence the response curve, and the controller fails to eliminate steady-state error, maintaining an error of approximately 0.025. Given these limitations, exploring alternative controllers might be beneficial.

Proportional Integral Derivative: for P = 18 & I = 9 & D = 2:

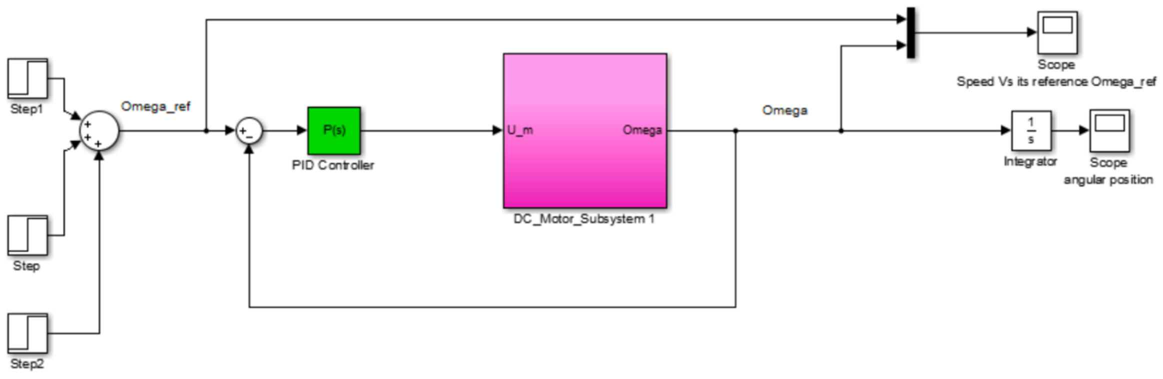


Advantages: The PID controller removes the offset found in proportional mode and responds quickly to disturbances.

Disadvantages: The final results for the reference and the measurement align at the end of the curve. However, the gap between 0 and approximately 4 could not be eliminated.

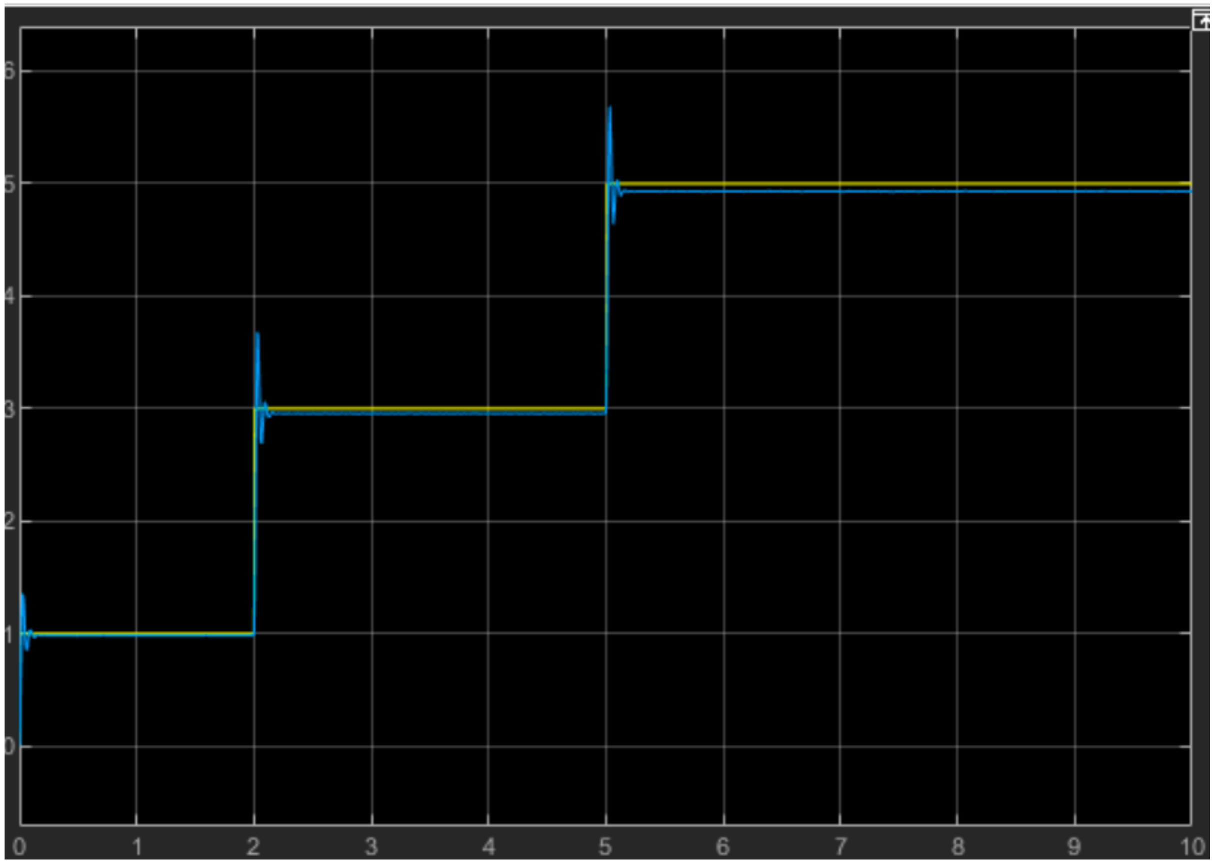
Part2: Control of the angular velocity

In this subsequent section, the aim is to regulate the angular velocity of the DC motor.



Control of the angular velocity with a variable input reference

QUESTION 8:

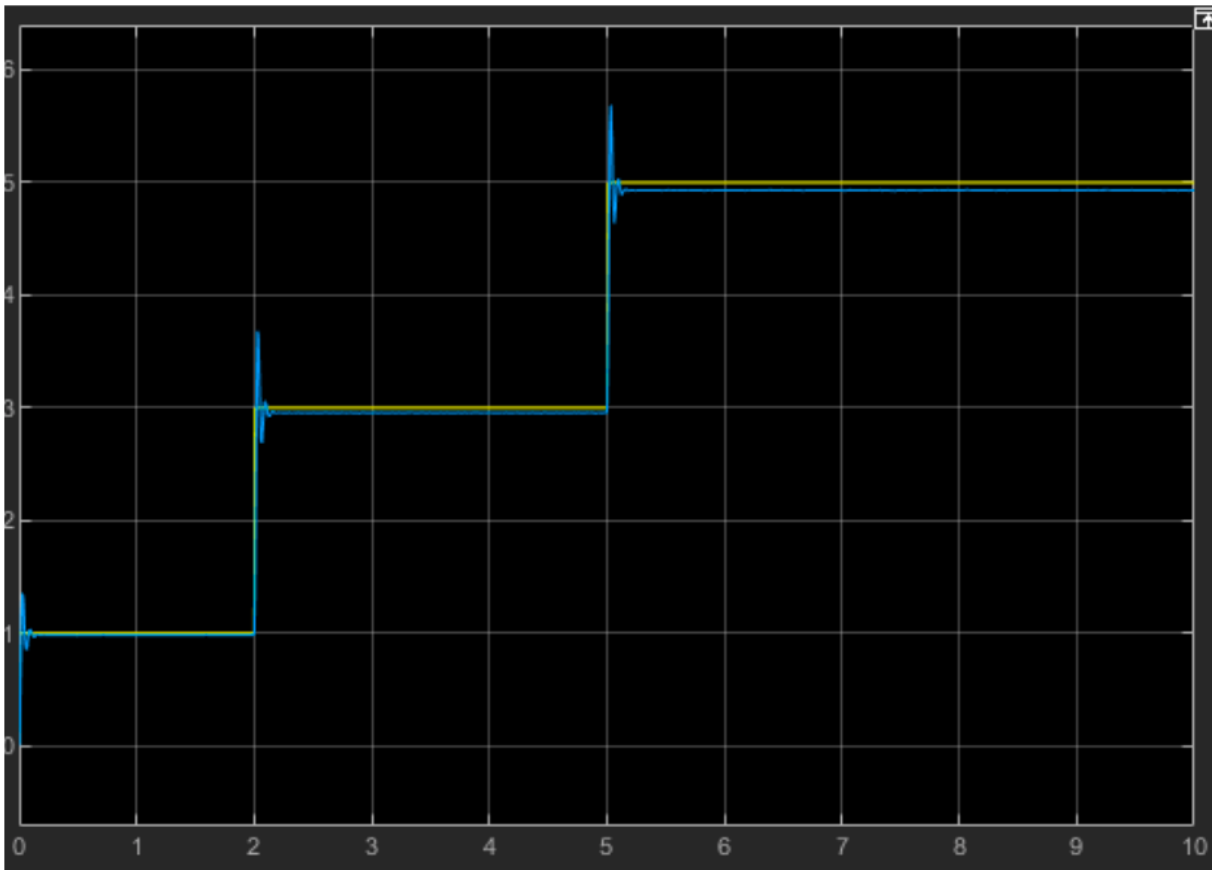


In our system utilizing a Proportional Controller, the steady-state angular velocity is achieved at around 5 seconds, coinciding with the timing of the last of three step inputs, each varying in parameters. This is expected, as each additional input amplifies the error—typical behavior for a Proportional Controller, which sacrifices stability for quicker response times. The final experimental value of angular velocity reaches approximately 4.95 rad/s, slightly under the theoretical target of 5 rad/s, illustrating a minor 1% discrepancy. Notably, this system exhibits significant overshoots with each step input, with a peak overshoot reaching more than 5.6 rad/s, which is about 12% above the reference value of 5 rad/s derived from the sum of the step inputs. These overshoots are accompanied by visible oscillations, further demonstrating the system's reduced stability under the influence of a Proportional Controller.

QUESTION 9:

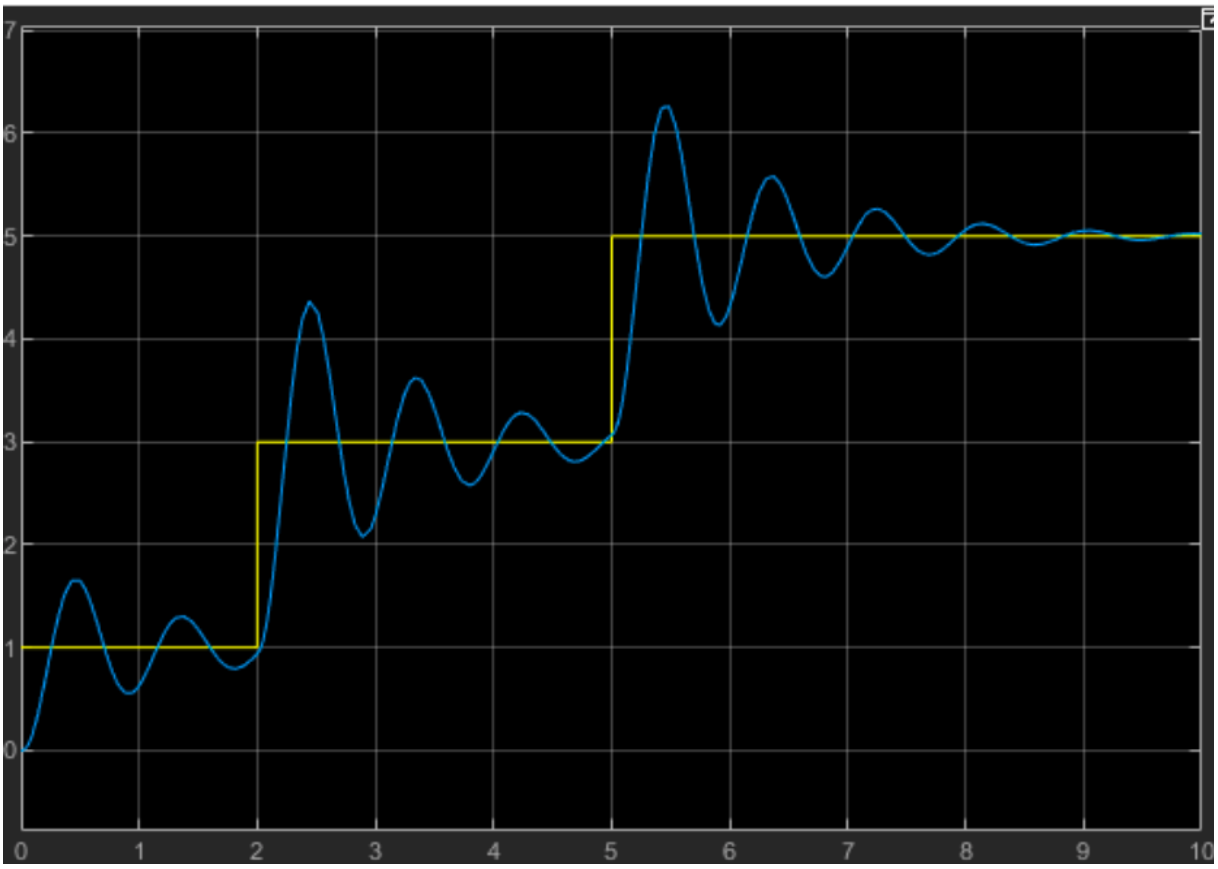
In our experiment, we employed the angular velocity control block diagram, systematically replacing the proportional controller with various alternatives such as I, PI, PD, and PID controllers. Our objective throughout this process was to fine-tune the parameters of these controllers to achieve a target control signal value, (I(s)), of 1.

• Proportional integral: P = 18 we get:



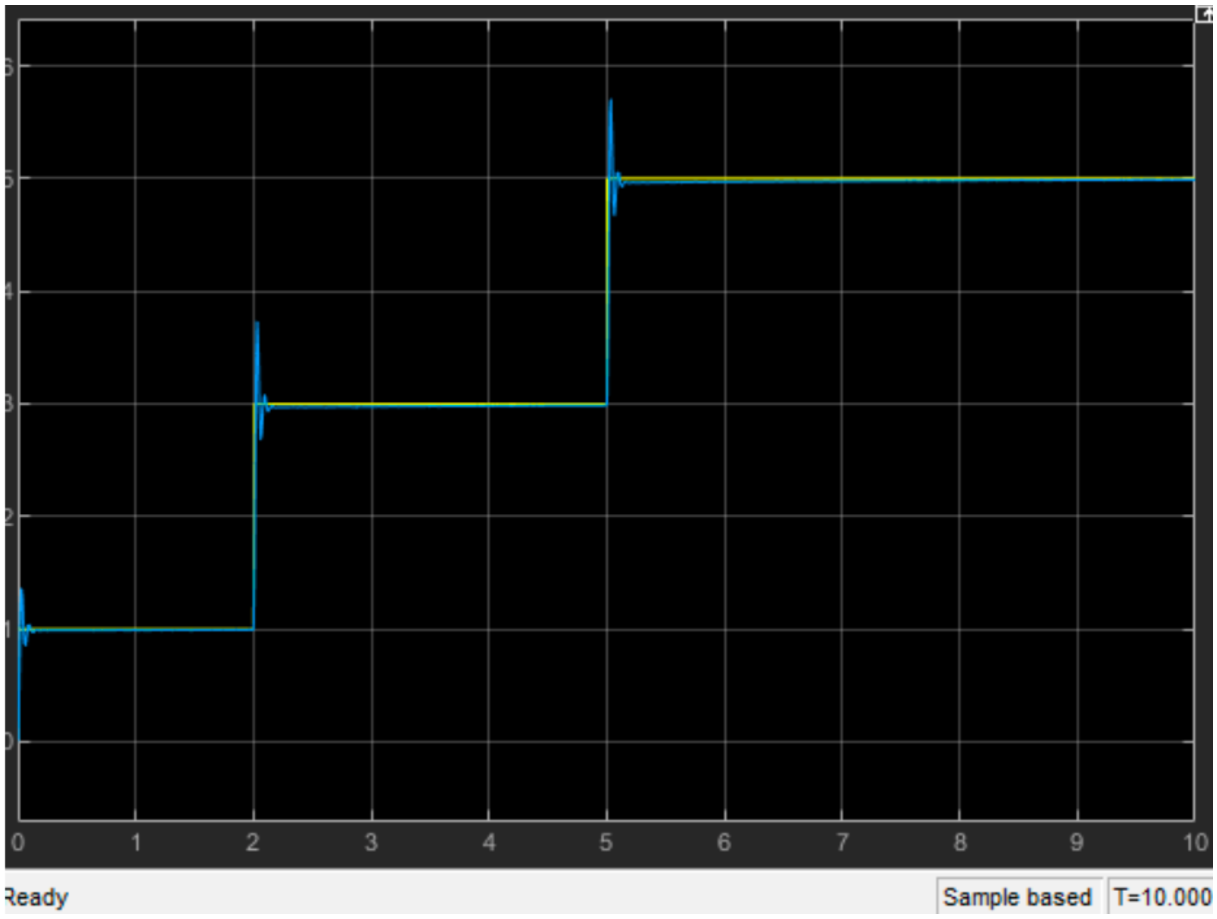
Despite achieving identical values for the target and measured values, overshoot persists at each step.

• Integral controller: I = 5 we get:



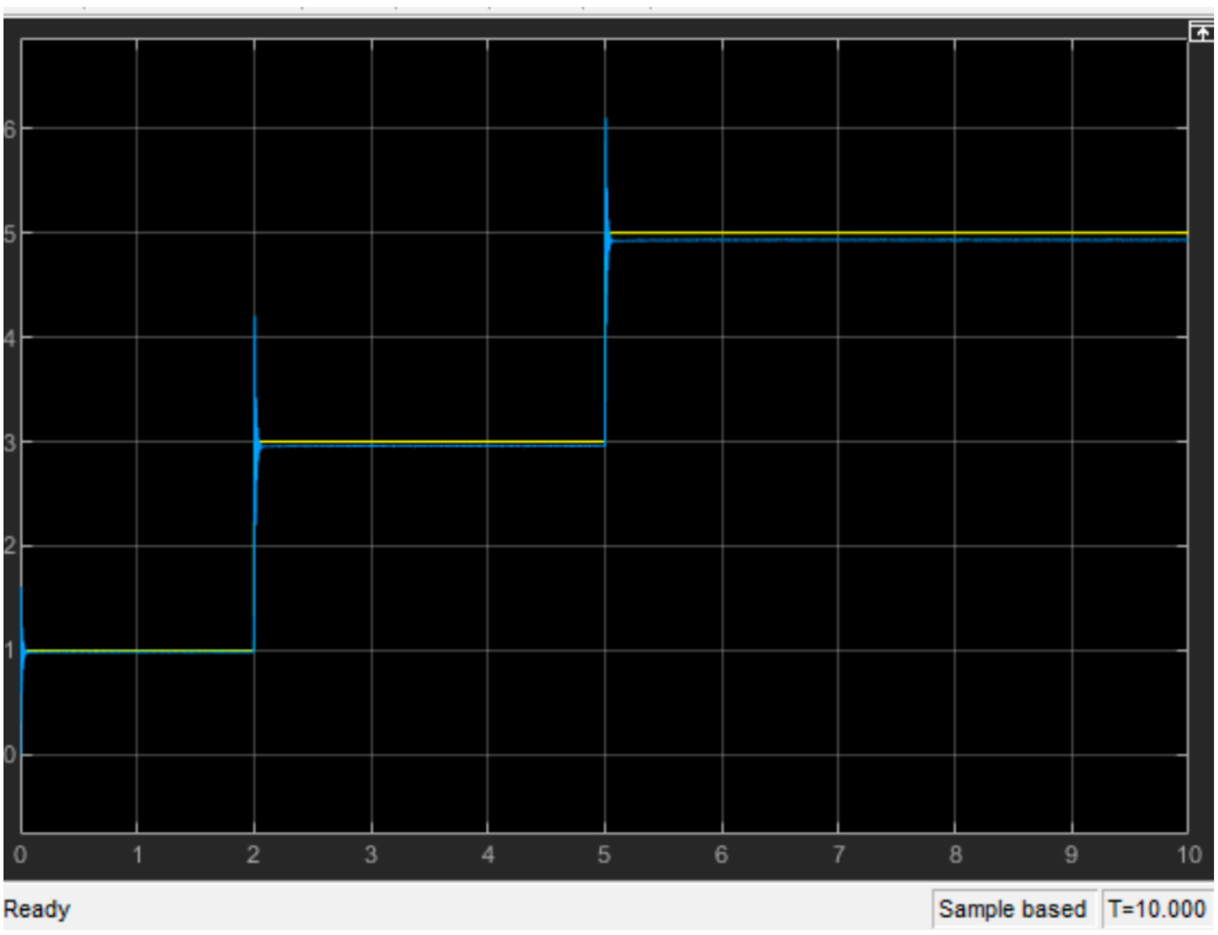
During our experiments, we observed that the measured value oscillates significantly with each new step, and these oscillations increase as the value of (I) rises. Despite this great overshoot and persistent fluctuations, the final measurement approximates the target value, although no fixed value is reached to halt the oscillations.

Proportional integral: P = 18 & I = 5:



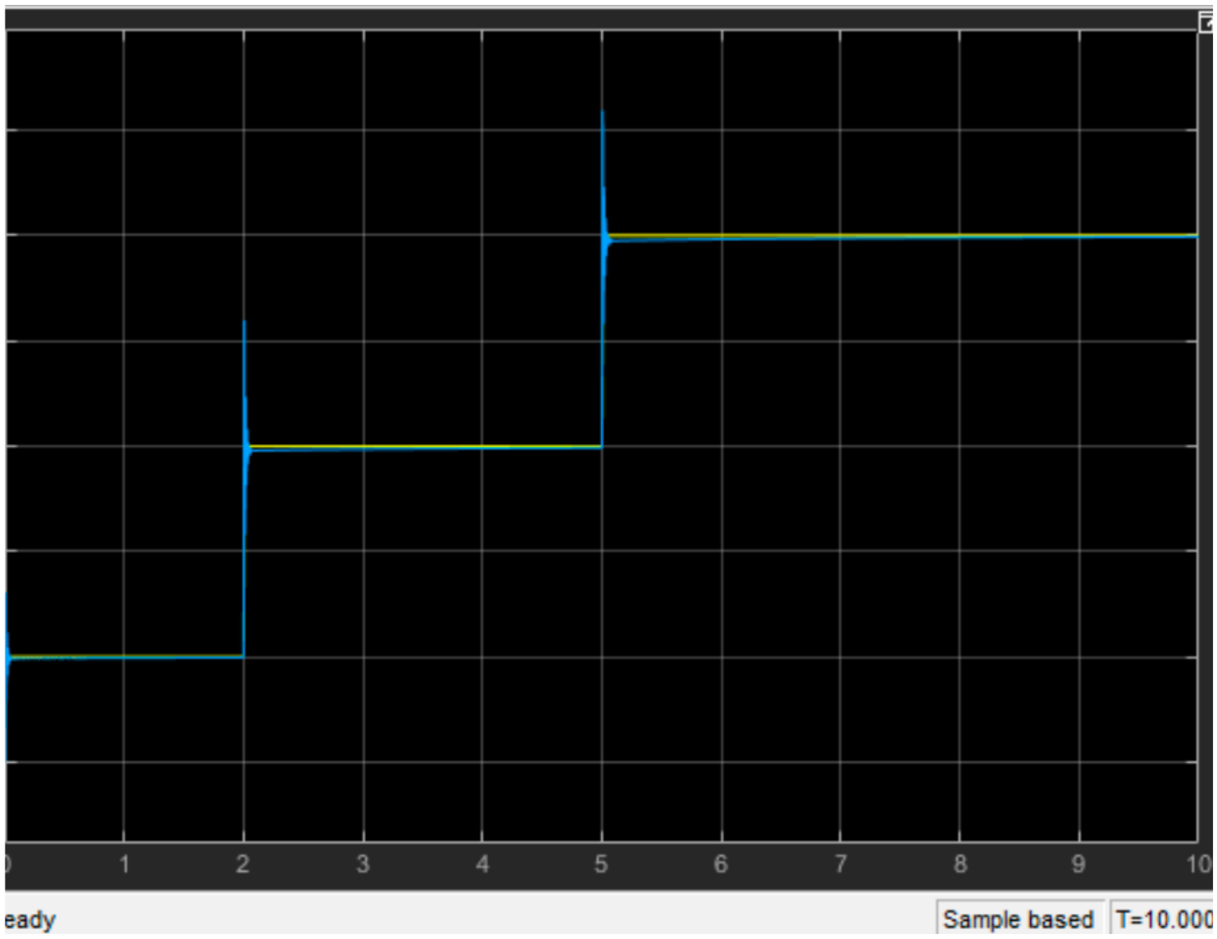
The proportional integral controller exhibits overshoot with each new step in the process, where the graph first overshoots, then dips below before stabilizing in sync with the target level.

• Proportional derivative: P = 18 & D = 2:



This graph exhibits overshoot followed by stabilization just little below the curve with each step.

Proportional Integral Derivative: P = 18 & I = 5 & D = 5:



Despite observing overshoot at each step, the measured values ultimately align with the target values.

In short, our study of the DC motor highlighted challenges in minimizing steady-state error across different controllers. While aiming for precise parameter selection, discrepancies persisted between theoretical and actual armature current values. Despite achieving parity in angular velocities, overshoots and oscillations persisted, indicating ongoing system imperfections.

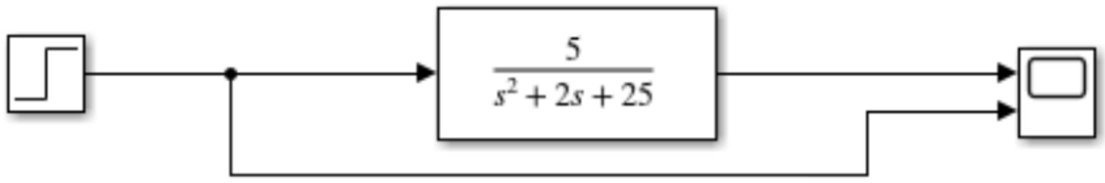
Task 2:

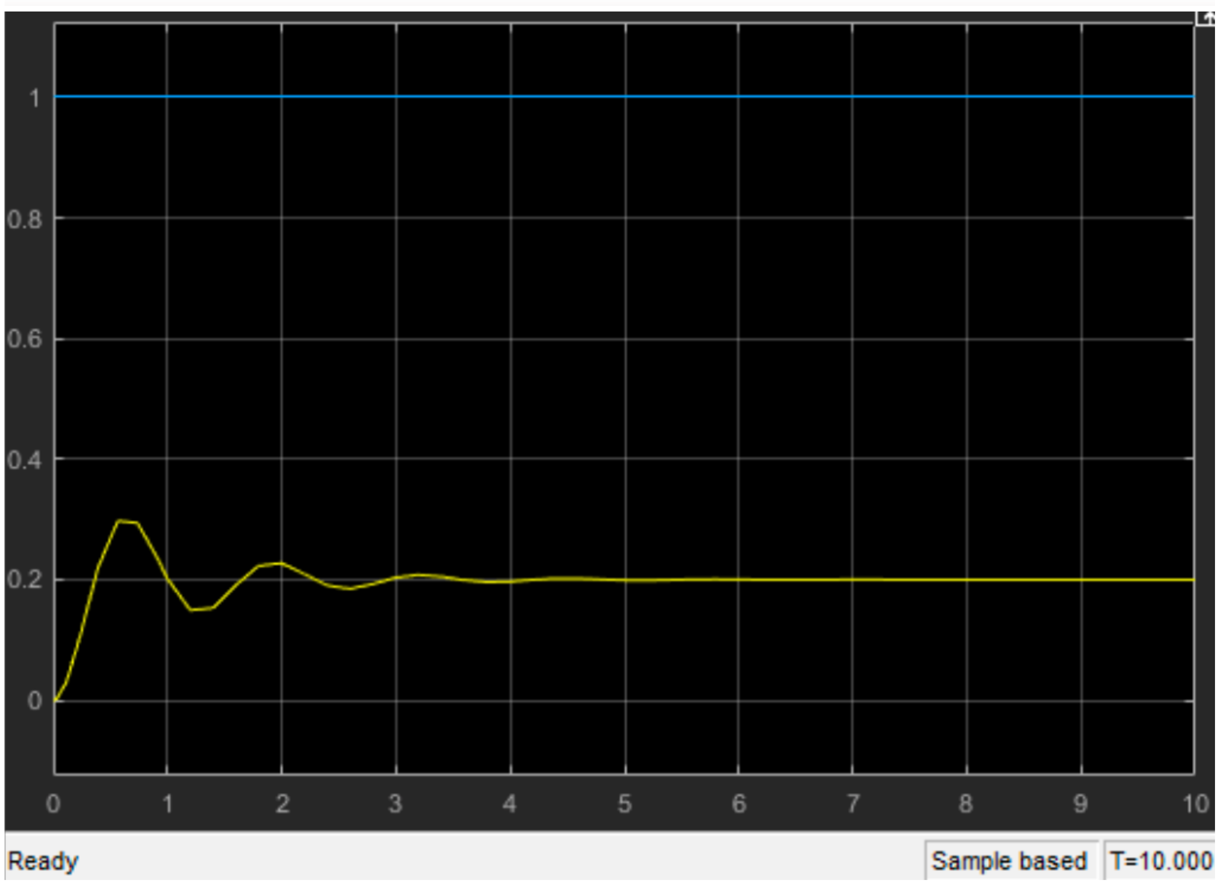
$$G(s) = \frac{5}{s^2 + 2s + 25}$$

our transfer function corresponds to a second order system .

So we can use canonical form to describe it .
we obtain :

- A Gain of= $\frac{1}{5}$
- A damping ratio $\xi = \frac{1}{5}$
- An Undamped natural (resonant) frequency $\omega_0 = 5$ rad/s





In our system, the steady-state value measured is 0.2, which confirms the steady-state error of 0.8 (1 - 0.2 = 0.8). This error, coupled with the characteristic oscillations observed in the system, indicates that we are dealing with an underdamped system.

Task 3

To solve Task 3, we need to develop a MATLAB script to find the value of k*k* that ensures the percent overshoot to a unit step input is between 1% and 10%. We will also verify that the steady-state error to a unit step input is zero.

$$T(s) = \frac{Y(s)}{R(s)}$$

Given the controller $\frac{10}{s}$ and the process $\frac{1}{s+k}$, the open-loop transfer function is:

$$G(s) = \frac{10}{s} \cdot \frac{1}{s+k} = \frac{10}{s(s+k)}$$

The closed-loop transfer function $T(s)$ is:

$$T(s) = \frac{G(s)}{1+G(s)} = \frac{\frac{10}{s(s+k)}}{1+\frac{10}{s(s+k)}} = \frac{10}{s(s+k)+10}$$

Matlab Scripte

```
% Define the range of k values to search

k_values = linspace(0.1, 10, 1000); % Adjust the range and resolution as needed

overshoot_target_range = [1, 10]; % Percent overshoot target range


% Initialize variables to store results

best_k = NaN;

best_overshoot = Inf;


% Loop through k values to find the best one

for k = k_values

  % Define the transfer function

  num = 10;

  den = [1, k, 10];

  sys = tf(num, den);


  % Compute the step response

  [y, t] = step(sys);
```

```
    % Compute percent overshoot
    S = stepinfo(sys);
    overshoot = S.Overshoot;

    % Check if overshoot is within the target range and is the best so far
    if overshoot >= overshoot_target_range(1) && overshoot <= 
overshoot_target_range(2) && overshoot < best_overshoot
        best_k = k;
        best_overshoot = overshoot;
    end
end

% Display the best k value and corresponding overshoot
fprintf('Best k: %.4f\n', best_k);
fprintf('Corresponding Percent Overshoot: %.2f%%\n', best_overshoot);

% Plot the step response for the best k value
if ~isnan(best_k)
    num = 10;
    den = [1, best_k, 10];
    best_sys = tf(num, den);
    step(best_sys);
    title(sprintf('Step Response for k = %.4f', best_k));
end

% Verify that the steady-state error is zero
% Since the system type is 1 (integrator in the loop), the steady-state error for a 
step input is zero.
```

The script optimizes the gain kk for a control system by:

1. **Searching** k: In a specified range and resolution.
2. **Defining** T(s)T: The transfer function.
3. **Computing Step Response**: Using MATLAB's **step** function.
4. **Calculating Overshoot**: With **stepinfo**.
5. **Finding Optimal** k: For 1%-10% overshoot.
6. **Plotting**: The step response for the best kk.
7. **Steady-State Error**: Assumes zero due to system type 1.

Task 4:

To address Task 4, we will consider two scenarios for the control system depicted in Figure 14:

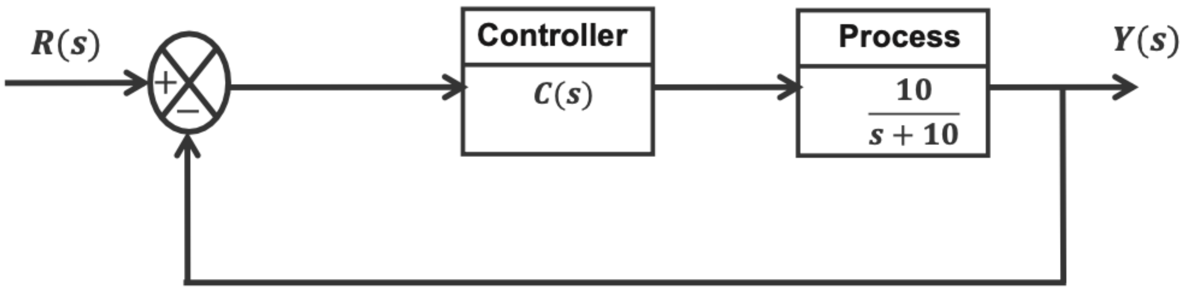


Figure 14: control system

Using a simple proportional controller $C(s) = K$ with $K = 2$.

Using a more complex Proportional-Integral (PI) controller $C(s) = K_0 + \frac{K_1}{s}$ with $K_0 = 2$ and $K_1 = 20$.

We will plot the unit step response for each controller and determine the steady-state error from the plots. Finally, we will compare the results to discuss the trade-off between controller complexity and steady-state tracking error performance.

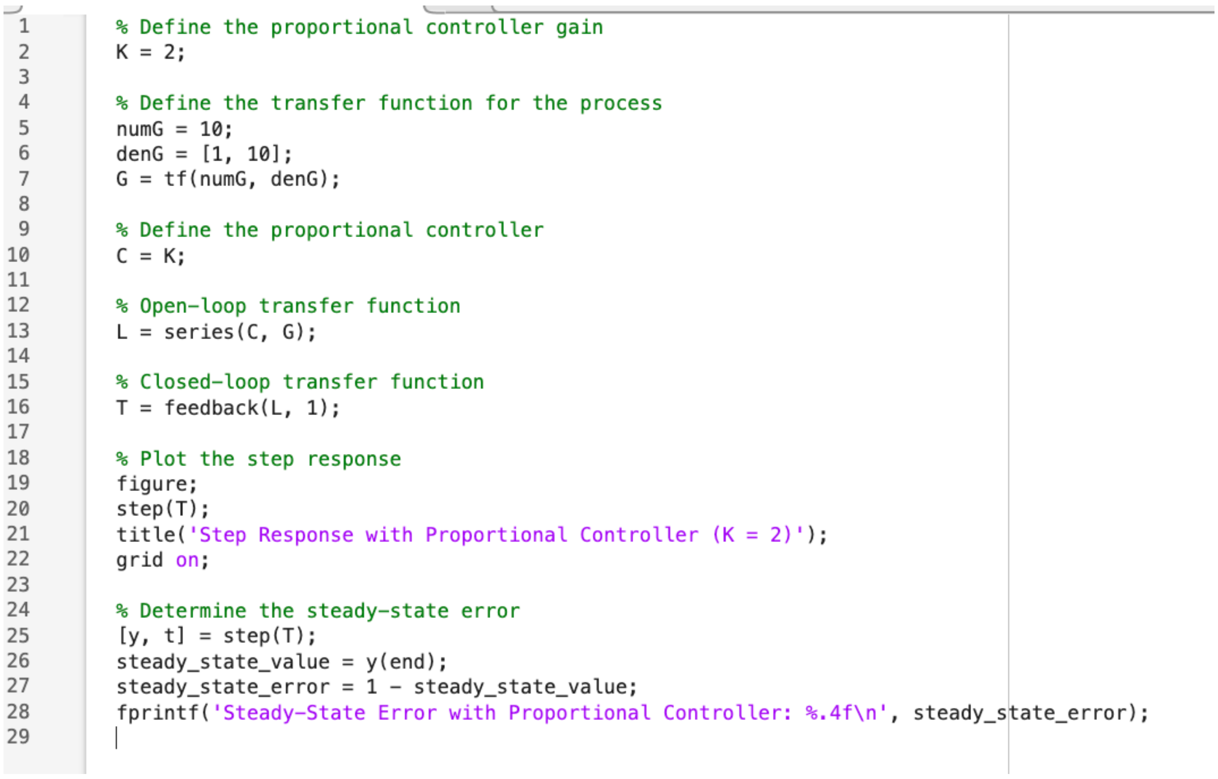
$$G(s) = \frac{10}{s+10}$$

With the proportional controller $C(s) = K$, the open-loop transfer function becomes:

$$L(s) = K \cdot G(s) = \frac{10K}{s+10}$$

The closed-loop transfer function is:

$$T(s) = \frac{L(s)}{1+L(s)} = \frac{\frac{10K}{s+10}}{1+\frac{10K}{s+10}} = \frac{10K}{s+10+10K}$$



```
% Define the proportional controller gain
K = 2;

% Define the transfer function for the process
numG = 10;
denG = [1, 10];
G = tf(numG, denG);

% Define the proportional controller
C = K;

% Open-loop transfer function
L = series(C, G);

% Closed-loop transfer function
T = feedback(L, 1);

% Plot the step response
figure;
step(T);
title('Step Response with Proportional Controller (K = 2)');
grid on;

% Determine the steady-state error
[y, t] = step(T);
steady_state_value = y(end);
steady_state_error = 1 - steady_state_value;
fprintf('Steady-State Error with Proportional Controller: %.4f\n', steady_state_error);
```

Part 2: Proportional-Integral (PI) Controller

$$C(s) = K_0 + \frac{K_1}{s}$$

With $K_0 = 2$ and $K_1 = 20$, the PI controller becomes:

$$C(s) = 2 + \frac{20}{s} = \frac{2s+20}{s}$$

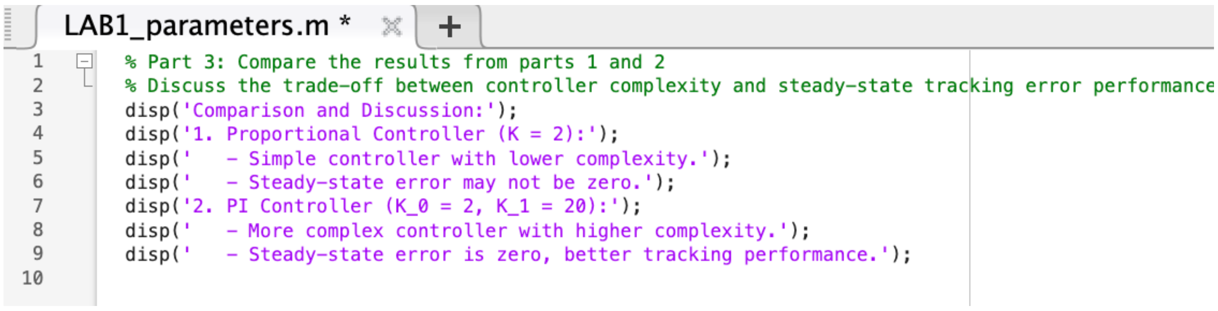
Closed-Loop Transfer Function:

$$L(s) = C(s) \cdot G(s) = \frac{(2s+20)\cdot 10}{s(s+10)} = \frac{20s+200}{s(s+10)}$$

The closed-loop transfer function is:

$$T(s) = \frac{L(s)}{1+L(s)} = \frac{\frac{20s+200}{s(s+10)}}{1+\frac{20s+200}{s(s+10)}} = \frac{20s+200}{s^2+30s+200}$$

Part 3: Comparison and Discussion


LAB1_parameters.m *

```
% Part 3: Compare the results from parts 1 and 2
% Discuss the trade-off between controller complexity and steady-state tracking error performance
disp('Comparison and Discussion:');
disp('1. Proportional Controller (K = 2):');
disp('    - Simple controller with lower complexity.');
disp('    - Steady-state error may not be zero.');
disp('2. PI Controller (K_0 = 2, K_1 = 20):');
disp('    - More complex controller with higher complexity.');
disp('    - Steady-state error is zero, better tracking performance.');
```