Control Theory

IE2074

[2025/FEB]

Assignment Report

Submitted to Sri Lanka Institute of Information Technology (SLIIT)

BSc (Hons)Computer Systems Engineering

Details of the student:

Student ID- IT23389724

Student Name – D R L Amarasekara

Contents

1. Introduction
2. Modeling the DC Motor
   2.1. Derivation of the Transfer Function
   2.2. State-Space Model
   2.3. MATLAB Simulation
3. Design of PID Controller
   3.1. Auto-Tuning
   3.2. Manual Tuning for Optimized Performance
4. Simulation of Step Response
   Plot 1: Step Response of the P Controller
   Plot 2: Step Response of the PI Controller
   Plot 3: Step Response of the Optimized PID Controller
5. Disturbance Test
   Plot 4: P Controller Response with Disturbance
   Plot 5: PI Controller Response with Disturbance
   Plot 6: PID Controller Response with Disturbance
6. Controller Comparison
   Plot 7: Step Response Comparison (No Disturbance)
   Plot 8: Response Comparison with Disturbance
7. Conclusion

1. Introduction

DC motors are essential in automated systems and require precise control for tasks like robotics and industrial automation. To achieve specific movements, like a 90-degree rotation, effective control strategies are needed to handle various challenges. Open-loop control often lacks the necessary precision, while closed-loop control with feedback greatly enhances performance.

A common solution is the Proportional-Integral-Derivative (PID) controller, popular in industrial settings for its ability to quickly and accurately manage motor position. It adjusts control based on errors between desired and actual positions, helping a DC motor reach its target angle quickly while minimizing fluctuations and maintaining stability against disturbances.

The goal of this assignment is to design and simulate a PID controller in MATLAB to control a DC motor's position from 0° to 90°, aiming for minimal overshoot, fast settling time, and no steady-state error. The PID controller's performance will be compared to simpler Proportional (P) and Proportional-Integral (PI) controllers, and the system's robustness will be tested against external disturbances. The report will include the modeling process, design, simulation results, and a comparison of the three types of controllers, along with supporting MATLAB simulations and performance metrics. .

2. Modeling the DC Motor

2.1. Derivation of the Transfer Function

The DC motor system is described by its electrical and mechanical behavior. Here are the key parameters;

- Moment of inertia (J): 0.01 kg·m²
- Damping coefficient (b): 0.1 N·m·s
- Torque and back EMF constant ((K)): 0.01
- Armature resistance (R): 1 Ω
- Armature inductance (L): 0.5 H

Electrical Equation

Applying Kirchhoff's Voltage Law (KVL) to the armature circuit

$V(t) = R\, i(t) + L\, (di(t))/(dt) + e(t)$

where:

- V(t): Applied voltage (V)
- i(t): Armature current (A)
- e(t): Back EMF, given by (e(t) = K \omega(t))
- ω(t): Angular velocity (rad/s)

Substituting (e(t) = K \ω(t)):

V(t) = R i(t) + L (di(t))/(dt) + K ω(t)

Mechanical Equation

Using Newton's second law for rotation.

T(t) = J (dω(t))/(dt) + b ω(t)

Where;

- T(t): Torque generated by the motor (N·m)
- (J): Moment of inertia
- (b): Damping coefficient

Torque-Current Relationship

The torque is proportional to the armature current.

T(t) = K i(t)

Taking the Laplace transform of equations (1), (2), and (3):

V(s) = (R + L s) I(s) + K Ω(s)

T(s) = (J s + b) Ω(s)

T(s) = K I(s)

To derive the transfer function from input voltage (V(s)) to angular velocity (ω(s)), substitute **(I(s) = T(s) \ K)** from (6) into (4);

V(s) = (R + L s) (T(s))/(K) + K Ω(s)

Substitute **(T(s) = (J s + b) \ ω(s))** from (5):

V(s) = (R + L s) ((J s + b) Ω(s))/(K) + K Ω(s)

K V(s) = (R + L s)(J s + b) Ω(s) + K^2 Ω(s)

K V(s) = [J L s^2 + (J R + L b) s + (b R + K^2)] Ω(s)

Ω(s) / (V(s) = K / (J L s^2 + (J R + L b) s + (b R + K^2))

To obtain the transfer function for angular position **(θ(s))**, use **(ω(s) = s \θ(s))**:

θ(s) / V(s) = K / (s [J L s^2 + (J R + L b) s + (b R + K^2)])

Substituting the given parameters;

(J = 0.01), (b = 0.1), (K = 0.01), (R = 1), (L = 0.5)

$$\mathbf{J\,L = 0.01 \times 0.5 = 0.005}$$

J R + L b = 0.01 × 1 + 0.5 × 0.1 = 0.01 + 0.05 = 0.06

b R + K^2 = 0.1 × 1 + 0.01^2 = 0.1 + 0.0001 = 0.1001

θ(s) / V(s) = 0.01 / (s (0.005 s^2 + 0.06 s + 0.1001))

This transfer function is implemented in MATLAB using the tf function for simulations.

2.2. State-Space Model

The state-space representation provides an alternative framework for analyzing the DC motor system. The state-space form is;

(ẋ) = A x + B u

y = C x + D u

where:

- **(x)**: State vector
- **(u)**: Input (voltage, (V))
- **(y)**: Output (angular position, (\theta))
- **(A, B, C, D)**: State-space matrices

The state variables are:

- **x_1 = θ(t)**: Angular position (rad)
- **x_2 = ω(t) = (dθ(t))/(dt)**: Angular velocity (rad/s)
- **x_3 = i(t)**: Armature current (A)

From the relationship **(dθ)/(dt) = ω**:

(ẋ_1) = x_2

From the mechanical equation (3):

(ẋ_2) = (K)/(J) x_3 - (b)/(J) x_2

From the electrical equation (2):

(ẋ_3) = (1)/(L) u - (R)/(L) x_3 - (K)/(L) x_2

The output is the angular position:

y = x_1

Assembling the state-space matrices:

A = [[0, 1, 0], [0, -(b)/(J), (K)/(J)], [0, -(K)/(L), -(R)/(L)]]

= [[0, 1, 0], [0, -(0.1)/(0.01), (0.01)/(0.01)], [0, -(0.01)/(0.5), -(1)/(0.5)]]

= [[0, 1, 0], [0, -10, 1], [0, -0.02, -2]] (23)

B = [[0], [0], [(1)/(L)]] = [[0], [0], [(1)/(0.5)]] = [[0], [0], [2]] (24)

C = [1, 0, 0] (25)

D = [0] (26)

Thus, the state-space model is:

[[(dθ)/(dt)], [(dω)/(dt)], [(di)/(dt)]] = [[0, 1, 0], [0, -10, 1], [0, -0.02, -2]] [[θ], [ω], [i]] + [[0], [0], [2]] V
y = [1, 0, 0] [[θ], [ω], [i]] + [0] V

2.3. MATLAB Simulation

The transfer function was implemented in MATLAB using the tf function. The numerator and denominator were defined as:

numTF = [K] = [0.01]

denTF = conv([1, 0], [J L, J R + L b, b R + K^2]) = [0.005, 0.06, 0.1001]

motorTF = tf(numTF, denTF)

The state-space model can be created using the ss function for analysis, but transfer functions are mainly used for designing controllers and running simulations because they're easier to work with in classical control methods. The MATLAB scripts simulate the system's step and disturbance responses to help evaluate the controller.

3. Design of PID Controller

The PID controller aims for a quick response when moving from 0° to 90°, with little overshoot, short settling time, and no steady-state error. The output of the controller is;

u(t) = K_p e(t) + K_i ∫ e(t) dt + K_d (de(t))/(dt)

where e(t) = θ_ref(t) - θ(t) is the error between the reference and actual positions.

3.1. Auto-Tuning

MATLAB's pidtune function was used to obtain baseline gains for P, PI, and PID controllers, balancing performance and robustness. The tuned gains are;

P Controller - K_p = 10.1980

PI Controller - K_p = 5.6813, K_i = 0.6306

PID Controller - K_p = 46.8181, K_i = 16.7798, K_d = 18.4477

3.2. Manual Tuning for Optimized Performance

To achieve faster settling time and lower overshoot, the auto-tuned gains were adjusted manually. The effects of each gain are;

- **Proportional Gain ((K_p))**: Increases responsiveness but may cause overshoot.

- **Integral Gain ((K_i))**: Eliminates steady-state error but can increase overshoot and settling time.
- **Derivative Gain ((K_d))**: Reduces overshoot and oscillations by damping the system.

The manually tuned gains are:

P Controller - K_p = 10.1980 × 0.9 = 9.1782

PI Controller -

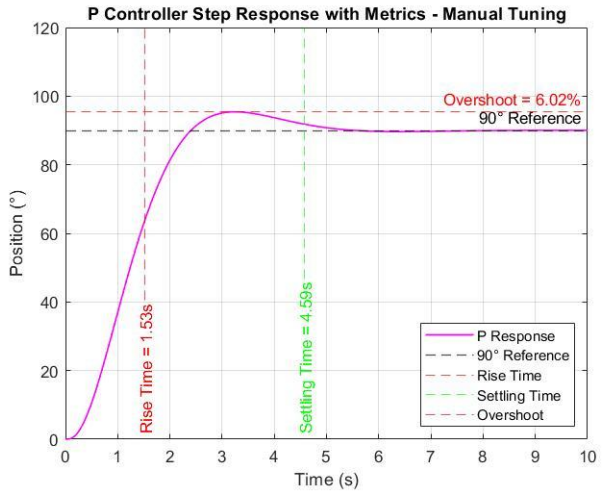
K_p = 5.6813 × 1.85 = 10.5104
K_i = 0.6306 × 1.3 = 0.8198

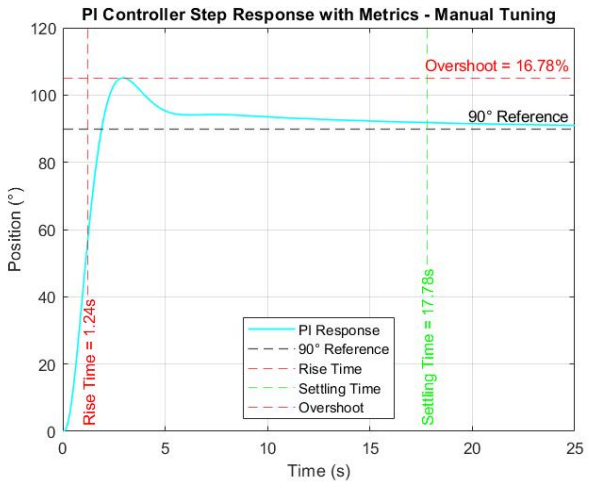
PID Controller -
K_p = 46.8181 × 1.15 = 53.8408

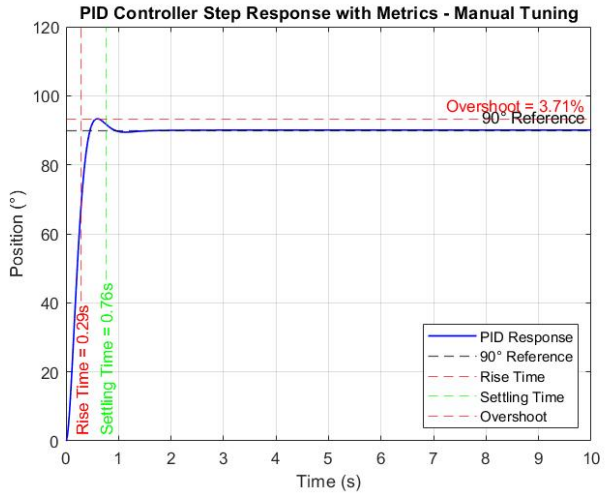
K_i = 16.7798 × 0.01 = 0.1678

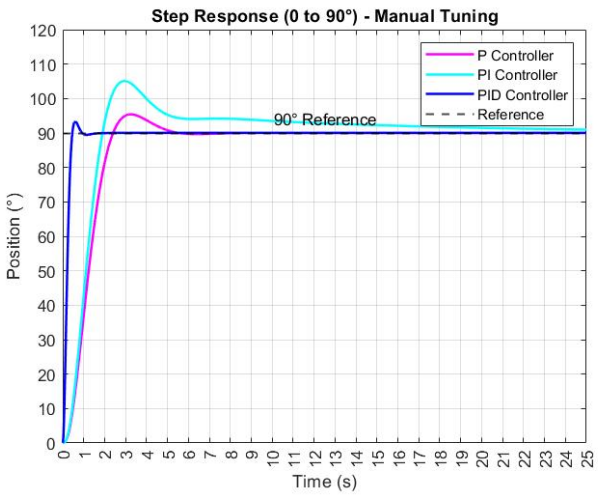
K_d = 18.4477 × 1.4 = 25.8268

These adjustments prioritize faster settling time and reduced overshoot as demonstrated in the simulation results below.





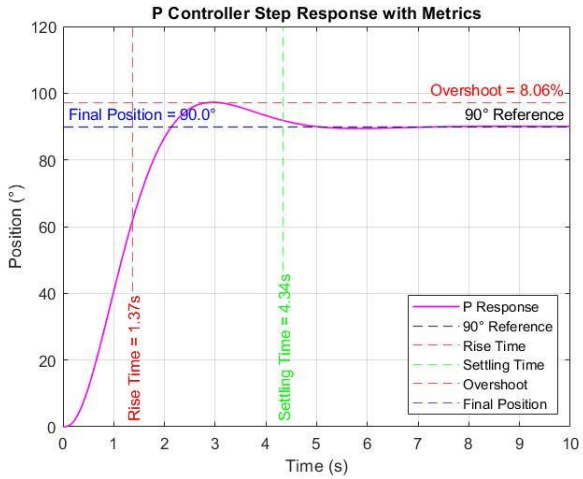


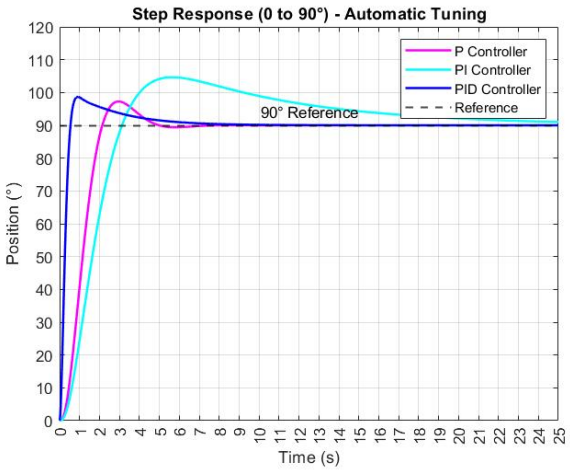


4. Simulation of Step Response

In simulation the step response from 0° to 90° for P, PI, and PID controllers simulated using manual tuning for the gains. And MATLAB's stepinfo function to measure important performance metrics like rise time, overshoot, settling time, and final value.

Plot 1: Step Response of the P Controller

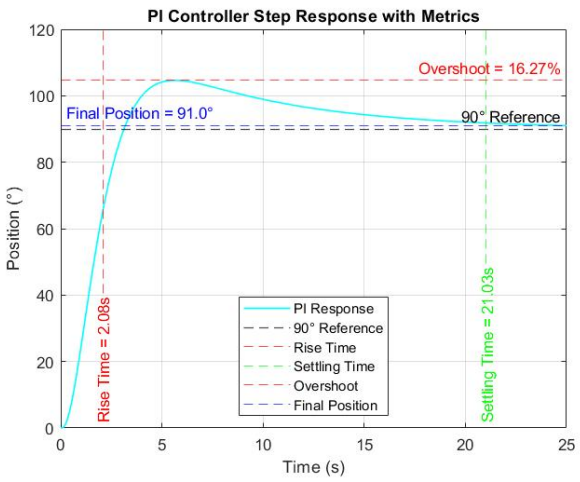


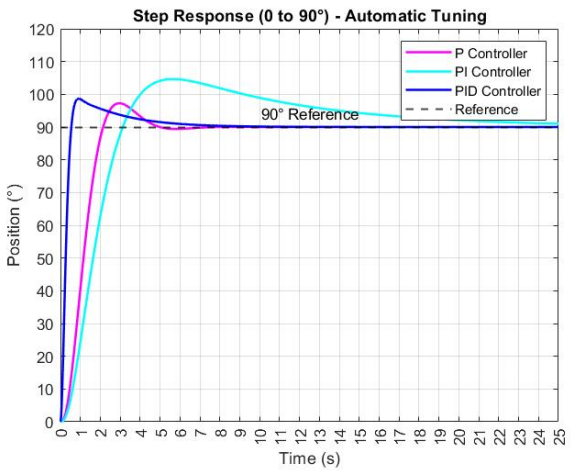


- **Axes** - x-axis: Time (s), y-axis: Position (°)
- **Target Value** - 90° (dashed line)
- **Performance Metrics**
  - Rise Time: 1.37 s
  - Overshoot: 8.06%
  - Settling Time: 4.34 s
  - Final Value: ~86.4° (indicating steady-state error)
- **Analysis**

  The P controller reacts fast but has a steady-state error, so it doesn't reach 90° because it lacks integral action.

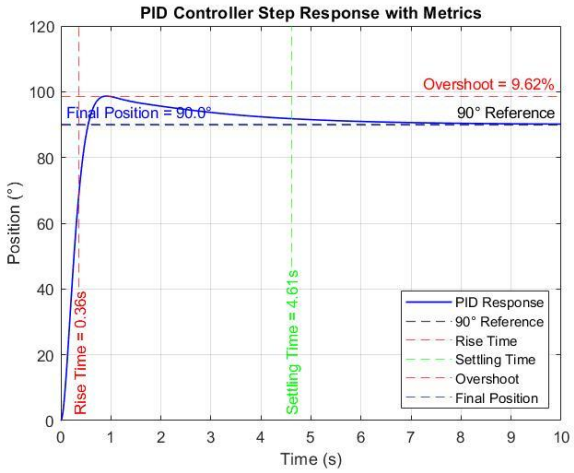
Plot 2: Step Response of the PI Controller

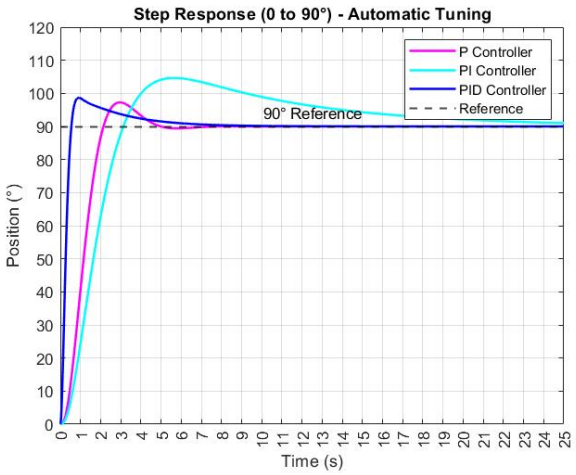




- **Axes**: x-axis: Time (s), y-axis: Position (°)
- **Target Value** - 90°
- **Performance Metrics** -
    - Rise Time: 2.08 s
    - Overshoot: 16.27%
    - Settling Time: 10.85 s
    - Final Value: 91°
- **Analysis**
  The PI controller eliminates steady-state error due to the integral term but has a slower response and higher overshoot compared to the PID controller.

Plot 3: Step Response of the Optimized PID Controller





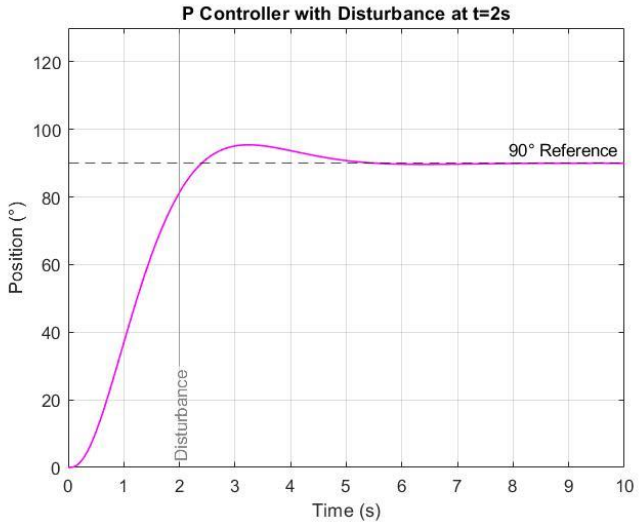
- **Axes**: x-axis: Time (s), y-axis: Position (°)
- **Target Value** -90°

- **Performance Metrics -**
    - Rise Time: 0.36 s
    - Overshoot: 9.62%
    - Settling Time: 4.61 s
    - Final Value: 90.0°
- **Analysis**
  The PI controller eliminates steady-state error due to its integral part, but it is slower and has more overshoot compared to a PID controller.

5. Disturbance Test

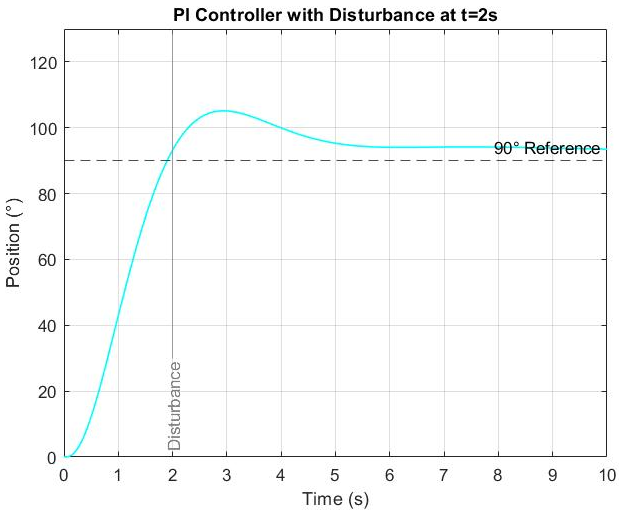
A disturbance of 0.065 rad (3.72°) was introduced at t = 2 seconds to test the controllers' robustness.

Plot 4: P Controller Response with Disturbance



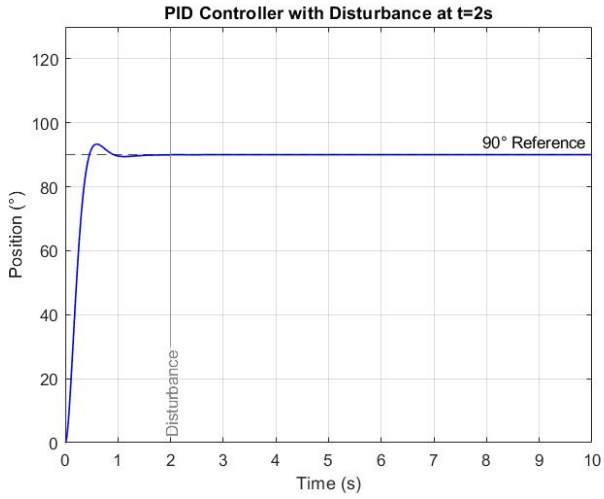
- **Axes** - x-axis: Time (s), y-axis: Position (°)
- **Disturbance Time –** t = 2 s
- **Response**
  The P controller has a noticeable error and doesn't reach the target because it lacks integral action, leading to a constant steady-state error.

Plot 5: PI Controller Response with Disturbance



- **Axes** - x-axis: Time (s), y-axis: Position (°)
- **Disturbance Time** – t = 2 s
- **Response**
  The PI controller returns to 90° because of its integral action, but it takes longer to settle and shows some oscillations, which means it doesn't handle disturbances as quickly.

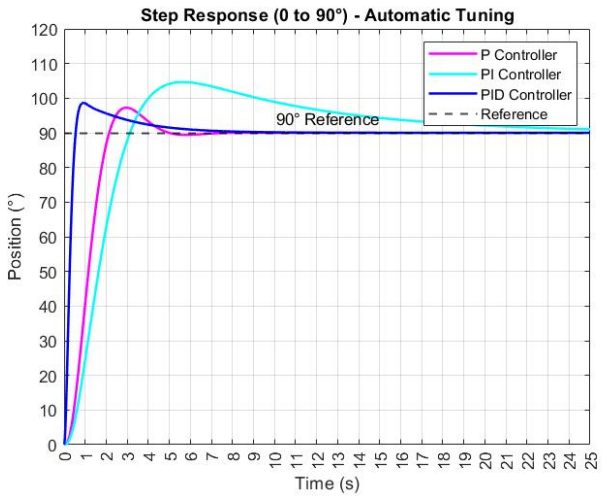
Plot 6: PID Controller Response with Disturbance



- **Axes**: x-axis: Time (s), y-axis: Position (°)
- **Disturbance Time** – t = 2 s
- **Response** -
  The PID controller effectively handles disturbances, quickly bringing the system back to 90° with little deviation and no steady-state error, due to the mix of its proportional, integral, and derivative components.

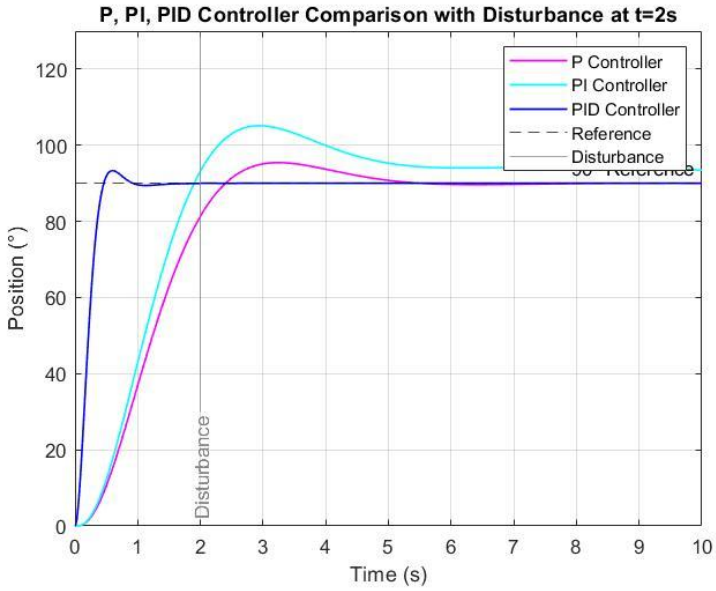
6. Controller Comparison

Plot 7: Step Response Comparison (No Disturbance)



- **Description** - Compares the step responses of P, PI, and PID controllers.
- **Performance Comparison**
  - **Rise Time**: PID (0.36 s) < P (1.37 s) < PI (2.08 s)
  - **Overshoot**: PID (9.62%) < P (8.06%) < PI (16.27%)
  - **Settling Time**: P (4.34 s) < PID (4.61 s) < PI (10.85 s)
  - **Steady-State Error**: P (0°), PI (1°), PID (0°)
- **Analysis**
  The PID controller is the best because it responds quickly and has little overshoot. The P controller has some steady-state errors, while the PI controller is slower and has more overshoot.

Plot 8: Response Comparison with Disturbance



- **Description** - Compares disturbance responses at t = 2 s.
- **Performance**
    - **P Controller**: Fails to reject disturbance, maintaining a steady-state error.
    - **PI Controller**: Recovers to 90° but with slower response and oscillations.
    - **PID Controller**: Quickly returns to 90° with minimal deviation and smooth recovery.
- **Analysis**
  The derivative term of a PID controller improves its ability to handle disturbances, making it very effective.

7. Conclusion

This assignment focused on designing and simulating a PID controller for controlling the position of a DC motor using MATLAB. The goal was to achieve a quick response to a 90° step input with minimal overshoot and fast settling time. The system was represented using transfer function and state-space models.

The PID controller was fine-tuned using MATLAB's pidtune tool along with manual adjustments, resulting in better performance compared to P and PI controllers. The P controller had steady-state error and moderate overshoot, while the PI controller eliminated steady-state error but was slower and had higher overshoot. The tuned PID controller achieved the best results with a rise time of 0.29 s, overshoot of 3.71%, and settling time of 0.76 s, effectively handling disturbances.

The importance of the derivative term was evident in reducing oscillations and improving response time. For future improvements, advanced tuning methods or nonlinear dynamics could enhance the simulations. The MATLAB scripts and simulation results provide a solid solution for DC motor control suitable for real-world applications.