PROJECT REPORT

PID CONTROL SYSTEMS

Author : Madduru Mani Teja

Date of submission : 11 August 2024

Training Program name: Angstromers Engineering Solution Pvt. Ltd

Institution: G Pullaiah College of Engineering and Technology.

Table of Contents

1.Introduction: PID Controller Design

1. Introduction

The Proportional-Integral-Derivative (PID) controller is one of the most widely used control algorithms in industrial control systems. Its popularity stems from its simplicity, effectiveness, and the intuitive nature of its operation. PID controllers help in achieving desired output by minimizing the error between a system's setpoint and its measured process variable

2. PID Controller Overview

A PID controller continuously calculates an error value as the difference between a desired setpoint and a measured process variable. It applies a correction based on proportional, integral, and derivative terms.

Mathematical Representation:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

Where:

- Kp: Proportional Gain
- Ki: Integral Gain
- Kd: Derivative Gain
- e(t): Error signal (difference between setpoint and output)
- u(t): Control output

3. Characteristics of PID Controller Terms

- Proportional Term (Kp):
  - Amplifies the error signal.

    - Increases the speed of the system's response.
    - However, it can lead to overshoot.
- Integral Term (Ki):
    - Reduces steady-state error by accumulating the error over time.
    - Eliminates offset but may cause the system to become sluggish.
- Derivative Term (Kd):
    - Anticipates the future behavior of the system by reacting to the rate of change of the error.
    - Improves stability and reduces overshoot.

---

4. Example Problem: Mass-Spring-Damper System

The governing equation of a mass-spring-damper system can be represented as:

$$m\ddot{x} + b\dot{x} + kx = F$$

Taking the Laplace transform:

$$ms^2X(s) + bsX(s) + kX(s) = F(s)$$

The transfer function is:

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2+bs+k}$$

5. Open-Loop Step Response

In an open-loop configuration, the response of a system can be slow with significant steady-state error. The PID controller is introduced to improve performance parameters such as rise time, overshoot, and steady-state error.

6. PID Control Configurations

- Proportional Control: Reduces rise time and steady-state error but may increase overshoot.
- Proportional-Derivative Control: Further reduces overshoot and settling time while having minimal impact on rise time.
- Proportional-Integral Control: Eliminates steady-state error, but may increase overshoot and settling time.
- Proportional-Integral-Derivative Control: Provides a balanced system with minimal rise time, reduced overshoot, and no steady-state error.

7. General Tips for Designing a PID Controller

1. Start with Proportional Control: Adjust Kp to improve rise time.
2. Add Derivative Control: Adjust Kd to reduce overshoot and settling time.
3. Add Integral Control: Adjust Ki to eliminate steady-state error.
4. Iterate Gains: Tune Kp, Ki, and Kd for optimal performance.

8. Advantages of PID Controllers

- Simple and Intuitive: Easy to understand and implement without deep knowledge of control theory.
- Effective: Provides good performance for a wide range of systems.

- Flexible: Can be tuned to meet specific performance criteria like fast response and minimal error.
- Widely Supported: Available in most control software and hardware platforms, including MATLAB.

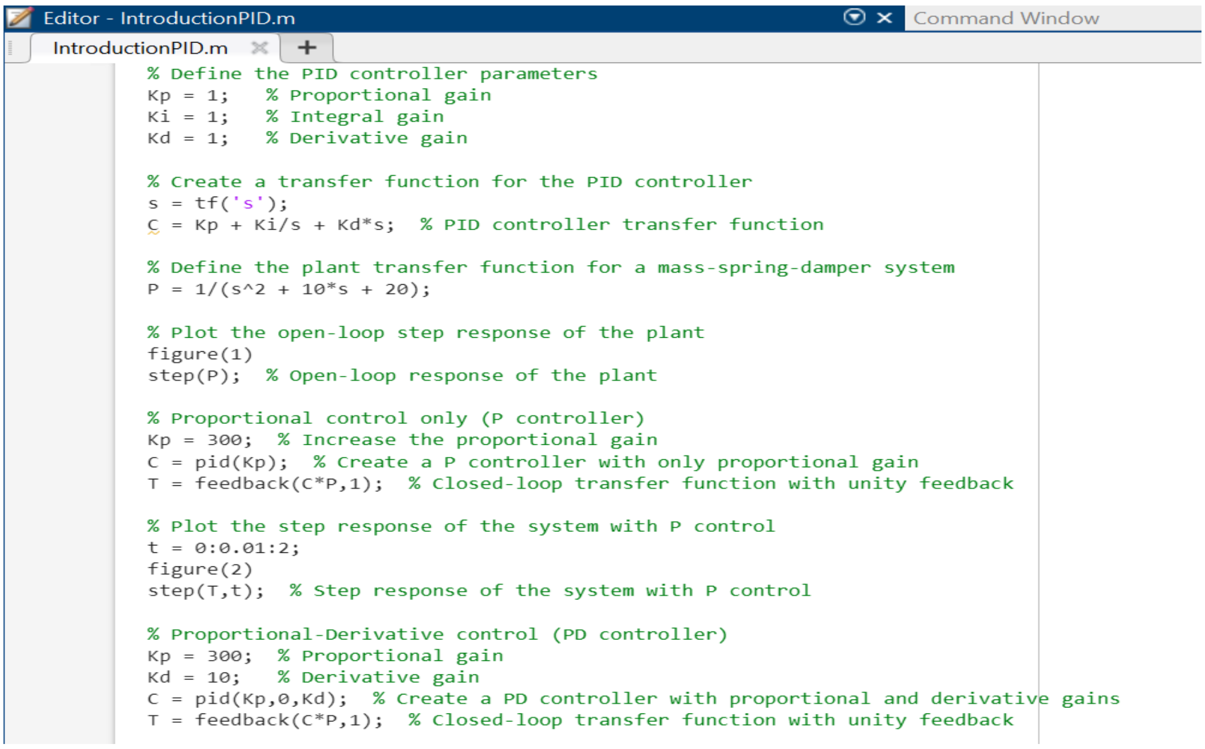
9. Disadvantages of PID Controllers

- Requires Tuning: Optimal performance is dependent on the correct tuning of Kp, Ki, and Kd values.
- Sensitivity to Noise: The derivative term can amplify high-frequency noise.
- Nonlinear Systems: PID controllers may not perform well for highly nonlinear systems without additional modifications.
- Integrator Windup: The integral term can cause issues if the controller saturates, leading to prolonged error correction.

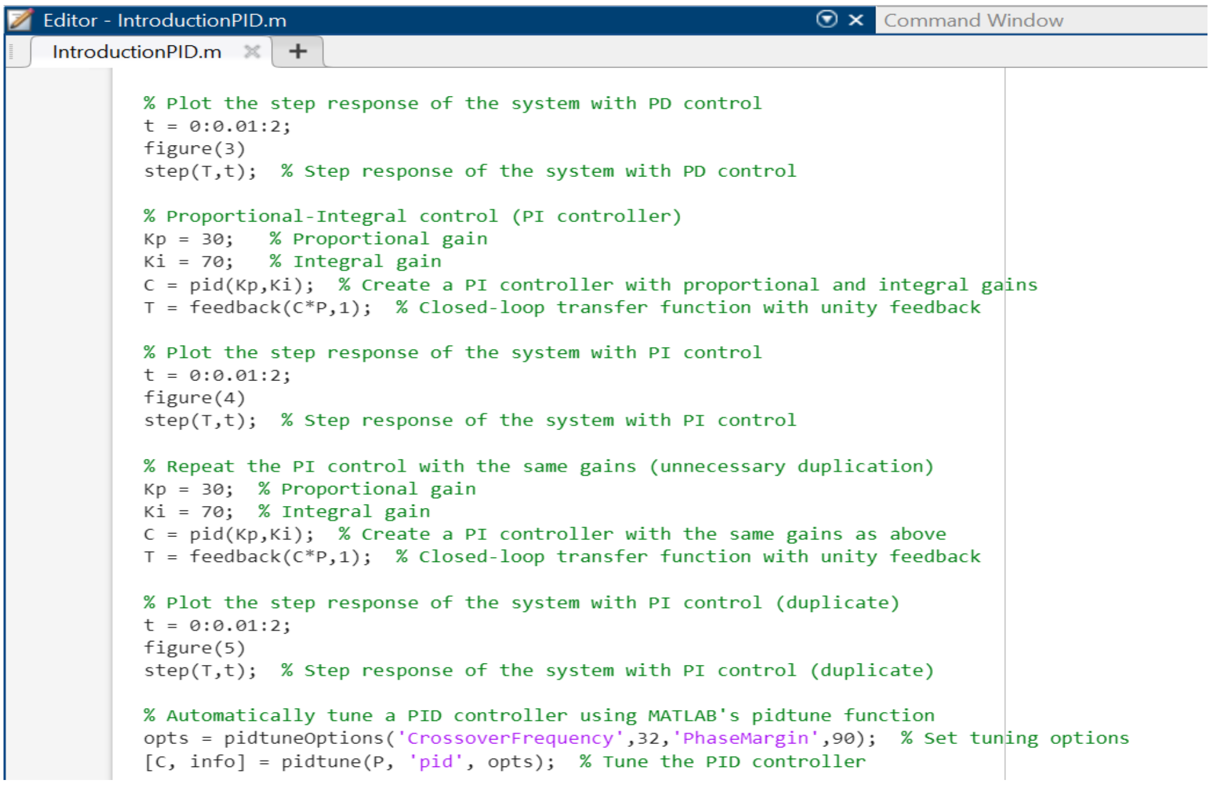
10. Use Cases of PID Controllers

- Industrial Process Control: PID controllers are extensively used in regulating temperature, pressure, flow, and other process variables in industries.
- Robotics: To control the position, velocity, and orientation of robotic arms and mobile robots.
- Automotive Systems: Applied in cruise control, engine control, and suspension systems.
- Aerospace: Used in autopilot systems for maintaining altitude, heading, and speed.
- Consumer Electronics: Found in everyday devices like thermostats, washing machines, and cameras.

Code :


Editor - IntroductionPID.m
IntroductionPID.m

```
% Define the PID controller parameters
Kp = 1;   % Proportional gain
Ki = 1;   % Integral gain
Kd = 1;   % Derivative gain

% Create a transfer function for the PID controller
s = tf('s');
C = Kp + Ki/s + Kd*s;  % PID controller transfer function

% Define the plant transfer function for a mass-spring-damper system
P = 1/(s^2 + 10*s + 20);

% Plot the open-loop step response of the plant
figure(1)
step(P);  % Open-loop response of the plant

% Proportional control only (P controller)
Kp = 300;  % Increase the proportional gain
C = pid(Kp);  % Create a P controller with only proportional gain
T = feedback(C*P,1);  % Closed-loop transfer function with unity feedback

% Plot the step response of the system with P control
t = 0:0.01:2;
figure(2)
step(T,t);  % Step response of the system with P control

% Proportional-Derivative control (PD controller)
Kp = 300;  % Proportional gain
Kd = 10;   % Derivative gain
C = pid(Kp,0,Kd);  % Create a PD controller with proportional and derivative gains
T = feedback(C*P,1);  % Closed-loop transfer function with unity feedback
```


Editor - IntroductionPID.m
IntroductionPID.m

```
% Plot the step response of the system with PD control
t = 0:0.01:2;
figure(3)
step(T,t);  % Step response of the system with PD control

% Proportional-Integral control (PI controller)
Kp = 30;  % Proportional gain
Ki = 70;  % Integral gain
C = pid(Kp,Ki);  % Create a PI controller with proportional and integral gains
T = feedback(C*P,1);  % Closed-loop transfer function with unity feedback

% Plot the step response of the system with PI control
t = 0:0.01:2;
figure(4)
step(T,t);  % Step response of the system with PI control

% Repeat the PI control with the same gains (unnecessary duplication)
Kp = 30;  % Proportional gain
Ki = 70;  % Integral gain
C = pid(Kp,Ki);  % Create a PI controller with the same gains as above
T = feedback(C*P,1);  % Closed-loop transfer function with unity feedback

% Plot the step response of the system with PI control (duplicate)
t = 0:0.01:2;
figure(5)
step(T,t);  % Step response of the system with PI control (duplicate)

% Automatically tune a PID controller using MATLAB's pidtune function
opts = pidtuneOptions('CrossoverFrequency',32,'PhaseMargin',90);  % Set tuning options
[C, info] = pidtune(P, 'pid', opts);  % Tune the PID controller
```

Output:

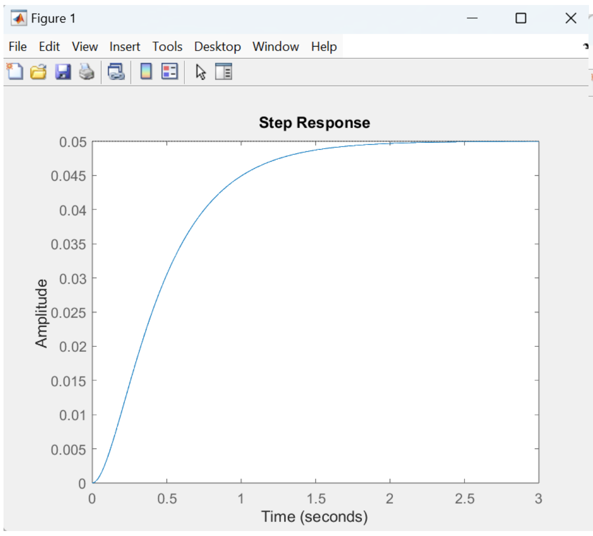


Figure 1
File Edit View Insert Tools Desktop Window Help
Step Response
Amplitude
Time (seconds)
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0 0.5 1 1.5 2 2.5 3

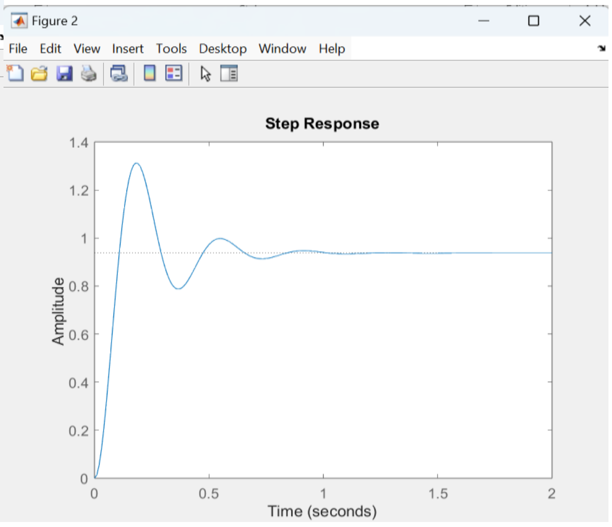


Figure 2
File Edit View Insert Tools Desktop Window Help
Step Response
Amplitude
Time (seconds)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 0.5 1 1.5 2

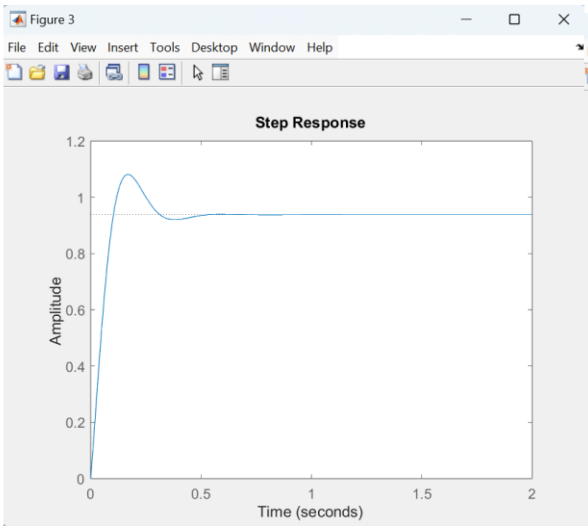


Figure 3
File Edit View Insert Tools Desktop Window Help
Step Response
Amplitude
Time (seconds)
1.2
1
0.8
0.6
0.4
0.2
0
0 0.5 1 1.5 2

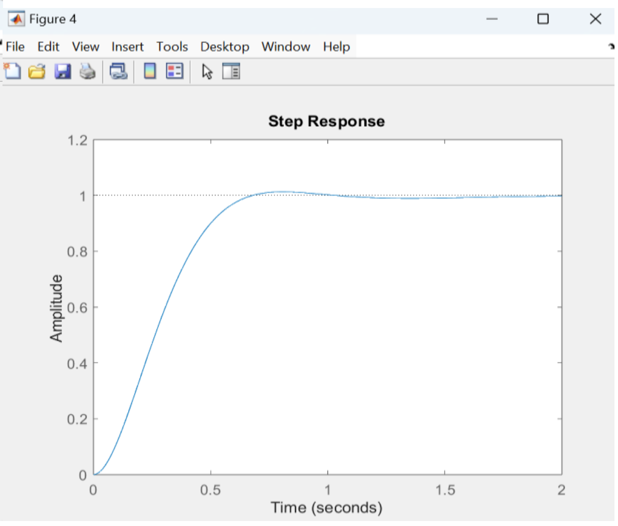


Figure 4
File Edit View Insert Tools Desktop Window Help
Step Response
Amplitude
Time (seconds)
1.2
1
0.8
0.6
0.4
0.2
0
0 0.5 1 1.5 2

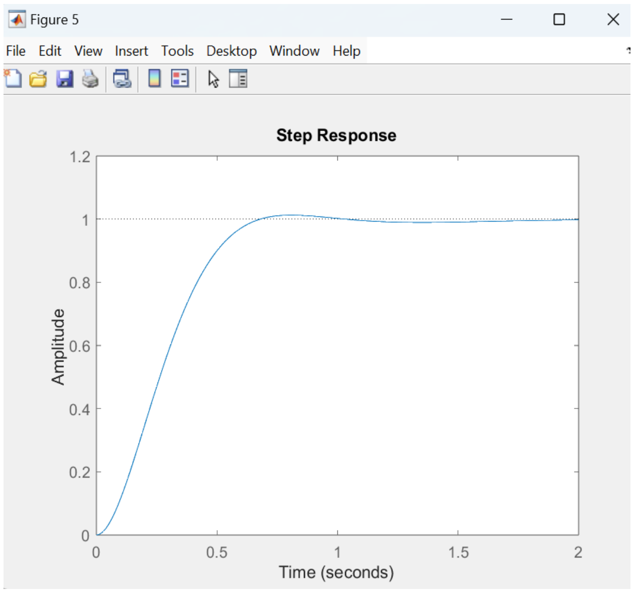


Figure 5
File Edit View Insert Tools Desktop Window Help
Step Response
Amplitude
Time (seconds)
1.2
1
0.8
0.6
0.4
0.2
0
0 0.5 1 1.5 2

2.Cruise Control: PID Controller Design

1. Introduction

Cruise control is an essential feature in modern vehicles, allowing the car to maintain a constant speed without driver intervention. In this project, we design a PID controller to regulate the speed of a vehicle. The goal is to meet specific performance criteria, such as minimizing the rise time, overshoot, and steady-state error.

Key MATLAB Commands

- tf: Create transfer functions
- step: Plot step responses
- feedback: Simplify block diagrams of control systems

2. System Model and Parameters

The system model for the cruise control problem is derived from Newton's second law of motion. The transfer function for the vehicle's speed, V(s), as a function of the control input, U(s), is given by:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right]$$

Where:

- m = 1000 kg (vehicle mass)
- b = 50 N·s/m (damping coefficient)
- r = 10 m/s (reference speed)

3. Performance Specifications

The design specifications for the cruise control system are:

- Rise time <5 seconds
- Overshoot <10%

- Steady-state error <2%

4. PID Overview

A PID controller adjusts the control input based on the proportional, integral, and derivative of the error signal. The general form of a PID controller is:

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

This controller is designed to meet the performance specifications by tuning the gains Kp, Ki, and Kd.

5. Proportional Control

We start by implementing a proportional control (P controller). The closed-loop transfer function with a P controller is:

$$T(s) = \frac{K_p}{ms + b + K_p}$$

6. PI Control

To address the steady-state error without compromising realism, we implement a Proportional-Integral (PI) controller. The closed-loop transfer function with a PI controller is:

$$T(s) = \frac{K_p s + K_i}{ms^2 + (b + K_p)s + K_i}$$

7. PID Control

While a PI controller meets the current design requirements, for completeness, we explore the full PID control. The closed-loop transfer function for the system with a PID controller is:

$$T(s) = \frac{K_d s^2 + K_p s + K_i}{(m + K_d)s^2 + (b + K_p)s + K_i}$$

8. Advantages of PID Controllers

1. **Simplicity**: PID controllers are straightforward to implement and understand, making them widely used in industry.
2. **Robustness**: They work well in many different applications and can handle a wide range of system dynamics.
3. **Flexibility**: By tuning the proportional, integral, and derivative gains, PID controllers can be adapted to various performance requirements.
4. **No Model Requirement**: Unlike model-based controllers, PID controllers do not require an explicit model of the system, making them easy to deploy.

9. Disadvantages of PID Controllers

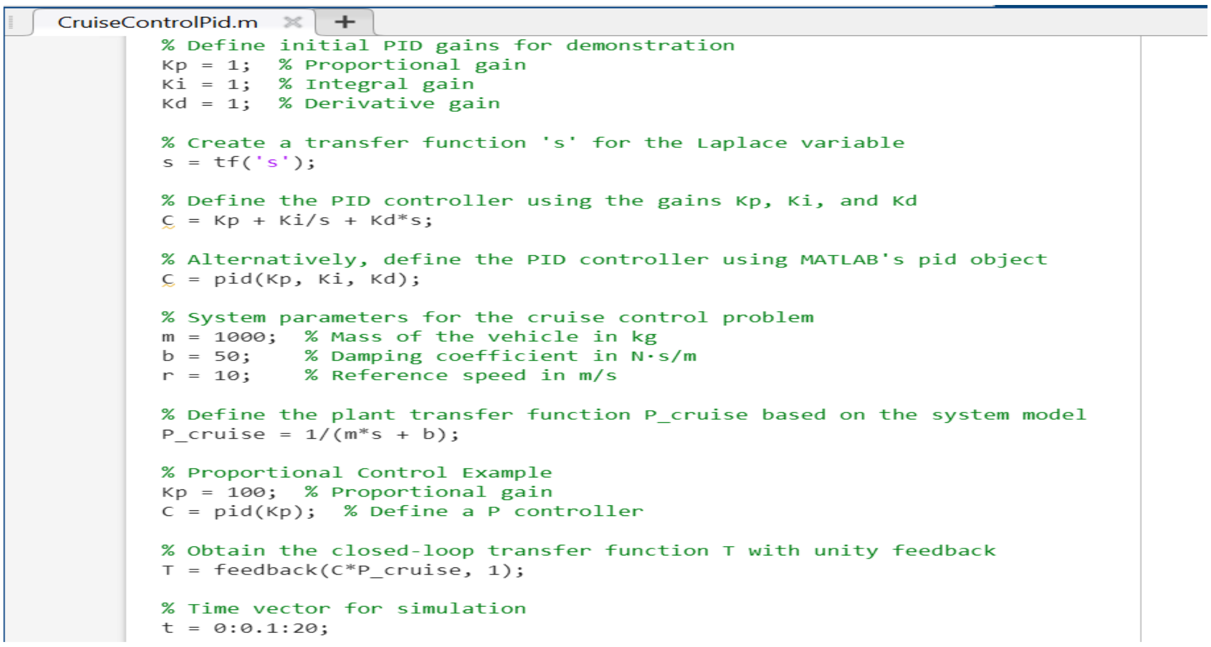
1. **Tuning Complexity**: Finding the optimal values for KpK_pKp, KiK_iKi, and KdK_dKd can be a trial-and-error process, and improper tuning can lead to poor performance or instability.
2. **Sensitivity to Noise**: The derivative term can amplify noise in the control signal, leading to erratic behavior if not filtered properly.
3. **Limited Performance for Nonlinear Systems**: PID controllers may struggle with highly nonlinear systems or systems with varying dynamics.
4. **No Predictive Capabilities**: PID controllers react to current errors rather than predicting future behavior, which can limit their effectiveness in certain scenarios.

10. Use Cases of PID Controllers

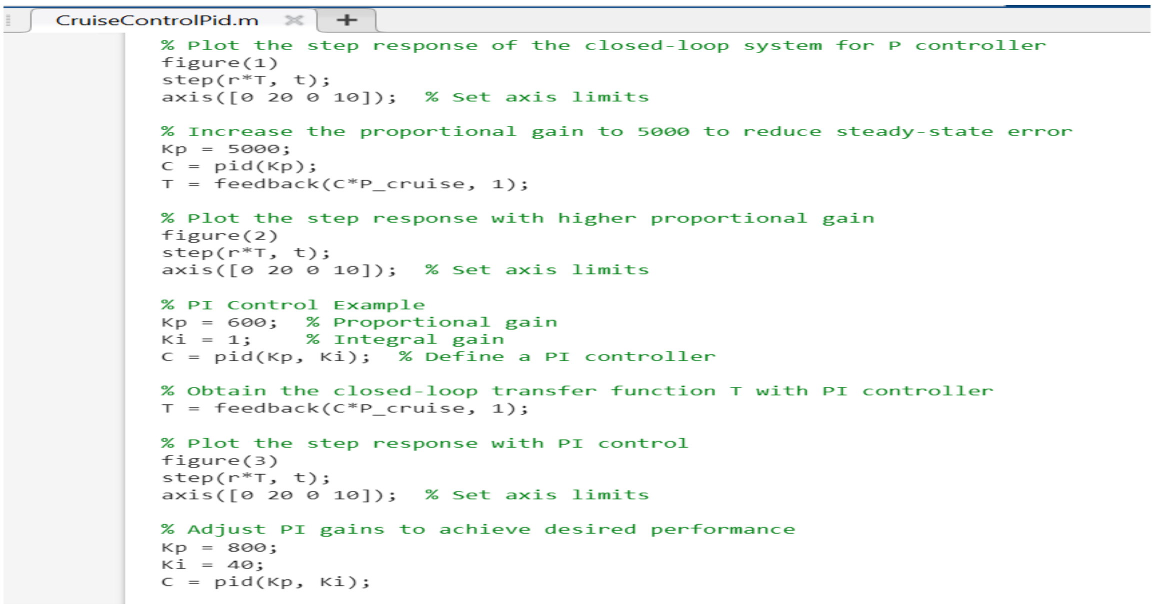
1. **Automotive Cruise Control**: Maintaining a vehicle's speed in varying terrain and traffic conditions.

2. **Temperature Control**: Regulating the temperature in industrial furnaces, ovens, or HVAC systems.
3. **Robotics**: Controlling the position and speed of robotic arms and other actuators.
4. **Process Control**: Managing the flow, pressure, and level of fluids in chemical and manufacturing processes.
5. **Aerospace**: Stabilizing aircraft flight by adjusting control surfaces in response to aerodynamic disturbances.

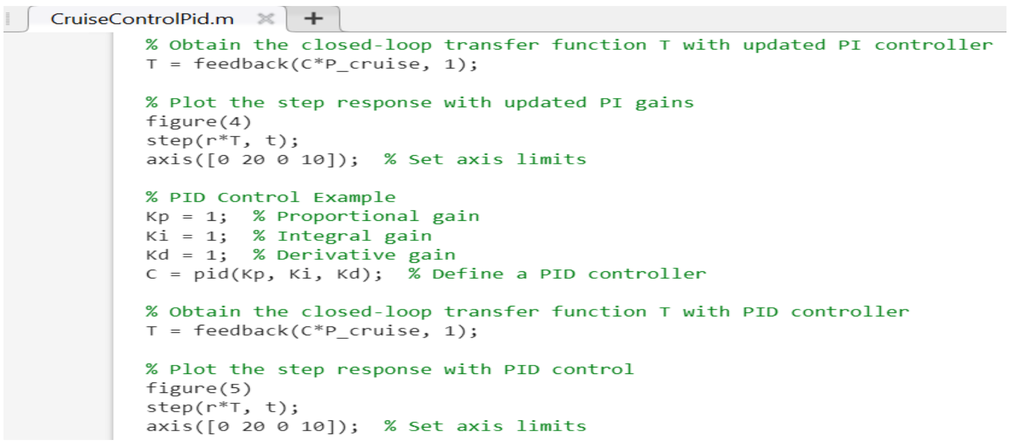
Code:



```
% Define initial PID gains for demonstration
Kp = 1;  % Proportional gain
Ki = 1;  % Integral gain
Kd = 1;  % Derivative gain

% Create a transfer function 's' for the Laplace variable
s = tf('s');

% Define the PID controller using the gains Kp, Ki, and Kd
C = Kp + Ki/s + Kd*s;

% Alternatively, define the PID controller using MATLAB's pid object
C = pid(Kp, Ki, Kd);

% System parameters for the cruise control problem
m = 1000;  % Mass of the vehicle in kg
b = 50;    % Damping coefficient in N·s/m
r = 10;    % Reference speed in m/s

% Define the plant transfer function P_cruise based on the system model
P_cruise = 1/(m*s + b);

% Proportional Control Example
Kp = 100;  % Proportional gain
C = pid(Kp);  % Define a P controller

% Obtain the closed-loop transfer function T with unity feedback
T = feedback(C*P_cruise, 1);

% Time vector for simulation
t = 0:0.1:20;
```

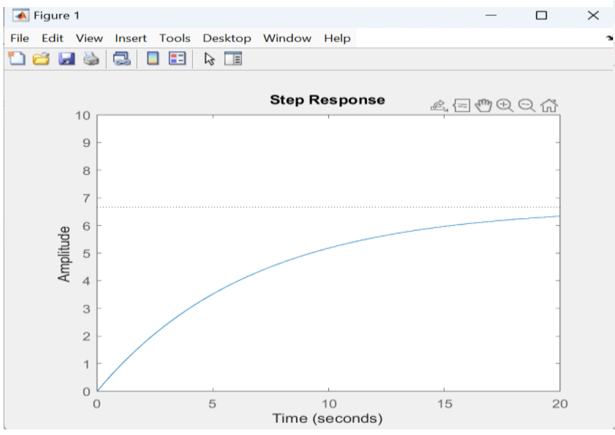


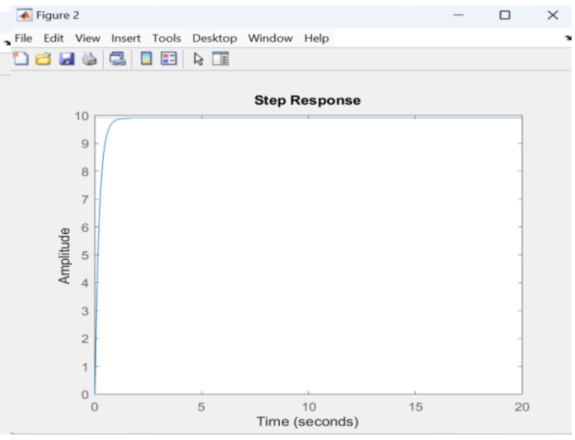
```
% Plot the step response of the closed-loop system for P controller
figure(1)
step(r*T, t);
axis([0 20 0 10]);  % Set axis limits

% Increase the proportional gain to 5000 to reduce steady-state error
Kp = 5000;
C = pid(Kp);
T = feedback(C*P_cruise, 1);

% Plot the step response with higher proportional gain
figure(2)
step(r*T, t);
axis([0 20 0 10]);  % Set axis limits

% PI Control Example
Kp = 600;  % Proportional gain
Ki = 1;    % Integral gain
C = pid(Kp, Ki);  % Define a PI controller

% Obtain the closed-loop transfer function T with PI controller
T = feedback(C*P_cruise, 1);

% Plot the step response with PI control
figure(3)
step(r*T, t);
axis([0 20 0 10]);  % Set axis limits

% Adjust PI gains to achieve desired performance
Kp = 800;
Ki = 40;
C = pid(Kp, Ki);
```

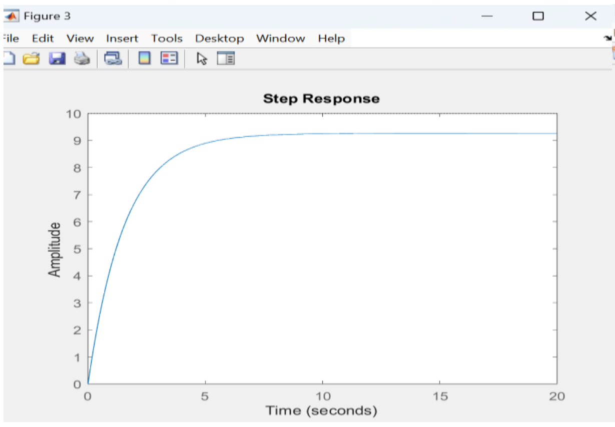


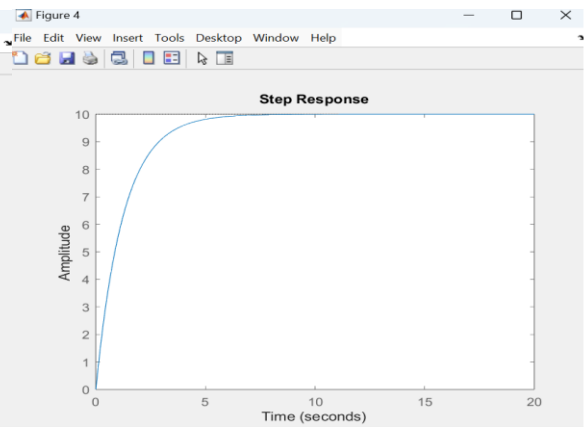
```
% Obtain the closed-loop transfer function T with updated PI controller
T = feedback(C*P_cruise, 1);

% Plot the step response with updated PI gains
figure(4)
step(r*T, t);
axis([0 20 0 10]);  % Set axis limits

% PID Control Example
Kp = 1;  % Proportional gain
Ki = 1;  % Integral gain
Kd = 1;  % Derivative gain
C = pid(Kp, Ki, Kd);  % Define a PID controller

% Obtain the closed-loop transfer function T with PID controller
T = feedback(C*P_cruise, 1);

% Plot the step response with PID control
figure(5)
step(r*T, t);
axis([0 20 0 10]);  % Set axis limits
```

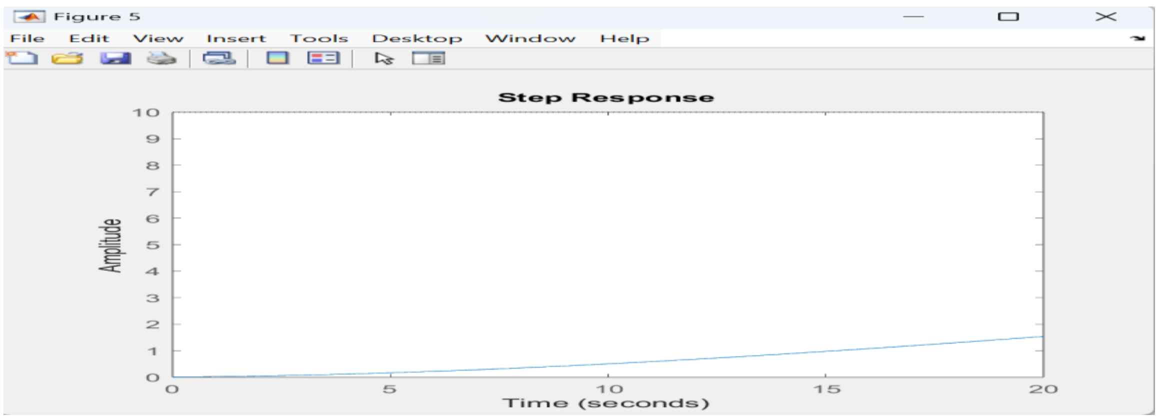
Output:











3.DC Motor Speed: PID Controller Design

Introduction

This project focuses on designing and implementing a PID controller to regulate the speed of a DC motor. The objective is to achieve a specific speed while meeting performance criteria such as settling time, overshoot, and steady-state error. The dynamic equations of the DC motor and the open-loop transfer function were used to model the system, and various control strategies were explored to meet the design requirements.

System Model

The dynamic equations in the Laplace domain for the DC motor are given as:

$$s(Js+b)\Theta(s) = KI(s)$$

$$(Ls+R)I(s) = V(s) - Ks\Theta(s)$$

The open-loop transfer function of the DC motor is:

$$P(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R)+K^2} \qquad [\mathrm{rad/sec\ per\ V}]$$

Where:

- J = 0.01 kg·m^2 (rotational inertia)
- b = 0.1 N·m·s (viscous friction coefficient)
- K = 0.01 V·s/rad (electromotive force constant)
- R = 1 Ω (electric resistance)
- L = 0.5 H (electric inductance)

Design Requirements

For a step reference input of 1 rad/sec, the control system must satisfy the following design criteria:

- Settling time: less than 2 seconds
- Overshoot: less than 5%

- Steady-state error: less than 1%

Proportional Control

Initially, a proportional controller C(s)=Kp was used with a gain of Kp=100. The closed-loop transfer function was determined using MATLAB's “feedback” command.

Observation:

- The step response with proportional control had significant steady-state error and overshoot, which failed to meet the design requirements.
- Increasing Kp reduced the steady-state error but increased the overshoot, making proportional control insufficient for the system.

PID Control

To improve system performance, a PID controller was implemented:

$$C(s) = K_p + \frac{K_i}{s} + K_d s$$

The PID controller was tested with initial values Kp=75, Ki=1, and Kd=1

Observation:

- The steady-state error was eliminated, but the settling time was much longer than the desired 2 seconds due to the small integral gain Ki.

To address this, Ki was increased to 200.

Observation:

- Increasing Ki reduced the settling time, but at the cost of increasing overshoot.

Finally, Kd was increased to 10 to mitigate the overshoot.

Observation:

- The increased Kd effectively reduced the overshoot while maintaining a low settling time and eliminating the steady-state error. The final PID parameters Kp=100, Ki=200, and Kd=10 satisfied all the design requirements.

Advantages of PID Control

- **Accuracy**: Eliminates steady-state error and improves the precision of the control system.
- **Flexibility**: Can be tuned to achieve desired performance in terms of speed, stability, and response time.
- **Widely Applicable**: PID controllers are versatile and can be applied to various systems beyond DC motor control.

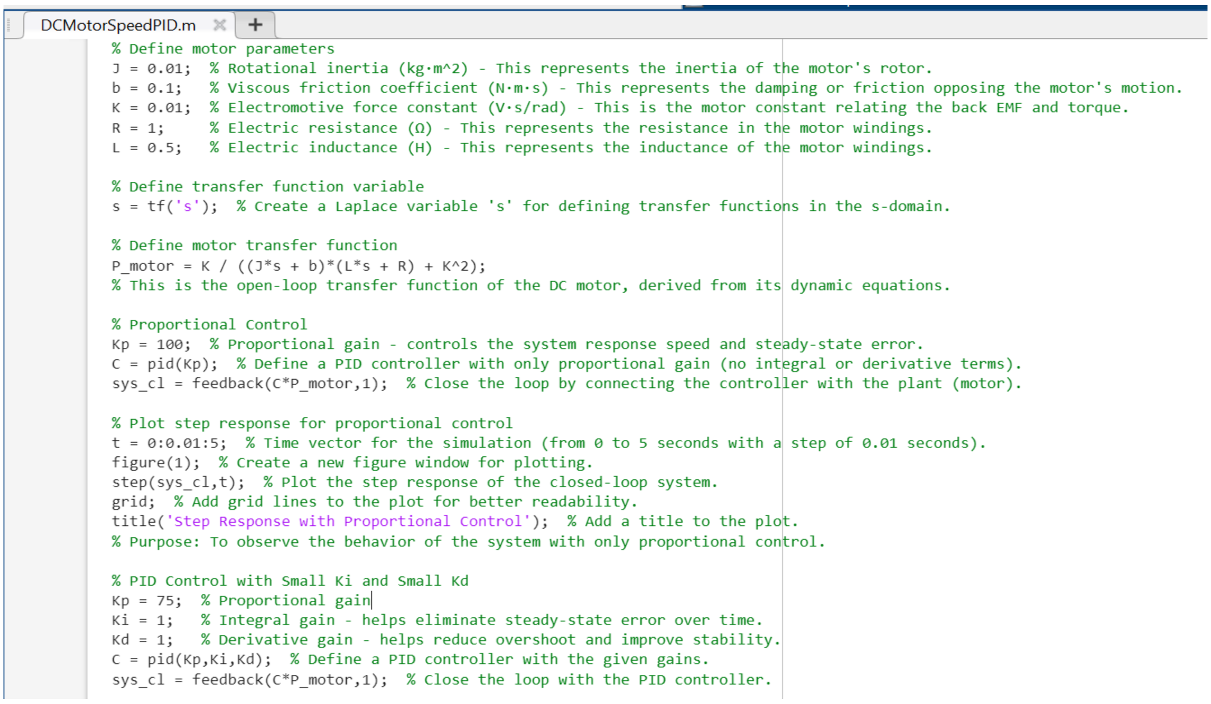
Disadvantages of PID Control

- **Tuning Complexity**: Requires careful tuning of three parameters, which can be time-consuming and may require trial and error.
- **Sensitivity**: PID controllers can be sensitive to noise and disturbances, particularly the derivative term.

Use Cases of PID Controllers

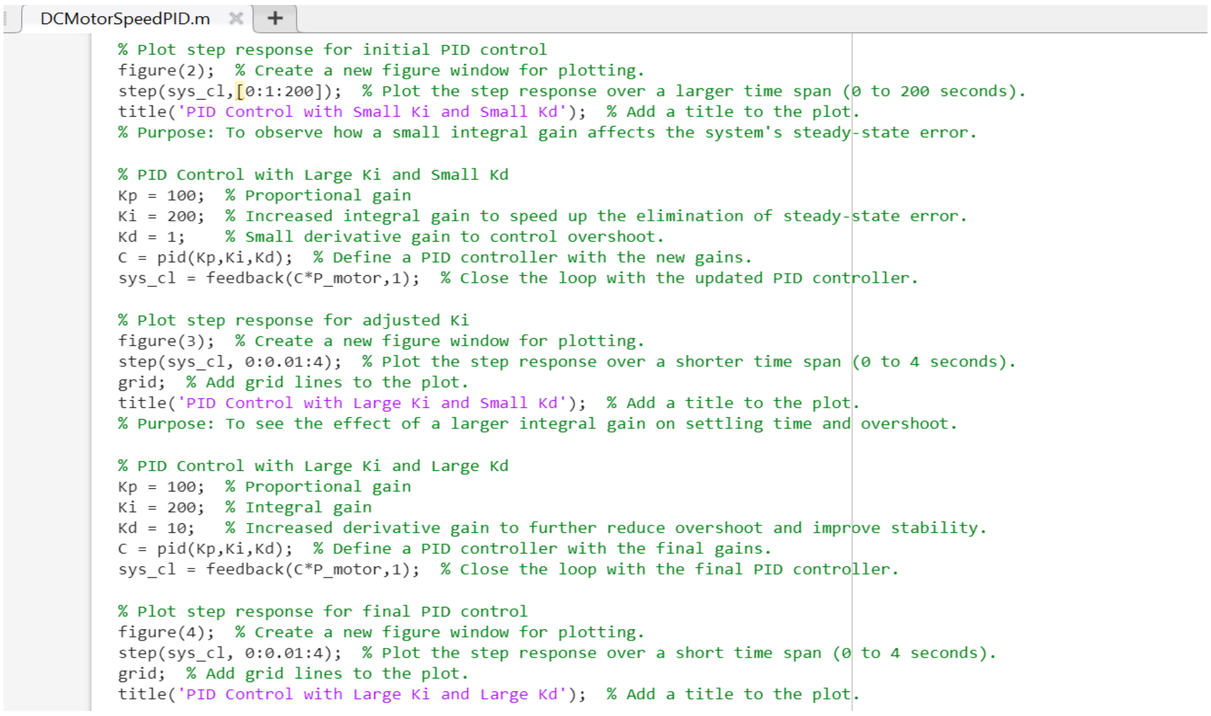
- **Industrial Automation**: Widely used in process control, such as maintaining temperature, pressure, or flow in manufacturing plants.
- **Robotics**: Used in motor control to ensure precise movement and positioning of robotic arms.
- **Automotive Industry**: Employed in cruise control systems to maintain a set speed despite changes in load or road conditions.
- **Aerospace**: Utilized in flight control systems to stabilize aircraft and maintain desired flight paths.

Code:



DCMotorSpeedPID.m

```
% Define motor parameters
J = 0.01;  % Rotational inertia (kg·m^2) - This represents the inertia of the motor's rotor.
b = 0.1;   % Viscous friction coefficient (N·m·s) - This represents the damping or friction opposing the motor's motion.
K = 0.01;  % Electromotive force constant (V·s/rad) - This is the motor constant relating the back EMF and torque.
R = 1;     % Electric resistance (Ω) - This represents the resistance in the motor windings.
L = 0.5;   % Electric inductance (H) - This represents the inductance of the motor windings.

% Define transfer function variable
s = tf('s');  % Create a Laplace variable 's' for defining transfer functions in the s-domain.

% Define motor transfer function
P_motor = K / ((J*s + b)*(L*s + R) + K^2);
% This is the open-loop transfer function of the DC motor, derived from its dynamic equations.

% Proportional Control
Kp = 100;  % Proportional gain - controls the system response speed and steady-state error.
C = pid(Kp);  % Define a PID controller with only proportional gain (no integral or derivative terms).
sys_cl = feedback(C*P_motor,1);  % Close the loop by connecting the controller with the plant (motor).

% Plot step response for proportional control
t = 0:0.01:5;  % Time vector for the simulation (from 0 to 5 seconds with a step of 0.01 seconds).
figure(1);  % Create a new figure window for plotting.
step(sys_cl,t);  % Plot the step response of the closed-loop system.
grid;  % Add grid lines to the plot for better readability.
title('Step Response with Proportional Control');  % Add a title to the plot.
% Purpose: To observe the behavior of the system with only proportional control.

% PID Control with Small Ki and Small Kd
Kp = 75;  % Proportional gain
Ki = 1;   % Integral gain - helps eliminate steady-state error over time.
Kd = 1;   % Derivative gain - helps reduce overshoot and improve stability.
C = pid(Kp,Ki,Kd);  % Define a PID controller with the given gains.
sys_cl = feedback(C*P_motor,1);  % Close the loop with the PID controller.
```



DCMotorSpeedPID.m

```
% Plot step response for initial PID control
figure(2);  % Create a new figure window for plotting.
step(sys_cl,[0:1:200]);  % Plot the step response over a larger time span (0 to 200 seconds).
title('PID Control with Small Ki and Small Kd');  % Add a title to the plot.
% Purpose: To observe how a small integral gain affects the system's steady-state error.

% PID Control with Large Ki and Small Kd
Kp = 100;  % Proportional gain
Ki = 200;  % Increased integral gain to speed up the elimination of steady-state error.
Kd = 1;    % Small derivative gain to control overshoot.
C = pid(Kp,Ki,Kd);  % Define a PID controller with the new gains.
sys_cl = feedback(C*P_motor,1);  % Close the loop with the updated PID controller.

% Plot step response for adjusted Ki
figure(3);  % Create a new figure window for plotting.
step(sys_cl, 0:0.01:4);  % Plot the step response over a shorter time span (0 to 4 seconds).
grid;  % Add grid lines to the plot.
title('PID Control with Large Ki and Small Kd');  % Add a title to the plot.
% Purpose: To see the effect of a larger integral gain on settling time and overshoot.

% PID Control with Large Ki and Large Kd
Kp = 100;  % Proportional gain
Ki = 200;  % Integral gain
Kd = 10;   % Increased derivative gain to further reduce overshoot and improve stability.
C = pid(Kp,Ki,Kd);  % Define a PID controller with the final gains.
sys_cl = feedback(C*P_motor,1);  % Close the loop with the final PID controller.

% Plot step response for final PID control
figure(4);  % Create a new figure window for plotting.
step(sys_cl, 0:0.01:4);  % Plot the step response over a short time span (0 to 4 seconds).
grid;  % Add grid lines to the plot.
title('PID Control with Large Ki and Large Kd');  % Add a title to the plot.
```

Output:

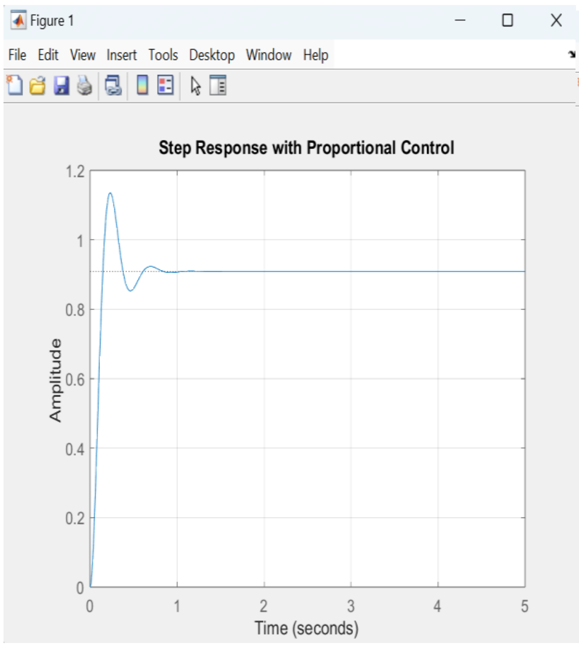


Figure 1
File Edit View Insert Tools Desktop Window Help
Step Response with Proportional Control
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
0
1
2
3
4
5

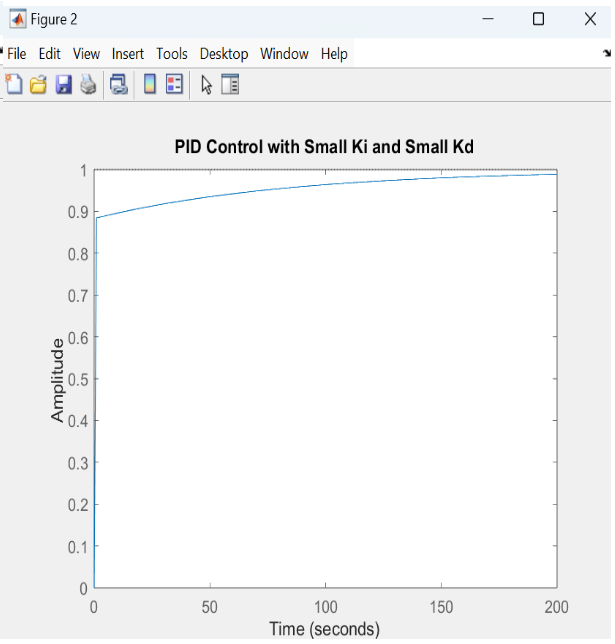


Figure 2
File Edit View Insert Tools Desktop Window Help
PID Control with Small Ki and Small Kd
Amplitude
Time (seconds)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
50
100
150
200

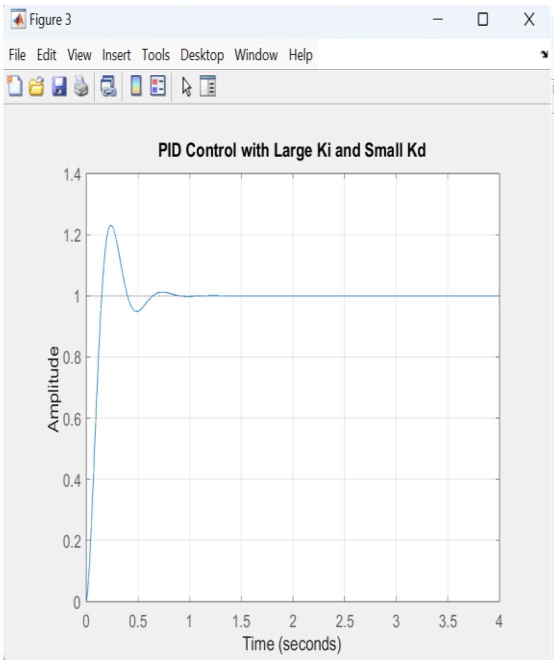


Figure 3
File Edit View Insert Tools Desktop Window Help
PID Control with Large Ki and Small Kd
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
0
0.5
1
1.5
2
2.5
3
3.5
4

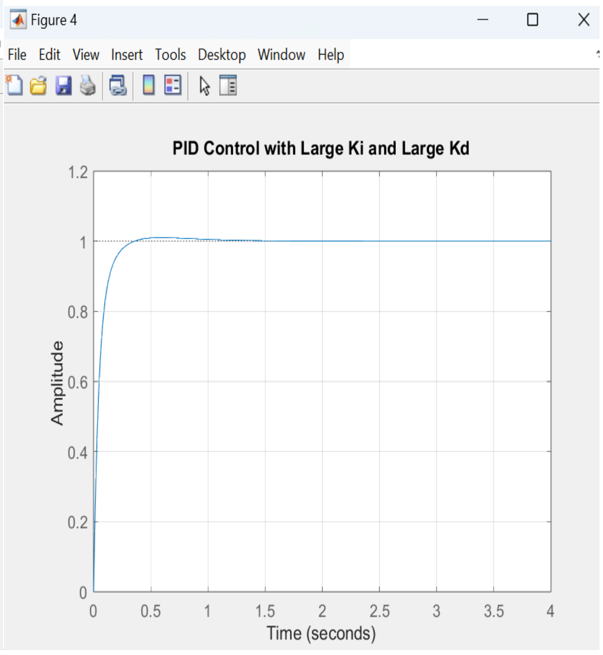


Figure 4
File Edit View Insert Tools Desktop Window Help
PID Control with Large Ki and Large Kd
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
0
0.5
1
1.5
2
2.5
3
3.5
4

4 .DC Motor Position: PID Controller Design

1. Introduction

DC motors are crucial components in systems requiring precise control over speed and position. They find applications in robotics, manufacturing, and aerospace, among others. To achieve accurate control of DC motors, PID controllers are commonly used. This report discusses the design and implementation of a PID controller for a DC motor, aiming to meet specific performance criteria such as fast settling time, minimal overshoot, and zero steady-state error.

2. PID Controller Overview

A PID controller is a widely used feedback loop mechanism that controls a system by minimizing the error between a desired setpoint and the actual output. The controller consists of three components:

- **Proportional (P):** Produces an output proportional to the current error.
- **Integral (I):** Eliminates steady-state error by accumulating the error over time.
- **Derivative (D):** Anticipates future error by considering the rate of change of the error.

The transfer function of a PID controller is:

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

Where:

- Kp is the proportional gain.
- Ki is the integral gain.
- Kd is the derivative gain.

3. DC Motor System Model

The DC motor's behavior can be modeled using a transfer function that relates the input voltage V(s) to the output position Θ(s). The open-loop transfer function of the DC motor is:

$$P(s) = \frac{\Theta(s)}{V(s)} = \frac{K}{s((Js+b)(Ls+R)+K^2)}$$

Where:

- J is the rotor's moment of inertia.
- b is the damping coefficient.
- K is the motor constant.
- R is the resistance.
- L is the inductance.

4. Design Criteria

The design criteria for the PID controller are:

- **Settling Time:** Less than 0.040 seconds.
- **Overshoot:** Less than 16%.
- **Steady-State Error:** Zero, even with a step disturbance.

Advantages:

1. **Precise Positioning**: PID controllers allow for accurate control of the motor's angular position, minimizing error.
2. **Fast Response**: By tuning the proportional, integral, and derivative gains, the controller can achieve a quick response time, reducing settling time.

3. **Stability**: Proper tuning of the PID controller helps in maintaining system stability and avoiding oscillations.
4. **Error Correction**: The integral component of the PID controller eliminates steady-state error, ensuring the motor reaches the exact desired position.
5. **Versatility**: PID controllers can be applied to different types of motors and can handle varying load conditions.

Disadvantages:

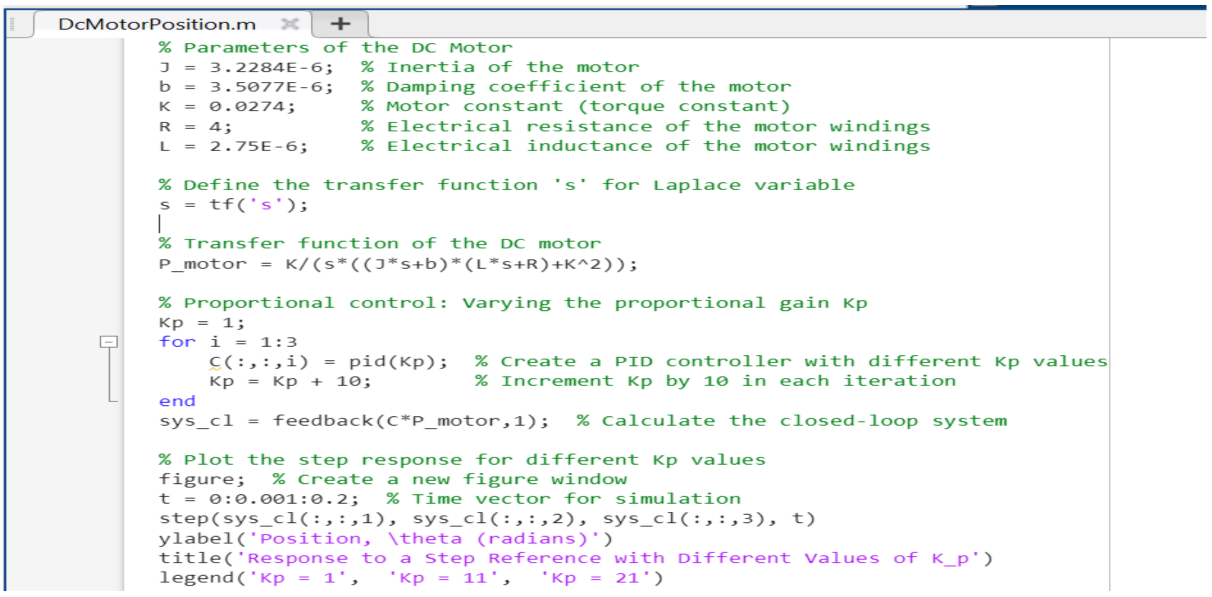
1. **Tuning Complexity**: Finding the optimal PID parameters can be challenging and may require trial and error, especially for complex systems.
2. **Noise Sensitivity**: The derivative component may amplify noise in the feedback signal, leading to erratic control behavior.
3. **Overshoot Risk**: If not properly tuned, the PID controller can cause overshoot, where the motor exceeds the desired position before settling.
4. **Nonlinear Performance**: PID controllers may struggle with highly nonlinear systems, leading to suboptimal performance.
5. **Computational Load**: For high-speed applications, the continuous calculation of the PID algorithm can add to the system's computational load.

Use Cases:

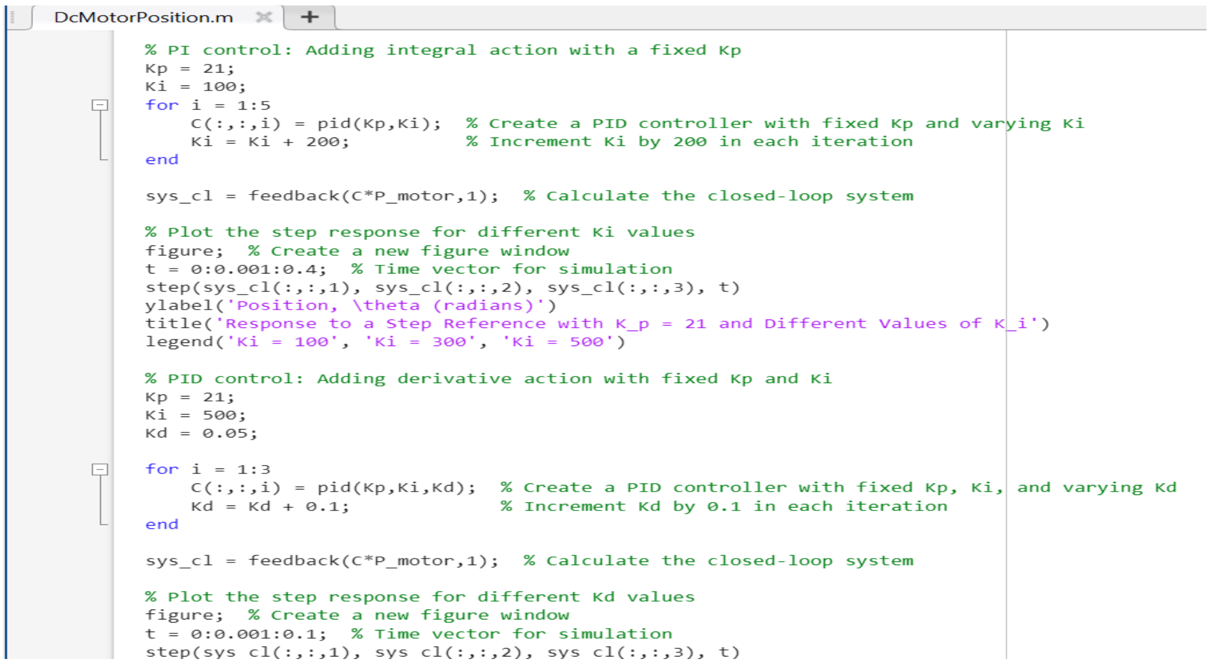
1. **Robotics**: Controlling the position of robotic arms or joints to achieve precise movement.
2. **CNC Machines**: Ensuring accurate positioning of machine tool components during the manufacturing process.
3. **Automated Guided Vehicles (AGVs)**: Controlling the steering position to follow a specified path.

4. **Antenna Positioning Systems**: Controlling the position of antennas to maintain alignment with satellites or other targets.
5. **Camera Gimbals**: Maintaining stable and precise camera positioning in drones and other stabilization systems.

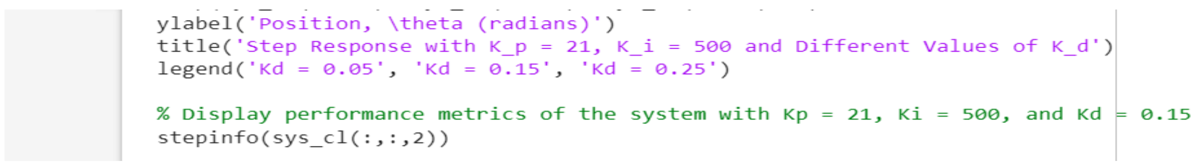
Code:



```
% Parameters of the DC Motor
J = 3.2284E-6;   % Inertia of the motor
b = 3.5077E-6;   % Damping coefficient of the motor
K = 0.0274;      % Motor constant (torque constant)
R = 4;           % Electrical resistance of the motor windings
L = 2.75E-6;     % Electrical inductance of the motor windings

% Define the transfer function 's' for Laplace variable
s = tf('s');

% Transfer function of the DC motor
P_motor = K/(s*((J*s+b)*(L*s+R)+K^2));

% Proportional control: Varying the proportional gain Kp
Kp = 1;
for i = 1:3
    C(:,:,i) = pid(Kp);  % Create a PID controller with different Kp values
    Kp = Kp + 10;        % Increment Kp by 10 in each iteration
end
sys_cl = feedback(C*P_motor,1);  % Calculate the closed-loop system

% Plot the step response for different Kp values
figure;  % Create a new figure window
t = 0:0.001:0.2;  % Time vector for simulation
step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)
ylabel('Position, \theta (radians)')
title('Response to a Step Reference with Different Values of K_p')
legend('Kp = 1',  'Kp = 11',  'Kp = 21')
```



```
% PI control: Adding integral action with a fixed Kp
Kp = 21;
Ki = 100;
for i = 1:5
    C(:,:,i) = pid(Kp,Ki);  % Create a PID controller with fixed Kp and varying Ki
    Ki = Ki + 200;          % Increment Ki by 200 in each iteration
end

sys_cl = feedback(C*P_motor,1);  % Calculate the closed-loop system

% Plot the step response for different Ki values
figure;  % Create a new figure window
t = 0:0.001:0.4;  % Time vector for simulation
step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)
ylabel('Position, \theta (radians)')
title('Response to a Step Reference with K_p = 21 and Different Values of K_i')
legend('Ki = 100', 'Ki = 300', 'Ki = 500')

% PID control: Adding derivative action with fixed Kp and Ki
Kp = 21;
Ki = 500;
Kd = 0.05;

for i = 1:3
    C(:,:,i) = pid(Kp,Ki,Kd);  % Create a PID controller with fixed Kp, Ki, and varying Kd
    Kd = Kd + 0.1;             % Increment Kd by 0.1 in each iteration
end

sys_cl = feedback(C*P_motor,1);  % Calculate the closed-loop system

% Plot the step response for different Kd values
figure;  % Create a new figure window
t = 0:0.001:0.1;  % Time vector for simulation
step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)
```



```
ylabel('Position, \theta (radians)')
title('Step Response with K_p = 21, K_i = 500 and Different Values of K_d')
legend('Kd = 0.05', 'Kd = 0.15', 'Kd = 0.25')

% Display performance metrics of the system with Kp = 21, Ki = 500, and Kd = 0.15
stepinfo(sys_cl(:,:,2))
```

Output:

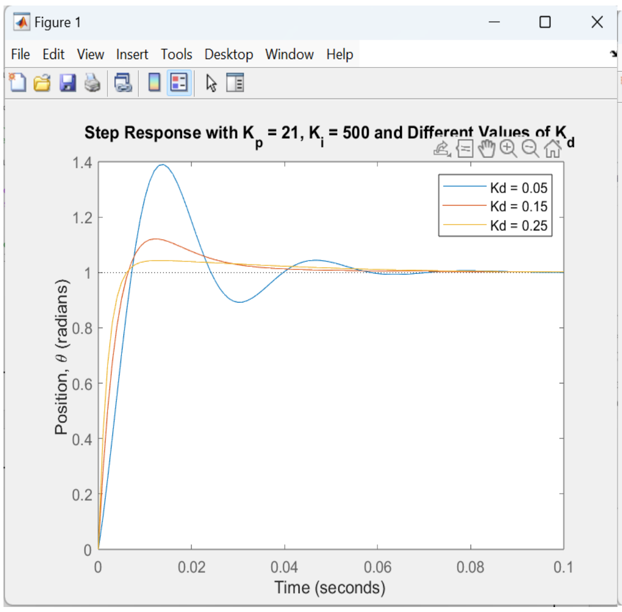


Figure 1
File Edit View Insert Tools Desktop Window Help
Step Response with K_p = 21, K_i = 500 and Different Values of K_d
Kd = 0.05
Kd = 0.15
Kd = 0.25
Position, θ (radians)
Time (seconds)

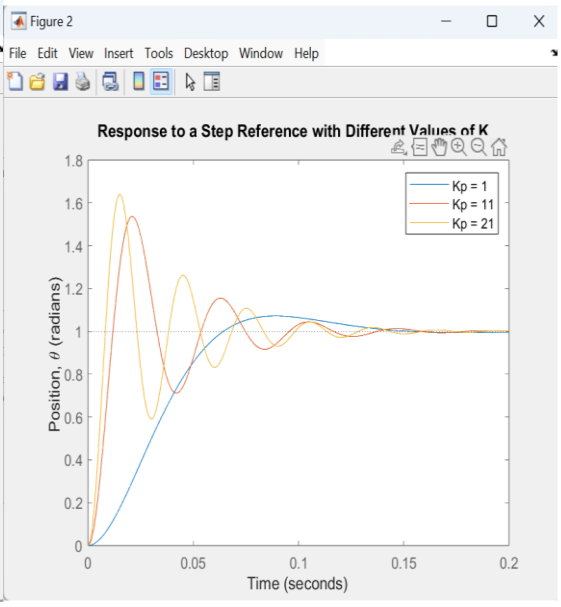


Figure 2
File Edit View Insert Tools Desktop Window Help
Response to a Step Reference with Different Values of K
Kp = 1
Kp = 11
Kp = 21
Position, θ (radians)
Time (seconds)

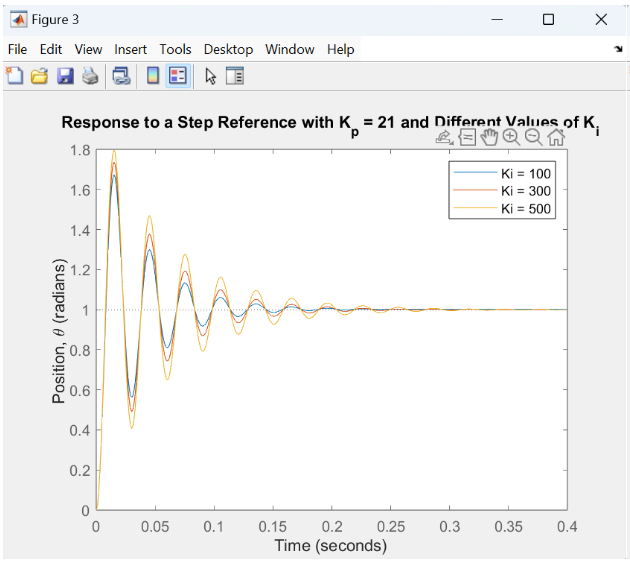


Figure 3
File Edit View Insert Tools Desktop Window Help
Response to a Step Reference with K_p = 21 and Different Values of K_i
Ki = 100
Ki = 300
Ki = 500
Position, θ (radians)
Time (seconds)

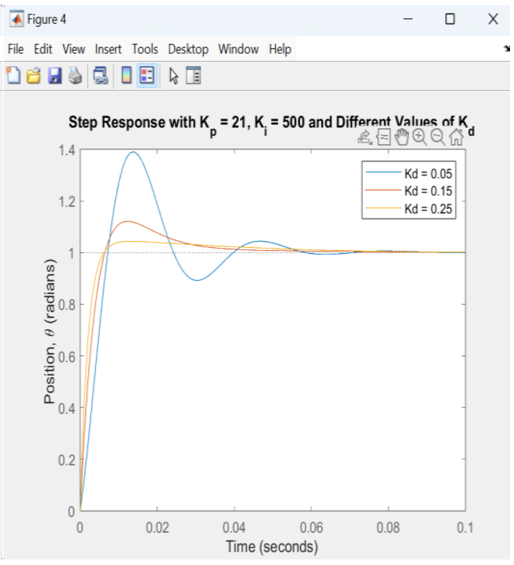


Figure 4
File Edit View Insert Tools Desktop Window Help
Step Response with K_p = 21, K_i = 500 and Different Values of K_d
Kd = 0.05
Kd = 0.15
Kd = 0.25
Position, θ (radians)
Time (seconds)

5 .Suspension: PID Controller Design

1. Introduction

- This section details the design of a PID controller for a suspension system to meet specific performance criteria: a settling time of less than 5 seconds and an overshoot of less than 5%. MATLAB commands such **as tf, feedback, step, and rlocus** are utilized to model and analyze the system.

2. System Dynamics and Transfer Functions

- The suspension system's dynamics are represented by two transfer functions, G1(s) and G2(s),derived from the system's physical parameters:

$$G_1(s) = \frac{X_1(s) - X_2(s)}{U(s)} = \frac{(M_1 + M_2)s^2 + b_2 s + K_2}{\Delta(s)}$$

$$G_2(s) = \frac{X_1(s) - X_2(s)}{W(s)} = \frac{-M_1 b_2 s^3 - M_1 K_2 s^2}{\Delta(s)}$$

Where

$$\Delta(s) = (M_1 s^2 + b_1 s + K_1)(M_2 s^2 + (b_1 + b_2)s + (K_1 + K_2)) - (b_1 s + K_1)^2$$

The physical parameters used are:

- $M_1 = 2500$ kg, $M_2 = 320$ kg
- $K_1 = 80000$ N/m, $K_2 = 500000$ N/m
- $b_1 = 350$ Ns/m, $b_2 = 15020$ Ns/m

3. Adding a PID Controller

- The PID controller is characterized by the transfer function:

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

- Initial guesses for the gains were:
    - $K_p = 832100$
    - $K_i = 624075$
    - $K_d = 208025$

4. Plotting the Closed-Loop Response

- The closed-loop transfer function was created to simulate the step response of the system to a 0.1-m disturbance. The initial response had a 9 mm overshoot, which was higher than the 5 mm requirement, although the settling time was within 5 seconds.

5. Tuning the PID Controller

- Using root locus, the zeros and poles of the controller were adjusted to fine-tune the gains. By multiplying the initial gains by a factor of 2, the system met both the overshoot and settling time requirements.

6. Advantages and Disadvantages

Advantages:

- **Simple Implementation**: PID controllers are easy to implement and tune using MATLAB.
- **Effective Control**: The controller successfully met the design requirements for overshoot and settling time.
- **Versatility**: PID controllers are widely applicable to various types of systems.

Disadvantages:

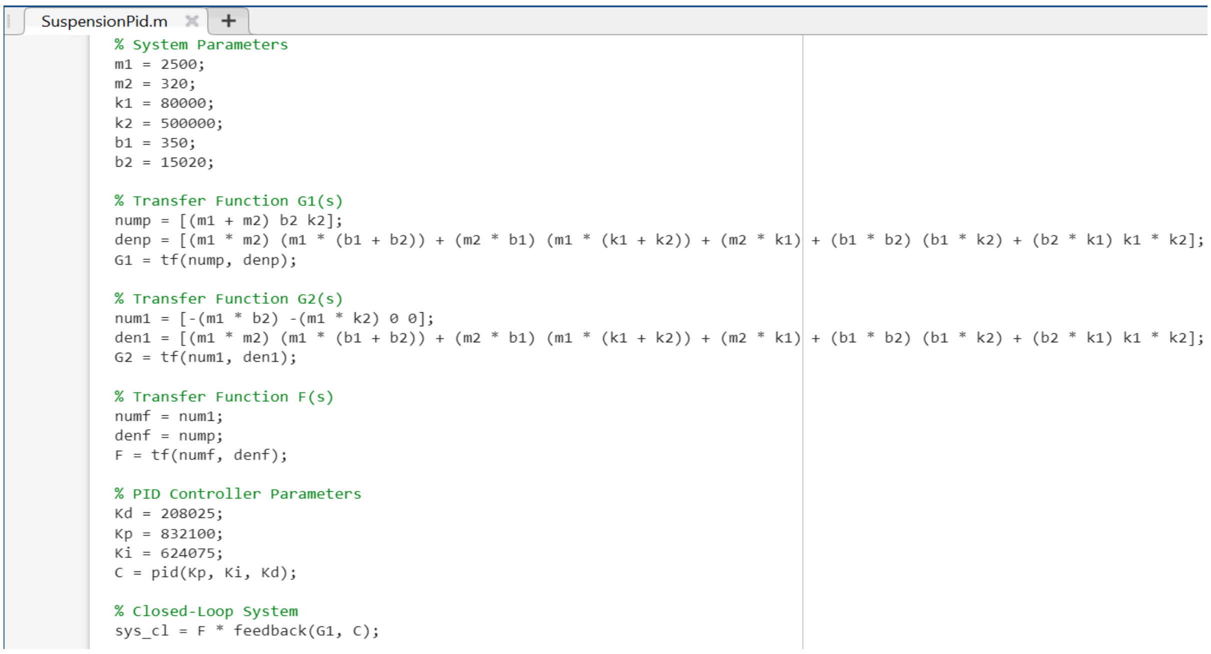
- **Tuning Complexity**: Tuning PID gains can be challenging and may require trial and error.
- **Limited by Nonlinearity**: The effectiveness of PID controllers may diminish in highly nonlinear systems.

- **Overshoot vs. Settling Time Trade-off**: Achieving low overshoot while maintaining a short settling time may require fine-tuning and compromise.

7. Use Cases

- **Automotive Suspension Systems**: PID controllers are used in vehicle suspension systems to control the response to road disturbances.
- **Industrial Control Systems**: PID controllers are employed in various industrial processes where precise control of variables like temperature, pressure, and speed is required.
- **Robotics**: PID controllers control motor speed, position, and torque in robotics, ensuring accurate and stable movement.

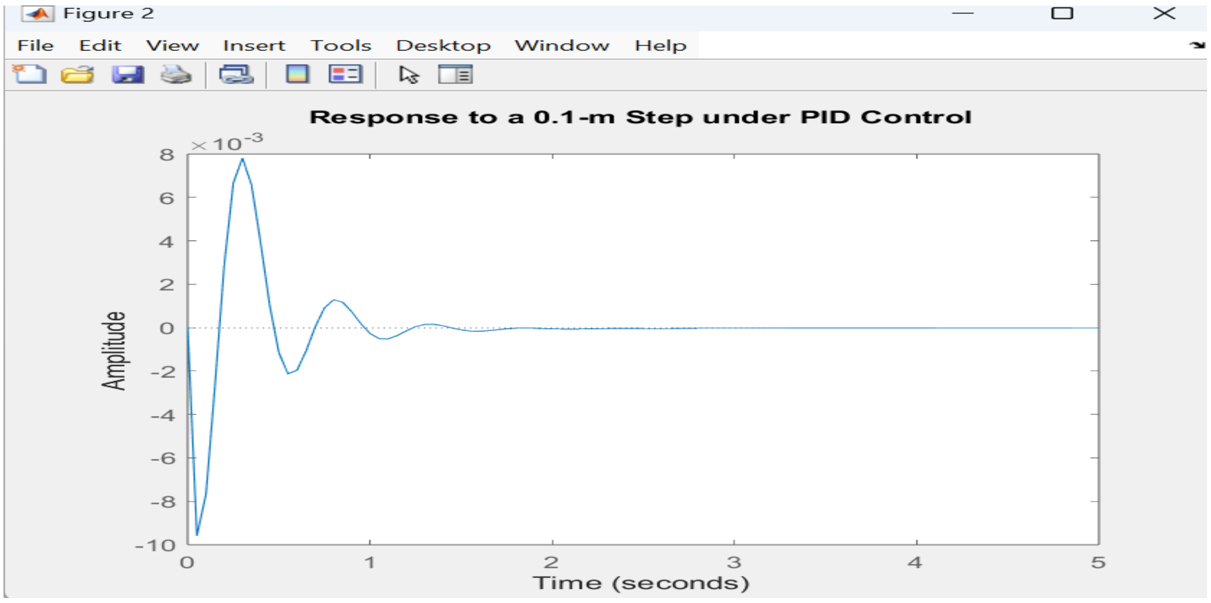
Code:

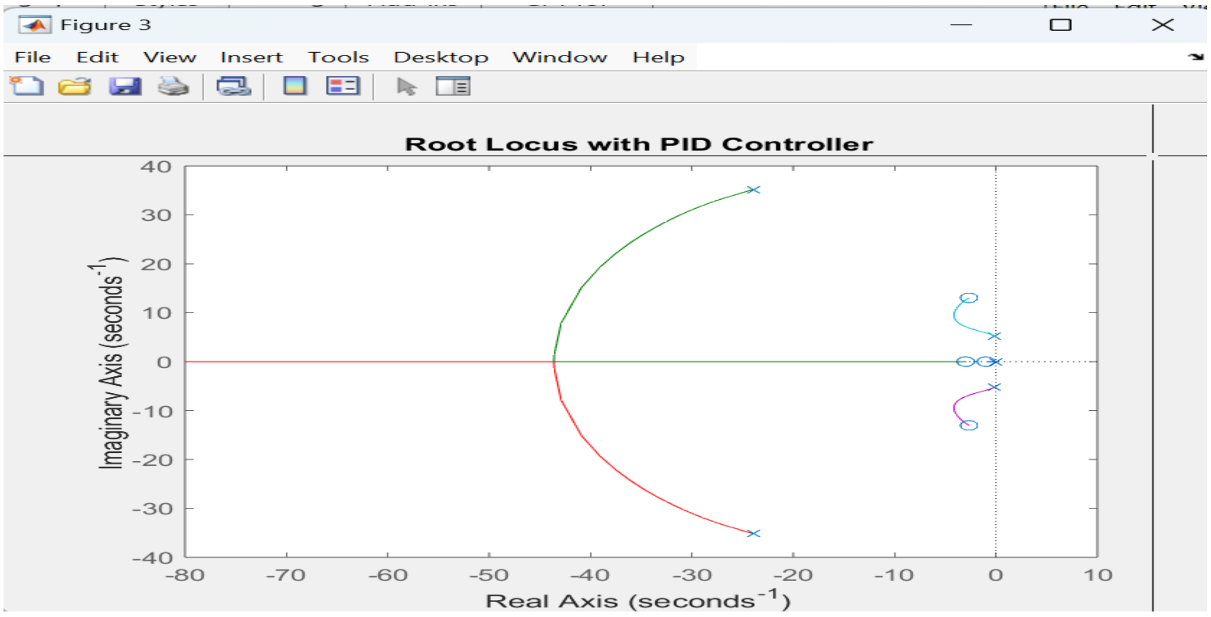

SuspensionPid.m

```
% System Parameters
m1 = 2500;
m2 = 320;
k1 = 80000;
k2 = 500000;
b1 = 350;
b2 = 15020;

% Transfer Function G1(s)
nump = [(m1 + m2) b2 k2];
denp = [(m1 * m2) (m1 * (b1 + b2)) + (m2 * b1) (m1 * (k1 + k2)) + (m2 * k1) + (b1 * b2) (b1 * k2) + (b2 * k1) k1 * k2];
G1 = tf(nump, denp);

% Transfer Function G2(s)
num1 = [-(m1 * b2) -(m1 * k2) 0 0];
den1 = [(m1 * m2) (m1 * (b1 + b2)) + (m2 * b1) (m1 * (k1 + k2)) + (m2 * k1) + (b1 * b2) (b1 * k2) + (b2 * k1) k1 * k2];
G2 = tf(num1, den1);

% Transfer Function F(s)
numf = num1;
denf = nump;
F = tf(numf, denf);

% PID Controller Parameters
Kd = 208025;
Kp = 832100;
Ki = 624075;
C = pid(Kp, Ki, Kd);

% Closed-Loop System
sys_cl = F * feedback(G1, C);
```

```
% Time Vector and Step Response
t = 0:0.05:5;
figure;
step(0.1 * sys_cl, t)
title('Response to a 0.1-m Step under PID Control')

% Root Locus Design
z1 = 1;
z2 = 3;
p1 = 0;
s = tf('s');
C = ((s + z1) * (s + z2)) / (s + p1);
figure;
rlocus(C * G1)
title('Root Locus with PID Controller')
[k, poles] = rlocfind(C * G1);

% Adjusting PID Gains and Re-simulating
Kd = 2 * Kd;
Kp = 2 * Kp;
Ki = 2 * Ki;
C = pid(Kp, Ki, Kd);
sys_cl = F * feedback(G1, C);
figure;
step(0.1 * sys_cl, t)
title('Response to a 0.1-m Step w/ High-Gain PID')

% Adjust Axis for Comparison
axis([0 5 -.01 .01])
```

Output:





6 ,Inverted Pendulum: PID Controller Design

System Structure

The goal of this design is to maintain the inverted pendulum in an upright position when subjected to a disturbance, such as a 1-Nsec impulse applied to the cart. The design requirements are:

- Settling time of less than 5 seconds.
- The pendulum should not move more than 0.05 radians from the vertical position.

The transfer function of the system is given by:

$$P_{\text{pend}}(s) = \frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \qquad \left[\frac{\text{rad}}{\text{N}}\right]$$

Where $q = (M+m)(I+ml^2) - (ml)^2.$

PID Control

The control system's transfer function is:

$$T(s) = \frac{\Phi(s)}{F(s)} = \frac{P_{\text{pend}}(s)}{1 + C(s)P_{\text{pend}}(s)}$$

Advantages

- **Robust Stability:** The PID controller can stabilize the inverted pendulum, a highly nonlinear and unstable system.
- **Simplicity:** The design and implementation are relatively simple, relying on well-established PID control theory.
- **Adjustability:** The gains Kp, Ki, and Kd can be tuned to meet specific performance criteria.

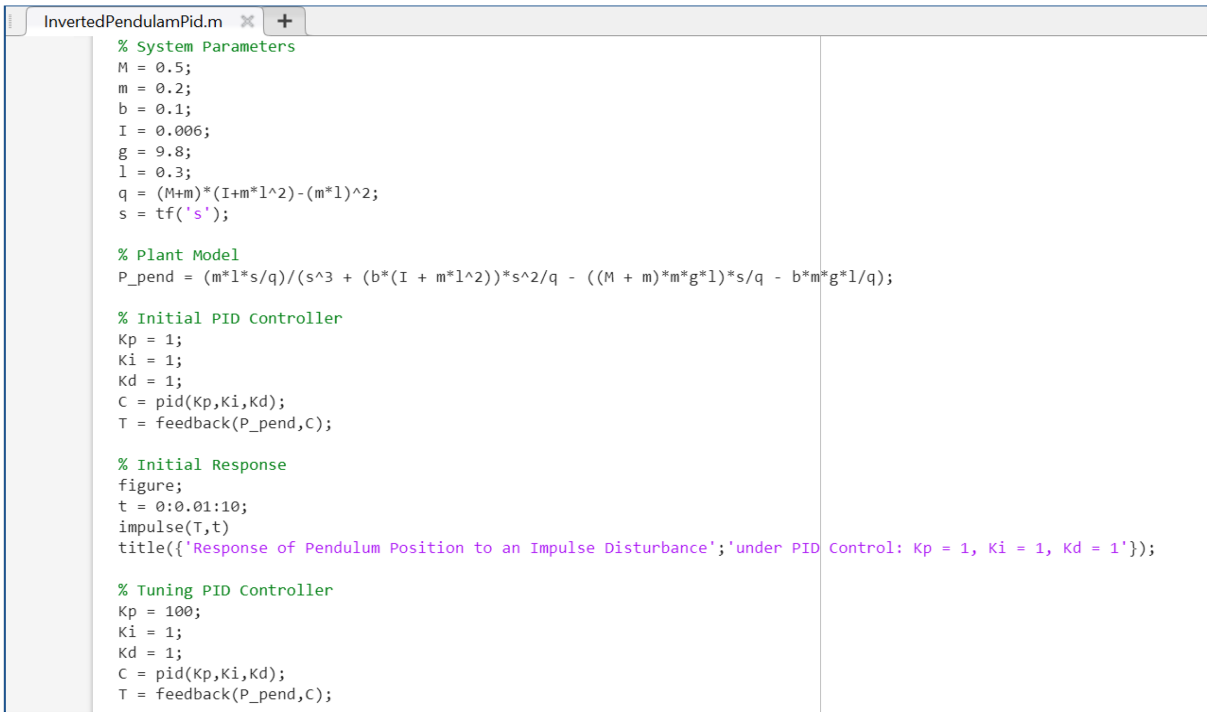
Disadvantages

- **Limited by Physical Constraints:** The controller stabilizes the pendulum but may not prevent the cart from drifting, which is not feasible in a physical implementation.
- **Sensitivity to Parameters:** The performance heavily depends on the accurate tuning of the PID gains, which may require trial and error.

Use Cases

- **Teaching and Learning:** The inverted pendulum is a classic control problem used in educational settings to demonstrate control system principles.
- **Robotics:** Balancing robots and two-wheeled vehicles use similar control strategies to maintain balance.
- **Aerospace:** Concepts from the inverted pendulum can be applied to control systems in aerospace for attitude control.

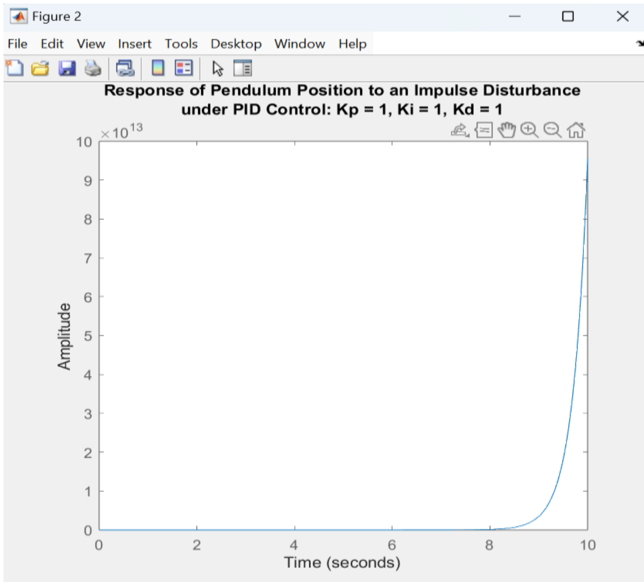
Code:

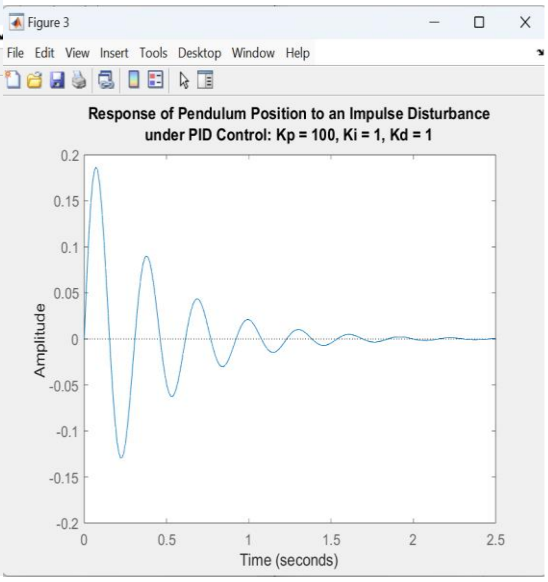


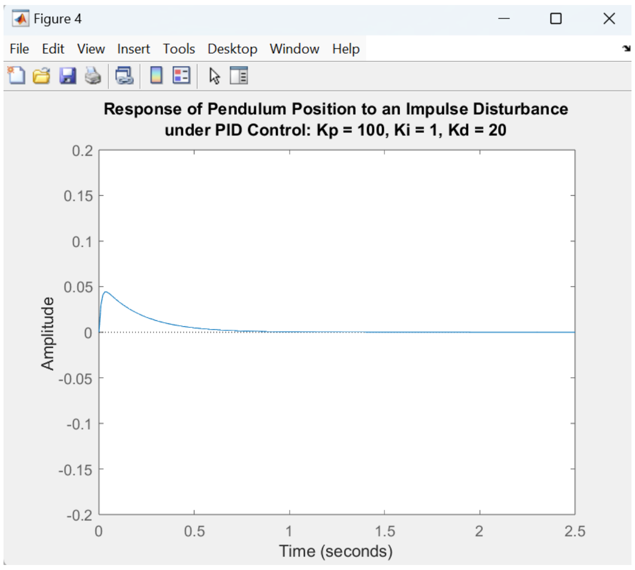
```matlab
% System Parameters
M = 0.5;
m = 0.2;
b = 0.1;
I = 0.006;
g = 9.8;
l = 0.3;
q = (M+m)*(I+m*l^2)-(m*l)^2;
s = tf('s');

% Plant Model
P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((M + m)*m*g*l)*s/q - b*m*g*l/q);

% Initial PID Controller
Kp = 1;
Ki = 1;
Kd = 1;
C = pid(Kp,Ki,Kd);
T = feedback(P_pend,C);

% Initial Response
figure;
t = 0:0.01:10;
impulse(T,t)
title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 1, Ki = 1, Kd = 1'});

% Tuning PID Controller
Kp = 100;
Ki = 1;
Kd = 1;
C = pid(Kp,Ki,Kd);
T = feedback(P_pend,C);
```

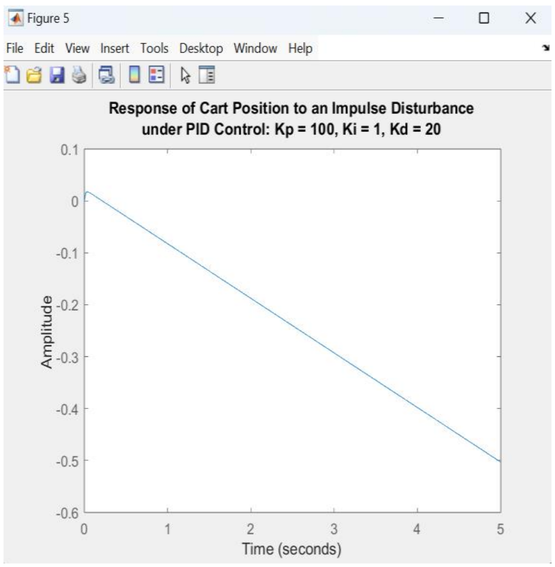
```
figure;
t = 0:0.01:10;
impulse(T,t)
axis([0, 2.5, -0.2, 0.2]);
title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 1'});

% Further Tuning with Derivative Gain
Kp = 100;
Ki = 1;
Kd = 20;
C = pid(Kp,Ki,Kd);
T = feedback(P_pend,C);

figure;
t = 0:0.01:10;
impulse(T,t)
axis([0, 2.5, -0.2, 0.2]);
title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 20'});

% Cart Position Analysis
P_cart = (((I+m*l^2)/q)*s^2 - (m*g*l/q))/(s^4 + (b*(I + m*l^2))*s^3/q - ((M + m)*m*g*l)*s^2/q - b*m*g*l*s/q);
T2 = feedback(1,P_pend*C)*P_cart;

figure;
t = 0:0.01:5;
impulse(T2, t);
title({'Response of Cart Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 20'});
```

Output:









7 .Aircraft Pitch: PID Controller Design

Introduction

In this report, we explore the design of a PID controller to regulate the pitch angle of an aircraft. The system is modeled as a third-order transfer function representing the dynamics of the aircraft pitch. The objective is to design a controller that meets specific performance criteria, including overshoot, rise time, settling time, and steady-state error, for a step input of 0.2 radians.

System Modeling

The open-loop transfer function for the aircraft pitch dynamics is given by:

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s}$$

where Θ(s) is the pitch angle of the aircraft, and Δ(s) is the elevator deflection angle.

The goal is to design a PID controller to regulate the pitch angle based on this transfer function.

Design Requirements

- **Overshoot:** Less than 10%
- **Rise time:** Less than 2 seconds
- **Settling time:** Less than 10 seconds
- **Steady-state error:** Less than 2%

Controller Design

Proportional Control

A proportional controller of the form C(s)=Kp was first implemented. The Control System Designer tool in MATLAB was used to analyze and tune the proportional gain Kp. Initial tuning with Kp=1.1269 met the rise time requirement, but the settling time was too large. Proportional control alone did not provide sufficient control over the system's performance.

PI Control

To reduce steady-state error and improve system performance, a PI controller was designed. The PI controller transfer function is given by:

$$C(s) = 0.026294\frac{1+43s}{s} \approx \frac{0.0263}{s} + 1.13$$

While the PI controller helped reduce the steady-state error more quickly, it did not significantly improve the system's oscillation and settling time.

PID Control

Finally, a PID controller was designed to meet all the performance requirements. The transfer function for the PID controller is:

$$C(s) = 1.74\frac{1+2.98s+1.716s^2}{s} \approx \frac{1.74}{s} + 5.1852 + 2.98s$$

This controller successfully met all design criteria:

- **Overshoot:** 7.5% (< 10%)
- **Rise time:** 0.413 seconds (< 2 seconds)
- **Settling time:** 9.25 seconds (< 10 seconds)
- **Steady-state error:** 0% (< 2%)

Advantages of PID Control

- **Versatility:** PID controllers can be applied to a wide range of processes and systems.
- **Ease of Use:** PID controllers are relatively easy to understand and implement.

- **Robustness:** They can handle system uncertainties and external disturbances effectively.
- **Automatic Tuning:** Tools like MATLAB's Control System Designer simplify the tuning process.

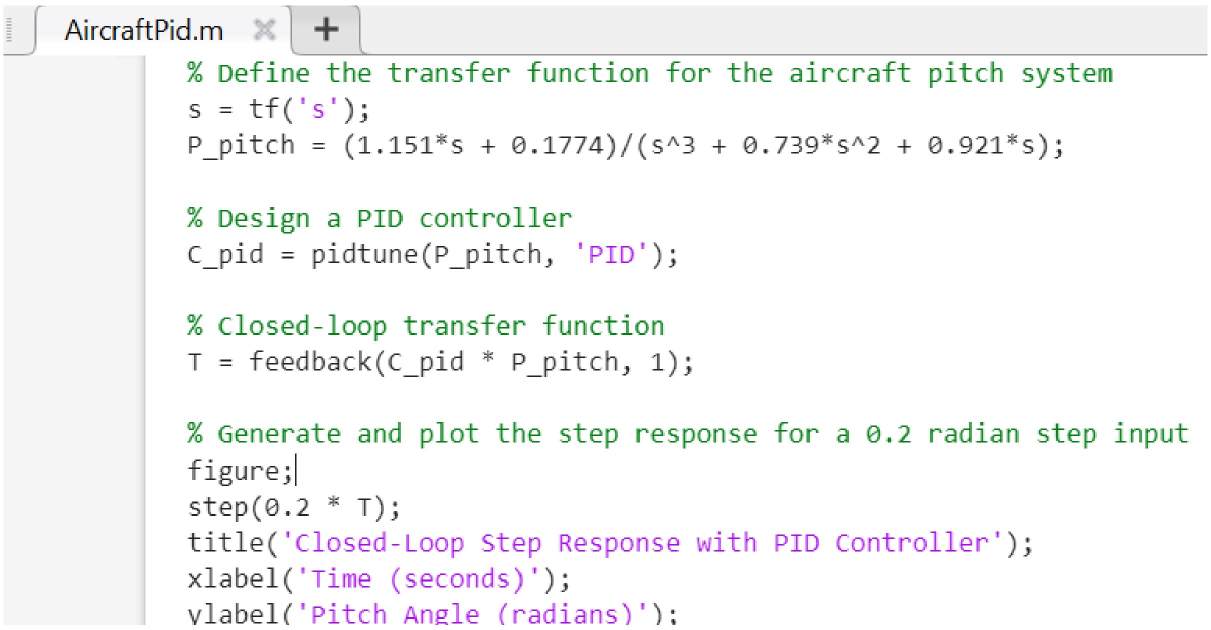
Disadvantages of PID Control

- **Complex Tuning:** Tuning PID parameters for optimal performance can be challenging, especially for higher-order systems.
- **Overreliance on Manual Tuning:** Despite automation, some systems may require manual tuning and experience to achieve the best performance.
- **Oscillations:** Poorly tuned PID controllers can introduce oscillations into the system.

Use Cases of PID Control in Aircraft Pitch

- **Stabilization:** Maintaining a stable pitch angle during flight operations, especially during takeoff, landing, and turbulence.
- **Autopilot Systems:** Ensuring smooth and accurate pitch control in autopilot systems.
- **Training Simulators:** Providing realistic control feedback for pilot training simulators.

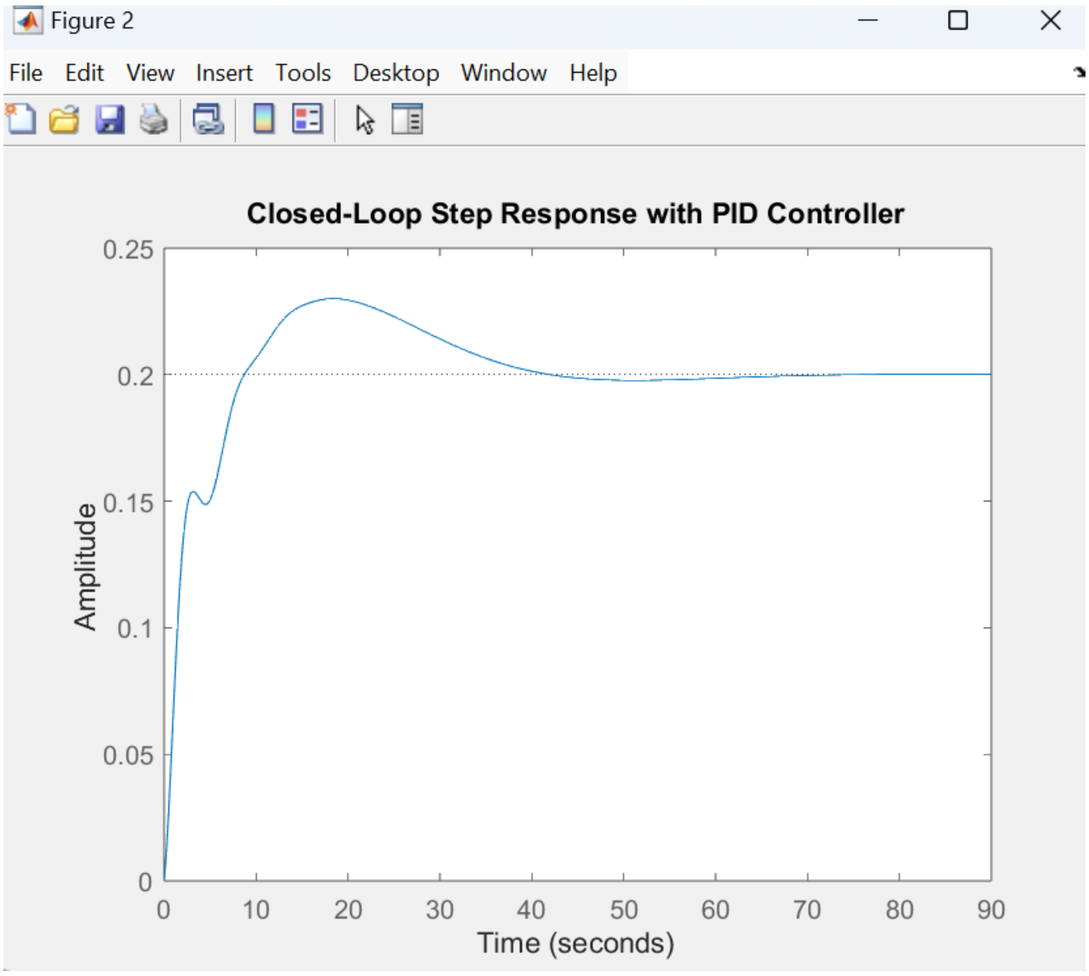
Code:



AircraftPid.m

```
% Define the transfer function for the aircraft pitch system
s = tf('s');
P_pitch = (1.151*s + 0.1774)/(s^3 + 0.739*s^2 + 0.921*s);

% Design a PID controller
C_pid = pidtune(P_pitch, 'PID');

% Closed-loop transfer function
T = feedback(C_pid * P_pitch, 1);

% Generate and plot the step response for a 0.2 radian step input
figure;
step(0.2 * T);
title('Closed-Loop Step Response with PID Controller');
xlabel('Time (seconds)');
ylabel('Pitch Angle (radians)');
```

Output:



8 .Ball & Beam: PID Controller Design

Introduction

The Ball & Beam system is a classic control problem that demonstrates the principles of PID (Proportional-Integral-Derivative) control. The goal is to maintain the position of a ball on a beam by adjusting the beam's angle. This project focuses on designing a PID controller to meet specific design criteria using MATLAB.

System Overview

The open-loop transfer function of the Ball & Beam system is given by:

$$P(s) = \frac{R(s)}{\Theta(s)} = -\frac{mgd}{L\left(\frac{J}{R^2}+m\right)}\frac{1}{s^2} \qquad [\frac{m}{rad}]$$

Where:

- m is the mass of the ball,
- g is the acceleration due to gravity,
- d is the distance from the pivot to the point of contact,
- L is the length of the beam,
- J is the moment of inertia,
- R is the radius of the ball.

Closed-Loop Representation

The closed-loop system is formed by applying a feedback loop with the PID controller. The block diagram of the system with unity feedback is considered.

Controller Design

Proportional Control

Using a proportional gain Kp, the system's response is tested. Initially, the proportional gain is set to 1. However, the system remains marginally stable and does not meet the design criteria.

Proportional-Derivative Control

To improve the system's performance, a derivative term is added to the controller. By tuning the gains Kp and Kd, the system meets the design criteria of settling time less than 3 seconds and overshoot less than 5%.

Advantages of PID Control

1. **Simplicity**: PID controllers are relatively simple to design and implement.
2. **Robustness**: They are robust to model uncertainties and can handle a variety of system dynamics.
3. **Widely Applicable**: PID control is applicable to a broad range of systems, including mechanical, electrical, and chemical processes.
4. **Automated Tuning**: Many modern tools, like MATLAB, offer automated tuning for PID controllers, making the design process easier.

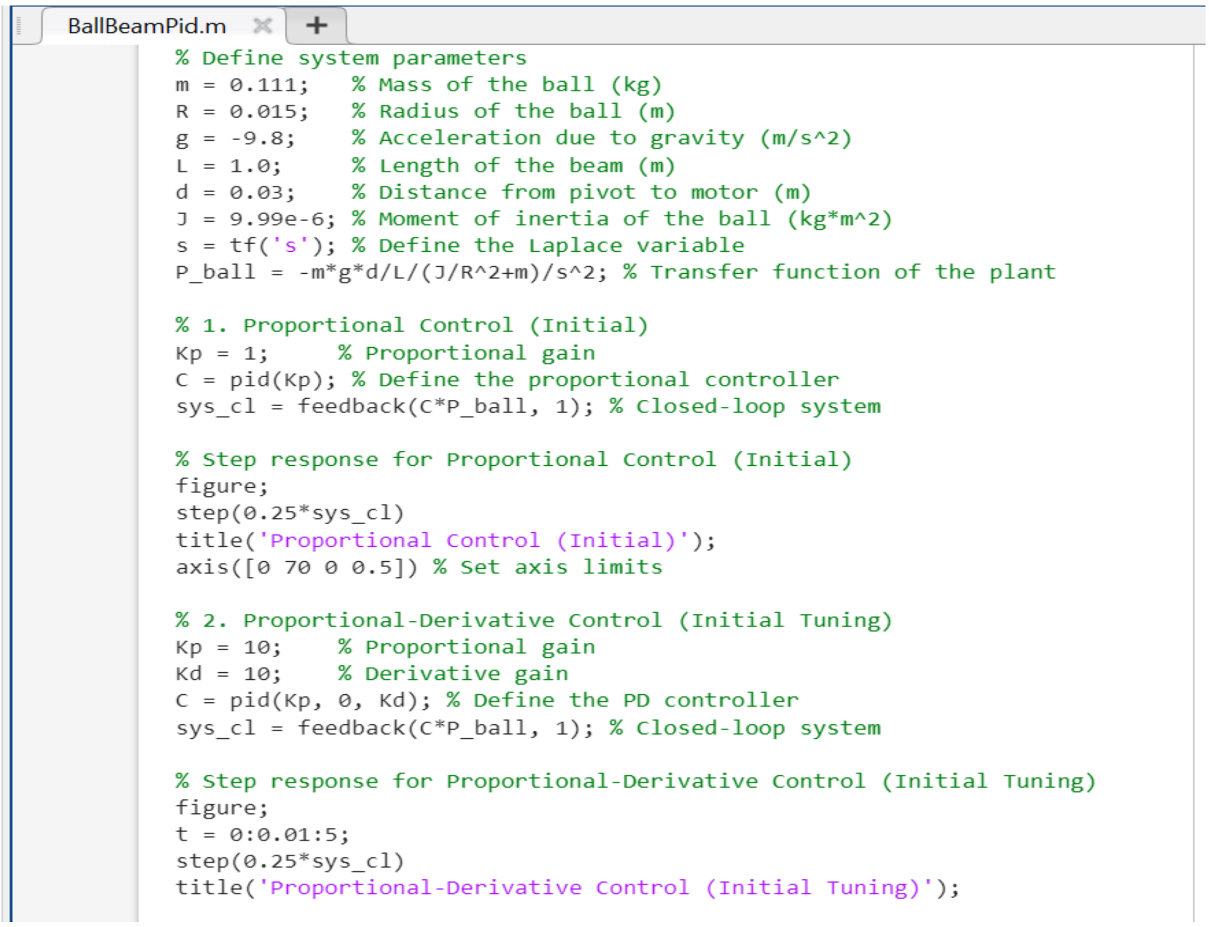
Disadvantages of PID Control

1. **Tuning Complexity**: While simple, tuning PID controllers can be challenging, especially for systems with complex dynamics.
2. **Sensitivity to Noise**: The derivative term in PID control can amplify noise in the system, leading to undesirable oscillations.
3. **Limited Control**: PID controllers may not be suitable for highly nonlinear or time-varying systems without additional modifications or advanced control strategies.

Use Cases of PID Control

1. **Industrial Automation**: PID controllers are extensively used in industrial processes, such as temperature control, pressure control, and speed control of motors.
2. **Robotics**: PID control is employed in robotic arms and vehicles to maintain stability and follow desired paths.
3. **Aerospace**: In aerospace engineering, PID controllers are used for autopilot systems, including maintaining the pitch, roll, and yaw of an aircraft.
4. **Consumer Electronics**: Devices like thermostats and washing machines often use PID control to maintain desired operating conditions.

Code:

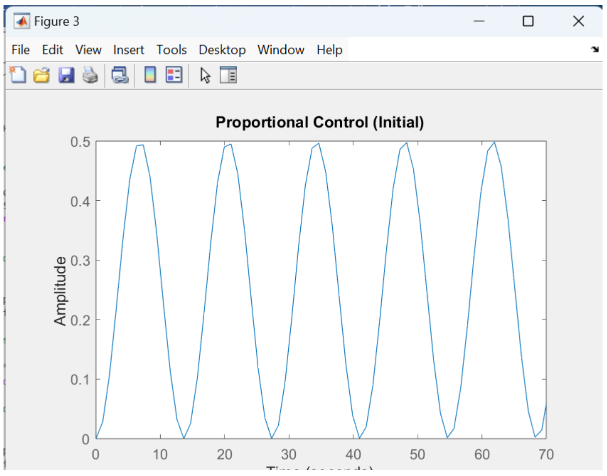


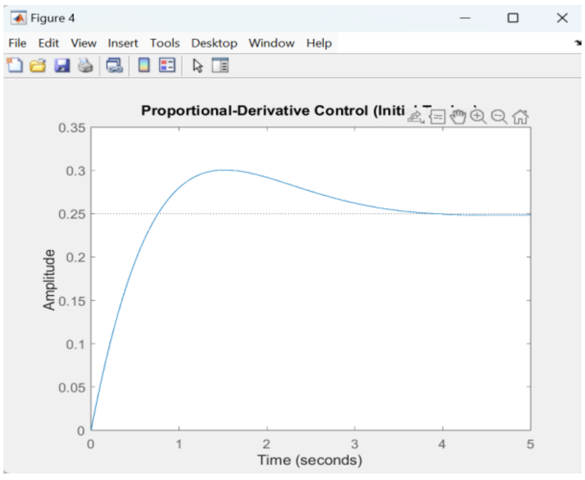
BallBeamPid.m

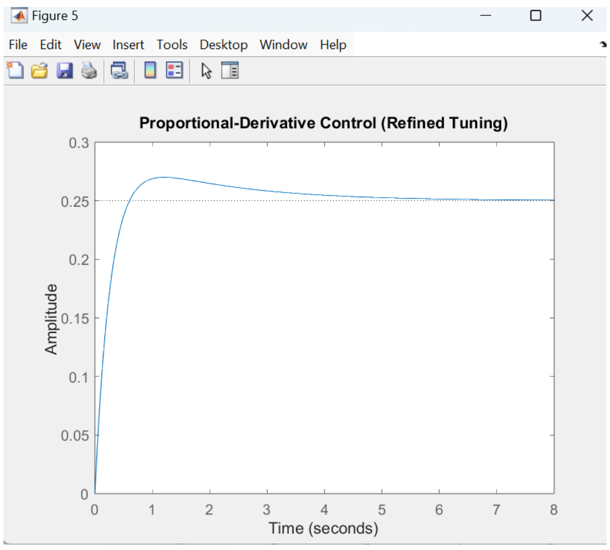
```
% Define system parameters
m = 0.111;   % Mass of the ball (kg)
R = 0.015;   % Radius of the ball (m)
g = -9.8;    % Acceleration due to gravity (m/s^2)
L = 1.0;     % Length of the beam (m)
d = 0.03;    % Distance from pivot to motor (m)
J = 9.99e-6; % Moment of inertia of the ball (kg*m^2)
s = tf('s'); % Define the Laplace variable
P_ball = -m*g*d/L/(J/R^2+m)/s^2; % Transfer function of the plant

% 1. Proportional Control (Initial)
Kp = 1;     % Proportional gain
C = pid(Kp); % Define the proportional controller
sys_cl = feedback(C*P_ball, 1); % Closed-loop system

% Step response for Proportional Control (Initial)
figure;
step(0.25*sys_cl)
title('Proportional Control (Initial)');
axis([0 70 0 0.5]) % Set axis limits

% 2. Proportional-Derivative Control (Initial Tuning)
Kp = 10;    % Proportional gain
Kd = 10;    % Derivative gain
C = pid(Kp, 0, Kd); % Define the PD controller
sys_cl = feedback(C*P_ball, 1); % Closed-loop system

% Step response for Proportional-Derivative Control (Initial Tuning)
figure;
t = 0:0.01:5;
step(0.25*sys_cl)
title('Proportional-Derivative Control (Initial Tuning)');
```

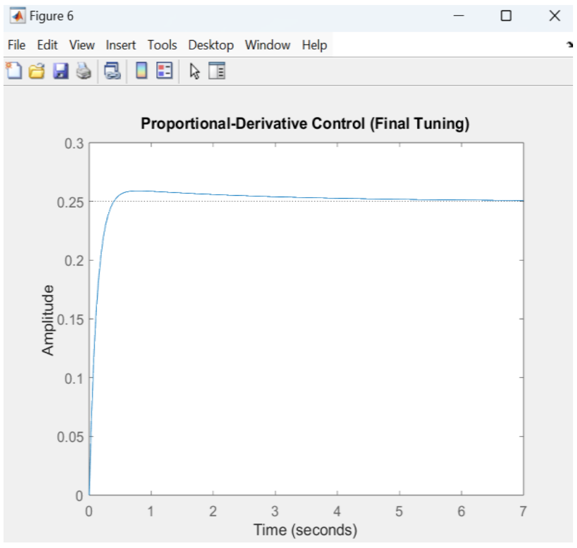
```
% 3. Proportional-Derivative Control (Refined Tuning)
Kp = 10;     % Proportional gain
Kd = 20;     % Refined Derivative gain
C = pid(Kp, 0, Kd); % Redefine the PD controller
sys_cl = feedback(C*P_ball, 1); % Closed-loop system

% Step response for Proportional-Derivative Control (Refined Tuning)
figure;
step(0.25*sys_cl)
title('Proportional-Derivative Control (Refined Tuning)');

% 4. Proportional-Derivative Control (Final Tuning)
Kp = 15;     % Final Proportional gain
Kd = 40;     % Final Derivative gain
C = pid(Kp, 0, Kd); % Redefine the PD controller
sys_cl = feedback(C*P_ball, 1); % Closed-loop system

% Step response for Proportional-Derivative Control (Final Tuning)
figure;
step(0.25*sys_cl)
title('Proportional-Derivative Control (Final Tuning)');
```

Output:









Referred Link:
https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID

Github repository :

https://github.com/Madduru-ManiTeja/PIDControlSystem