CONTROL SYSTEMS – EC5030

LABORATORY SESSION 2

MODELING CONTROL SYSTEMS IN MATLAB SIMULINK

WIMALASOORIYA G.H.N.P.D.

2022/E/039

GROUP CG 03

SEMESTER 05

09 MAY 2025

PART 1 – MODELING AND ANALYSING A SIMPLE CONTROL SYSTEM USING MATLAB SIMULINK

APPARATUS

- MATLAB with Simulink
- Simulink Library blocks: Sum, Gain, Integrator, Constant, Step, Scope, MUX

PROCEDURE

1.Open Simulink Library Browser:

o Start MATLAB, open Simulink Library Browser.
o Explore “Math,” “Source,” “Sink,” and “Continuous” libraries.
o Familiarize with the following blocks:

- Sum: Configurable for any number of inputs and signs.
- Gain: Set gain value via dialog box.
- Integrator: Initial condition can be set explicitly or implicitly.
- Constant: Set constant value via dialog box.
- Step: Configure step level and time.
- Scope: Visualize system response; use MUX to display multiple signals

3. Model a Second Order System:

- Build the open-loop system as per the provided transfer function.
- Applied a Step input and observed output on Scope.
- Used “Autoscale” for proper visualization.

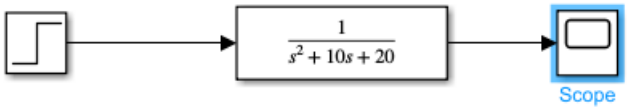


FIG 01: SECOND ORDER OPEN LOOP SYSTEM

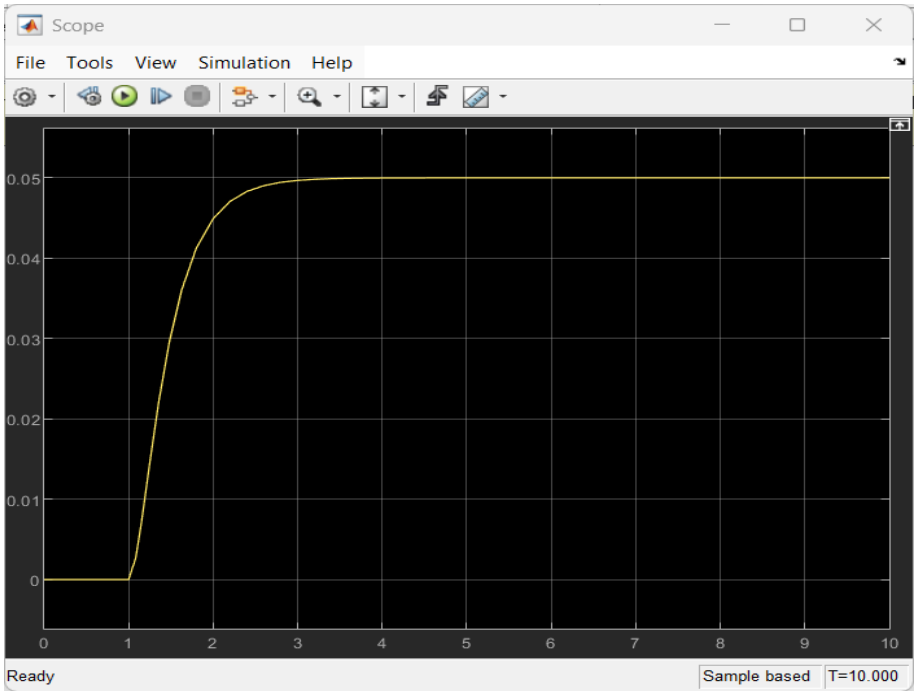


FIG 02: OUTPUT OF OPEN LOOP SECOND ORDER OPEN LOOP SYSTEM

4. Model a Closed Loop System with Proportional Controller:

- Added a proportional controller block with Kp=100.
- Observed and plotted the output waveform.
- Changed Kp to 300, re-run, and observed the effect.

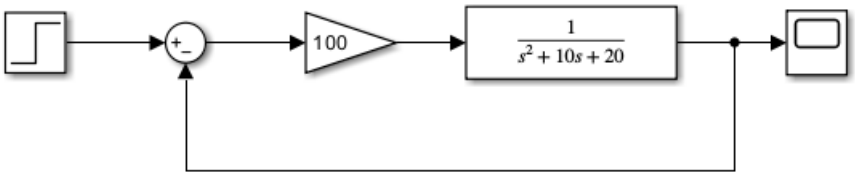


FIG 03: PLANT WITH PROPORTIONAL CONTROLLER KP = 100

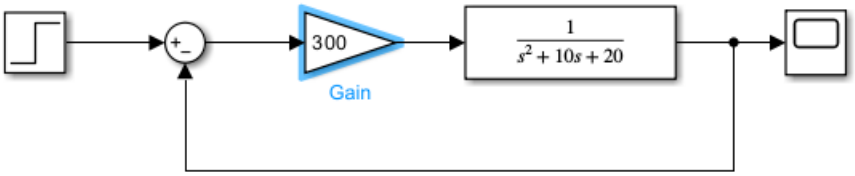


FIG 04: PLANT WITH PROPORTIONAL CONTROLLER Kp = 300

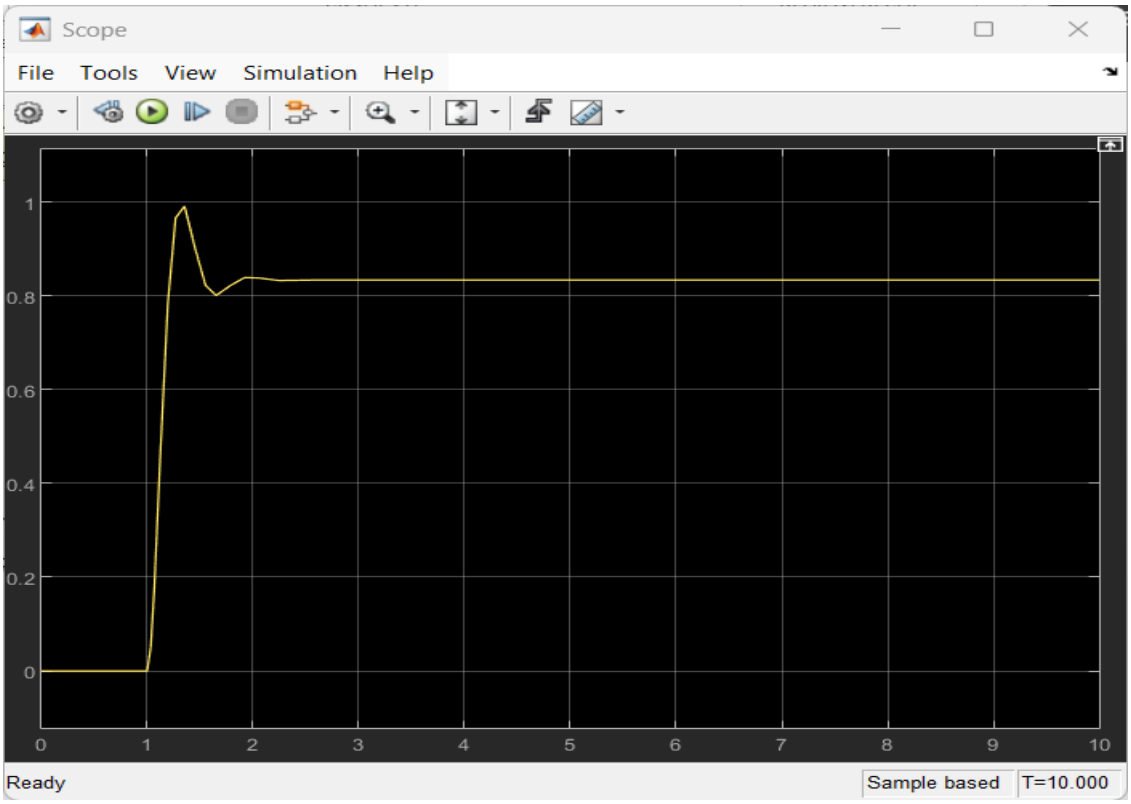


FIG 05: OUTPUT OF PLANT WITH PROPORTIONAL CONTROLLER Kp = 100

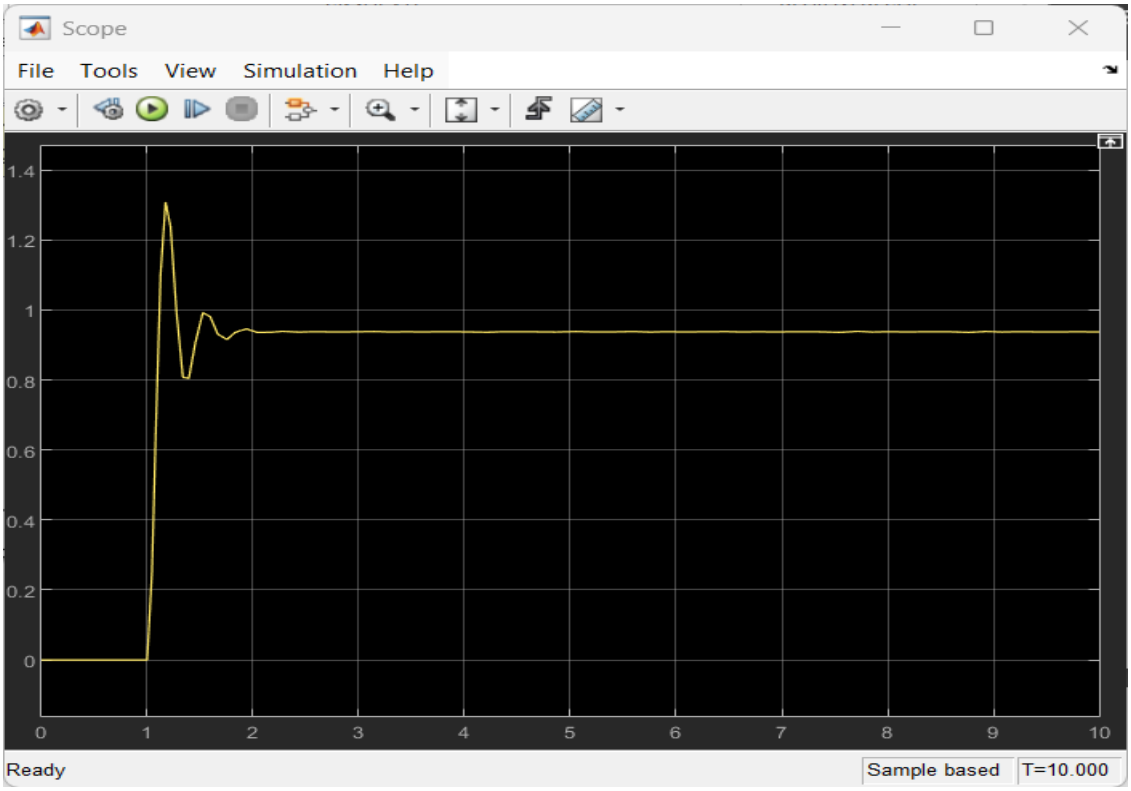


FIG 06: OUTPUT OF PLANT WITH PROPORTIONAL CONTROLLER Kp = 300

5. PI Controller:

- Added an integrator in the controller path (PI structure).
- Vary Kp from 70 to 100, observed and plotted the output.

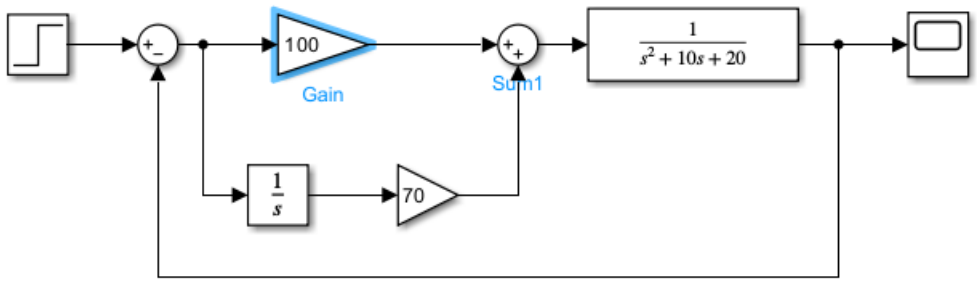


FIG 07: CLOSED LOOP SYSTEM WITH PI CONTROLLER

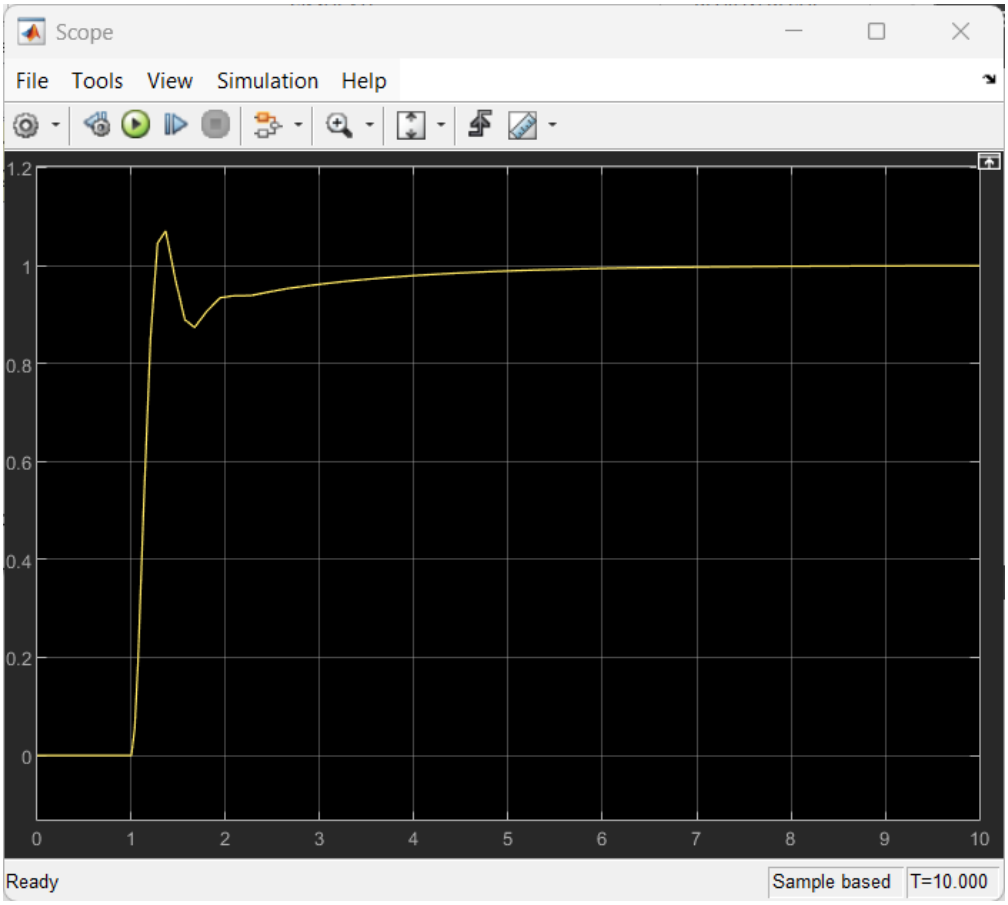


FIG 08: CLOSED LOOP SYSTEM WITH PI CONTROLLER Kp = 70

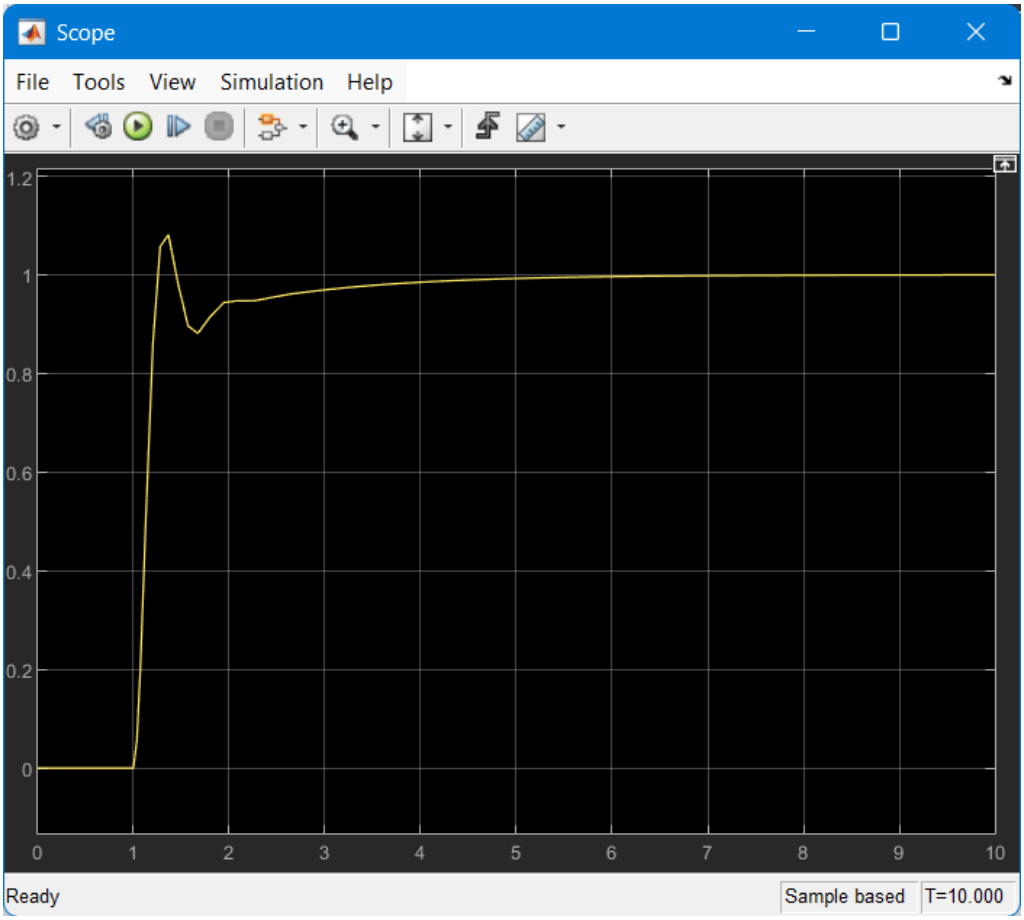


FIG 09: CLOSED LOOP SYSTEM WITH PI CONTROLLER Kp = 80

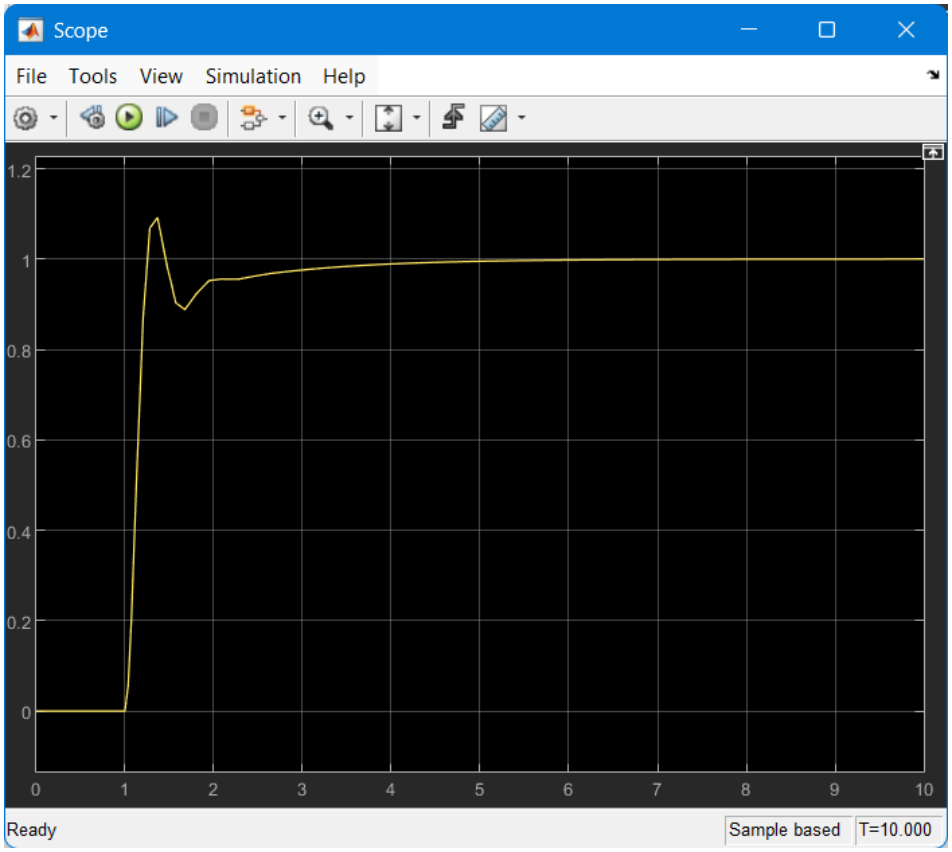


FIG 10: CLOSED LOOP SYSTEM WITH PI CONTROLLER Kp = 90

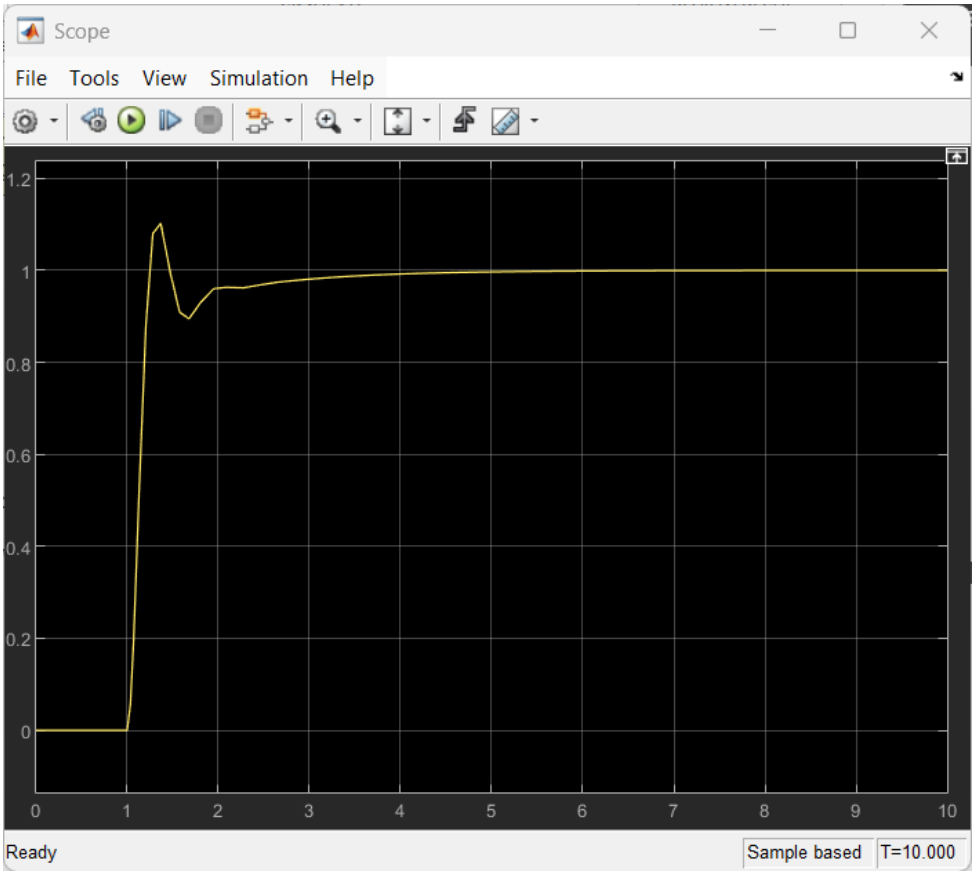


FIG 11: CLOSED LOOP SYSTEM WITH PI CONTROLLER Kp = 100

6. PD Controller:

- Added a differentiator in the controller path (PD structure).
- Vary Kd from 10 to 20, observed and plotted the output.

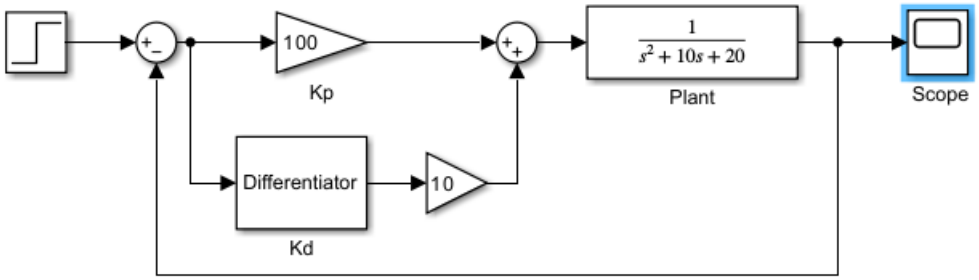


FIG 12: CLOSED LOOP SYSTEM WITH PD CONTROLLER

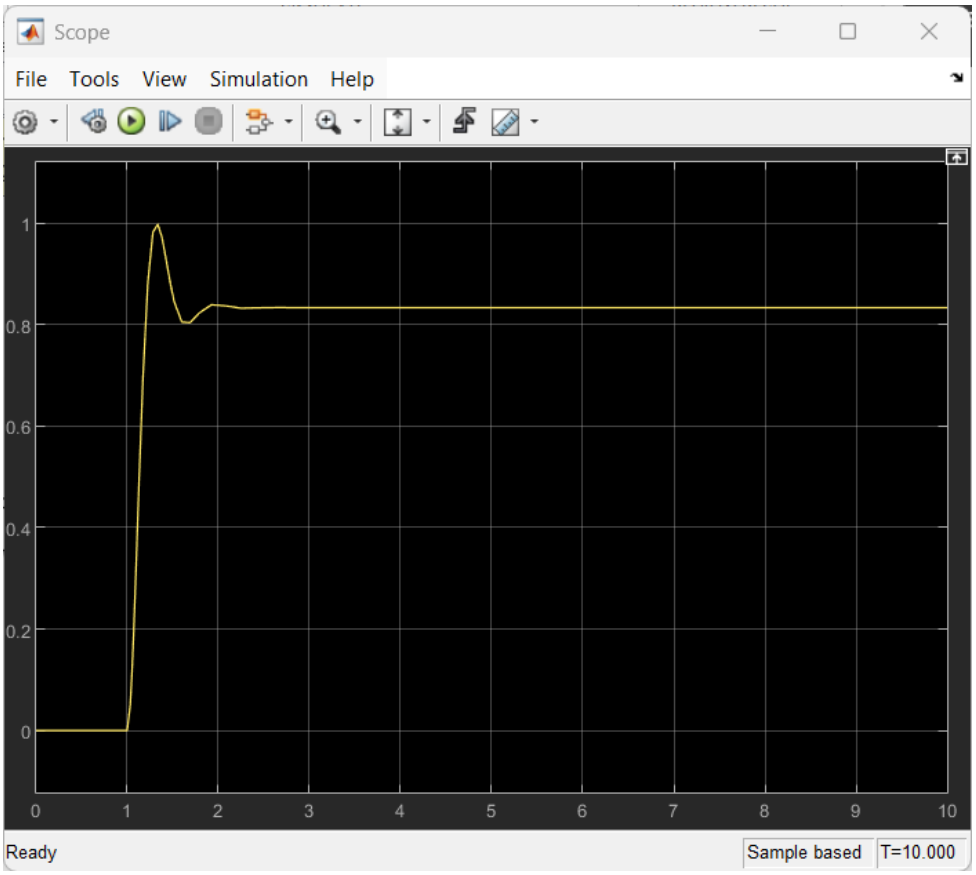


FIG 13: CLOSED LOOP SYSTEM WITH PD CONTROLLER KI = 10

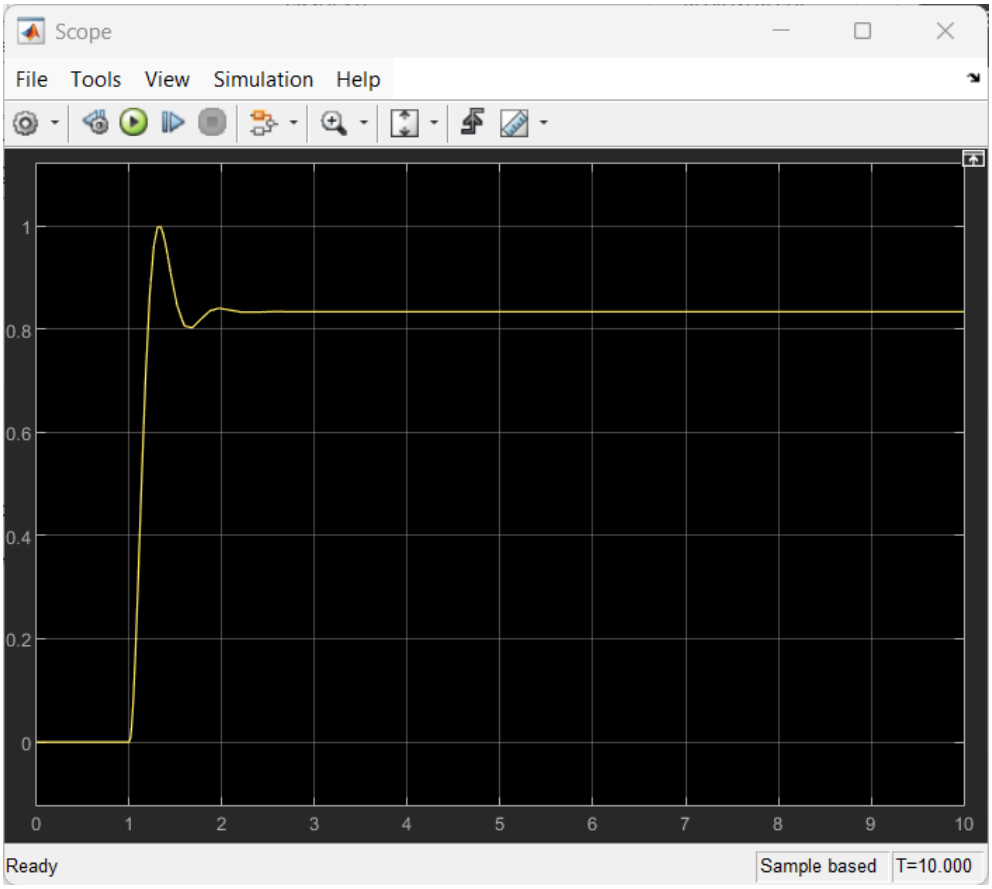


FIG 14: CLOSED LOOP SYSTEM WITH PD CONTROLLER KI = 20

7. PID Controller:

- Combined proportional, integral, and derivative actions.
- Set Kp=100, Ki=10,Kd=70.
- Use a MUX to plot both input and output on the same Scope.

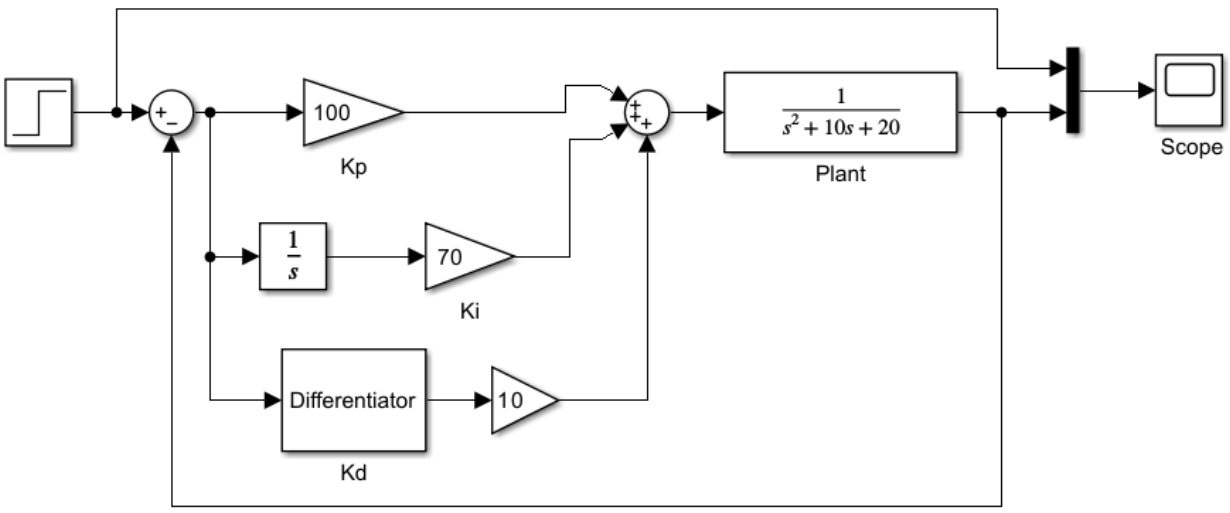


FIG 15: CLOSED LOOP SYSTEM WITH PID CONTROLLER (DIAGRAM 01)

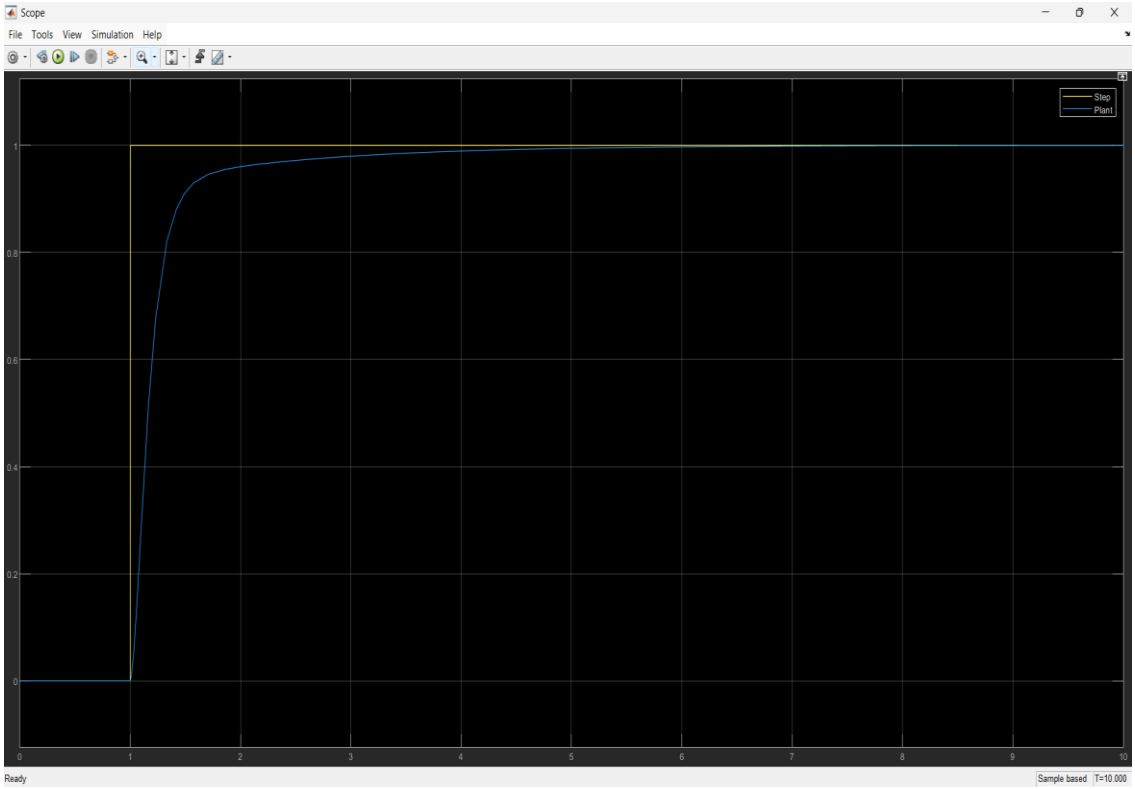


FIG 16: OUTPUT OF CLOSED LOOP SYSTEM WITH PID CONTROLLER (DIAGRAM 1)

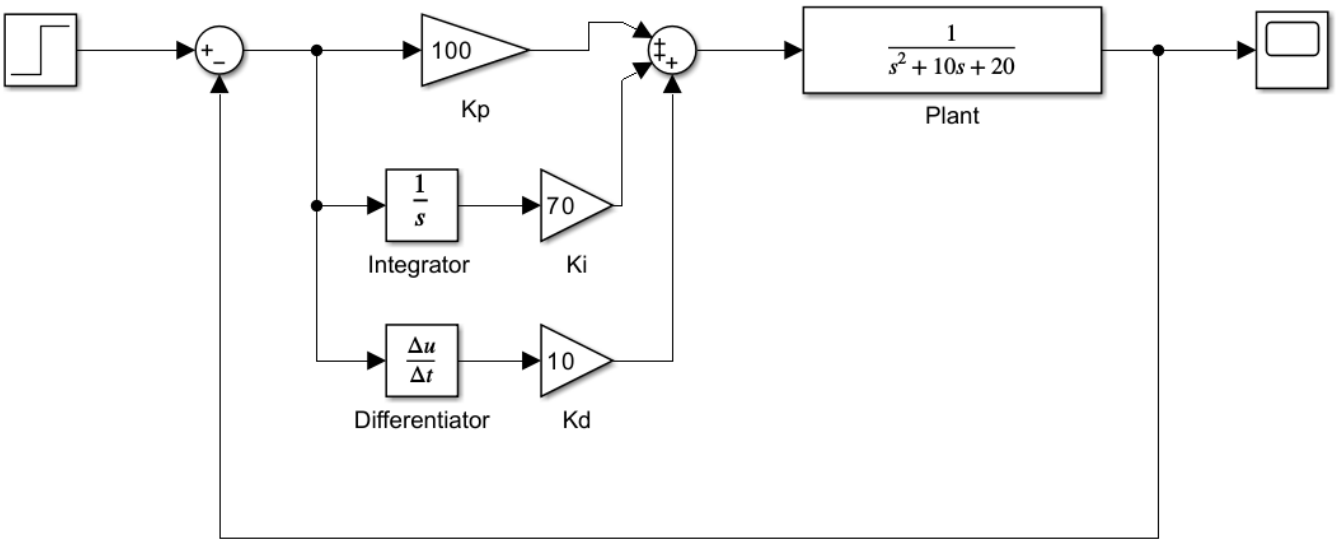


FIG 17: CLOSED LOOP SYSTEM WITH PID CONTROLLER (DIAGRAM 2)

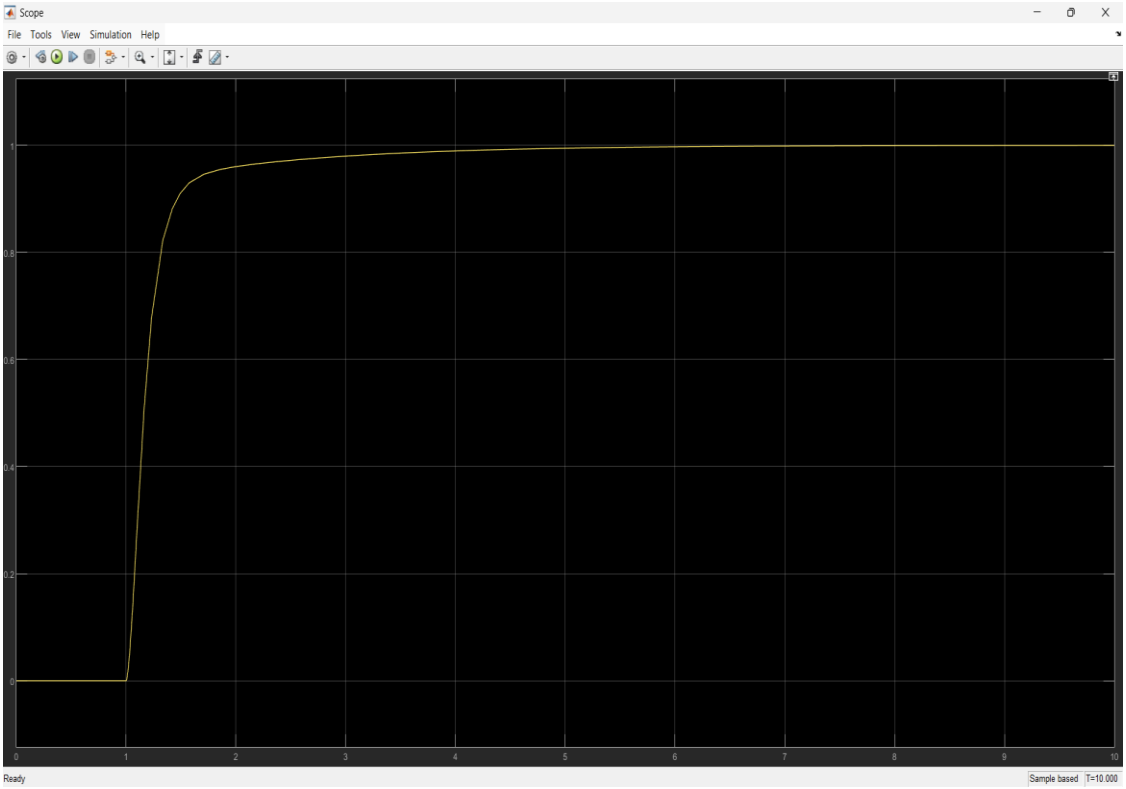


FIG 18: OUTPUT OF CLOSED LOOP SYSTEM WITH PID CONTROLLER (DIAGRAM 2)

PART 2: MODELING A DC MOTOR AND CONTROLLER

2. MATLAB Representation:

- Defined parameters in an M-file.

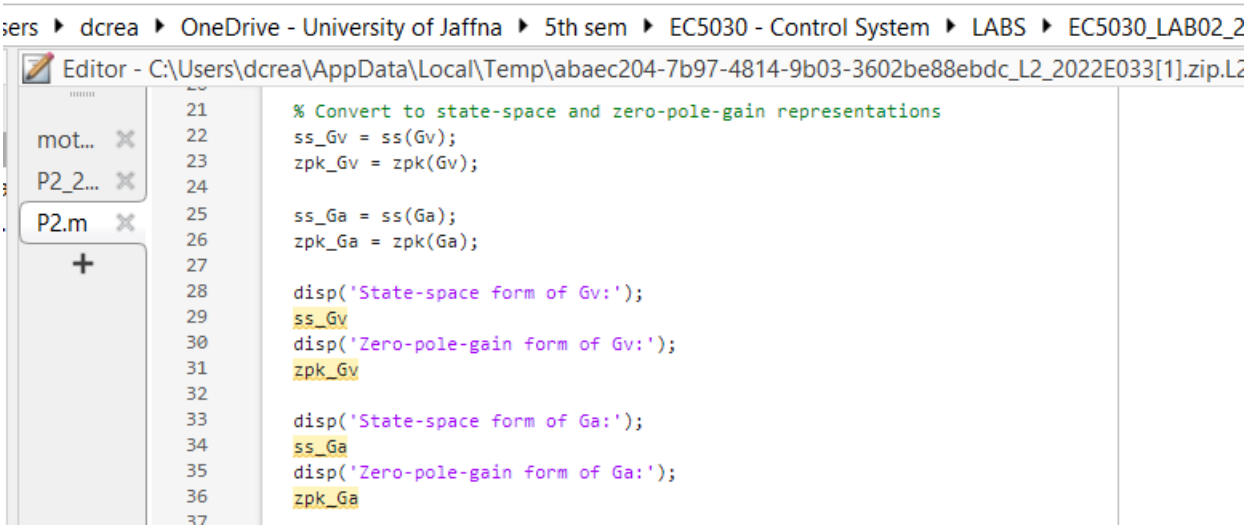
Editor - C:\Users\dcrea\OneDrive - University of Jaffna\5th sem\EC5030 - Control System\LABS\EC5030_LAB02_202

mot...

```
%WIMALASOORIYA G.H.N.P.D.
%% DC Motor Model
J=0.01;
b=0.1;
K=0.01;
R=1;
L=0.5;

%% Method(1)
aux = tf(K,conv([L R],[J b])); % or aux = tf(K,[L R])*tf(1,[J b]);
Gv = feedback(aux,K); % transfer function for voltage and angle
Ga = tf(1,[1 0])*Gv; % transfer function for voltage and angular velocity
%% Method (2)
s = tf([1 0],1);
Gv = K/((L*s + R)*(J*s + b) + K^2); % transfer function for voltage and angle
Ga = Gv/s; % transfer function for voltage and angular velocity
% Labeling the input and output
Gv.InputName = 'Voltage';
Gv.OutputName = 'Velocity';
Ga.InputName = 'Voltage';
Ga.OutputName = 'Angle';

```

3. State-Space and Zero-Pole-Gain Representation:

- Converted Gv and Ga using ss() and zpk() functions

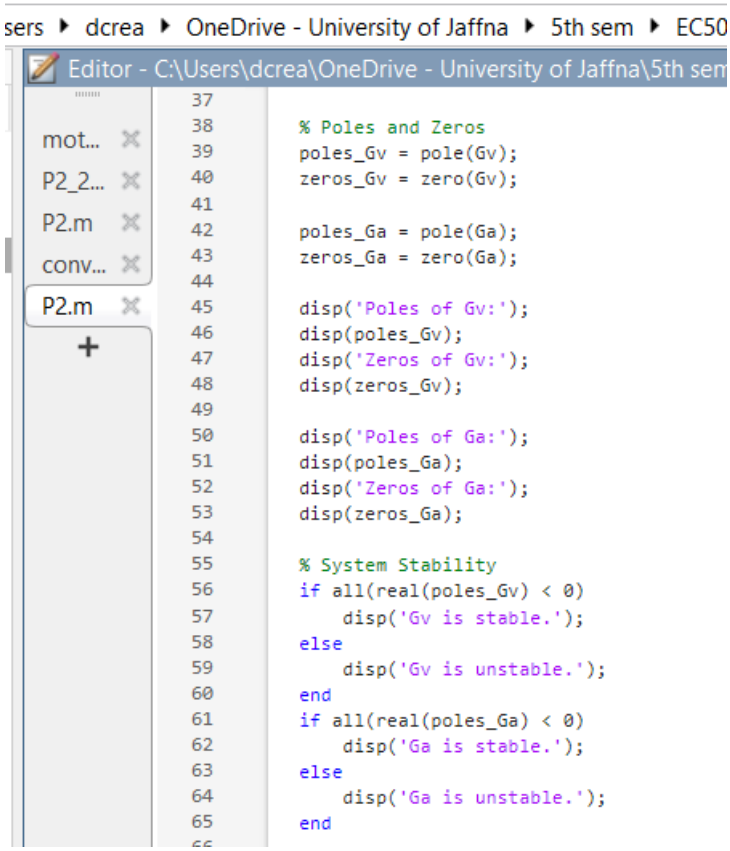



```
>> P2
State-space form of Gv:

ss_Gv =

  A =
          x1      x2
   x1     -12  -5.005
   x2       4       0

  B =
       Voltage
   x1      0.5
   x2        0

  C =
             x1  x2
   Velocity   0   1

  D =
             Voltage
   Velocity        0
```

```
ss_Ga =

  A =
          x1      x2      x3
   x1     -12  -5.005       0
   x2       4       0       0
   x3       0    0.25       0

  B =
       Voltage
   x1        2
   x2        0
   x3        0

  C =
          x1  x2  x3
   Angle   0   0   1

  D =
          Voltage
   Angle        0
```

4. Poles and Zeros:

- Used pole() and zero() to find system poles and zeros.
- Assess stability: System is stable if all poles have negative real parts.




```
Model Properties
Poles of Gv:
   -9.9975
   -2.0025

Zeros of Gv:
Poles of Ga:
         0
   -9.9975
   -2.0025

Zeros of Ga:
Gv is stable.
Ga is unstable.
```

5. Controllability and Observability:

- Used ctrb() and obsv() to compute controllability and observability matrices.

```
76  % Controllability and Observability
77  [A, B, C, D] = tf2ss(Gv.num{1}, Gv.den{1});
78  Co = ctrb(A, B);
79  Ob = obsv(A, C);
80
81  is_controllable = rank(Co) == size(A,1);
82  is_observable = rank(Ob) == size(A,1);
83
84  disp(['Is the system controllable? ', mat2str(is_controllable)]);
85  disp(['Is the system observable? ', mat2str(is_observable)]);
```

```
Is the system controllable? true
Is the system observable? true
>>
```

6. Time and Frequency Domain Responses: ‘

- Plotted step, impulse, and Bode responses:

```
87  % Time and Frequency Domain Responses
88  figure(1);
89  step(Ga);
90  hold on;
91  step(Gv);
92  title('Step Responses');
93  legend('Angle (Ga)', 'Velocity (Gv)');
94  grid on;
95
96  figure(2);
97  impulse(Gv);
98  hold on;
99  impulse(Ga);
100 title('Impulse Responses');
101 legend('Velocity (Gv)', 'Angle (Ga)');
102 grid on;
103
104 figure(3);
105 bode(Ga);
106 hold on;
107 bode(Gv);
108 title('Bode Plots');
109 legend('Angle (Ga)', 'Velocity (Gv)');
110 grid on;
```

7. PID Controller Design: • Design PID controller:

- Used rlocus() and rlocfind() to tune gains for stability and desired pole locations.

ers ▸ dcrea ▸ OneDrive - University of Jaffna ▸ 5th sem ▸ EC5030 - Control Syst

Editor - C:\Users\dcrea\AppData\Local\Temp\0e65acae-af66-4474-9d60-52ce

mot...
P2_2...
P2.m
conv...
P2.m
+

```
%% Adding a PID controller
Kp = 1;
Ki = 0.8;
Kd = 0.3;

C = tf([Kd Kp Ki],[1 0]); % (Kd*s^2+Kp*s+Ki)/s
Gc = feedback(Ga*C,1);
figure;
step(Gc);

figure;
rlocus(Ga*C);
Kp = rlocfind(Ga*C);
Gc1 = feedback(Ga*C*Kp,1);
figure(6);
step(Gc1)
```

8. Simulink Model for DC Motor:

- Build the model as per the block diagram.
- Plot step responses for both angle and angular velocity.

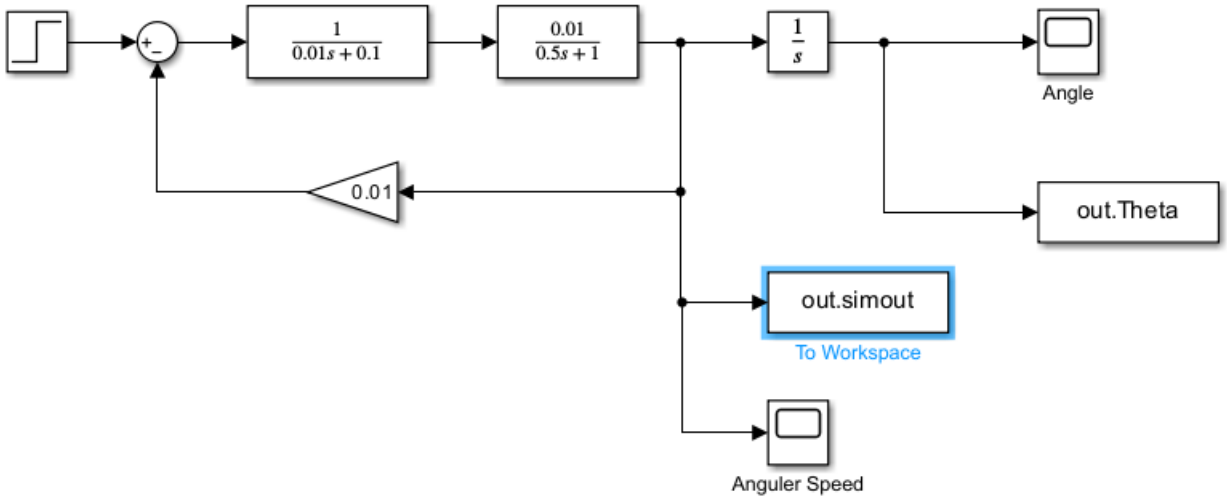


FIG 19: SIMULINK MODEL FOR DC MOTOR

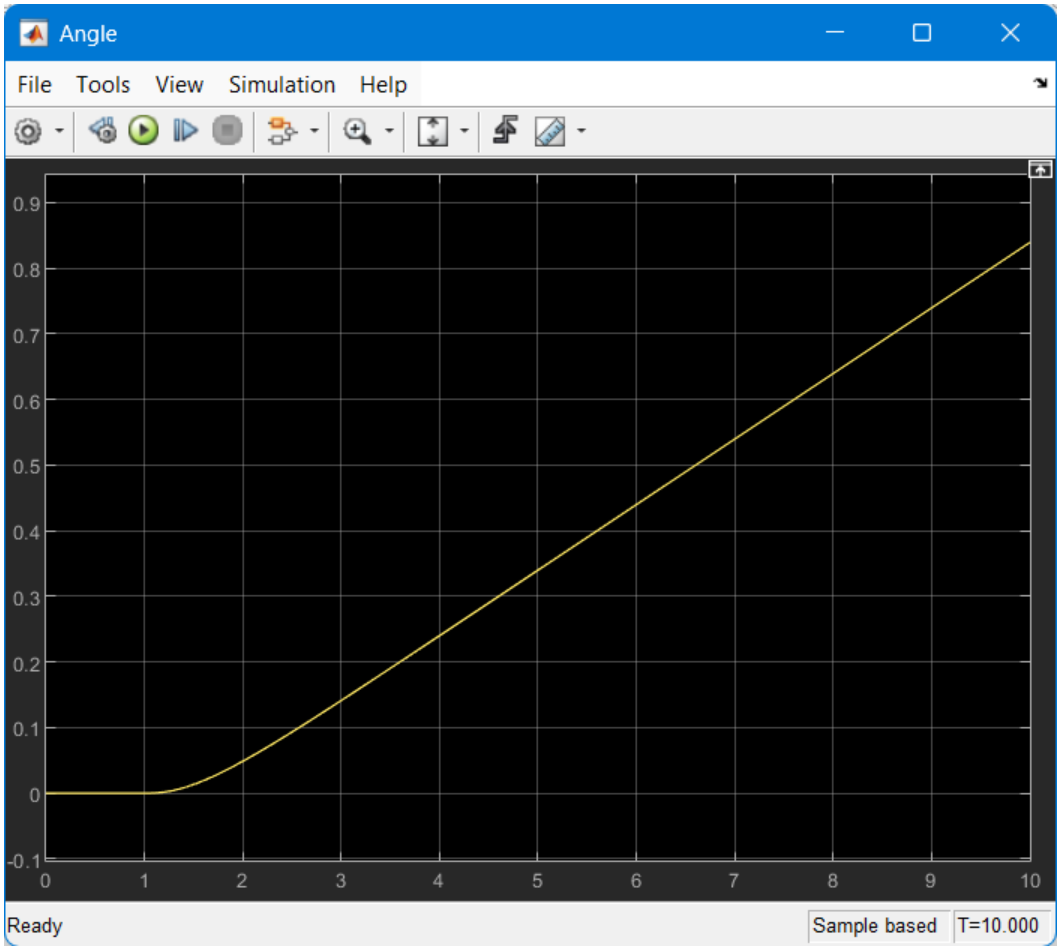


FIG 20: PLOT OF ANGLE FOR SIMULINK MODEL FOR DC MOTOR)

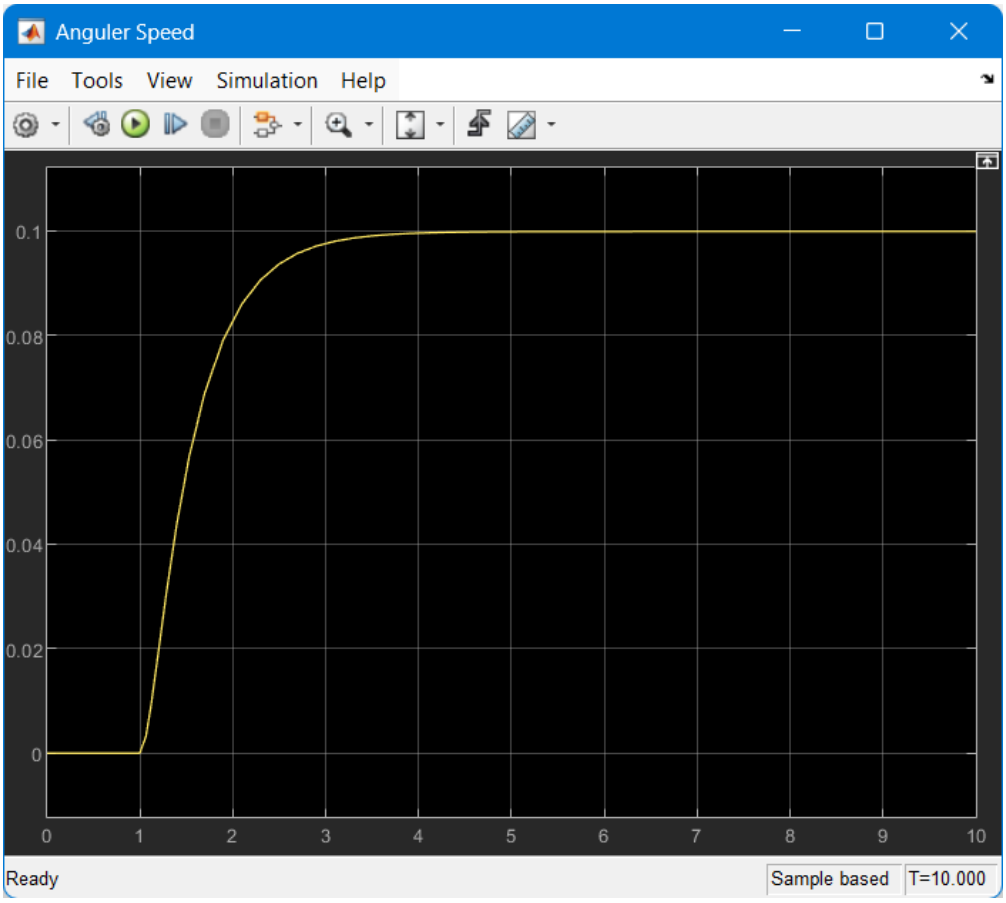


FIG 21: PLOT OF ANGLE FOR SIMULINK MODEL FOR DC MOTOR)

9. Simulink Model with PID Controller:

- Created a subsystem for the DC motor.

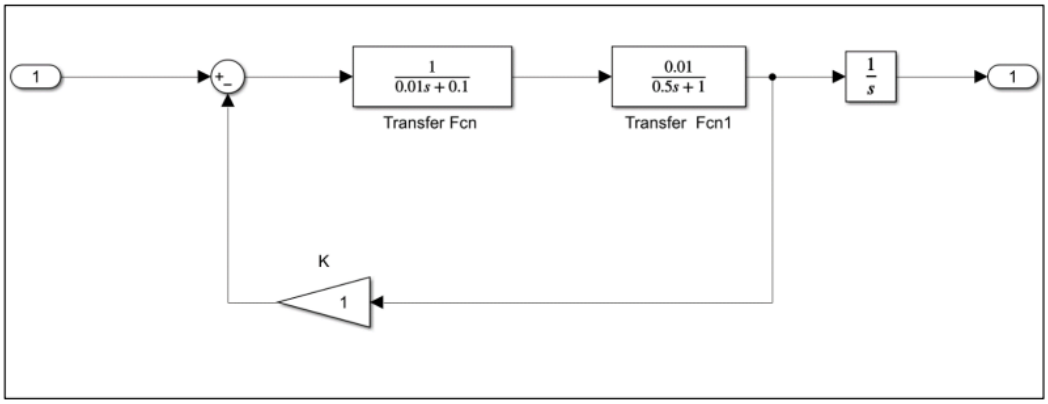


FIG 22: SUBSYSTEM FOR THE DC MOTOR.

- Implemented the PID controller to control the motor.

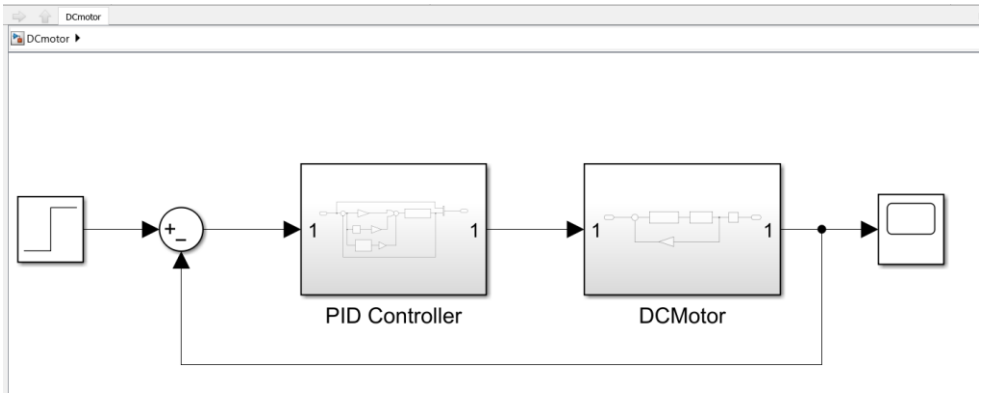


FIG 23: DC MOTOR WITH PID CONTROLLER

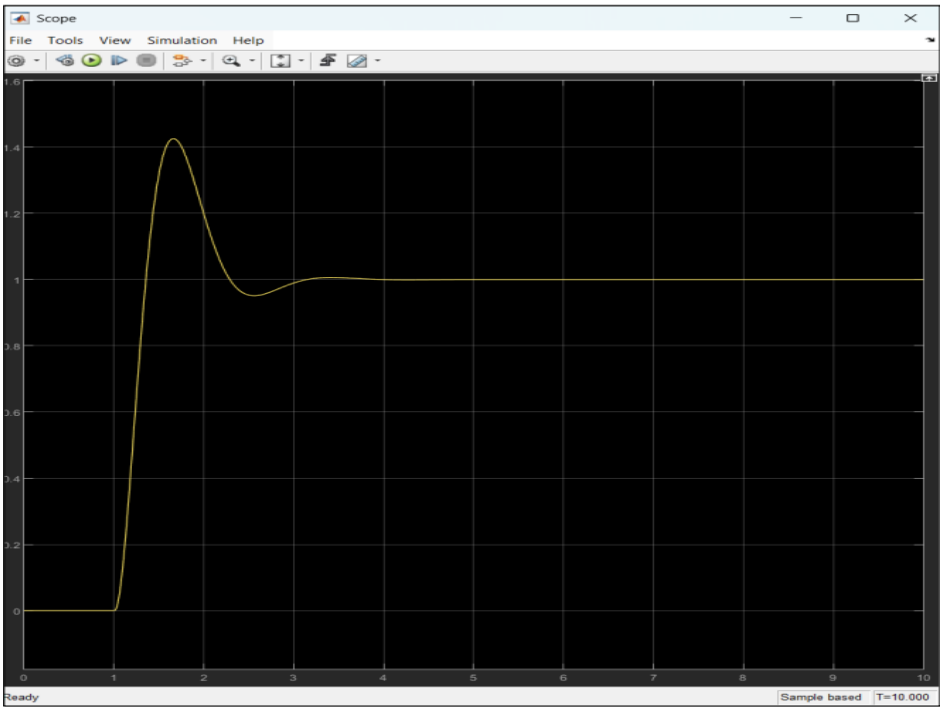


FIG 24: OUTPUT OF THE MODEL OF DC MOTOR WITH PID CONTROLLER KP = 90, KI = 100, KD = 20

PART 2: MODELING A DC MOTOR AND CONTROLLER

Discussion

Part 1: Understanding and Modeling a Basic Control System

1. **Simulink Exploration:**
   I began by exploring the Simulink Library Browser and familiarized myself with key blocks such as Sum, Gain, Integrator, Constant, Step, Scope, and MUX. This initial step helped me build confidence in creating and analyzing block diagrams that represent dynamic systems.
2. **Modeling a Second-Order System:**
   I developed both open-loop and closed-loop models of a second-order system in Simulink. A step input was applied, and the system's response was monitored using the Scope block. To better observe the output, I utilized the "Autoscale" feature. I experimented with various controller types—P, PI, PD, and PID—and observed how adjusting the gains influenced the system behavior.
3. **Controller Design and Analysis:**
   By implementing and tuning different controllers, I gained insight into how each one affects the performance of the system. I included Simulink block diagrams and output responses in my report for each configuration, clearly illustrating the impact on rise time, overshoot, settling time, and steady-state accuracy.

Part 2: DC Motor Modeling and Control

- **Parameter Definition and Transfer Function Derivation:**
  I identified all relevant parameters for the DC motor and derived the corresponding transfer functions: one from voltage to angular velocity (Gv), and another from voltage to shaft angle (Ga). This laid the groundwork for accurate simulation and controller development.
- **MATLAB Implementation:**
  I wrote MATLAB code to define transfer functions, assign appropriate input and output labels, and convert these models into state-space and zero-pole-gain formats. This demonstrates a strong command of MATLAB's Control System Toolbox and ensures the model is ready for further analysis.
- **System Poles, Zeros, and Stability:**
  I determined the poles and zeros of both Gv and Ga. Since all poles lie in the left half of the s-plane, the system was confirmed to be stable—an essential requirement for physical systems.
- **Controllability and Observability Checks:**
  Using state-space representation, I evaluated the controllability and observability matrices and found them to be full-rank. This verifies that the entire system is both controllable and observable, which is crucial for designing effective feedback and estimation schemes.
- **Time and Frequency Domain Behavior:**
  I analyzed the system's step, impulse, and frequency responses. These plots provided valuable insights into how the system behaves under different inputs, and revealed key frequency characteristics such as bandwidth and phase margin.
- **PID Controller Design:**
  A PID controller was designed and its performance was simulated. By using root locus analysis, I tuned the controller gains to achieve a stable and efficient closed-loop system. The improved response was verified through step response plots.
- **Simulink Modeling of DC Motor System:**
  I translated the mathematical model into Simulink, building both open-loop and closed-loop systems. I used subsystems for organization and tested the system using the PID controller. The simulation outputs for both angular velocity and angle confirmed that the model and controller function as expected.

Conclusion

This laboratory session provided valuable hands-on experience in modeling and controlling dynamic systems using MATLAB and Simulink. For the second-order system, I successfully developed open- and closed-loop models, experimented with different types of controllers, and observed how tuning affected system performance. Among the controllers tested, the PID offered the best balance of speed, accuracy, and stability.

In the second part, I modeled a DC motor system by deriving its transfer functions and representing it in different forms. I confirmed system stability, controllability, and observability through analytical methods. Using both time and frequency domain analyses, I gained a deeper understanding of the system's dynamics.

The PID controller design process, supported by root locus tuning and simulations in both MATLAB and Simulink, resulted in a well-performing control system. This lab enhanced my ability to link theoretical concepts with practical tools and simulations, reinforcing key principles of control systems engineering.