COURSE CONTENT

CONTROL SYSTEMS LABORATORY								
IV Semester: EEE								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
AEEE15	**Core**	-	-	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 36**				**Total Classes: 36**		
Prerequisite: There are no prerequisites to take this course								

I. COURSE OVERVIEW:

The Control Systems laboratory course is indeed to train the students practically on the modelling, analysis and design of linear feedback control systems. This course deals with modelling of dynamical systems, and the control components and designing the compensator. The hands-on training in the laboratory enables students to apply and modelling control principles in various areas of industrial applications.

II. COURSES OBJECTIVES:

The students will try to learn

I. The estimation of stability of dynamical systems using Digital simulation.
II. The various techniques of modeling and analyzing system's performance.
III. Design the time and frequency response of system by both classical and modern techniques.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1: Make use of the knowledge of digital simulation tool for system analysis with different standard Inputs.
CO 2: Model the dynamic systems in transfer function using digital simulation tool and validate the performance characteristics of motors.
CO 3: Analyze and select various electronics devices for improving system performance along with tuning mechanism in virtual environment.
CO 4: Experiment the types of compensation techniques for improving the system's accuracy.
CO 5: Analyze the system's stability in time and frequency domain by computing gain and phase margin.

IV. COURSE CONTENT:

DO's

1. Once the operation is completed pull the plug itself rather chord attached to it.
2. To repair the equipment switch-off the supply and go on.
3. To operate the equipment on supply, see that hands are dry, if that is not possibly hide the hand in the pockets.
4. If a person comes in contact with current unexpectedly don't touch the person with hands but immediately use any insulator material and shut down the power (like leather belts, wood and plastic bars etc).
5. If water is nozzles on the equipment, immediately shunt down the power using circuit breaker or pull out the plug.
6. Use the connecting wires of good continuity, short circuit of connecting wire leads damage of circuit parameters

DON'Ts

1. Do not wear loose clothing and do not hold any conducting materials in contact with skin when the power is on.
2. Do not pull out the connections until unless all the currents are dead.
3. Do not wait for switches to de-magnetize when there is a delay but pull out the plug.
4. Do not overload the circuit by plugging in too many appliances.
5. If you are mentally and physically stressed don't operate the power equipment.
6. Never operate the equipment under wet conditions.
7. Do not interconnect two or more wires, take appropriate length of wire.

SAFETY NORMS

1. The lab must be equipped with fire extinguisher.
2. See that the connections are made tight.
3. Use single plug for each equipment.
4. Cover the body completely to avoid arc effect.
5. To change the connections during the experiment, switch off the supply and carry on.
6. Used equipment may get heated, so take care handling the equipment after it is used.
7. Do the wiring, all set ups and check the circuit connections before the supply is on

IV. COURSE CONTENT:

EXERCISES FOR CONTROL SYSTEMS LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice session

Getting Started Exercises

Introduction to MATLAB

Identify the symbols, tool kits and connections in Simulink environment for computing the quantities associated with electrical circuits.

MATLAB is a powerful software environment widely used in various fields, including electrical engineering and circuit analysis. It provides a range of symbols, toolkits, and connections related to electrical circuits to assist engineers and researchers in designing, analyzing, and simulating electrical circuits. Here's an introduction to some of the key features and tools MATLAB offers in this regard:

1. Symbolic Math Toolbox: MATLAB's Symbolic Math Toolbox allows you to perform symbolic computations, which are crucial for electrical circuit analysis. You can define symbolic variables for circuit components like resistors, capacitors, and inductors and manipulate algebraic equations symbolically. This is useful for solving circuit equations and obtaining transfer functions.
2. Simulink: Simulink is a graphical environment within MATLAB for modelling, simulating, and analyzing dynamic systems, including electrical circuits. It provides a vast library of pre-built electrical components and blocks, making it easier to create circuit models and simulate their behaviour.
3. Control System Toolbox: This toolbox contains tools and functions for analyzing and designing control systems, which are essential for understanding the behaviour of electrical circuits. You can use it to design controllers, analyze stability, and simulate the response of electrical systems.
4. Electrical Circuit Analysis Toolbox: There are also third-party toolboxes available for MATLAB, such as the Electrical Circuit Analysis Toolbox, that provide specialized tools for electrical circuit analysis. These toolboxes often include features like circuit simulation, AC/DC analysis, and more.
5. SimPowerSystems (formerly known as Sim Electronics): SimPowerSystems is a specialized toolbox in Simulink for modeling and simulating electrical circuits and systems. It includes a wide range of electrical components, such as transformers, generators, and motors, allowing you to create detailed electrical system models.
6. Simscape Electrical: Simscape Electrical is another Simulink toolbox that focuses on modelling and simulating electrical power systems. It includes components like power electronics devices, electrical machines, and power generation and distribution elements.
7. Signal Processing Toolbox: This toolbox is valuable for analyzing signals within electrical circuits. You can use it for tasks like filtering noisy signals, performing Fourier analysis, and extracting important information from sensor data.
8. Data Acquisition Toolbox: If you're working with hardware experiments or data acquisition systems, this toolbox can help you connect MATLAB to external hardware, making it easier to acquire and analyse data from real-world electrical circuits.
9. Instrument Control Toolbox: This toolbox enables you to communicate with and control external instruments like oscilloscopes, signal generators, and mustimeters, which is beneficial for experimental work in electrical engineering.
10. Custom Functions and Scripts: MATLAB allows you to create custom functions and scripts to model and analyze specific aspects of electrical circuits. You can define your own functions for

circuit equations, transfer functions, and more.

Creating syntax programs for electrical circuit analysis in MATLAB often involves using built-in functions, toolboxes, and custom code to define circuits, simulate their behaviour, and analyse results. Below are some examples of syntax programs for common tasks in electrical circuit analysis using MATLAB:

Hint

1. Calculate the Circuit Components and Equations:

```
 % Define symbolic variables
syms V1 V2 R1 R2 C1 L1 s

% Define resistor values
R1 = 100; % Ohms
R2 = 200; % Ohms

% Define capacitor and inductor values
C1 = 0.1; % Farads
L1 = 0.5; % Henrys

% Define Kirchhoff's laws for an RC circuit
Eq1 = V1 - R1 * I1 - V2 == 0;
Eq2 = V2 - R2 * I2 - I1/C1 == 0;

% Define Laplace domain equations
Eq1_Laplace = laplace (Eq1, t, s);
Eq2_Laplace = laplace (Eq2, t, s);
```

2.Solve Circuit Equations:

```
% Solve the Laplace domain equations symbolically
I1_Laplace = solve (Eq1_Laplace, I1);
I2_Laplace = solve (Eq2_Laplace, I2);

% Inverse Laplace transform to get time-domain solutions
I1_time = ilaplace (I1_Laplace, s, t);
I2_time = ilaplace (I2_Laplace, s, t);
```

3.Simulate the Circuits in Simulink:

```
% Create a Simulink model for an RLC circuit
model = 'RLC_Circuit';
open_system(new_system(model));

% Add components and connections using Simulink blocks
add_block ('Simulink/Continuous/Resistor', [model '/R1']);
add_block ('Simulink/Continuous/Inductor', [model '/L1']);
add_block ('Simulink/Continuous/Capacitor', [model '/C1']);
add_block ('Simulink/Sources/Step', [model '/Step Voltage']);
add_line (model, 'Step Voltage/1', 'R1/1');
% Add connections as needed
% Set component values and simulation parameters
```

```
set_param ([model '/R1'], 'Resistance', '100');
set_param ([model '/L1'], 'Inductance', '0.5');
set_param ([model '/C1'], 'Capacitance', '0.1');
set_param (model, 'Stop Time', '5');

% Run the simulation
sim(model);
```

4.Analyze Frequency Response:

```
% Define a transfer function for an RC circuit
num = [1];
den = [R1*C1 1];
sys = tf (num, den);

% Bode plot for frequency response
bode(sys);
```

5.Evaluate the Data Analysis and Plotting:

```
% Import experimental data from a file
data = import data('experimental_data.csv');
time = data (: 1);
voltage = data (: 2);

% Plot voltage-time graph
plot (time, voltage);
xlabel ('Time (s)');
ylabel ('Voltage (V)');
title ('Voltage vs. Time');
```

Try:

a. **Circuit Equations:**
   i. Write down the Kirchhoff's Voltage Law (KVL) equation for a series RC circuit involving a resistor (R) and a capacitor (C). Assume a constant voltage source V_S.
   ii. Write down the Kirchhoff's Voltage Law (KVL) equation for a series RL circuit involving a resistor (R) and an inductor (L). Assume a time-varying current source I_S(t).

b. **MATLAB Code:** Write MATLAB code to:
   i. Solve the KVL equation for the RC circuit (from question 2a) symbolically for V_C as a function of time, assuming initial conditions.
   ii. Create a Simulink model for the RL circuit (from question 2b) with appropriate components and connections. Set the simulation parameters and run a transient analysis for a specified time duration.

   Hint: Save your MATLAB code as a separate script file (.m) for questions

c. **Transfer Function:**
   i. Define the transfer function H(s) for the RC circuit in terms of Laplace variables.
   ii. Plot the Bode plot of H(s) to analyze the frequency response of the RC circuit.

d. **Data Analysis:**
   i. Import experimental voltage and time data from a CSV file into MATLAB.
   ii. Plot the voltage-time graph and label the axes appropriately.

e. Circuit Toolbox:

i. Mention at least two MATLAB toolboxes that are useful for electrical circuit analysis. Explain briefly how each toolbox can assist in circuit analysis.

ii. If R = 10 Ohms and the current is increased from 0 to 10 A with increments of 2A, write a MATLAB program to generate a table of current, voltage and power dissipation.

1. TIME RESPONSE OF SECOND ORDER SYSTEM

To determine response of first order and second order systems for step input for several of constant 'K' using linear simulator unit and compare theoretical and practical results.

.

1.1. TIME RESPONSE OF SECOND ORDER SYSTEM IN MATLAB

Examine the time response of a given second order system with time domain specifications using MATLAB

To find the time domain specifications of a second-order system in MATLAB, you can use the "**stepinfo**" function, which provides information such as rise time, settling time, overshoot, etc. Here's an example with the necessary syntax:

Hint

```
% Define system parameters

Omega _n = 2;  % Natural frequency
zeta = 0.7;   % Damping ratio

% Create the transfer function
numerator = omega_n^2;
denominator = [1, 2*zeta*omega_n, omega_n^2];
sys = tf(numerator, denominator);

% Obtain time domain specifications
info = stepinfo(sys);

% Display time domain specifications
disp('Time Domain Specifications:');
disp(['Rise Time: ' num2str(info.RiseTime) ' seconds']);
disp(['Settling Time: ' num2str(info.SettlingTime) ' seconds']);
disp(['Overshoot: ' num2str(info.Overshoot) ' %']);
disp(['Peak Time: ' num2str(info.PeakTime) ' seconds']);
```

Try

1. MATLAB program for finding step, ramp parabolic input response for a closed-loop transfer function $G(s) = \frac{3}{s(s+4)}$
2. MATLAB program to find the unit step response of a unity feedback control system with $G(s) = \frac{10}{s(s+3)}$ and $H(s) = 0.1s + 1$

1.2. TIME RESPONSE OF SECOND ORDER SYSTEM

Examine the time response of a given second order system with time domain specifications.

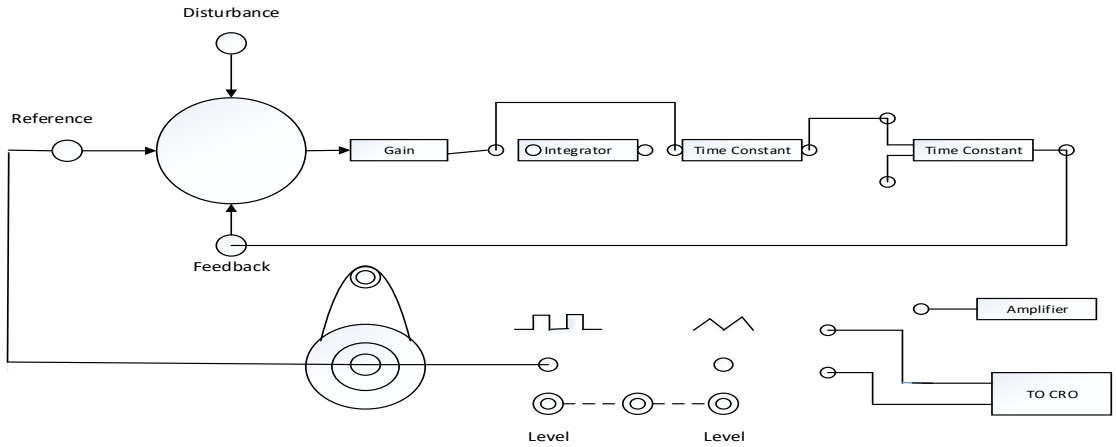

Figure 1 - Time Response of Second Order System

Try

1. Determine the time domain specifications for different zeta values for Figure 1
2. For different test signals obtain the time domain specifications for different zeta=1
3. Draw the comparison graph for first, second order system with zeta = 1, step signal.

2. TRANSFER FUNCTION OF DC MOTOR

To determine Transfer function of armature-controlled dc motor 1. By performing load test on dc motor and speed control by armature voltage control. 2. To plot characteristics between back e.m.f. and angular velocity and armature current Vs torque

2.1. TRANSFER FUNCTION OF DC MOTOR with MATLAB

Determine the transfer function, time response of DC motor and verification with MATLAB
The transfer function of a DC motor can be derived from the basic equations governing its dynamics. The typical transfer function for a DC motor is given by:

$$J(s) = \frac{K}{s(Js + b)K_b^2}$$

Hint

```
% Define system parameters
% Define DC Motor Parameters
K = 0.01;       % Motor gain or torque constant (Nm/A)
J = 0.01;       % Moment of inertia (kg*m^2)
b = 0.1;        % Viscous damping coefficient (Nms/rad)
K_b = 0.01;     % Back EMF constant (V/(rad/s))
% Formulate Transfer Function
numerator = K;
denominator = [J, b, 0; 0, K_b, 0; 0, 1, 0];
sys = tf(numerator, denominator);
% Time Response Analysis
t = 0:0.01:2; % Time vector from 0 to 2 seconds with a step size of 0.01 seconds
[y, t] = step (sys, t);
% Plot the Step Response
figure
plot (t, y);
```

```
title ('Step Response of DC Motor');
xlabel ('Time (seconds)');
ylabel ('Angular Position (rad)');
grid on;
```

Try

1. Create a Simulink model of the DC motor system. Compare the results obtained from Simulink with those obtained using the transfer function. Ensure consistency between the analytical and simulation results.
2. Use the **"bode"** function in MATLAB to analyse the frequency response of the DC motor system. Investigate how the system behaves at different frequencies.
3. Change one parameter at a time (e.g. R, L, J, b, K, Ke) and observe how it affects the step response. Explain the observed changes in terms of the motor dynamics.

2.2. TRANSFER FUNCTION OF DC MOTOR

Determine the transfer function, time response of DC motor

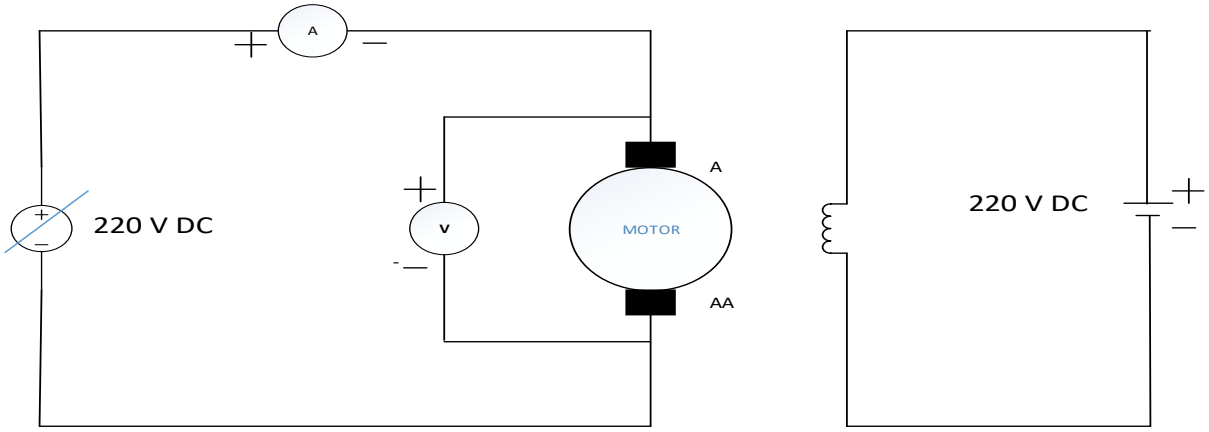

Figure 2 - Armature voltage control

Try

1. Calculate the different armature currents (i.e ia equal il plus if) for shunt motor under various loads for figure1
2. Draw the External characteristics using armature currents (0 to 13 A) with respect to the load currents.
3. Draw the Internal characteristics using generated induced emf (0 to 220V) with respect to the field currents (0 to 2A).

3. TRANSFER FUNCTION OF AC SERVO MOTOR

The servomotors used in the low power servomechanism are a.c. servomotors. The a.c. servomotor is basically two-phase induction motor. The output power of a.c. servomotor varies from fraction of watts to few hundreds of watts. The operating frequency is 50 Hz to 400 Hz.

3.1. TRANSFER FUNCTION OF AC SERVO MOTOR using MATLAB

To control an AC servo motor in MATLAB, you can use the Simulink environment along with the necessary blocks and functions. Here's a basic example of a MATLAB program for simulating the characteristics of an AC servo motor using Simulink:

```
% AC Servo Motor Characteristics Simulation Assignment
```

```
% Parameters
R = 2; % Resistance (ohms
L = 0.01;   % Inductance (Henry)
Kt = 0.1;   % Torque constant (Nm/A)
J = 0.001; % Moment of inertia (kg.m^2
B = 0.01;   % Viscous friction coefficient (Nm/(rad/s))
Ke = 0.1;   % Back EMF constant (V/(rad/s)
% Transfer function of the AC servo motor system
numerator = [Kt]
denominator = [J * L, (B * L + J * R), (R * B + Kt^2), 0];
motor = tf (numerator, denominator);
% Create a time vector for simulation
t = 0:0.01:2;

% Simulate the step response
step_response = step (motor, tf);

% Plot the step response
figure;
plot (t, step_response);
title ('Step Response of AC Servo Motor System');
xlabel ('Time (seconds)');
ylabel ('Angular Position');
grid on;

% Calculate characteristics
info = step info(motor_tf);
% Display characteristics
fprintf ('Overshoot: %.2f%%\n', info. Overshoot);
fprintf ('Settling Time: %.2f seconds\n', info. Settling Time);
fprintf ('Rise Time: %.2f seconds\n', info. Risetime);
```

Try

1. Consider an AC servo motor as system with the following specifications:
   Find the Transfer function of Motor for given Parameters:
   a) Resistance (R): 2 ohms
   b) Inductance (L): 0.01 Henries
   c) Torque constant (Kt): 0.1 Nm/A
   d) Moment of inertia (J): 0.001 kg.m^2
   e) Viscous friction coefficient (B): 0.01 Nm/(rad/s)
   f) Back EMF constant (Ke): 0.1 V/(rad/s)

3.2. AC SERVO MOTOR

Determine the Torque Speed characteristics of a AC servomotor

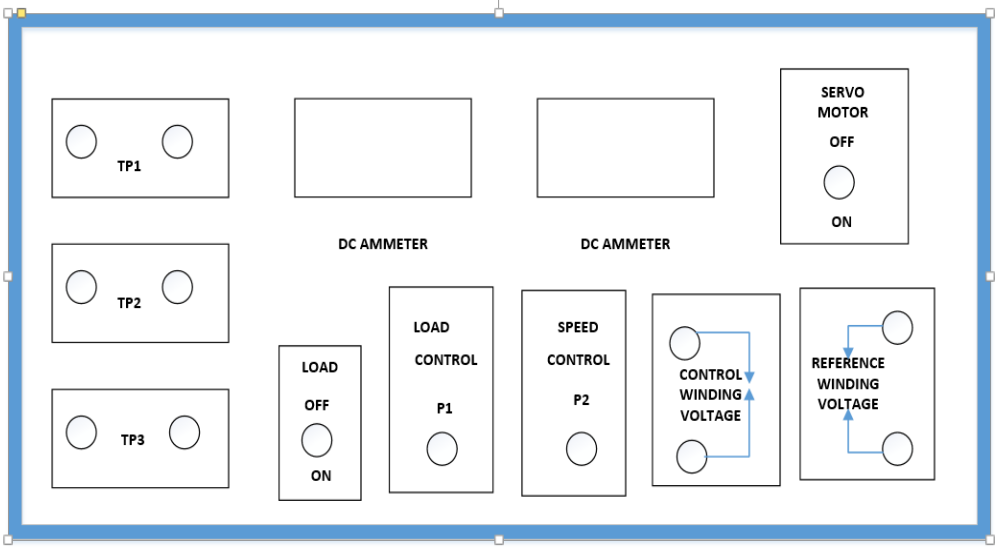

Figure 3 – Front panel of AC Servo-motor Controller

Try

1. Calculate the different armature currents (i.e I_a equal I_a plus if) for AC servo motor under various Loads for figure 3.
2. Draw the External characteristics using armature currents (0 to 13 A) with respect to the load currents.
3. Draw the Internal characteristics using generated induced emf (0 to 220V) with respect to the field currents (0 to 2A).

4. EFFECT OF VARIOUS CONTROLLERS ON SECOND ORDER SYSTEM USING MATLAB

PID controllers are adjusted on-site and many different types of tuning rules have been proposed in different literatures. Using these tuning rules, delicate and fine tuning of PID controllers can be made on-site. Also, automatic tuning methods have been developed and some of the PID controllers may possess on-line automatic tuning capabilities. Modified forms of PID control, such as PID control and two-degrees-of- freedom PID control, are currently in use in industry.

4.1. EFFECT OF VARIOUS CONTROLLERS ON SECOND ORDER SYSTEM USING MATLAB

Analyze the effects of various controllers on a second-order system using the control system toolbox. Here's a basic example to demonstrate the impact of different controllers on a second-order system:

```
% Define the second-order system
numerator = 1;
denominator = [1, 2, 1]; % Second-order system with damping ratio 1 (critically damped)
sys = tf(numerator, denominator);
% Define time vector
t = 0:0.01:5;
% Step response of the original system
figure;
```

```
step (sys, t);
title ('Step Response - Original System');
% Proportional (P) controller
Kp = 1;
controller_p = tf([Kp], [1]);
% Closed-loop system with P controller
sys_p = feedback (series (controller_p, sys), 1);
% Step response of the system with P controller
figure;
step (sys_p, t);
title ('Step Response - Proportional (P) Controller');
% Proportional-Integral (PI) controller
Kp = 1;Ki = 0.5;
controller_pi = tf([Kd, Ki], [1, 0]);
% Closed-loop system with PI controller
sys_pi = feedback (series (controller_pi, sys), 1);
% Step response of the system with PI controller
figure;
step (sys_pi, t);
title ('Step Response - Proportional-Integral (PI) Controller');
% Proportional-Derivative (PD) controller
Kp = 1; Kd = 0.2;
controller_pd = tf ([Kp, Kd], [1]);
% Closed-loop system with PD controller
sys_pd = feedback (series (controller_pd, sys), 1);
% Step response of the system with PD controller
figure;
step (sys_pd, t);
title ('Step Response - Proportional-Derivative (PD) Controller');
```

Try

1. Compare the step responses of a second-order system with Proportional (P), Proportional-Integral (PI), and Proportional-Derivative (PD) controllers.
2. Analyze and discuss the impact of each controller on system performance.

4.2. EFFECT OF VARIOUS CONTROLLERS ON SECOND ORDER SYSTEM

Determine time domain specifications of the second order system by using PID Controller

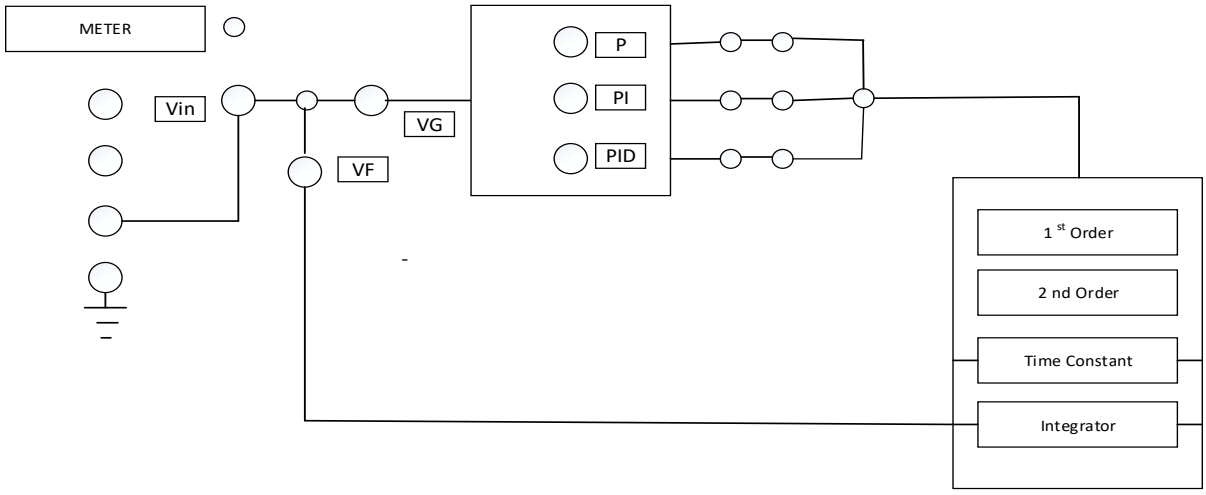

Figure 4 – Block Diagram Effect of P, PI, PID Controller on second order systems

Try

1. Choose values for natural frequency (ω_n) and damping ratio(ζ) to represent an underdamped, critically damped, and overdamped system.
2. How the proportional controller affects the system's rise time, overshoot, and steady-state error.
3. Analyze the effect of varying the proportional gain (K_p) and integral gain (K_i) on the closed-loop system.

5. COMPENSATOR

A lead compensator is commonly used for improving stability margins. Lag compensators are used to improve the steady state performance. The lead compensator achieves the desired results through the merits of its phase-lead contribution. The lag compensator accomplishes its result through the merits of its attenuation property at high frequencies

Study of Lag, Lead, Lead - Lag compensation networks and obtain its frequency response.

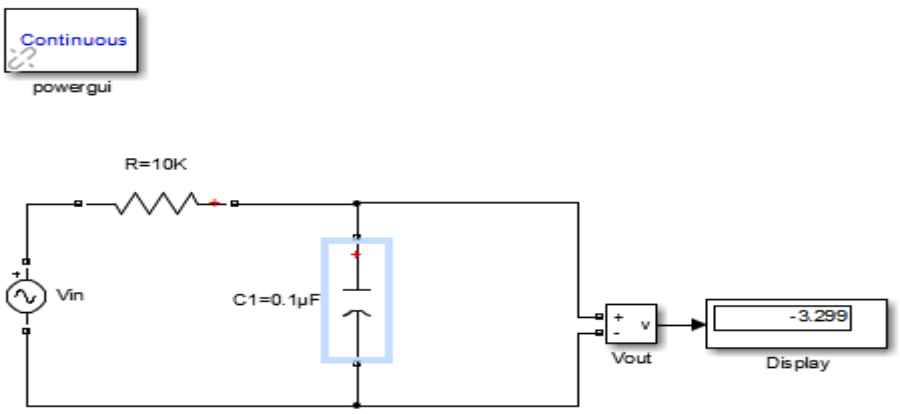

Figure 5.1 – Lag Compensation

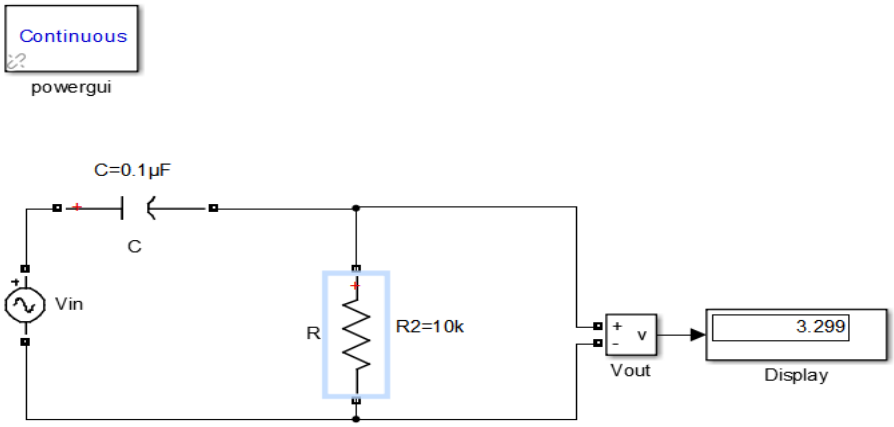

Figure 5.2 – Lead Compensation

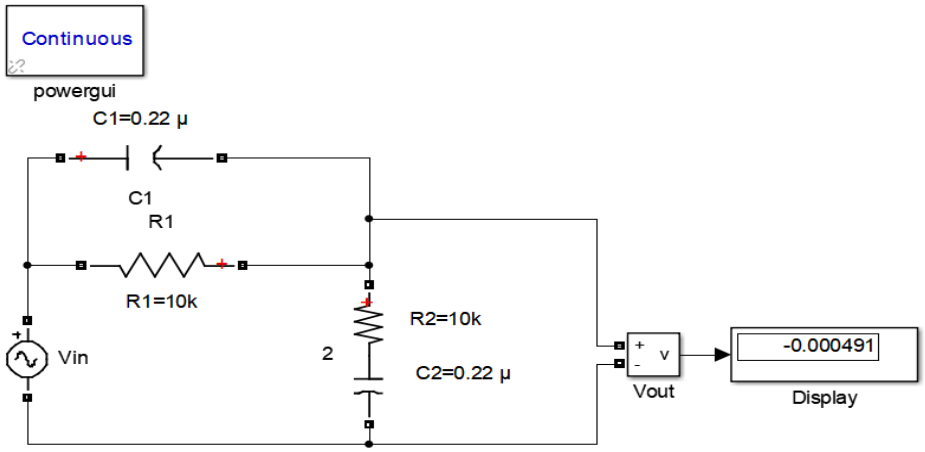

Figure 5.3 – Lag - Lead Compensation

Try

1. Consider a unity feedback control system with an open-loop transfer function:

$$G(s) = \frac{K}{s(s+3)(s+5)}$$

The system is required to meet the following specifications:
Settling Time: Less than 2 seconds.
Overshoot: Less than 10%.
Phase Margin: Greater than 40 degrees.

a. Design a lead-lag compensator with parameters($K_{lead}, \alpha, T_{lead}, K_{lag}, T_{lag}$ to meet the given specifications.
b. Provide explanations for your design choices and how each parameter affects the system response.
c. Construct the closed-loop transfer function by connecting the lead-lag compensator to the original system.

6. TEMPERATURE CONTROLLER

PID temperature control is a loop control feature found on most process controllers to improve the accuracy of the process. PID temperature controllers work using a formula to calculate the difference between the desired temperature set point and current process temperature, then predicts how much power to use in subsequent process cycles to ensure the process

temperature remains as close to the set point as possible by eliminating the impact of process environment changes.

6.1. TEMPERATURE CONTROLLER

Study the performance of PID controller used to control the temperature of an oven

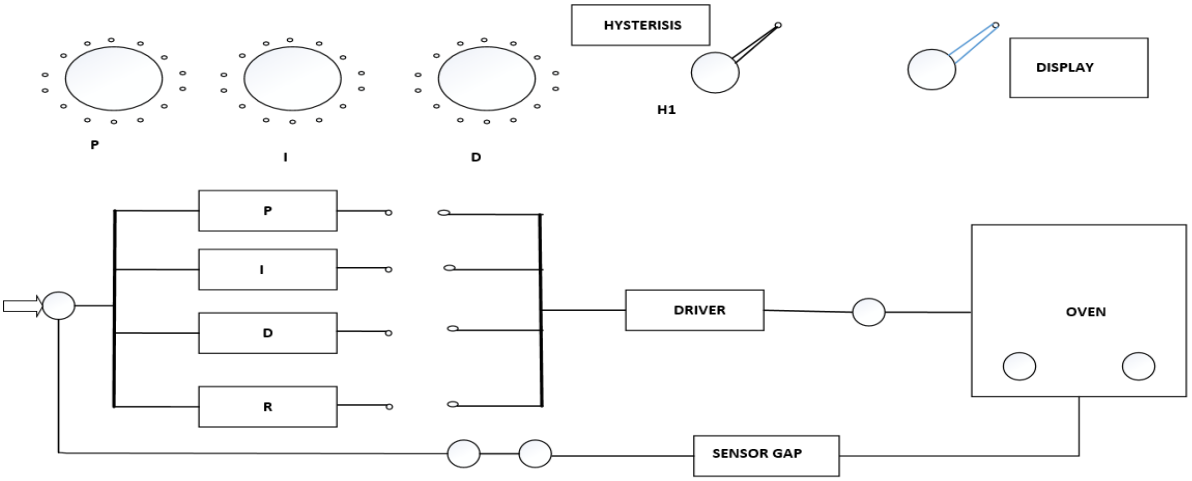

Figure 6 - Temperature controller of an oven

Try

1. Design a PID controller for the oven temperature control. Experiment with different values of $K_p, K_i, and\ K_d$ to achieve desired performance for figure 6
2. Compare the performance of the PID controller with alternative controllers (e.g., P, PI, PD) in terms of temperature regulation for figure 6

6.2. TEMPERATURE CONTROLLER using MATLAB

To study the performance of a PID controller in controlling the temperature of an oven, you can use MATLAB and the Simulink environment. Below is a basic example with MATLAB code for simulating a PID-controlled oven temperature system and analyzing its performance:

Hint

```
% PID Controller for Oven Temperature Control

% System parameters
tau = 10;  % Time constant of the oven (in seconds)

% Create a transfer function representing the open-loop system (oven)
oven_sys = tf(1, [tau, 1]);

% PID Controller parameters
Kp = 1;     % Proportional gain

Ki = 0.1;   % Integral gain
Kd = 0.01;  % Derivative gain

% Create a PID controller
```

```
pid_controller = pid(Kp, Ki, Kd);

% Connect the PID controller to the oven system

closed_loop_sys = feedback(pid_controller * oven_sys, 1);

% Simulate the closed-loop system response to a step change in setpoint

time = 0:0.1:100;  % Time vector for simulation

setpoint = 100;   % Desired temperature setpoint

% Simulate the system response
[y, t] = step (setpoint * closed_loop_sys, time);

% Plot the temperature response
figure;
plot (t, y);
title ('PID Controller Performance - Oven Temperature Control');
xlabel ('Time (seconds)');
ylabel('Temperature');
grid on;

% Analyze the step response characteristics
step_info = step info(closed_loop_sys);
disp ('Step Response Characteristics:');
disp(step_info);
```

Try

1. Experiment with different tuning methods (e.g., manual tuning, Ziegler-Nichols) to optimize the PID gains.
2. Compare the system's performance before and after tuning.

7. DESIGN AND VERIFICATION OF OP-AMP BASED PID CONTROLLER

The PID control is the most commonly known for control process utilized as a part of industries for controlling action. The basic technique for PID controllers makes it simple to coordinate the process output. The main focus of the experiment is about study of OPAMP and fabrication of an PID Controller using the three control parameters. The Controller design is demonstrated through simulation in order to get an output of better dynamic and static performance. The controller is fabricated on hardware after the test of individual terms: -proportional, integral and derivative

7.1. DESIGN AND VERIFICATION OF OP-AMP BASED PID CONTROLLER USING MATLAB

Implementation of PID controller using Op-Amps and verification using MATLAB

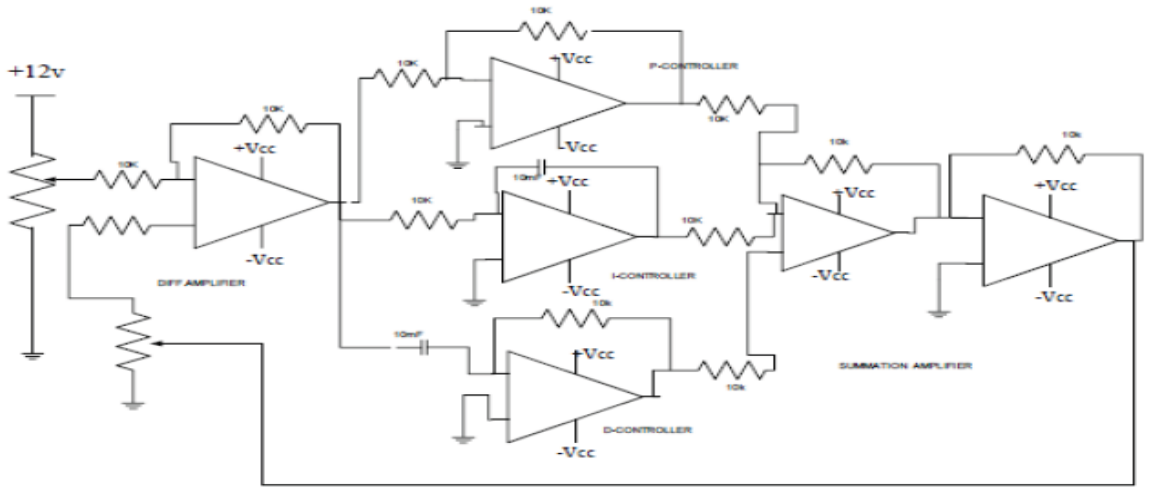

Figure 7.1 – Circuit Diagram of OP-AMP based PID Controller

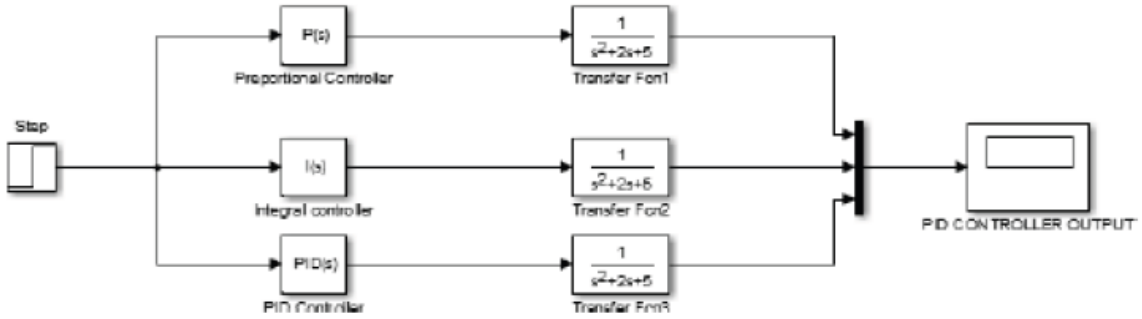

Figure 7.2 – Simulation circuit diagram of OP-AMP PID Controller

Try

1. Design an op-amp based PID controller for the chosen system.
2. Express the PID controller transfer function $H(s)$ using operational amplifiers.
3. Explain the design choices and constraints in selecting resistor and capacitor values.

7.2. DESIGN AND VERIFICATION OF OP-AMP BASED PID CONTROLLER

Implementing the design and verification of an op-amp based PID controller involves a combination of analog circuit design and simulation using tools like MATLAB. Below is an example MATLAB code to illustrate the op-amp based PID controller design and verification. Please note that this is a basic example, and you may need to adapt it according to the specifics of your system.

```
% PID Controller for Oven Temperature Control

% System parameters

% Design and Verification of Op-Amp Based PID Controller


% Plant Transfer Function (example: second-order system)
numerator = 1;
denominator = [1, 2, 1];
```

```
plant_tf = tf(numerator, denominator);

% Op-Amp Based PID Controller Parameters
R1 = 10e3; % Resistance values (adjust as needed)
R2 = 10e3;
C1 = 1e-6; % Capacitor value (adjust as needed)

% Op-Amp PID Controller Transfer Function
s = tf('s');
pid_opamp_tf = (R2 * (R1 + R2) * s + R1) / (R1 * R2 * C1 * s^2 + (R1 + R2) * s);

% Closed-Loop System Transfer Function
closed_loop_tf = feedback (pid_opamp_tf * plant_tf, 1);

% Step Response Simulation
time = 0:0.01:5;
input_signal = ones(size(time));  % Step input
output_response = lsim(closed_loop_tf, input_signal, time);

% Plot Step Response
figure;
plot (time, input_signal, 'r--', 'LineWidth', 1.5, 'DisplayName', 'Step Input');
hold on;
plot (time, output_response, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Output Response');
title ('Op-Amp Based PID Controller - Step Response');
xlabel ('Time (seconds)');
ylabel('Amplitude');
legend('show');
grid on;

% Bode Plot of the Closed-Loop System
figure;

bode(closed_loop_tf);
title ('Op-Amp Based PID Controller - Bode Plot');

% Stability Analysis
margin_info = margin(closed_loop_tf);
disp ('Stability Analysis:');
disp(margin_info);
```

Try:

1. Compare the op-amp based PID controller with an ideal PID controller in terms of performance.
2. Experiment with different resistor and capacitor values to observe the effect on the closed-loop system's performance.

8. STABILITY ANALYSIS USING DIGITAL SIMULATION

stability analysis of a linear time-invariant system using root locus, Bode plot, polar plot, and Nyquist criterion through digital simulation. MATLAB can be used for this purpose.

8.1. STABILITY ANALYSIS USING ROOT LOCUS

Stability analysis using Root Locus in MATLAB.

Hint

```
% Stability Analysis using Root Locus in MATLAB

% System parameters
numerator = 1;
denominator = [1, 3, 2]; % Example second-order system: s^2 + 3s + 2
plant_tf = tf(numerator, denominator);

% Plot the root locus
figure;
rlocus(plant_tf);
title ('Root Locus Analysis');
grid on;

% Customizing the plot (optional)
% set (findobj(gcf, 'Type', 'Line'), 'LineWidth', 2); % Increase line width
% xlabel ('Real Part');
% ylabel ('Imaginary Part');
% legend('show');



% Comment out or remove the customization lines if not needed
% Zoom into the plot (optional)
% xlim([-5, 1]);
% ylim([-5, 5]);
% Comment out or remove the xlim and ylim lines if not needed
```

Try:

1. MATLAB program for complete root locus for the system with open-loop transfer function
$$G(s) = \frac{K(S+4)}{S(S+2)}$$
2. MATLAB program for complete root locus for the system with open-loop transfer function
$$G(s) = \frac{K(S+3)}{S(s^2+S+2)}$$

8.2. STABILITY ANALYSIS USING BODE PLOT

Stability analysis using Bode plots in MATLAB involves examining the frequency response of a system.

Hint

```
% Stability Analysis using BODE PLOT in MATLAB
Step 1: Define the Transfer Function
% Example: Second-order system
numerator = 1;
denominator = [1, 3, 2];  % s^2 + 3s + 2
system_tf = tf(numerator, denominator);
Step 2: Generate Bode Plot
% Plot Bode magnitude and phase
figure;
```

```
bode(system_tf);
title ('Bode Plot Analysis');
grid on;
Step 3: Customization (Optional)
% Customize the Bode plot
set (findobj(gcf, 'Type', 'Line'), 'LineWidth', 2); % Increase line width
xlabel ('Frequency (rad/s)');
ylabel ('Magnitude (dB) and Phase (degrees)');
legend('show');
Step 4: Zooming and Exploration (Optional)
% Zoom into specific frequency range
xlim([0.1, 10]);
Step 5: Analytical Insights
% Obtain crossover frequency, gain margin, and phase margin
[mag, phase, wout] = bode(system_tf);
crossover_frequency = wout(find (mag <= 0, 1)); % Find the frequency where magnitude is 0 dB
gain_margin = 1/mag (find (wout <= crossover_frequency, 1)); % Gain at crossover frequency
phase_margin = 180 + phase (find (wout <= crossover_frequency, 1)); % Phase margin
Step 6: Conclusion
Summarize the stability characteristics of the system based on the Bode plot.
```

Try:

1. MATLAB program for complete BODE PLOT for the system with open-loop transfer function

$$G(S) = \frac{KS^2}{(1+S)(1+0.1S)(1+0.01S)}$$

2. MATLAB program for complete BODE PLOT for the system with open-loop transfer function

$$(S) = \frac{Ke^{-0.2S}}{(1+S)(1+0.1S)(1+0.2S)}$$

8.3. STABILITY ANALYSIS USING POLAR PLOT

Stability analysis using polar plots in MATLAB involves examining the polar representation of the system's frequency response. Below are the MATLAB steps for generating a polar plot and performing stability analysis:

Hint

```
Step 1: Define the Transfer Function
% Example: Second-order system
numerator = 1;
denominator = [1, 3, 2]; % s^2 + 3s + 2
system_tf = tf(numerator, denominator);
Step 2: Generate Polar Plot
% Plot polar representation
figure;

polarplot(angle(system_tf), abs(system_tf));
title('Polar Plot Analysis');
grid on;
Step 3: Customization (Optional)
% Customize the polar plot
ax = gca;
ax.ThetaZeroLocation = 'top';
ax.ThetaDir = 'clockwise';
```

Step 4: Analytical Insights
% Analyze polar plot to understand system stability
Step 5: Conclusion
Summarize the stability characteristics of the system based on the polar plot

Try:

1. MATLAB program for complete POLAR PLOT for the system with open-loop transfer function
$$G(S) = \frac{1}{(1+S)(1+4S)}$$
2. MATLAB program for complete POLAR PLOT for the system with open-loop transfer function
$$G(S) = \frac{(1+0.1S)(1+0.2S)}{S^3(1+0.004S)(1+0.001S)}$$

8.4. STABILITY ANALYSIS USING NYQUIST PLOT

Stability analysis using NYQUIST PLOTS in MATLAB involves examining the frequency response of a system.

Hint

Step 1: Define the Transfer Function
% Example: Second-order system
numerator = 1;
denominator = [1, 3, 2]; % s^2 + 3s + 2
system_tf = tf(numerator, denominator);
Step 2: Generate Nyquist Plot
% Plot Nyquist plot
figure;

nyquist(system_tf);
title ('Nyquist Plot Analysis');
grid on;
Step 3: Customization (Optional)
% Customize the Nyquist plot
set (findobj(gcf, 'Type', 'Line'), 'LineWidth', 2); % Increase line width
xlabel ('Real Part');
ylabel ('Imaginary Part');
legend('show');
Step 4: Analytical Insights
% Analyze Nyquist plot to understand system stability
Step 5: Conclusion
Summarize the stability characteristics of the system based on the Nyquist plot.

Try:

1. MATLAB program for complete NYQUIST PLOT for the system with open-loop transfer function
$$\boldsymbol{G(S) = \frac{1}{(1+S)(1+4S)}}$$
2. MATLAB program for complete NYQUIST PLOT for the system with open-loop transfer function
$$\boldsymbol{G(S) = \frac{1}{S(1+S)(1+4S)}}$$

9. STATE SPACE MODEL USING DIGITAL SIMULATION

Verify a state-space model from a transfer function and vice versa using digital simulation in MATLAB.

1.Verification of State-Space Model from Transfer Function:

```
% Define the transfer function
numerator = [1];
denominator = [1, 2, 1];
sys_tf = tf(numerator, denominator);
% Convert transfer function to state-space representation
sys_ss = tf2ss(sys_tf);
% Display state-space matrices
disp ('State-Space Matrices:');
disp ('A matrix:');
disp(sys_ss.A);
disp ('B matrix:');
disp(sys_ss.B);
disp ('C matrix:');
disp(sys_ss.C);
disp ('D matrix:');
disp(sys_ss.D);
```

2. Verification of Transfer Function from State-Space Model:

```
% Define state-space matrices
A = [-2 -1; 1 0];
B = [1; 0];
C = [1 0];
D = 0;
% Convert state-space model to transfer function
sys_tf_verified = ss2tf(A, B, C, D);
% Display transfer function coefficients
numerator_verified = sys_tf_verified.Numerator{1};
denominator_verified = sys_tf_verified.Denominator{1};
disp ('Transfer Function Coefficients:');
disp ('Numerator Coefficients:');
disp(numerator_verified);
disp ('Denominator Coefficients:');
disp(denominator_verified);
```

3.Digital Simulation for Verification:

```
% Define time vector
t = 0:0.01:5;
% Simulate the transfer function response
y_tf = lsim(sys_tf, ones(size(t)), t);
% Simulate the state-space model response
u = ones(size(t));
x0 = [0; 0]; % Initial state
[y_ss, t_ss, x_ss] = lsim(sys_ss, u, t, x0)

% Plot results for verification
figure;
subplot (2,1,1);
plot (t, y_tf, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Transfer Function');
hold on;
```

```
plot (t_ss, y_ss, 'b--', 'LineWidth', 1.5, 'DisplayName', 'State-Space Model');
title ('Output Response');
xlabel('Time');
ylabel('Amplitude');
legend('show');

subplot(2,1,2);
plot(t_ss, x_ss(:,1), 'b-', 'LineWidth', 1.5, 'DisplayName', 'State 1');
hold on;
plot(t_ss, x_ss(:,2), 'g-', 'LineWidth', 1.5, 'DisplayName', 'State 2');
title('State Variables');
xlabel('Time');
ylabel('Amplitude');
legend('show');
```

Try

1. MATLAB Program to obtain the state model of the system

$$T(S) = \frac{Y(S)}{U(S)} = \frac{2S^3 + 7S^2 + 12S + 8}{S^3 + 6S^2 + 11S + 9}$$

2. MATLAB program to obtain the transfer function for the state model

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -9 & -11 & -6 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, C = [-10 \quad -10 \quad -5]$$

9.1 Introduction to PLC and Industrial Automation

A Programmable Logic Controller (PLC) is a specialized digital computer used in industrial automation and control systems. It plays a crucial role in automating processes in manufacturing plants, power generation facilities, water treatment plants, and various other industrial applications. PLCs are designed to monitor inputs, make decisions based on a programmed logic, and control outputs to automate machinery and processes.

Basic introduction to a Programmable Logic Controller (PLC) and an industrial automation laboratory:

Programmable Logic Controller (PLC):

a. **Purpose:**
   - PLCs are used for automating and controlling industrial processes.
   - They replace traditional relay-based control systems with a more flexible and efficient digital control.

b. **Components:**
   - CPU (Central Processing Unit): The brain of the PLC, responsible for executing the control program.
   - Input Modules: Receive signals from sensors and other devices.
   - Output Modules: Control actuators, motors, and other output devices.
   - Memory: Stores the control program and data.
   - Facilitate communication with other devices and systems.

c. **Programming:**
   - PLCs are programmed using specialized software that often employs ladder logic diagrams, resembling relay logic.
   - Programming involves defining inputs, outputs, and the logic that determines the system's behavior.

d. Applications:

- PLCs are used in various industries, including manufacturing, chemical processing, power plants, and more.
- They control processes such as assembly lines, batch processing, and continuous production.

Industrial Automation Laboratory:

An industrial automation laboratory is a controlled environment where students, engineers, or technicians can learn and experiment with various aspects of industrial automation using PLCs. Such a lab typically includes:

a. Hardware Setup:

- PLCs and associated input/output modules.
- Sensors and actuators representing real-world industrial equipment.
- Communication interfaces for networking and data exchange.

b. Programming Tools:

- Computers with PLC programming software.
- Simulation tools to test programs before deploying them to actual PLCs.

c. Training Stations:

- Workstations equipped with PLCs, input/output devices, and connection terminals for hands-on learning.

d. Experiments and Projects:

- Laboratory sessions designed for students to implement control strategies, troubleshoot, and optimize industrial processes.
- Projects that involve designing and implementing automation solutions.

e. Safety Measures:

- Protocols to ensure safe experimentation, especially when dealing with real-world industrial equipment.

f. Documentation and Analysis:

- Tools and methodologies for documenting PLC programs, analyzing system performance, and identifying area for improvement.

Try:

PLC Ladder Diagrams:

- Analyzing the results of operations of timers, counters, DC/AC motors and traffic signal control using PLC ladder diagrams in terms of concepts introduced in the Industrial Automation and Control course.

Hardware Connections:

- Trained to get hands-on experience on wiring the Delta make DVP series Programmable Logic Controller and Human Machine Interface.

10. LADDER DIAGRAMS USING PLC

Input output connection, simple programming, ladder diagrams, uploading, running the program and debugging in

1. Input-Output Connection:

PLCs interface with the external world through input and output connections. Inputs can be connected to sensors, switches, or other devices, while outputs can be connected to actuators, motors, or other controlled devices. These connections are typically made through terminals on the PLC.

2. Simple Programming using Ladder Diagrams:
Ladder Logic is a graphical programming language widely used in PLCs. Below is an example of a simple start-stop motor control program using ladder diagrams:

```
--[ Start ]----( )----[ Motor Coil ]--
                          |
--[ Stop ]-----[ ]--------------------
```

In this example, the motor will start when the "Start" button is pressed (closed), and it will stop when the "Stop" button is pressed (closed).

3. Uploading a Program to PLC:
Uploading involves transferring the ladder logic program from your development environment (PLC programming software) to the PLC. This is typically done through a communication cable. The uploaded program is stored in the PLC's memory.

4. Running the Program:
Once the program is uploaded, the PLC can be set to "Run" mode. In this mode, it executes the ladder logic continuously based on the current state of input devices. Outputs change state accordingly.

5. Debugging:
Debugging in PLC programming involves identifying and correcting errors in the ladder logic or addressing. Common debugging tools include:
Online Monitoring: Observe the current status of inputs and outputs in real-time.
Force I/O: Force inputs or outputs to specific states for testing.
Setting Breakpoints: Pause execution at specific points to inspect variables or conditions

```
Example: Emergency Stop Circuit
--[ Start ]----( )----[ Motor Coil ]----
              |
--[ E-Stop ]----[ ]----------------------
```

In this example, the motor will start when the "Start" button is pressed, but it will immediately stop if the emergency stop ("E-Stop") button is pressed.

Try:
Design a ladder diagram using a PLC to control a conveyor belt system

11. TRUTH TABLES USING PLC

In programmable logic controllers (PLCs), truth tables are often used to describe the relationship between inputs and outputs in a logic circuit. Each row in the truth table represents a combination of input states and the corresponding output state. Here is an example of a truth table using ladder logic for a basic AND gate:

Example: AND Gate Truth Table and Ladder Logic
Consider a simple AND gate with two inputs (I1 and I2) and one output (O).
Truth Table:

I1	I2	O
0	0	0
0	1	0
1	0	0
1	1	1

```
|----[ I1 ]------------------( )----[ O ]--|
|                                  |
|----[ I2 ]---------------------------------|
```

In this example:

1. If both I1 and I2 are 0, the output O is 0.
2. If either I1 or I2 is 1 (but not both), the output O is 0.
3. If both I1 and I2 are 1, the output O is 1.

Steps for Creating a Truth Table using PLC:
Identify Inputs and Outputs:
Determine the number of inputs and outputs in your logic circuit.
List All Input Combinations:
Enumerate all possible combinations of input states. This creates the input columns of your truth table.
Determine Output for Each Combination:
For each combination of input states, determine the corresponding output state.

Create Truth Table:
Organize the input and output states into a table format.
Implement Ladder Logic:
Create ladder logic based on the truth table. Use symbols such as contacts (representing inputs) and coils (representing outputs).

Try

1. Draw the OR, NAND, NOT, XNOR LOGIC gates.

12. IMPLEMENTATION OF COUNTER

In programmable logic controllers (PLCs), counters are often used to count events or pulses. There are different types of counters, such as up counters, down counters, and up/down counters. Here, I'll demonstrate the implementation of a basic up counter using ladder logic in a PLC.
Example: Up Counter Implementation
Consider an up counter with one input (Start) and one output (Count). The counter increments each time the Start input is activated.

Ladder Logic:

```
|----[ Start ]----[ ]----( )----[ Count ]--|
```

In this ladder logic diagram:
[Start]: Represents the input contact. When this input is closed (activated), the counter will increment.
[ ]: Represents the coil. This is the output that changes state when the counter increments.
(): Represents the counter. Each time the Start input is activated, the counter increments by 1, and the Count output turns on.
Steps for Implementing an Up Counter:
Identify Inputs and Outputs:
Determine the input (e.g., a Start button) and output (e.g., Count) for your up counter.
Create Ladder Logic Diagram:
Use ladder logic symbols to represent the input contact, the counter, and the output coil. Connect them appropriately to form the ladder diagram.
Configure Counter Parameters:

In your PLC programming software, configure the counter parameters such as the preset value (the value at which the counter resets to zero) and the counting direction (up, down, or up/down).

Program the Counter Logic:

Write the ladder logic program based on the ladder diagram. Define the conditions for the counter to increment and reset based on the input.

Upload and Run the Program:

Upload the ladder logic program to the PLC. Set the PLC to "Run" mode to execute the program.

Test and Debug:

Test the counter by activating the Start input and observing the Count output. Use debugging tools provided by your PLC programming software to identify and fix any issues.

Try

1. Write the ladder diagram for modifying the Up Counter for Reset counter.

13. BLINKING LIGHTS USING PLC

Implementing blinking lights using a Programmable Logic Controller (PLC) involves creating a simple ladder logic program. Below is an example of ladder logic for a PLC program that controls two blinking lights alternately.

Consider two lights (L1 and L2) that should blink alternately. The program will turn on one light while turning off the other and vice versa

```
|----[ Timer1 ]----[ Coil_L1 ]----|
|                                 |
|----[ Timer2 ]----[ Coil_L2 ]----|
```

In this ladder logic diagram:

- **[Timer1]:** Represents the first timer. When this timer reaches its preset time, it activates the associated output, Coil_L1.
- **[Coil_L1]:** Represents the output coil for Light 1. It turns Light 1 on when activated.
- **[Timer2]:** Represents the second timer. When this timer reaches its preset time, it activates the associated output, Coil_L2.
- **[Coil_L2]:** Represents the output coil for Light 2. It turns Light 2 on when activated.

Steps for Implementing Blinking Lights:

Identify Outputs:

Determine the outputs (lights) that you want to control. In this example, we have Light 1 (L1) and Light 2 (L2).

Configure Timers:

Configure two timers (Timer1 and Timer2) in your PLC programming software. Set the preset time for each timer according to the desired blink rate.

Create Ladder Logic:

Use ladder logic symbols to represent the timers and output coils. Connect them appropriately to create the ladder diagram.

Program the Logic:

Write the ladder logic program based on the ladder diagram. Define the conditions for the timers to activate the output coils.

Upload and Run the Program:

Upload the ladder logic program to the PLC. Set the PLC to "Run" mode to execute the program.

Observe Blinking Lights:

Observe the lights connected to Coil_L1 and Coil_L2 alternately blinking based on the timer settings.

Try

1. Write the ladder diagram for modifying for Additional Lights

14. WATER-LEVEL CONTROL USING PLC

Let's consider a basic scenario where you have a water tank, a pump to fill the tank, and a valve to drain the tank. We'll use two float switches: one for the low water level (LW) and one for the high water level (HW).

```
Start ----[ ]----[ ]---------------------------------------+
          |                                 |
          +----[ ]----( )----[ ]------------------------+
             |    |    |                 |
             +------+     +-----[ ]----( )----------+
                 |        |    |
                 +------------+------+
                 |
                 |       ( )
                 +----------[ ]----- Output (Pump)
                 |
                 +---------------------- Output (Valve)
```

Explanation:

Start: Represents the start of the ladder logic program.
[ ]: Normally open contact (input condition).
( ): Coil (output or action).

Ladder Logic Steps:

1. When the start condition is true, check if the low water level float switch ([ ]) is open.
2. If the low water level float switch is open, activate the pump (Pump [ ]).
3. Check if the high water level float switch ([ ]) is open.
4. If the high water level float switch is open, activate the valve (Valve [ ]).
5. Continue monitoring and repeating the logic.

Try:

1. Explain different types of sensors used for water-level measurement in tanks.
2. Write a ladder logic program for water-level control using a PLC. Include conditions for starting and stopping a pump and a valve based on low and high water-level thresholds.

V. REFERENCE BOOKS:

1. J Nagrath, M Gopal, "Control Systems Engineering", New Age International, 3rd edition, 2007.
2. K Ogata, "Modern Control Engineering", Prentice Hall, 4th edition, 2003.
3. Benjamin Kuo, "Automatic Control Systems", PHI, 7th edition, 1987.

VI. ELECGRONIC RESOURCES:

1. https://www.ee.iitkgp.ac.in
2. https://www.iare.ac.in/

VII. MATERIALS ONLINE

1. Course Outline Description.
2. Laboratory Manual.
3. Laboratory Exercises.