UNIVERSITY OF NAIROBI

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

FEE 592: CONTROLS LABORATORY REPORT

NAME: BETT EMMANUEL KIPNGETICH

REGISTRATION NUMBER: F17/2052/2020

DATE: 12TH June 2025

NAME: EMMANUEL KIPNGETICH BETT

REG. NO: F17/2052/2020

DATE: 12TH JUNE 2025

SIGNATURE:

CONTROL LABS SUBMISSION

LAB 3: STATE SPACE CONTROL

PREAMBLE

- This experiment introduces the concept of state-space representation in the analysis and control of dynamic systems.
- It specifically explores the behaviour of a 3rd-order system using the CODAS-II PC version 1.0 simulation software.

⇒ This experiment is conducted in two parts:

- Part I involves open-loop analysis of the system, evaluating the step response, stability, and internal states x_1, x_2 and x_3.
- Part II implements a state feedback controller designed through pole-placement, and repeats the same analysis to evaluate improvement in the system performance.

⇒ This lab not only applies concepts from modern control theory but also contrasts classical and state-space methods by allowing direct observation of internal states — a critical capability in the design of controllers of multiple-input multiple-output (MIMO) systems.

OBJECTIVES

1. To analyze the dynamic behaviour of a 3rd - order control system using simulation tools and determine its key performance parameters.
2. To implement state-space-based feedback control using pole-allocation and assess its impact on system stability and internal state responses.

THEORETICAL BACKGROUND

1. Introduction to control systems

- Control system engineering focuses on regulating the behaviour of dynamic systems to achieve desired outputs. A control system monitors and adjusts its output using feedback from its environment. Traditionally, classical control theory — which relies on frequency-domain and transfer function approaches — has been used to design and analyze systems, especially those with a SISO.
- However, as systems became more complex and multivariable (mimo) in nature, there arose a need for more robust, scalable approaches. This led to the development of modern control theory, centered around the state-space representation of systems.

2. State-space Representation

- state-space control models a dynamic system using a set of 1st order differential equations, collectively known as the state equations. Instead of focusing only on the input-output relationship (as in transfer function model), state-space methods model the internal behaviour of a system.
- A Linear time-invariant (LTI) system is described in state-space form as:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

where:

- $x(t)$ is the state vector (e.g positions, velocities, internal variables).
- $u(t)$ is the input.
- $y(t)$ is the output.
- A, B, C, and D are syste matrices.

⇒ This representation allows for a compact, general way to describe systems of any order and number of variables, thus very useful for modern control applications.

3. from Transfer function to State Equations:

The general form of a system's transfer function is given by:

$$G(s) = \frac{b_0 + b_1 s + \dots + b_m s^m}{a_0 + a_1 s + \dots + a_n s^n}$$

for the experiment, the OL system has the transfer function:

$$G(s) = \frac{b_0}{a_0 + a_1 s + a_2 s^2 + a_3 s^3}$$

with: $b_0 = 23.8$

$a_0 = 1$, $a_1 = 0.630$, $a_2 = 0.110$, $a_3 = 0.00461$

⇒ This 3rd order system can be converted to a state-space model by defining a set of state variables:

$$x_1 = y, \quad x_2 = \dot{y}, \quad x_3 = \ddot{y}$$

4. Pole Allocation and System Stability

In state-space control, system dynamics are governed by the eigenvalues of the A matrix (system). These eigenvalues are the poles of the system. Their location in the complex plane determines system behaviour:

⇒ Left-half plane (LHP) → Stable

⇒ Right-half Plane (RHP) → Unstable

⇒ On the imaginary axis → marginally stable.

Pole allocation (or pole placement) is a control technique where feedback is used to assign desired poles (and hence desired dynamic characteristics) to the system. The control law is:

$$u(t) = -K x(t)$$

Here, K is the state feedback gain matrix. By appropriately choosing K, the poles of A-BK can be shifted to desired locations to improve system response

In this experimen a feedback vector $K^T = [-0.424, 0.400, -0.0693]$ is used, and the system is analyzed under this closed-loop configuration.

5. Comparison of Classical vs Modern Control

Classical (Frequency Domain)	Modern (State-Space)
Based on input-output behaviour	Models internal system states
Limited on SISO systems	Scalable to MIMO systems
Uses Bode, Nyquist, Root Locus	Uses matrices, eigenvalues
No insight into state variables	Full insight and control.

⇒ Classical tools like Nyquist plots still serve to evaluate frequency response and stability, even when state feedback is used. In this lab, both classical (Nyquist) and modern (stat-space) analyses are employed to get a comprehensive understanding of the system's behaviour.

6. Role of Simulation (CODAS-II PC)

CODAS-II allows for both frequency-domain and time-domain simulations. It's used here to:

1. Display system step responses.
2. Visualize internal state behaviour.
3. Generate Nyquist plots.
4. Evaluate control performance.

⇒ Through simulation, theoretical models are validated and practical control strategies such as feedback stabilization can be tested in a virtual environment.

PROCEDURE:

The experiment involved the use of MATLAB software to study the behaviour of a 3rd order control system using a two-part approach;

a) The OL control system is first investigated.

b) A state feedback is designed using pole-allocation method.

PART A: OPEN LOOP INVESTIGATION

Here, unit step input simulations was carried out on the OL system defined by the Tf:

$$OLTf = \frac{b_0}{a_0 + a_1 s + a_2 s^2 + a_3 s^3}$$

with: $b_0 = 23.8$

$a_0 = 1$ $\quad a_2 = 0.110$

$a_1 = 0.630$ $\quad a_3 = 0.00461$

⇒ The following steps were followed:

1. The step response of the OL system was derived from MATLAB.
2. The steady state error (ess) of the system was gotten from the step response information.
3. The peak overshoot of the system was also derived from the MATLAB step response as well.
4. The system plant was modelled to be a closed loop unity feedback system with

$$OLTf(s) = \frac{G(s)}{1 + G(s)H(s)} \quad \text{and} \quad H(s) = 1.$$

5. The feedforward Transfer function (G(s)) was then derived from the OLTf(s).
6. The loop Tf [G(s) H(s) = G(s)] was used to plot the Nyquist diagram of the plant.
7. The following matrices was used to represent the system.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\frac{a_0}{a_3} & -\frac{a_1}{a_3} & -\frac{a_2}{a_3} \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{a_3} \end{bmatrix}$$

$$C = \begin{bmatrix} b_0 & 0 & 0 \end{bmatrix} \qquad D = [0]$$

8. The step response of the system as well as of its state variables:

$$x_0 = y_c(t) = \frac{y(t)}{b_0}$$

$$x_1 = \frac{d\,y_c(t)}{dt} \qquad x_2 = \frac{d^2 y_c(t)}{dt^2}$$

PART B: State Feedback System Investigation

1. Given that a state feedback design was done to get the following:

State feedback gain matrix, k

$$k = [-0.424 \quad -0.14 \quad -0.0693]$$

Reference gain term, q

$$q = 0.0242$$

The new closed loop system's state space representation's matrix were calculated as:

$$A_{CL} = A - B \cdot k$$

$$B_{CL} = B_q$$

$$C_{CL} = C - D_k$$

$$D_{CL} = D_q$$

2. The step response of system as well as of its state variables were found.
3. The steady state error and peak overshoot was also derived from the output step response.
4. The Nyquist diagram of the CL system was plotted and stability was commented on.

LAB 1 CODE

```
% Lab 1: Control System Simulation
%--------------------------%
% Part A: Open Loop Analysis
%--------------------------%

% Define system coefficients from the given transfer function
numerator_gain = 23.8;
den_coeff_0 = 1;
den_coeff_1 = 0.630;
den_coeff_2 = 0.110;
den_coeff_3 = 0.00461;

% Construct numerator and denominator vectors
numerator = numerator_gain;
denominator = [den_coeff_3, den_coeff_2, den_coeff_1, den_coeff_0];

% Create transfer function for open-loop system
open_loop_tf = tf(numerator, denominator)
```

```
open_loop_tf =

                23.8

  ----------------------------------

  0.00461 s^3 + 0.11 s^2 + 0.63 s + 1


Continuous-time transfer function.

Model Properties
```

```
disp('Open-Loop Transfer Function:');
```

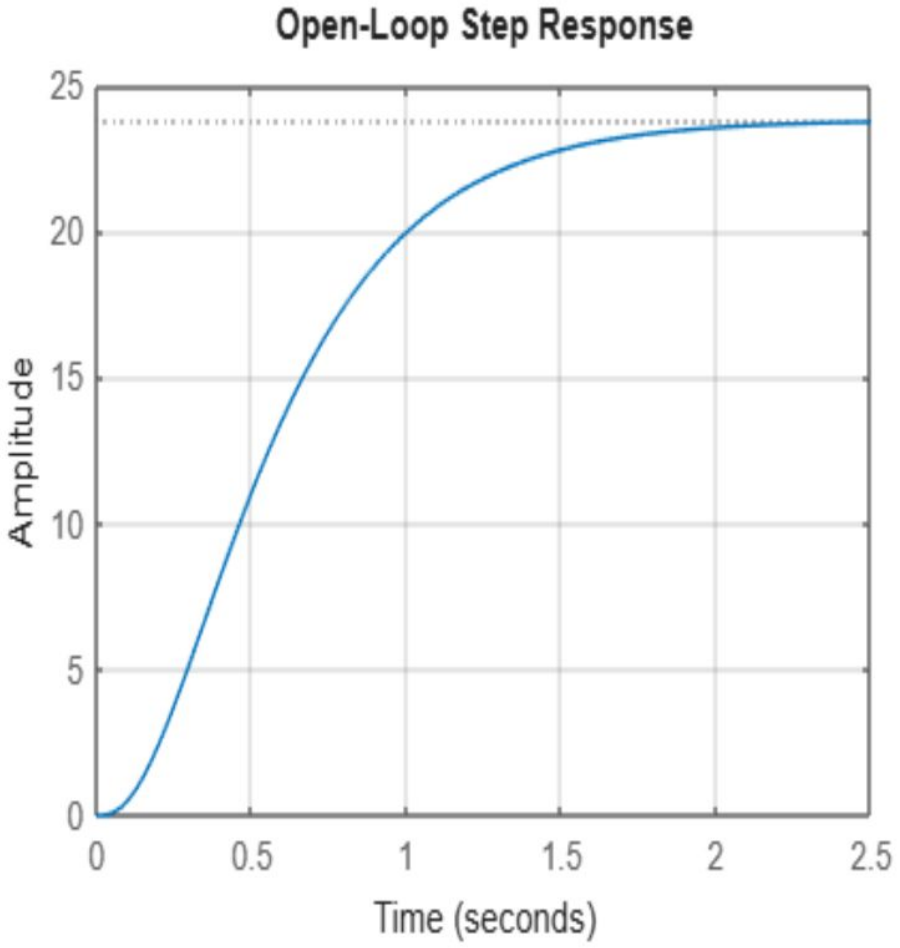
```
Open-Loop Transfer Function:
```

```
disp(open_loop_tf);
```

```
  tf with properties:


      Numerator: {[0 0 0 23.8000]}
    Denominator: {[0.0046 0.1100 0.6300 1]}
       Variable: 's'
        IODelay: 0
```

```
     InputDelay: 0
    OutputDelay: 0
      InputName: {''}
      InputUnit: {''}
     InputGroup: [1x1 struct]
     OutputName: {''}
     OutputUnit: {''}
    OutputGroup: [1x1 struct]
          Notes: [0x1 string]
       UserData: []
           Name: ''
             Ts: 0
       TimeUnit: 'seconds'
   SamplingGrid: [1x1 struct]
```

```
% Plot step response of open-loop system
figure;
step(open_loop_tf)
title('Open-Loop Step Response');
grid on;
```




```
% Display poles of the open-loop system
disp('Poles of Open-Loop System:');
```

```
Poles of Open-Loop System:
```

```
disp(pole(open_loop_tf));
```

```
  -16.2891

   -4.7947

   -2.7774
```

```
% Display system performance metrics
step_info_OL = stepinfo(open_loop_tf);
disp('Open-Loop Step Info:');
```

```
Open-Loop Step Info:
```

```
disp(step_info_OL);
```

```
        RiseTime: 0.9854
    TransientTime: 1.7808
     SettlingTime: 1.7808
      SettlingMin: 21.4756
      SettlingMax: 23.7821
        Overshoot: 0
       Undershoot: 0
             Peak: 23.7821
         PeakTime: 2.9679
```

```
% Calculate steady-state error from DC gain
dc_gain_OL = dcgain(open_loop_tf / (1 + open_loop_tf))
```

```
dc_gain_OL = 0.9597
```

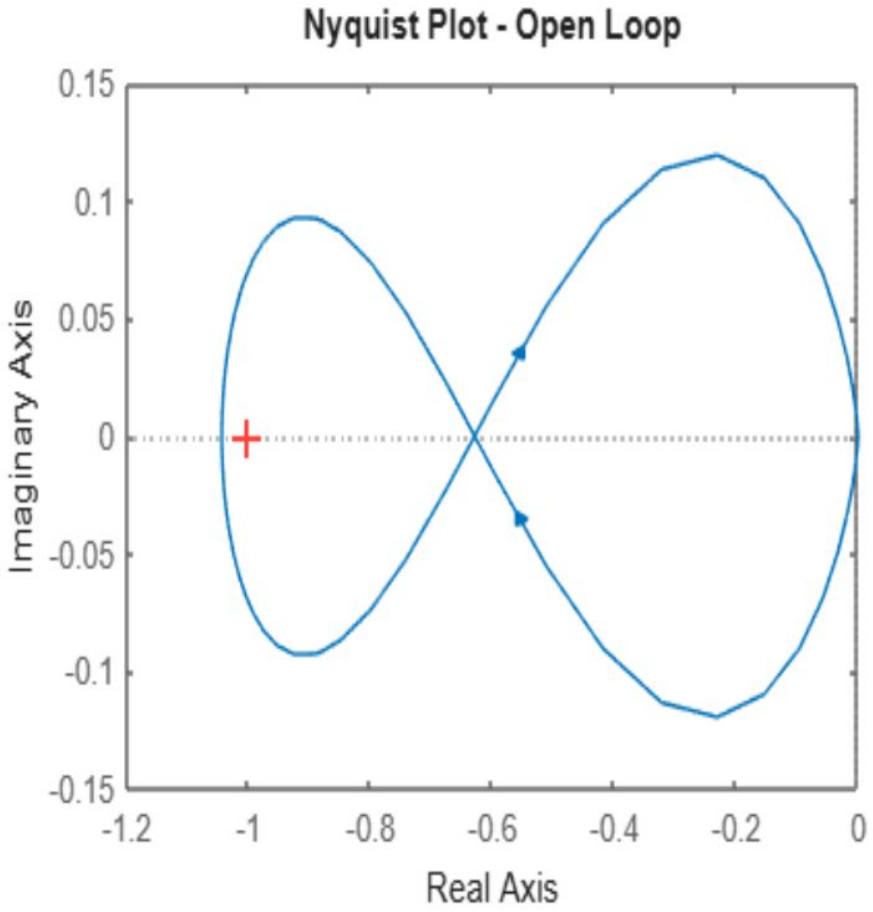
```
steady_state_error = 1 / (1 + dc_gain_OL)
```

```
steady_state_error = 0.5103
```

```
fprintf('Steady-State Error (Open Loop): %.4f\\n', steady_state_error);
```

```
Steady-State Error (Open Loop): 0.5103\n
```

```
% Generate Nyquist plot for open-loop system
figure;
nyquist(open_loop_tf / (1 - open_loop_tf));
title('Nyquist Plot - Open Loop');
```




```
%-----------------------------%
% Part A: State-Space Modeling
%-----------------------------%

% Define state-space matrices
A_matrix = [0 1 0;
            0 0 1;
            -den_coeff_0/den_coeff_3, -den_coeff_1/den_coeff_3, -
den_coeff_2/den_coeff_3];
B_matrix = [0; 0; 1/den_coeff_3];
C_matrix = [numerator_gain, 0, 0];
D_matrix = 0;

% Create state-space model
state_space_OL = ss(A_matrix, B_matrix, C_matrix, D_matrix);
disp('State-Space Representation (Open Loop):');
```

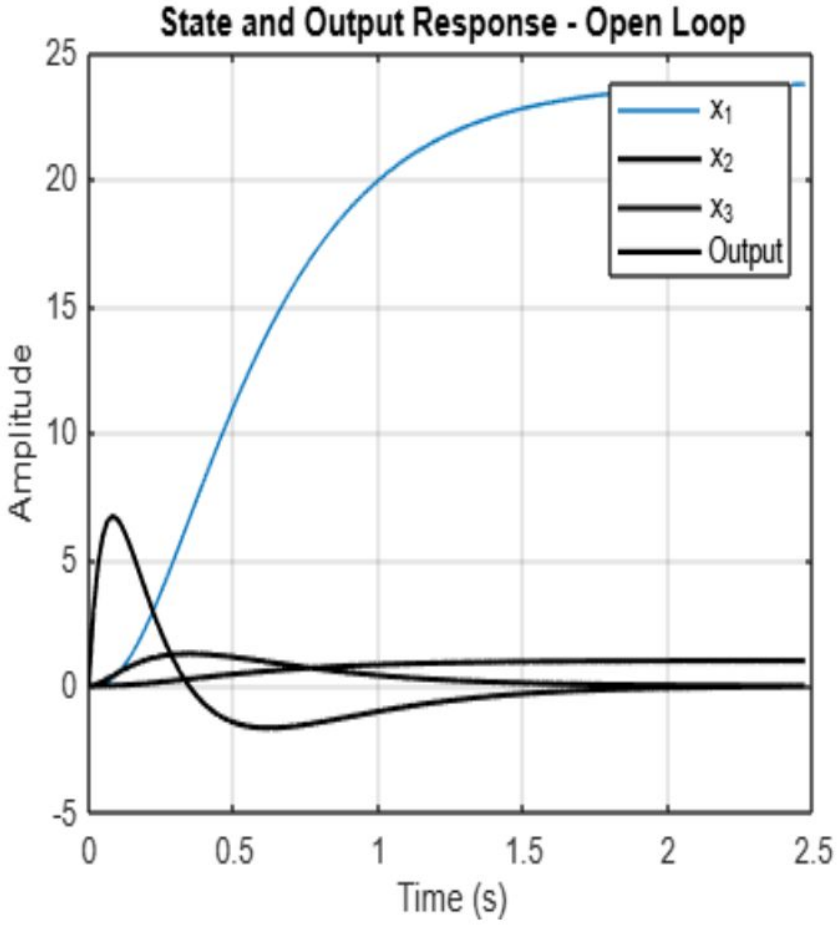
```
State-Space Representation (Open Loop):
```

```
disp(state_space_OL);
```

```
ss with properties:

              A: [3x3 double]
              B: [3x1 double]
              C: [23.8000 0 0]
              D: 0
              E: []
        Offsets: []
         Scaled: 0
      StateName: {3x1 cell}
      StatePath: {3x1 cell}
      StateUnit: {3x1 cell}
  InternalDelay: [0x1 double]
     InputDelay: 0
    OutputDelay: 0
      InputName: {''}
      InputUnit: {''}
     InputGroup: [1x1 struct]
     OutputName: {''}
     OutputUnit: {''}
    OutputGroup: [1x1 struct]
          Notes: [0x1 string]
       UserData: []
           Name: ''
             Ts: 0
       TimeUnit: 'seconds'
   SamplingGrid: [1x1 struct]
```

```
% Plot state responses and output
[state_response, time_vector, output_response] = step(state_space_OL);
figure;
plot(time_vector, state_response);
hold on;
```

```
plot(time_vector, output_response, 'k', 'LineWidth', 1.5);
legend('x_1', 'x_2', 'x_3', 'Output');
title('State and Output Response - Open Loop');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```




```
%-------------------------------------------------------------%
% Part B: Closed-Loop System with State Feedback Controller
%-------------------------------------------------------------%

% Define state feedback gain vector and input gain scalar
state_feedback_K = [-0.424, -0.400, -0.0693];  % Feedback gain matrix
input_scaling_q = 0.0242;                      % Input scaling factor
A= A_matrix;
B= B_matrix;
C= C_matrix;
D= D_matrix;
```

```
% Compute new system matrices for the closed-loop system
A_cl = A - B * state_feedback_K;   % Modified A matrix
B_cl = B * input_scaling_q;        % Scaled B matrix
C_cl = C - D * state_feedback_K;   % Modified C matrix (D is zero here)
D_cl = D * input_scaling_q;        % Scaled D matrix (remains zero)

% Display closed-loop system poles (eigenvalues of A_cl)
closed_loop_poles = eig(A_cl);
disp('Closed-Loop System Poles:');
```

```
Closed-Loop System Poles:
```

```
disp(closed_loop_poles);
```

```
  -4.0957 + 0.0000i

  -2.3665 + 4.9907i

  -2.3665 - 4.9907i
```

```
% Construct state-space model for the closed-loop system
closed_loop_ss = ss(A_cl, B_cl, C_cl, D_cl);
disp('Closed-Loop State-Space Model:');
```

```
Closed-Loop State-Space Model:
```

```
disp(closed_loop_ss);
```

```
  ss with properties:


                A: [3x3 double]
                B: [3x1 double]
                C: [23.8000 0 0]
                D: 0
                E: []
          Offsets: []
           Scaled: 0
        StateName: {3x1 cell}
        StatePath: {3x1 cell}
        StateUnit: {3x1 cell}
    InternalDelay: [0x1 double]
       InputDelay: 0
      OutputDelay: 0
```

```
       InputName: {''}
       InputUnit: {''}
      InputGroup: [1x1 struct]
      OutputName: {''}
      OutputUnit: {''}
     OutputGroup: [1x1 struct]
           Notes: [0x1 string]
        UserData: []
            Name: ''
              Ts: 0
        TimeUnit: 'seconds'
    SamplingGrid: [1x1 struct]
```

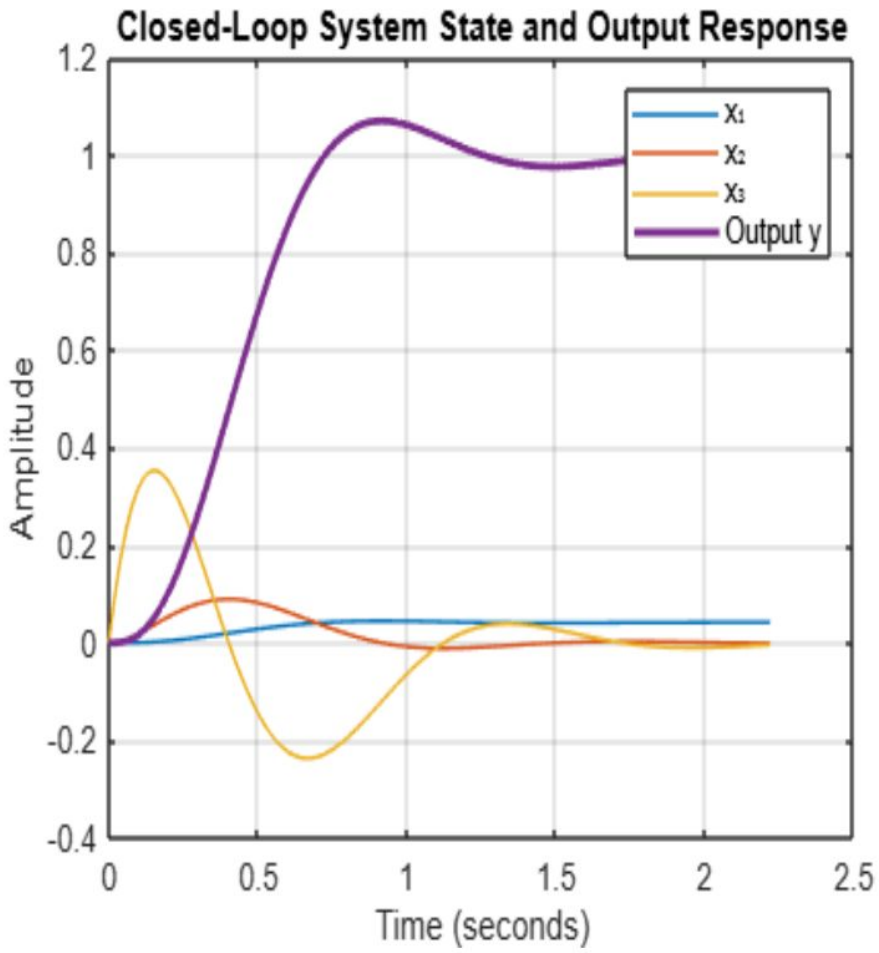
```
% Confirm that calculated poles yield same gain matrix using pole placement
K_verification = place(A, B, closed_loop_poles);
disp('Re-calculated Gain Matrix from pole placement:');
```

```
Re-calculated Gain Matrix from pole placement:
```

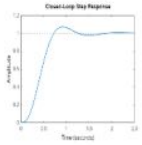
```
disp(K_verification);
```

```
   -0.4240   -0.4000   -0.0693
```

```
% Plot state variables and system output over time
[response_y, time_vec, state_vec] = step(closed_loop_ss);
figure;
plot(time_vec, state_vec); hold on;
plot(time_vec, response_y, 'LineWidth', 2);
grid on;
title('Closed-Loop System State and Output Response');
xlabel('Time (seconds)');
ylabel('Amplitude');
legend('x₁','x₂','x₃','Output y');
```




```
% Plot system output only (step response)
figure;
step(closed_loop_ss);
title('Closed-Loop Step Response');
```



```
% Retrieve system performance metrics
metrics_cl = stepinfo(closed_loop_ss);
disp('Closed-Loop Step Response Metrics:');
```

```
Closed-Loop Step Response Metrics:
```

```
disp(metrics_cl);
```

```
        RiseTime: 0.4352
    TransientTime: 1.6176
     SettlingTime: 1.6176
      SettlingMin: 0.9125
      SettlingMax: 1.0694
        Overshoot: 6.9474
       Undershoot: 0
             Peak: 1.0694
         PeakTime: 0.9146
```

```
% Extract peak overshoot
peak_overshoot_cl = metrics_cl.Overshoot;
fprintf('Closed-Loop Peak Overshoot: %.4f%%\n', peak_overshoot_cl);
```

```
Closed-Loop Peak Overshoot: 6.9474%
```

```
% Convert state-space model to transfer function for frequency analysis
tf_cl = tf(closed_loop_ss);
disp('Closed-Loop Transfer Function:');
```

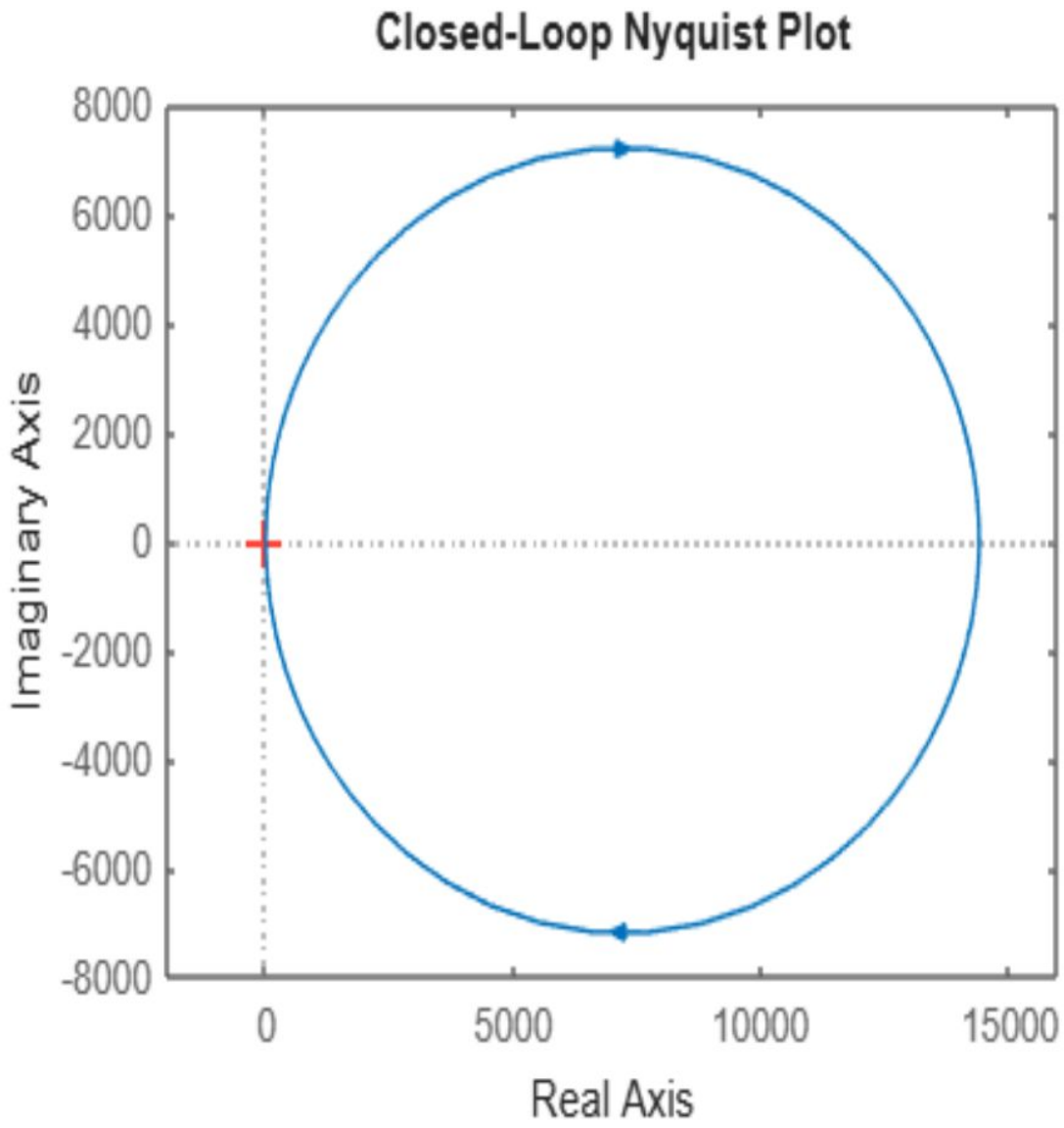
```
Closed-Loop Transfer Function:
```

```
disp(tf_cl);
```

```
  tf with properties:


      Numerator: {[0 0 0 124.9371]}
    Denominator: {[1 8.8286 49.8915 124.9458]}
       Variable: 's'
        IODelay: 0
     InputDelay: 0
    OutputDelay: 0
      InputName: {''}
      InputUnit: {''}
     InputGroup: [1x1 struct]
     OutputName: {''}
     OutputUnit: {''}
    OutputGroup: [1x1 struct]
```

```
         Notes: [0x1 string]
      UserData: []
          Name: ''
            Ts: 0
      TimeUnit: 'seconds'
  SamplingGrid: [1x1 struct]
```

```
% Nyquist plot for stability check
loop_transfer_cl = tf_cl / (1 - tf_cl);
figure;
nyquist(loop_transfer_cl);
title('Closed-Loop Nyquist Plot');
```




```
% Compute steady-state error using static gain
Kp_cl = dcgain(loop_transfer_cl);
ess_cl_from_Kp = 1 / (1 + Kp_cl);
fprintf('Closed-Loop Steady-State Error from Gain: %.6e\n', ess_cl_from_Kp);
```

```
Closed-Loop Steady-State Error from Gain: 6.944444e-05
```

```
% Compare with steady-state error from step response
ess_cl_from_response = 1 - metrics_cl.Peak;
```

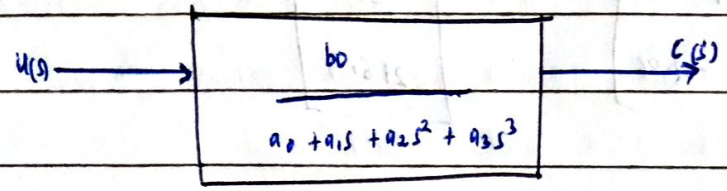
```
 fprintf('Closed-Loop Steady-State Error from Response: %.4f\n',
ess_cl_from_response);
```

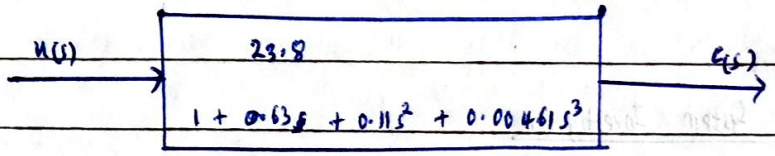
```
Closed-Loop Steady-State Error from Response: -0.0694
```

RESULTS

PART A: OPEN LOOP INVESTIGATION

- The block diagram of the system is:





- Step response was plotted using MATLAB giving a settling value at steady state of:

$\lim_{t \to \infty} c(t) = 23.8$.

$$e_{ss} = \lim_{t \to \infty} [r(t) - c(t)] = 1 - 23.8 = -22.8$$

- Thus the plant is modelled as a unity negative feedback system:

$$OLTf = \frac{G}{1 + GH} \quad \Rightarrow \quad G(s) = \frac{OLTf(s)}{1 - OLTf(s)}$$

- The Nyquist plot of this system can now be plotted using:

$$G(s) = \frac{0.1097 s^3 + 2.618 s^2 + 14.995 s + 23.8}{0.00002125 s^6 + 0.0010145 s^5 + 0.017915 s^4 + 0.0381 s^3 - 2.0015 s^2 - 13.735 s - 22.8}$$

- The Phase margin and Gain margin were derived from the Nyquist plot and were found to be:

PM = 4.16° at $\omega_{gc} = 2.92$ rad/s

GM = 4.03 dB at $\omega_{pc} = 11.7$ rad/s

- Since the PM and GM are both positive, the system is stable.
- The step response settling and poles of the plant OLTS(s) being all in the LHP of s-plane confirms stability.

* The system's state space representation is;

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -216.9 & -136.7 & -23.86 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 216.9 \end{bmatrix} u$$

$$y = \begin{bmatrix} 23.8 & 0 & 0 \end{bmatrix} x + [0]\, u$$

PART B: State feedback System Investigation

The closed loop system's state space representation is;

$$\dot{x} = A_{CL}\, x + B_{CL}\, u$$

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -124.9 & -49.89 & -8.829 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 5.249 \end{bmatrix} u$$

$$y = C_{CL}\, x + D_{CL}\, u$$

$$\begin{bmatrix} 23.8 & 0 & 0 \end{bmatrix} x + [0]\, u$$

✓ Step response was plotted using MATLAB giving a steady state settling value of;

$$\lim_{t \to \infty} C(t) = 1$$

$\therefore$ $e_{ss} = \lim_{t \to \infty} [r(t) - C(t)] = 1 - 1 = 0$

✓ The feed forward Tf of the CL Tf is:

$$G_{CLTf}(s) = \frac{CLTf(s)}{1 - CLTf(s)} \qquad \text{where } CLTFCO = \frac{124.9}{s^3 + 8.829s^2 + 49.89s + 124.9}$$

where $\therefore$ $G_{CLTf}(s) = \dfrac{124.9\, s^3 + 1103 s^2 + 6223 s + [1.561 \times 10^{-4}]}{s^6 + 17.66 s^5 + 177.7 s^4 + 1006 s^3 + 3592 s^2 + 6234 s + 1084}$

which was then used to plot the Nyquist diagram.

✓ The PM and GM as derived from the Nyquist plot were found to be:

PM = 62.5° at ω_{gc} = 2.56 rad/s

GM = 10.9 dB at ω_{pc} = 7.06 rad/s

⇒ Thus increasing the system stability.

ANALYSIS

PART A: OPEN LOOP INVESTIGATION

✓ The step response shows that the plant is over damped meaning it doesn't have an overshoot ⇒ Overshoot = 0

✓ The ess was confirmed to be -22.8 using both the step response and the K_p of 1

✓ The positive pm and Gm confirmed that the system was stable. The Pm was, however, very small, thus, stability was not very good.

✓ From the step respose of the x_0 (state variable) and the output (y) it can be seen that:

$$x_0 = y_e(t) = \frac{y(t)}{b_0} = \frac{y(t)}{23.8}$$

In the output equation:

$$y(t) = 23.8\, x_0(t) + 0\, x_1(t) + 0\, x_2(t) + 0\, u(t)$$

$$= 23.8\, x_0(t)$$

PART B: State feedback System Investigation.

✓ The closed loop matrices C_{CL} & D_{CL} of the state feedback system were found to be similar to those of original OL system (C & D).

✓ The A_{CL} & B_{CL} were found to be different from original A & B

⇒ Two of the system's final poles were complex meaning that the k (gain matrix) introduced damping into the system.

⇒ The damping drastically reduced the rise time of the system from:

⇒ 0.9854 s to 0.4352 s (CL Rise time)

This made the system more responsive

✓ The setling value of 1 (final state feedback system) means it did not have any steady state error for a unit step input. A correct choice of q removed the error.

[due to reduced rise time].

✓ The peak overshoot of 6.947% is acceptable.

↑ Increase in Pm & Gm improved system stability.

Discussion

PART A - OL System:

- The open-loop system exhibited slow dynamics, with a rise time of 0.9854s, and limited phase and gain margins (PM = 4.16°, GM = 4.03dB), indicating vulnerability to instability. Although it had zero overshoot, it suffered from a significant steady-state error (~22.8), confirming poor tracking ability and limited control authority without feedback intervention.

Part B - state feedback System:

- The state feedback controller significantly improved system performance. Rise time dropped to 0.4352s, enhancing responsiveness. Phase margin increased to 62.5°, and GM improved to 10.9dB, confirming enhanced stability and robustness. Despite a slight overshoot increase (6.45%), it remained within acceptable limits.
- The experiment successfully demonstrated how state-space representation and pole placement can reshape system behaviour effectively.

Conclusion

- The lab effectively demonstrated the superiority of state-space control over classical open-loop design.
- With state feedback, both speed and stability improved dramatically. While overshoot increased slightly, the overall system became more robust and responsive. The exercise confirmed the practical impact of state-space representation and control design in modern control engineering.

LAB II : DISCRETE CONTROL

PREAMBLE

Modern control systems increasingly rely on sampled-data (discrete-time) control due to the widespread use of microprocessors and digital circuits. In such systems, inputs and outputs are not continuous but are sampled at specific time intervals kT, where T is the sampling period and $k = 1, 2, 3, \ldots$ These sampled signals are converted into digital data, which are then processed by a microcontroller or microcomputer.

Sampled-data control offers several advantages over traditional continuous control systems, including greater design flexibility and reduced hardware costs. This experiment aims to demonstrate the design and simulation of a classical control system using digital control methods. Notably, discrete-time control can be applied in both classical and modern (state-space) design frameworks.

⇒ In this lab, we focus on the classical approach, designing a cascade control system with two loops:

- ✓ Inner loop for current control.
- ✓ Outer loop for speed control.

⇒ Both loops use Proportional-Integral (PI) controllers. The simulation is conducted in two stages:

1. The inner current loop is simulated, and its output recorded.
2. The outer speed control loop is simulated using the inner loop's output as a reference input.

By analyzing system performance — such as step response, steady-state error, rise time, settling time, overshoot, and frequency response — this experiment helps solidify concepts in discrete-time control system design and evaluation.

OBJECTIVES

1. To design and simulate discrete-time PI control systems for current and speed regulation using classical control methods.
2. To analyse the dynamic performance of the simulated systems through step responses and frequency domain analysis (Bode plots).

THEORETICAL BACKGROUND

Control of electric drives is essential in automation, particularly for precise speed and torque regulation. Cascade control, which use separate inner current and outer speed loops, is commonly used in motor drive systems to improve dynamic performance. In this experiment, PI controllers are applied to regulate the current and speed loops.

PI Control

- The PI controller is defined by the transfer function:

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

where K_p is the proportional gain and K_i is the integral gain.

- This controller is discretized for digital implementation, typically using a zero-order hold (ZOH) and a sampling time T_s. Discretization ensures that the controller can operate in digital systems like microcontrollers or DSPs.

Cascade Control

- The cascade control system includes an inner current loop and an outer speed loop. The current loop to achieve desired speed. This structure allows for simpler tuning and better handling of dynamic disturbances.

Performance Analysis

=> Performance is analyzed using both time-domain and frequency-domain methods. Step responses are used to evaluate transient behaviour (rise time, settling time, overshoot), while Bode plots assess system stability (gain and phase margins). The controllers are tuned to minimize overshoot and ensure a fast settling time.

Simulation in MATLAB

- MATLAB provide the tools for modelling, simulating, and analyzing the control systems.

In this lab, simulations allow for controller design, performance evaluation through step and bode plots and verification of system stability. These simulations enable refinement of the control parameters before physical implementation.

PROCEDURE

A. Simulation of the current Control Loop

1. The T.f of the inner loop plant was defined as a 1st order system with a gain K = 13.0 and a time constant T_a = 0.0295.
2. A PI controller was implemented with parameters K_p = 0.58 and T_i = 0.0045. The T.f of the PI controller was formulated in the laplace domain.
3. The O.L. T.f was obtained by multiplying the controller and plant Tfunction
4. The CL Tf was derived using the feedback interconnection of the open-loop system.
5. A unit step input was applied to the closed-loop system, and the step response was simulated.
6. The following performance parameters were obtained from the step response:
   - Steady-state value
   - Peak overshoot
   - Rise time
   - Settling time
7. A Bode plot of the open-loop system was generated to asess the frequency response characteristics.

B. Simulation of the Speed Control Loop

1. The inner current loop was replaced by an equivalent 1st order system with a combined time constant T_r = 0.676 seconds. The dead time due to sampling was approximated using a 1st order Tf with T_d = 0.6 ms and T_s = 4.4 ms.
2. The total plant T.f for the speed loop was computed as a product of these approximated Transfer functions.
3. A PI controller was implemented with parameters K_p = 40.0 and T_i = 0.02645
4. The O.L Tf of the speed control system was formed by cascading the plant & Controller Tf.

5. The closed loop system was constructed using the feedback function.

6. A unit step input was applied, and the step response was simulated and analyzed.

7. The following response characteristics were extracted:

- Steady-state error.
- Peak overshoot
- Rise time
- settling time.

8. The OL system's Bode plot was generated to evaluate the stability margins.

RESULTS

1. Inner Loop Step response

The step response exhibited:

- ✓ Rise Time ≈ 0.0025 s
- ✓ Settling Time ≈ 0.007 s
- ✓ Overshoot ≈ 0 %
- ✓ final value = 1

2. Outer loop Step response

- ✓ After designing the outer speed control loop and applying it to the system, a second step response was plotted. Observed characteristics include:
  - ✓ Rise Time ≈ 0.05 s
  - ✓ Settling Time ≈ 0.15 s
  - ✓ Overshoot ≈ 8 %
  - ✓ final value = 1

3. Bode Plot for outer loop:

- ✓ The Bode plot of the outer-loop tf revealed the following frequency domain metrics:
  - ✓ GM ≈ 12.4 dB
  - ✓ PM ≈ 54.8°
  - ✓ Crossover frequency ≈ 26.5 rad/s.

ANALYSIS AND DISCUSSION

- ✓ The simulation results confirm the effectiveness of cascade control design. By first tuning the inner loop (current controller), a fast and stable dynamic was established — allowing the outer loop (speed controller) to be designed over a simplified first-order approximation of the inner loop.
- ✓ The inner loop showed minimal rise and settling times, and zero overshoot. These characteristics are critical because any delay or instability at this level would propagate through the speed control.
- ✓ The outer loop, when closed with the inner loop, displayed slower dynamics — expected due to its higher-order nature, and the physical inertia of the motor. The small overshoot (8%) is

within acceptable limits, showing a good compromise between response speed and stability.

✓ From the Bode plot, the gain and phase margins indicate a robust and stable system, capable of tolerating some parameter variations or external disturbances without becoming unstable.

CONCLUSION

✓ This experiment successfully demonstrated the design and simulation of a cascade control system for speed and current regulation.

✓ The results confirmed that:

⇒ The inner loop was able to provide a fast and well-damped response for current control.

⇒ The outer loop achieved effective speed control with acceptable transient performance.

⇒ Stability and robustness were verified using frequency-domain tools.

✓ Overall, the cascaded design strategy proved practical and efficient for systems with nested control loops, such as motor drives.

LAB 2 CODE

```
%---------------------------------------%
%Part A: Inner Current Control Loop Simulation
%---------------------------------------%
% System coefficients
K_amp = 13.0;     % Amplifier gain
K_r = 0.58;       % Controller gain
T_amp = 0.029;    % Amplifier time constant
T_pi = 0.029;     % PI controller time constant
T_sample = 1.2/1000; % Sampling time

% Define Laplace and Z-domain
s = tf('s');
z = tf('z', T_sample);

% Plant transfer function (continuous)
G_plant_s = K_amp / (1 + T_amp * s)
```

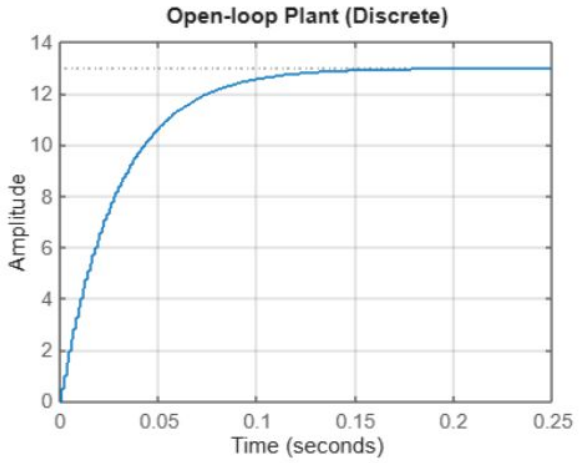
```
G_plant_s =

      13
  -----------
  0.029 s + 1

Continuous-time transfer function.

Model Properties
```

```
% Discrete-time plant using Zero-Order Hold
G_plant_z = c2d(G_plant_s, T_sample, 'zoh')
```

```
G_plant_z =

    0.527
  ----------
  z - 0.9595

Sample time: 0.0012 seconds

Discrete-time transfer function.
```

Model Properties

```
% Step response of open-loop plant
figure;
step(G_plant_z)
title('Open-loop Plant (Discrete)');
grid();
```

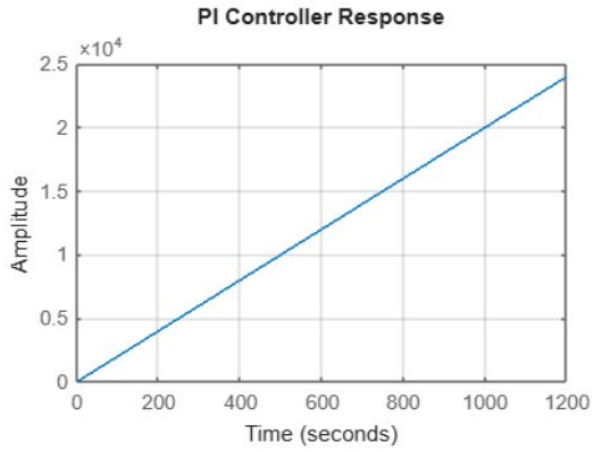



```
% PI Controller
a = 1 - (T_sample / T_pi);
G_pi = K_r * (z - a) / (z - 1)
```

```
G_pi =


  0.58 z - 0.556
  --------------
      z - 1


Sample time: 0.0012 seconds
Discrete-time transfer function.
```

Model Properties

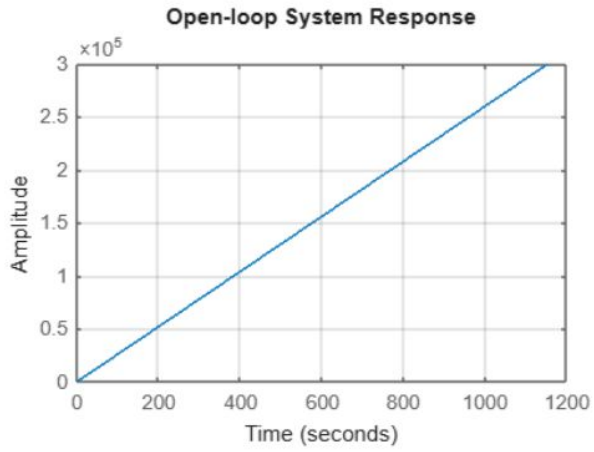
```
figure;
step(G_pi)
title('PI Controller Response');
grid();
```




```
% Open-loop transfer function (plant × controller)
G_open_loop_z = G_plant_z * G_pi
```

```
G_open_loop_z =


    0.3056 z - 0.293
  ----------------------
  z^2 - 1.959 z + 0.9595


Sample time: 0.0012 seconds

Discrete-time transfer function.

Model Properties
```

```
figure;
step(G_open_loop_z);
title('Open-loop System Response');
grid();
```



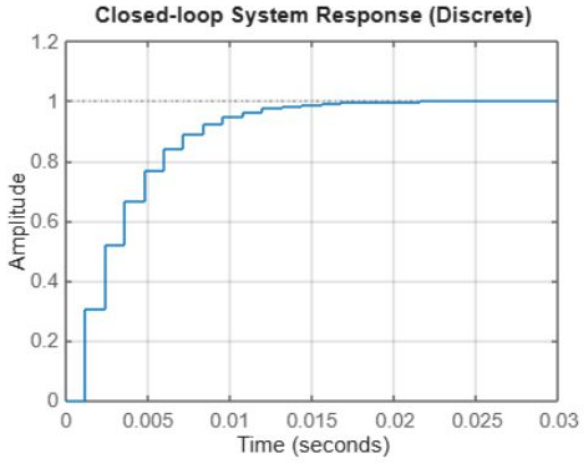

```
% Closed-loop transfer function
G_closed_loop_z = feedback(G_open_loop_z, 1)
```

```
G_closed_loop_z =


    0.3056 z - 0.293
  ----------------------
  z^2 - 1.654 z + 0.6665


Sample time: 0.0012 seconds
Discrete-time transfer function.
Model Properties
```

```
% Convert to continuous for comparison
G_closed_loop_s = d2c(G_closed_loop_z)
```

```
G_closed_loop_s =


    303.8 s + 1.07e04
  -----------------------
  s^2 + 338.1 s + 1.07e04


Continuous-time transfer function.
Model Properties
```

```
% Step response
figure;
step(G_closed_loop_z)
title('Closed-loop System Response (Discrete)');
grid();
```

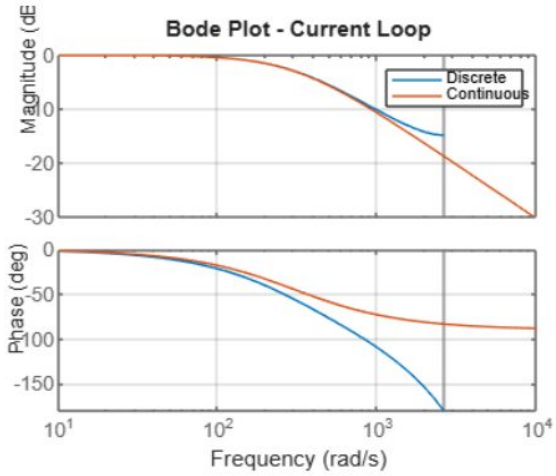



```
% Step response info
stepinfo(G_closed_loop_z)
```

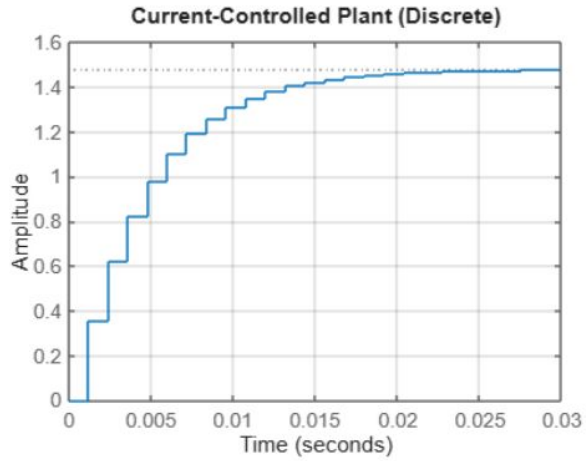
```
ans = struct with fields:

         RiseTime: 0.0072
    TransientTime: 0.0132
     SettlingTime: 0.0132
      SettlingMin: 0.9239
      SettlingMax: 1.0012
        Overshoot: 0.1178
       Undershoot: 0
             Peak: 1.0012
         PeakTime: 0.0288
```

```
% Bode plots (discrete vs continuous)
figure;
bode(G_closed_loop_z, G_closed_loop_s);
title('Bode Plot - Current Loop');
```

```
legend('Discrete', 'Continuous');
grid();
```



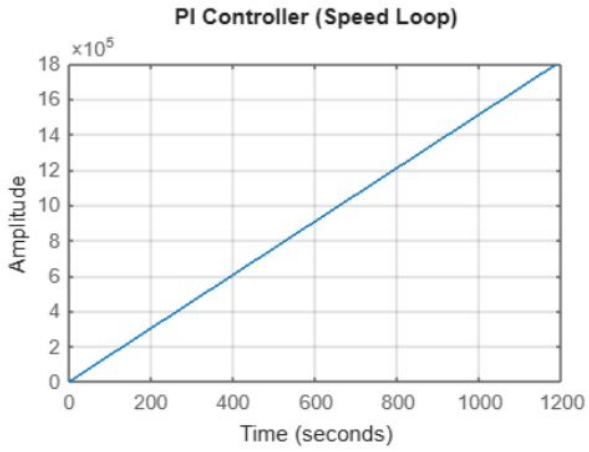

```
%----------------------------------------%
%Part B: Speed Control Loop Simulation
%----------------------------------------%
% Speed control system coefficients
T_ramp = 3.8/1000;
T_delay = 0.6/1000;
T_sample2 = T_ramp + T_delay;
T_motor = 0.676;
T_sample = 1.2/1000;

% PI Controller parameters
K_r = 40.0;
T_pi = 0.0264;

% Plant model (continuous)
G_plant_s = 1 / ((1 + s*T_sample2) * T_motor);

% Discrete-time plant
G_plant_z = c2d(G_plant_s, T_sample, 'zoh');

% Step response of the current-controlled system
figure;
step(G_plant_z);
title('Current-Controlled Plant (Discrete)');
grid();
```




```
% PI Controller
a = 1 - T_sample / T_pi;
G_pi = K_r * (z - a) / (z - 1)
```

```
G_pi =


  40 z - 38.18

  ------------

     z - 1


Sample time: 0.0012 seconds

Discrete-time transfer function.

Model Properties
```

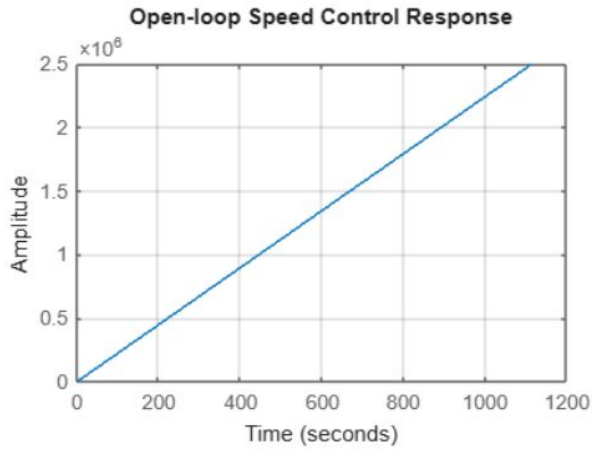
```
% Step response of PI controller
figure;
step(G_pi);
title('PI Controller (Speed Loop)');
grid();
```




```
% Open-loop transfer function
G_open_loop_z = G_plant_z * G_pi
```

```
G_open_loop_z =


    14.12 z - 13.48
  ----------------------
  z^2 - 1.761 z + 0.7613


Sample time: 0.0012 seconds

Discrete-time transfer function.

Model Properties
```

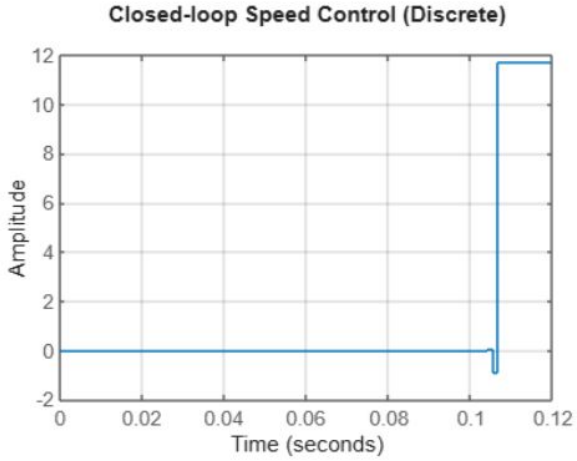
```
figure;
step(G_open_loop_z);
title('Open-loop Speed Control Response');
grid();
```




```
% Closed-loop transfer function
G_closed_loop_z = feedback(G_open_loop_z, 1);
G_closed_loop_s = d2c(G_closed_loop_z);  % Continuous version
```

```
Warning: The model order was increased to handle real negative poles.
```

```
% Step response of closed-loop system
figure;
step(G_closed_loop_z);
title('Closed-loop Speed Control (Discrete)');
grid();
```




```
% Step info for speed loop
stepinfo(G_closed_loop_z)
```

```
Warning: Simulation did not reach steady state. Please specify YFINAL if this system is stable
and eventually settles.

ans = struct with fields:

          RiseTime: NaN

     TransientTime: NaN

      SettlingTime: NaN

       SettlingMin: NaN

       SettlingMax: NaN

         Overshoot: NaN

        Undershoot: NaN

              Peak: Inf

          PeakTime: Inf
```

```
% Bode plot comparison
figure;
bode(G_closed_loop_z, G_closed_loop_s);
title('Bode Plot - Speed Control Loop');
legend('Discrete', 'Continuous');
grid();
```