CONTROL SYSTEM ENGINEERING

(DSC – 13)

Practical File

Submitted By

Goldi Kumari
Roll No.: 23294917143 (Batch: ECE B– B1)
B. Tech Electronics and Communication Engineering
(Semester V)

To

Dr. Vanita Jain
(Assistant Professor)
Department of Electronics and Communication Engineering

FACULTY OF TECHNOLOGY
UNIVERSITY OF DELHI
NEW DELHI – 110007
(2025 – 2026)

Experiment 1

To plot Poles and Zeroes of 1st order and 2nd order systems for given transfer function using MATLAB and Simulink.

0.1 Aim

- Plot pole-zero maps for given transfer functions.
- Simulate time responses (impulse, step, ramp, parabolic, sinusoidal) using MATLAB and Simulink and compare 1st- and 2nd-order system behaviors.

0.2 Theory

A transfer function for an LTI system in the Laplace domain is given by

$$G(s) = \frac{N(s)}{D(s)}$$

- **Poles**: roots of $D(s) = 0$ — determine stability and transient behavior.
- **Zeros**: roots of $N(s) = 0$ — influence response shape and phase.

First-order system:

$$G(s) = \frac{K}{\tau s + 1}$$

Single real pole at $s = -1/\tau$. Typical responses: exponential behaviour for step/impulse; ramp/parabolic show steady-state errors.

Second-order system:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

- Damping ratio ζ and natural frequency ω_n determine oscillatory/overdamped/critically-damped behaviour.

0.3 MATLAB Implementation

0.3.1 Pole-Zero & Time Responses (Script-style, interactive input)

```
clc; clear; close all;
% Accept numerator and denominator coefficients from user
num = input('write coefficient of numerator = ');
den = input('write coefficient of denominator = ');
sys = tf(num, den);

disp('The Transfer Function of the System is:');
sys

% Pole-zero map
figure; pzplot(sys); title('Pole-Zero Map of System'); grid on;

% Time vector and inputs
t = 0:0.01:20;
u_step = ones(size(t));
u_ramp = t;
u_parabolic = 0.5*t.^2; f = 1; u_sine = sin(2*pi*f*t);

% Simulate responses
y_step = lsim(sys, u_step, t);
y_ramp = lsim(sys, u_ramp, t);
y_parabolic = lsim(sys, u_parabolic, t);
y_sine = lsim(sys, u_sine, t);

% Plot responses
```

```
figure;
plot(t, y_step, 'g', 'LineWidth', 1.5); hold on;
plot(t, y_ramp, 'r', 'LineWidth', 1.5);
plot(t, y_parabolic, 'b', 'LineWidth', 1.5);
plot(t, y_sine, 'm', 'LineWidth', 1.5);
grid on; title('System Responses for Different Inputs'); xlabel('Time (s)'); ylabel('
    Output');
legend('Step','Ramp','Parabolic','Sine'); hold off;
```

0.3.2 Example: Third-order vs Second-order (symbolic -¿ numeric)

```
clc; clear; close all;
syms s t;
% Define example transfer functions
sys1 = (s+1)/(s^3+8*s^2+7*s+2); % 3rd order
sys2 = (s+1)/((s+9)*(s+10)); % 2nd order

% Convert to numeric transfer functions
[num1, den1] = numden(sys1);
[num2, den2] = numden(sys2);
n1 = sym2poly(num1); d1 = sym2poly(den1);
n2 = sym2poly(num2); d2 = sym2poly(den2);
tf_1 = tf(n1, d1);
tf_2 = tf(n2, d2);

% Impulse & step
figure; impulse(tf_1, 'r--'); hold on; impulse(tf_2, 'b-'); hold off; title('Impulse
    Response');
figure; step(tf_1, 'r--'); hold on; step(tf_2, 'b-'); hold off; title('Step Response');

% Ramp (analytical via laplace)
ri = t; R = laplace(ri);
Y1 = R * sys1; Y2 = R * sys2;
yt1 = ilaplace(Y1, s, t); yt2 = ilaplace(Y2, s, t);
figure; fplot(yt1, [0 10], 'r--', 'LineWidth', 1.5); hold on; fplot(yt2, [0 10], 'b-', '
    LineWidth', 1.5); hold off;
title('Ramp Response'); legend('3rd order','2nd order');

% Parabolic
p = t^2; P = laplace(p);
Y1_parabolic = P * sys1; Y2_parabolic = P * sys2;
yt1_parabolic = ilaplace(Y1_parabolic, s, t);
yt2_parabolic = ilaplace(Y2_parabolic, s, t);
figure; fplot(yt1_parabolic, [0 10], 'r--', 'LineWidth', 1.5); hold on; fplot(
    yt2_parabolic, [0 10], 'b-', 'LineWidth', 1.5); hold off;
title('Parabolic Response');
```

0.4 Simulink Implementation

- Create a new Simulink model and add `Transfer Function` block.
- For first-order: set numerator/denominator accordingly (e.g. `1/(s+1)`).
- Add input sources: `Step`, `Ramp` (use Signal Builder or a custom block), `Sine Wave`.
- Connect outputs to `Scope` and to `To Workspace` blocks to export `y_step`, `y_ramp`, `y_sine`, `y_impulse`.

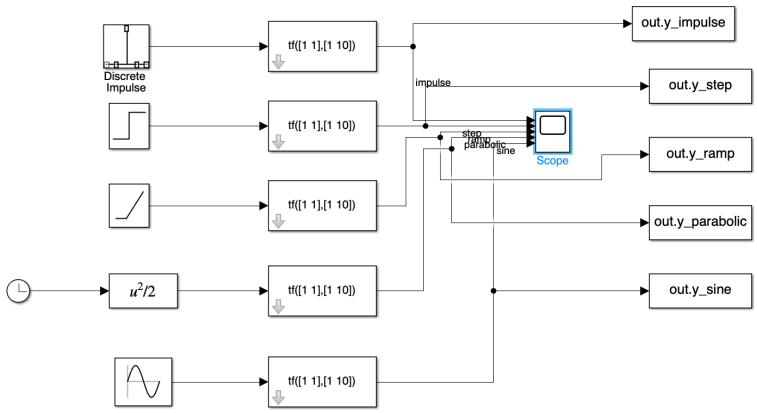


Figure 1: Simulink model for time response

- Example Simulink export plotting (MATLAB):

```
% simOut obtained from sim(modelName);
y_step = simOut.get('y_step');
y_ramp = simOut.get('y_ramp');
y_sine = simOut.get('y_sine');
y_impulse = simOut.get('y_impulse');

figure; hold on;
plot(y_step.Time, y_step.Data, 'LineWidth',1.5);
plot(y_ramp.Time, y_ramp.Data, 'LineWidth',1.5);
plot(y_sine.Time, y_sine.Data, 'LineWidth',1.5);
plot(y_impulse.Time, y_impulse.Data, 'LineWidth',1.5);
legend('Step','Ramp','Sine','Impulse'); title('System Responses exported from Simulink')
    ; grid on; hold off;

% Pole-zero example
s = tf('s'); sys = (s+4)/(s^2 + 1.2*s + 4);
figure; pzplot(sys); title('Pole-Zero Plot'); grid on;
```

0.5 Results

- **Pole-Zero plots:** For 1st order system :

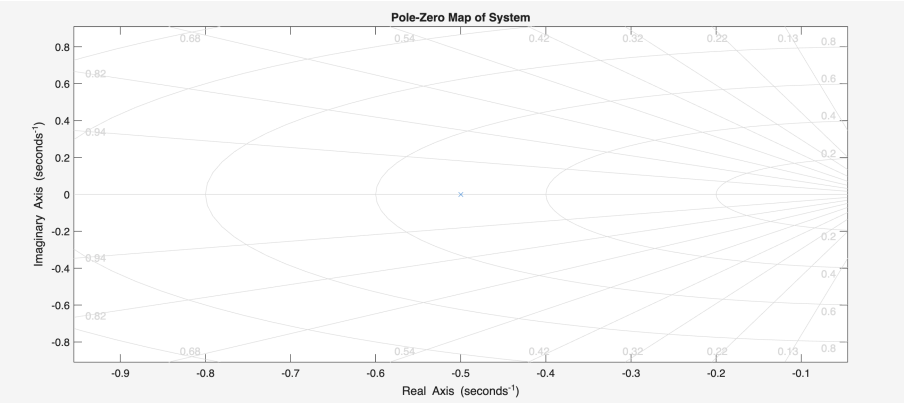


Figure 2: Pole zero plot for 1st order system

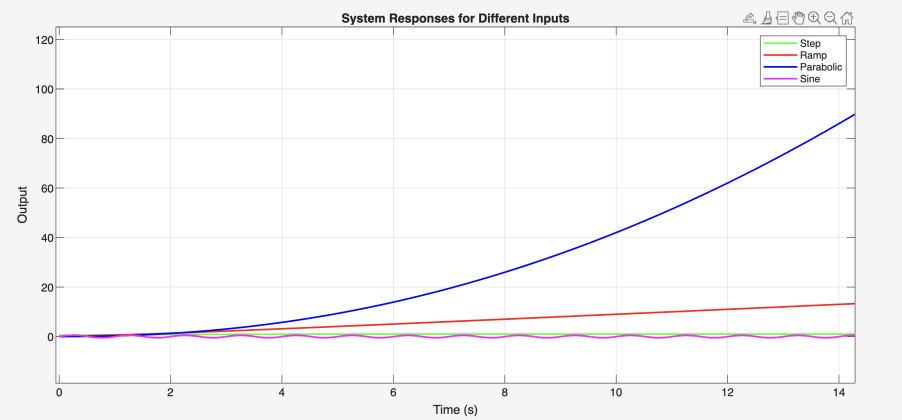


Figure 3: Time response for 1st order system

- **Pole-Zero plots:** For 2nd order system :

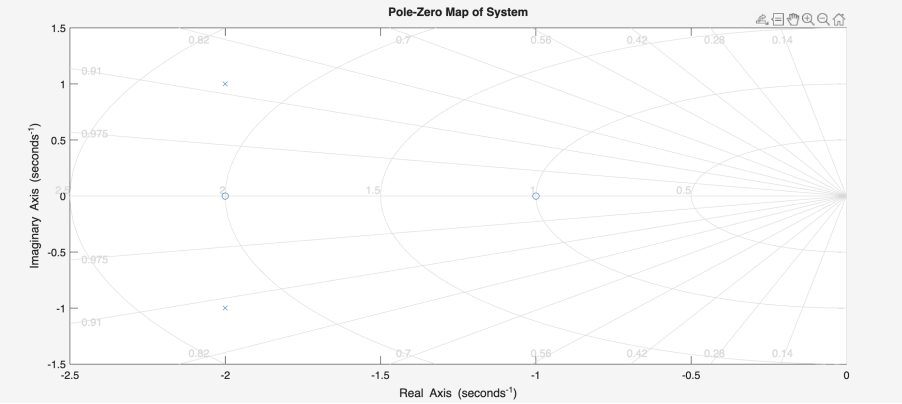


Figure 4: Pole zero plot for 2nd order system

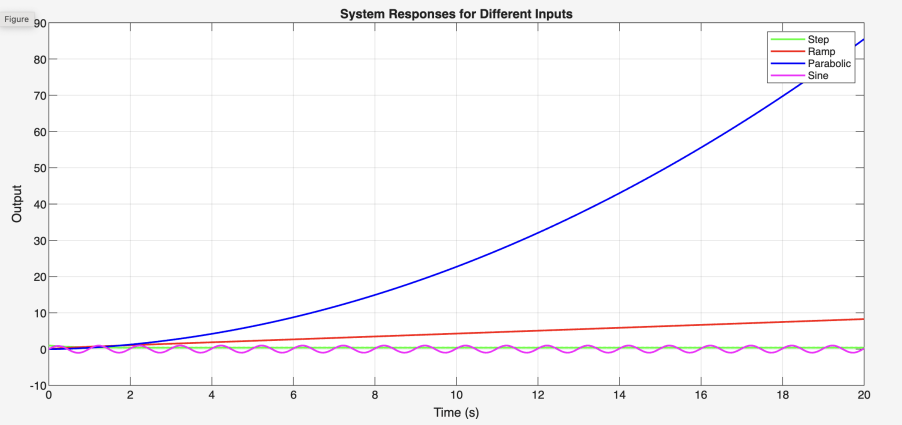


Figure 5: Time response for second order system

*R*esponse from simulink scope ::

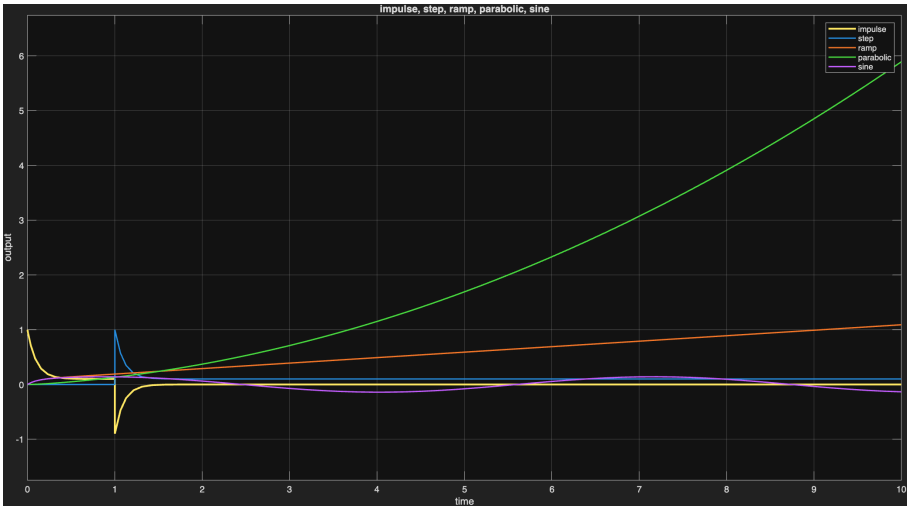


Figure 6: First order system response from simulink's scope

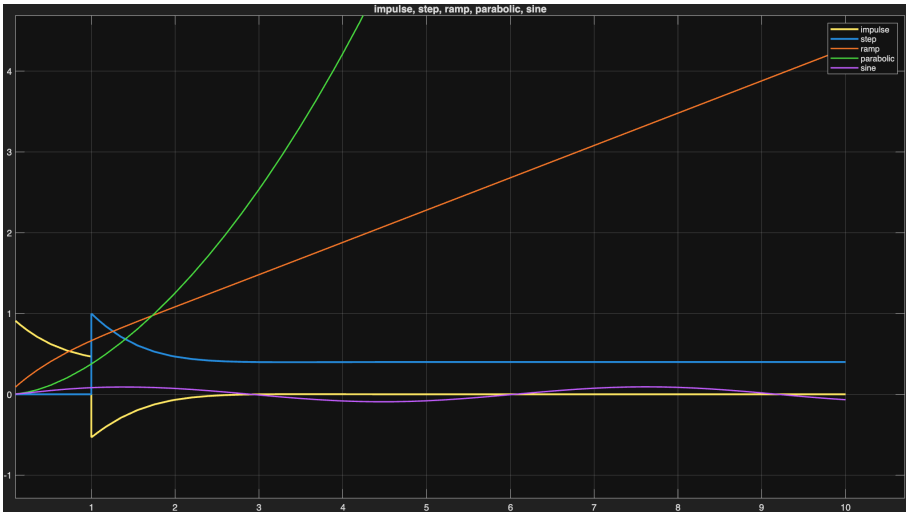


Figure 7: First order system response from simulink's scope

0.6 Discussion

The MATLAB and Simulink simulations successfully illustrated the difference in responses between first-order and second-order systems.

Pole-Zero Analysis:

- The first-order system had a single real pole, ensuring a stable but non-oscillatory response.
- The second-order system showed a pair of complex-conjugate poles with negative real parts, leading to damped oscillations.

Time-Domain Observations:

1. The step response of the first-order system was a smooth exponential rise, while the second-order system displayed oscillations before settling.
2. Ramp and parabolic inputs highlighted steady-state errors. The first-order system had larger errors, showing that higher-order systems improve accuracy.

3. The impulse response of the first-order system decayed monotonically, while the second-order response exhibited oscillations before stabilizing.

4. Under sinusoidal input, the second-order system revealed resonance effects when the input frequency neared its natural frequency, unlike the first-order system.

Both MATLAB code and Simulink models produced consistent results, validating the correctness of the experiment.

0.7 Conclusion

This experiment demonstrated that the pole-zero configuration of a system directly determines its dynamic performance. A first-order system is inherently stable with a simple exponential response, whereas a second-order system can show oscillations, critical damping, or sluggish behavior depending on its damping ratio.

Key findings include:

- Step inputs cause exponential or oscillatory responses depending on system order and damping.
- Ramp and parabolic inputs result in steady-state errors, more prominent in lower-order systems.
- Sinusoidal inputs emphasize the frequency response and potential resonance in second-order systems.

Overall, the experiment confirmed theoretical expectations, and the MATLAB/Simulink simulations provided clear visualization of system characteristics.

0.8 References

- Control system engineering 7th Edition - Norman S. Nise
- Control system engineering - LJ. NAGRATH• M. GOPAL

1 Experiment 2

Closed loop Transfer Function of block diagram via block diagram reduction method using Matlab Aim: To find the closed loop transfer function of multi-loop feedback block diagram via block diagram reduction method using Matlab.

Apparatus Required :

1. **MATLAB Software**

Theory

Block Diagram Reduction of Multi-loop Feedback System :

A block diagram is a way to visually represent the relationships between different parts of a control system using blocks for system components and arrows for signals. Each block represents a **transfer function**, and feedback paths show how the output can affect the input, either directly or indirectly.

In practical systems, we often see multiple feedback loops. These feedbacks are used to control the behavior of the system, such as stability and accuracy. To analyze or design such systems, engineers use **block diagram reduction methods**. This means simplifying the network of blocks and feedback loops into a single transfer function that relates the system's input to its output.

The basic rules for block diagram reduction are:

- **Series Connection:** When two blocks are in series, their transfer functions multiply.
- **Parallel Connection:** When two blocks are in parallel, their transfer functions add.
- **Feedback Loop:** For a block G with a feedback H, the total transfer function becomes:

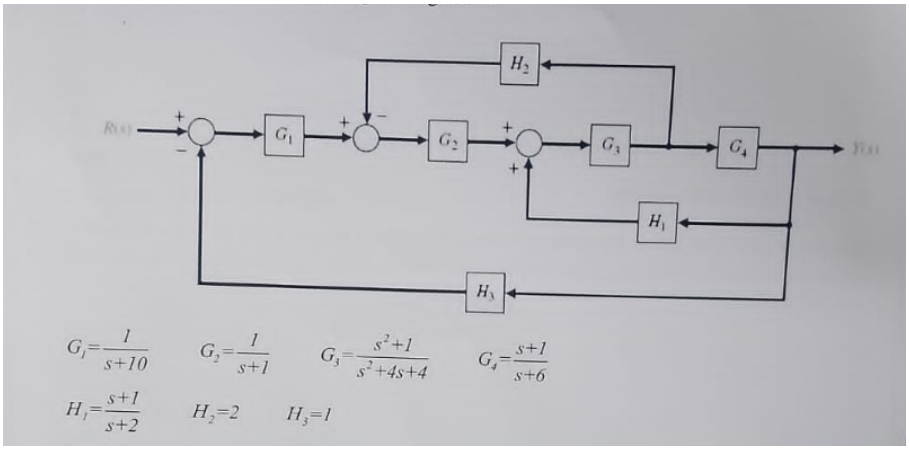
$$\frac{G}{1+GH}$$

  if the feedback is negative.

In this experiment, the block diagram has several blocks arranged in both forward and feedback paths. The aim is to reduce this complex multi-loop structure to a single transfer function, using the above rules step by step.

Once the block diagram is simplified, we find the overall transfer function $\frac{Y(s)}{R(s)}$, where $R(s)$ is the input and $Y(s)$ is the output. This transfer function tells us how the system reacts to any given input in terms of frequency response and dynamic behavior.

Diagram / Problem :



Code

```
clc;
clear all;
close all;

s = tf('s');

G1 = 1/(s + 10);
G2 = 1/(s + 1);
G3 = (s^2 + 1)/(s^2 + 4*s + 4);
G4 = (s + 1)/(s + 6);

H1 = (s + 1)/(s + 2);
H2 = 2;
H3 = 1;


sys = H2/G4 ;

G_3_4_series = series(G3,G4);

G_3_4_H_1_feedback = feedback(G_3_4_series,H1,+1);

G_3_4_H_1_G2_series = series(G2,G_3_4_H_1_feedback);

G_3_4_H_1_G2_sys_feedback = feedback(G_3_4_H_1_G2_series,sys,-1);


G_3_4_H_1_G2_sys_G1_series = series(G1,G_3_4_H_1_G2_sys_feedback);


G_3_4_H_1_G2_sys_G1_H3_feedback  = feedback(G_3_4_H_1_G2_sys_G1_series,H3,-1);

disp('Overall Transfer Function Y(s)/R(s):');

G_3_4_H_1_G2_sys_G1_H3_feedback
```

Output

```
Overall Transfer Function Y(s)/R(s):

G_3_4_H_1_G2_sys_G1_H3_feedback =

              s^5 + 4 s^4 + 6 s^3 + 6 s^2 + 5 s + 2
  ------------------------------------------------------------------
  12 s^6 + 205 s^5 + 1066 s^4 + 2517 s^3 + 3128 s^2 + 2196 s + 712

Continuous-time transfer function.
Model Properties
>>
```

Result and Conclusion

The overall transfer function of the given multi-loop feedback system is obtained as:

$$T(s) = \frac{s^5 + 4s^4 + 6s^3 + 6s^2 + 5s + 2}{12s^6 + 205s^5 + 1066s^4 + 2517s^3 + 3128s^2 + 2196s + 712}$$

By systematically applying the rules of block diagram reduction (series, parallel, and feedback simplification), the complex multi-loop system was reduced to a single equivalent transfer function. This simplified transfer function $T(s) = \dfrac{Y(s)}{R(s)}$ fully represents the system dynamics and can be further analyzed for stability, frequency response, and transient behavior.