MECH 467
Lab #3
By Bobsy Narayan
Date: November 29, 2024

Table of Contents

Abstract

This project aims to design and simulate trajectory planning for a two-axis machine. By implementing trajectory generation using various controllers using Simulink, we will study tracking accuracy, contouring accuracy, and system performance for different bandwidth configurations and for generated input plans, simulated output, and measured systems. The experimental and simulated results are compared to analyze the effects of feedrate, mismatched dynamics, and controller design on contouring performance. This project reviews concepts learned in MECH 467 and demonstrates their application to real-world mechatronics systems.

Introduction

Multi-axis trajectory design is a critical component in modern industrial applications, where precise control of machine movement is essential for automated tasks, such as CNC machining & additive manufacturing. This project explores the complete design and implementation of a two-axis machine controller, focusing on trajectory generation, controller design, and contouring performance evaluation. This report involves creating accurate toolpaths using linear and circular trajectory planning, designing lead-lag controllers for low and high bandwidth cases, and analyzing the system's performance through simulation and experimentation. The report is structured to present the methodology, simulation results, experimental validation, and the effects of feedrate and controller bandwidth on system accuracy.

Prelab Part A – Trajectory Generation

In this section, I generated discrete axis commands to control two motors for a 2-axis system to follow a given reference tool path. The expected reference toolpath can be seen in Figure 1.

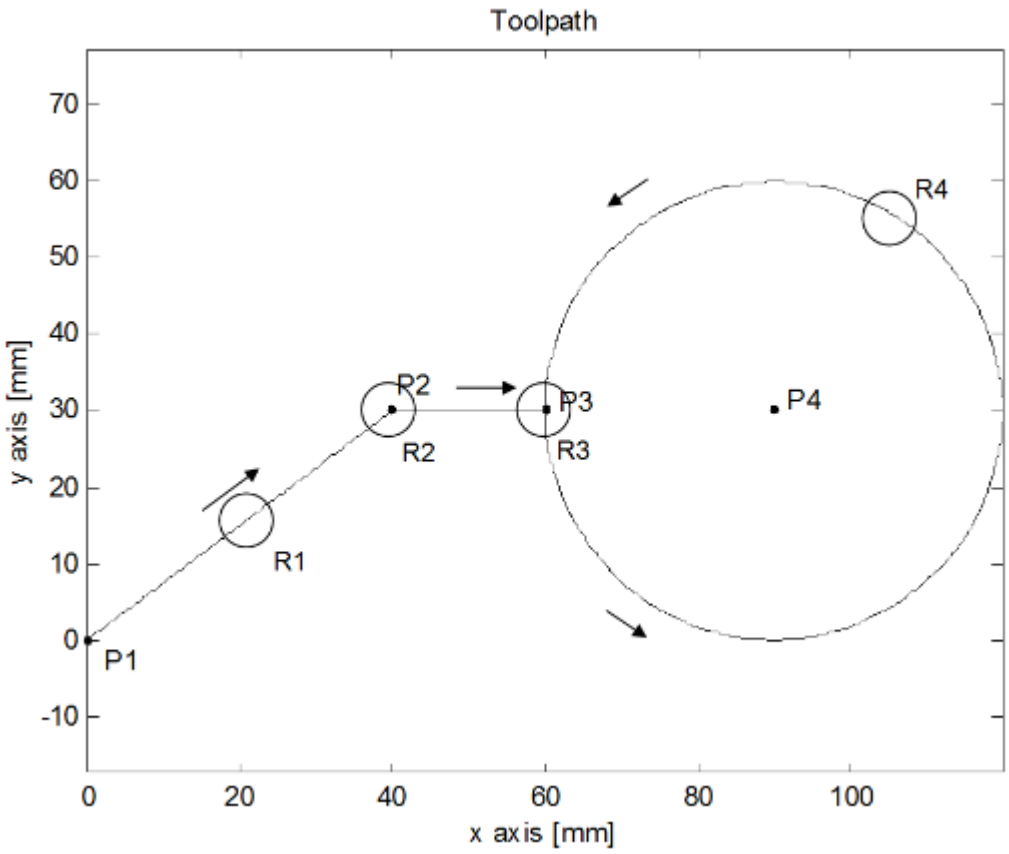


Figure 1: Reference Toolpath for 2 Axis Controller

Our system has set parameters describing the feed profile of the two-axis controller, which can be seen in Table 1.

Table 1: System Trajectory Parameters

Parameter	Value
Feedrate (mm/s)	200
Tangential Acceleration (mm/s^2)	1000
Tangential Deceleration (mm/s^2)	-1000
Interval Time-Steps (ms)	0.01

General steps for developing trajectory axis commands can be seen in the next section. We developed our trajectory planning values in MATLAB, which can be seen in the Appendix.

General Trajectory Generation Steps:

1. Determine System Endpoint, Startpoint, & trajectory parameters.
    a. For this system, these values are given to us.
2. Determine absolute distance values.
    a. For Linear Trajectory, this is linear total distance between the two points.
    b. For Circular Trajectory, this is curve radius, total circle travelled, start angle, and arc length.
3. Determine system acceleration periods & distances.
    a. Using kinematics, we can determine acceleration, deceleration, and constant vel distances and times.
4. Adjust system if constDist < 0.
    a. If this occurs, that means that our system will not have a constant velocity period. Instead, it will have a triangular profile of acceleration and deceleration symmetrically.
    b. For both trajectories, we will adjust our system time & total distances values accordingly. Our accel & decel periods will be equivalent & our const velocity distances and time will be 0.
5. Adjust parameters according to sampling frequency.
    a. Since our system has set discrete stepping times, we will need to change our parameters to match our discrete values.
    b. Firstly, we change our accel, decel, and const vel time ranges to be a multiple of our samplingTime.
    c. Secondly, we recalculate our distance values using our next time ranges.
6. Develop Tangenetial Motion Values
    a. Using kinematic equations, we develop distance, velocity, and acceleration commands for the acceleration, deceleration, and const Vel periods.
7. Break System into X & Y Axis.
    a. Using known angular geometries or direction values, we will break our motion values into the X & Y axis, allowing us to develop trajectory commands for our system & estimate our system velocity and acceleration through simulation.

A1) Plot Generated Toolpath

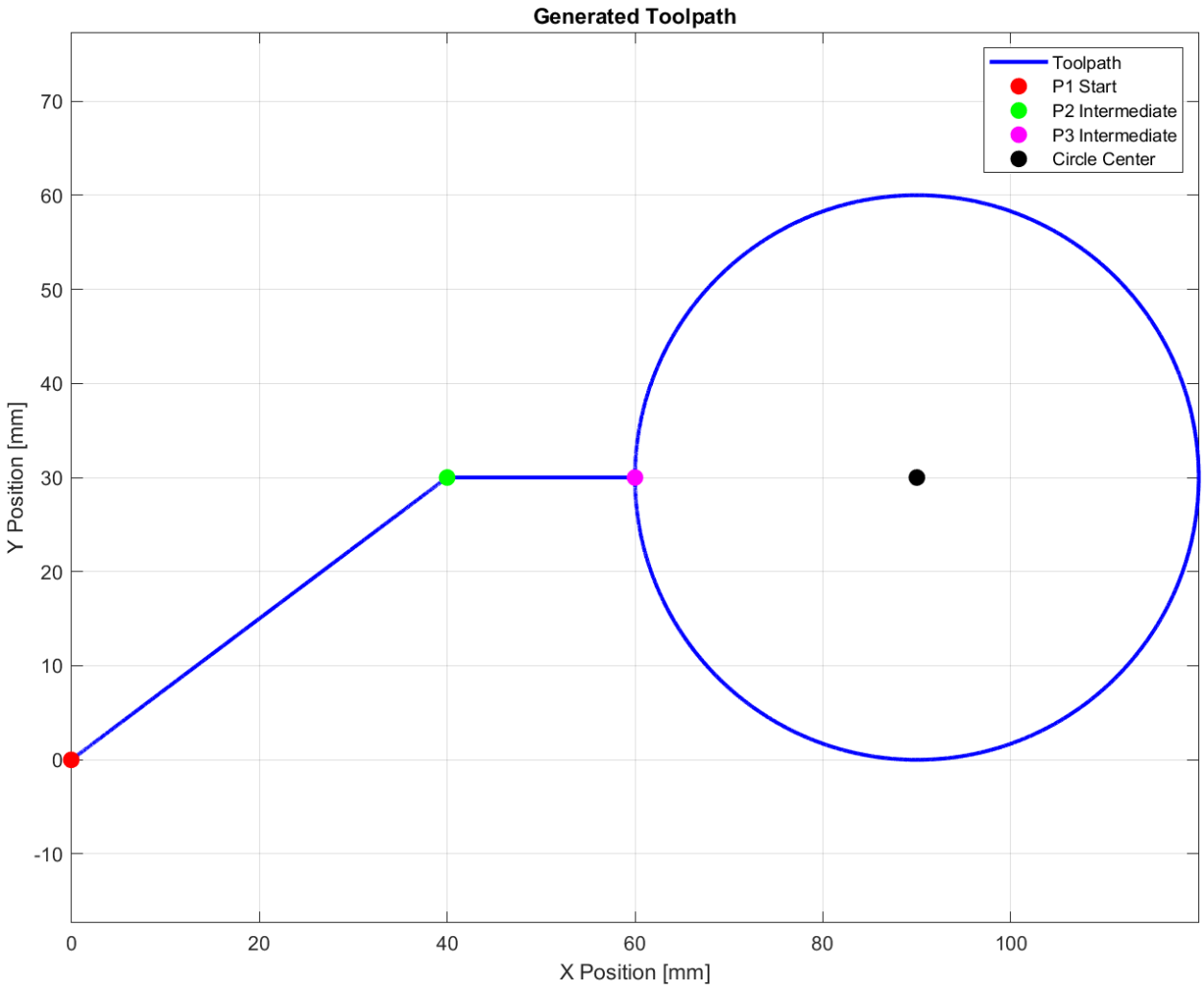


Figure 2: Generated Toolpath: X-Axis Commands vs Y-Axis Commands

A2) Plot Displacement, Feedrate, & Tangetial acceleration profiles as a function of time.

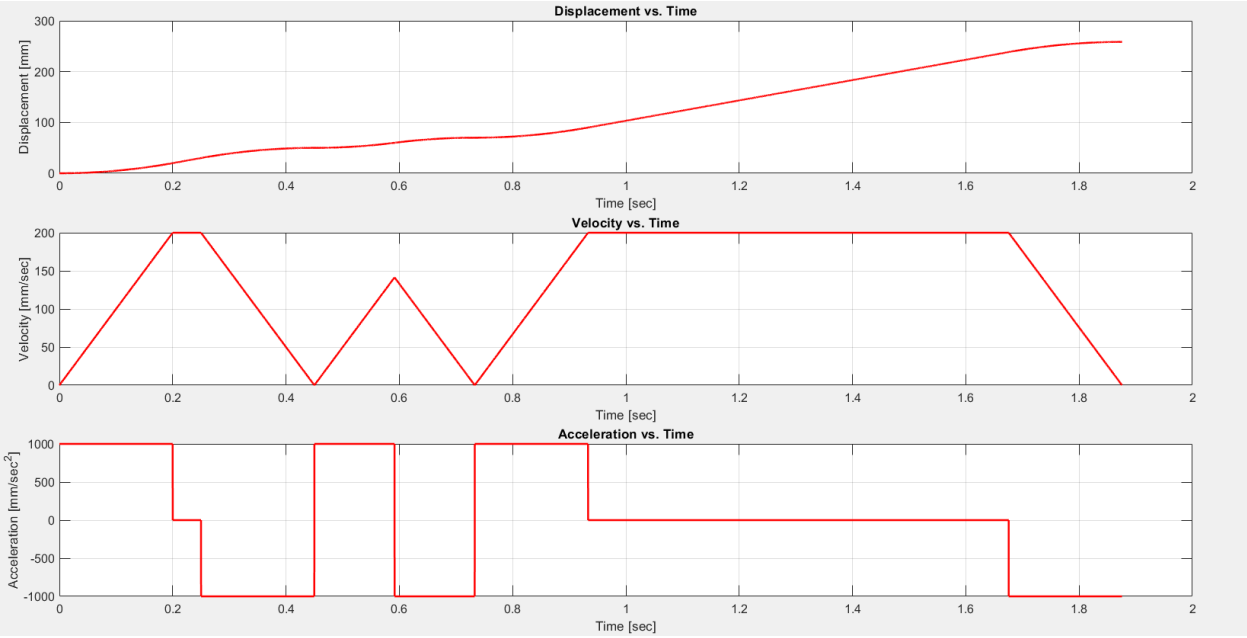


Figure 3: System Motion Profiles Vs Time

A2) Plot X & Y position, velocity, and acceleration as functions of time.

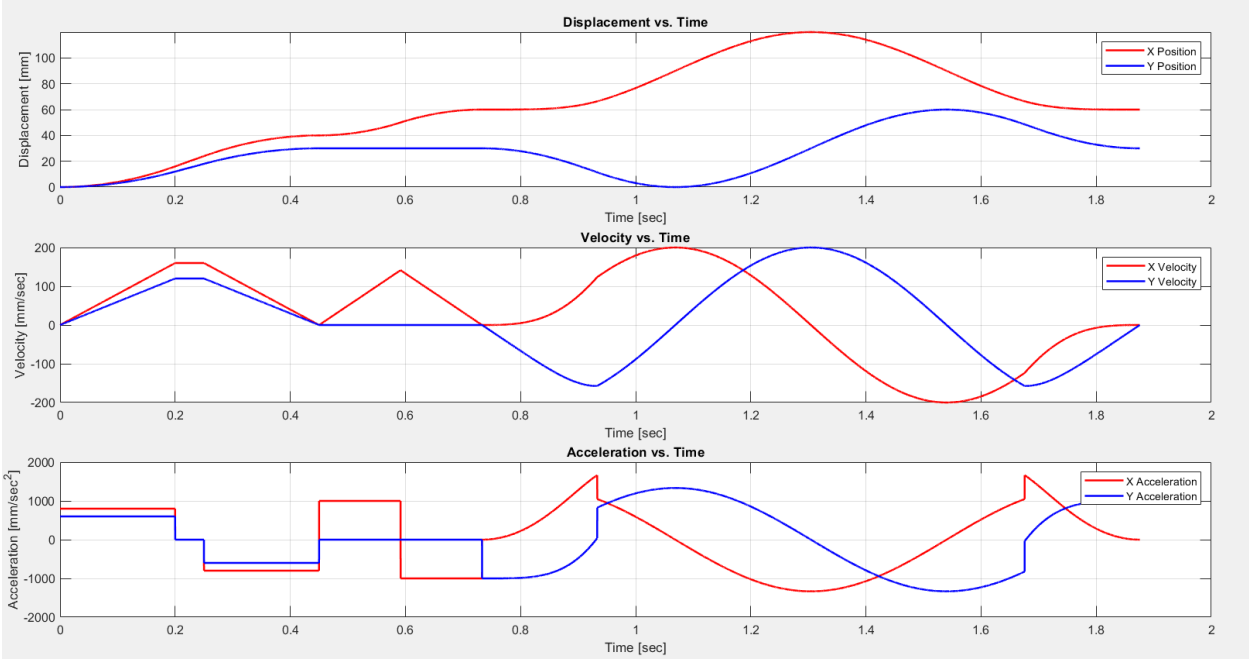


Figure 4: Axis Position, Velocity, and Acceleration Vs Time

Prelab Part B – Two-Axis Controller Design

In this project, an XY ball screw table will be modelled and simulated. The system's parameters and block diagram model can be seen in the following figures.

Table 2: 2 Axis Ball Screw Parameters

Axis Parameters	X Axis	Y Axis
Amplifier Gain (Ka [A/V])	1	1
Motor Torque Constant, (Kt [Nm/A])	0.49	0.49
Encoder Gain, (Ke [mm/rad])	1.59	1.59
Rotational Inertia, (Je [kgm2])	4.36x10^-4	3x10^-4
Viscous Friction, (B [Nm/(rad/sec)])	0.0094	0.0091

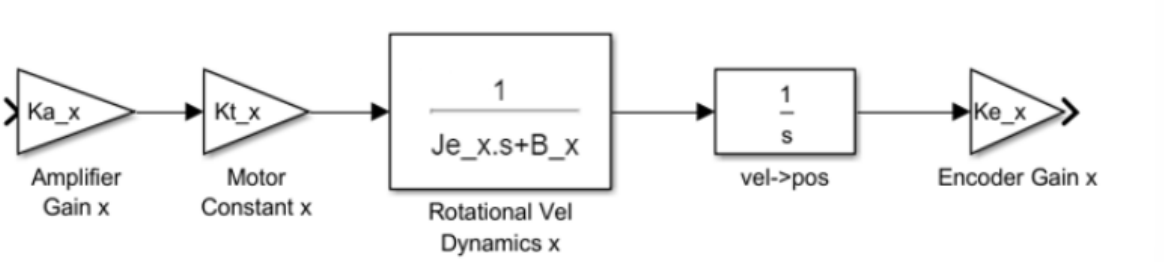


Figure 5: Linear Motor Driven Axis Model (X-Axis)

B1) For this part, we will be creating two different lead lag controllers for each axis, for 2 total lead lag controllers. An integral action will be cascaded in afterwards. PM = 60deg. LBW = 20Hz. HBW = 40Hz.

In the previous lab, we discussed how our Kp will affect the poles of our system, change system stability, and affect our system bandwidth. As Kp changes, our poles changes, which may lead to instability at higher frequencies. Since Kp is apart of our Loop Return Ratio, it will also affect our bandwidth. We want to optimize our Kp so that our unity gain cross over frequency, where the loop return ratio crosses a gain of 1, isn't close to 180, as well as use our lead compensator to add extra phase to compensate for this.

To do this, we will use our relationship between our Loop Return Ratio, Kp, and our unity gain crossover frequency to determine the parameters needed for a proper lead compensator controller.

Below, we have our major system transfer functions and the appropriate steps necessary for creating our lead-lag controllers.

Equation 1: System Forward Path Transfer Function

$$G(s) = \frac{K_a K_t K_e}{s(J_e s + B_e)}$$

Equation 2: System Loop Return Ratio Transfer Function (Since H(s) = 1)

$$G_{OpenLoop}(s) = LRR(s) = Kp \frac{K_a K_t K_e}{s(J_e s + B_e)} \frac{K_i + s}{s}$$

Equation 3: System Closed Loop Transfer Function

$$G_{CL}(s) = \frac{K_p G(s)}{1 + K_p G(s)}$$

Step 1: Determine Kp for Both Axis and for Both Unity Gain Crossover Frequencies.

Using the following equation, we can determine our Kp for a normal proportional gain controller.

Equation 4: Proportional Gain Kp Calculation

$$G_{OpenLoop}(s) = \frac{K_a K_t K_e}{jw_{cross}(J_e * jw_{cross} + B_e)}$$

Table 3: Kp Proportional Gain Parameter for Multiple Axis & Crossover Frequencies

Axis	**20Hz = Crossover Frequency**	**20Hz = Crossover Frequency**
X-Axis - Kp	8.97	35.48
Y-Axis - Kp	6.26	24.50

Step 2: Determine Current Phase for Both Axis & Crossover Frequencies

Using the Kp values calculated before, we can determine the current phase of our open-loop systems. These phase value will be used for determining our Lead Compensator parameters. We will determine where our system unity gain crossover frequency is, where our system gain is 1, and determine the current phase at that point.

This was done in MATLAB and can be seen in our Appendix.

Table 4: Current System Phase at Unity Gain Crossover for Multiple Axis & Crossover Frequencies

Axis	**20Hz = Crossover Frequency**	**20Hz = Crossover Frequency**
X-Axis - $\emptyset_m$ [Deg]	-170.27	-175.10
Y-Axis - $\emptyset_m$ [Deg]	-166.43	-173.12

Step 3: Determine Peak Phase for Both Axis & Crossover Frequencies

Using the following equation, we can determine our peak phase for each transfer function, used to calculate our lead compensator parameters. This was calculated in MATLAB. Our target phase is 60 degrees, as stated in *Lab Manual 3.*

Equation 5: Peak Phase Calculation

$$\emptyset_p = \emptyset_{target} - \emptyset_m - 180^\circ$$
$$\emptyset_p = 60^\circ - \emptyset_m - 180^\circ$$

Table 5: Current System Peak Phase for Multiple Axis & Crossover Frequencies

Axis	**20Hz = Crossover Frequency**	**20Hz = Crossover Frequency**
X-Axis - $\emptyset_p$ [Deg]	50.27	55.10
Y-Axis - $\emptyset_p$ [Deg]	46.43	53.12

Step 4: Determine Lead Compensator Parameters for Both Axis & Crossover Frequencies

Using the following equations, we can determine the necessary parameters for our lead compensator controllers.

Equation 6: Lead Compensator Parameter Calculations

$$\alpha = \frac{1 + \sin \emptyset_p}{1 - \sin \emptyset_p}$$
$$\tau = \frac{1}{w_c \sqrt[2]{\alpha}}$$

Table 6: Current System Alpha Values for Multiple Axis & Crossover Frequencies

Axis	20Hz = Crossover Frequency	20Hz = Crossover Frequency
X-Axis - α	7.6583	10.1186
Y-Axis - α	6.2602	8.9937

Table 7: Current System Alpha Values for Multiple Axis & Crossover Frequencies

Axis	20Hz = Crossover Frequency	20Hz = Crossover Frequency
X-Axis - τ	0.0029	0.0013
Y-Axis - τ	0.0032	0.0013

Step 5: Determine Kp Proportional Gain Parameter for Both Axis & Crossover Frequencies

Using the following equation, we can determine our new Kp value for our lead compensator controller.

Equation 7: Lead Compensator Proportional Gain Calculation

$$|L(jw_c) = 1 = |KC_0(jw_c)G(jw_c)|$$

$$C_0(jw_c) = \frac{\alpha\tau(jw_c) + 1}{\tau(jw_c) + 1}$$

$$G(jw_c) = \frac{K_a K_t K_e}{jw_c(J_e * jw_c + B_e)}$$

$$k_p = \frac{1}{|C_0(jw_c)G(jw_c)|}$$

Table 8: Lead Compensator Kp Proportional Gain Parameter for Multiple Axis & Crossover Frequencies

Axis	20Hz = Crossover Frequency	20Hz = Crossover Frequency
X-Axis - Kp	3.24	11.15
Y-Axis - Kp	2.50	8.17

By completing the above steps, we were able to create 4 different lead compensator controllers, with different crossover frequencies and for different axis. A general form can be seen below, alongside all 4 numerical transfer functions determined in MATLAB.

Equation 8: Lead Compensator Transfer Functions

$$G_{OpenLoop}(s) = Kp\frac{K_aK_tK_e}{s(J_es + B_e)}\frac{K_i + s}{s}$$

$$G_{OL\_LowBandwidth\_X}(s) = \frac{0.05559s^2 + 3.223s + 31.72}{1.254e^{-6}s^4 + 0.000463s^3 + 0.0094s^2}$$

$$G_{OL\_HighBandwidth\_X}(s) = \frac{0.11s^2 + 11.45s + 218.4}{5.454e^{-7}s^4 + 0.0004478s^3 + 0.0094s^2}$$

$$G_{OL\_LowBandwidth\_Y}(s) = \frac{0.03878s^2 + 2.435s + 24.48}{9.542e^{-7}s^4 + 0.0003289s^3 + 0.0091s^2}$$

$$G_{OL\_HighBandwidth\_Y}(s) = \frac{0.07595s^2 + 8.273s + 160.0}{3.98e^{-7}s^4 + 0.0003121s^3 + 0.0091s^2}$$

B2) Overlay the bode plots of the open loop system of X axis with the LBW and HBW controllers. On another figure, plot the same graphs for the closed loop system in s domain.

Bode Plots for our open loop and closed loop systems for the X-Axis can be seen below. Injected phase can be seen inputted into our system in our open-loop controllers bode plots, as expected.

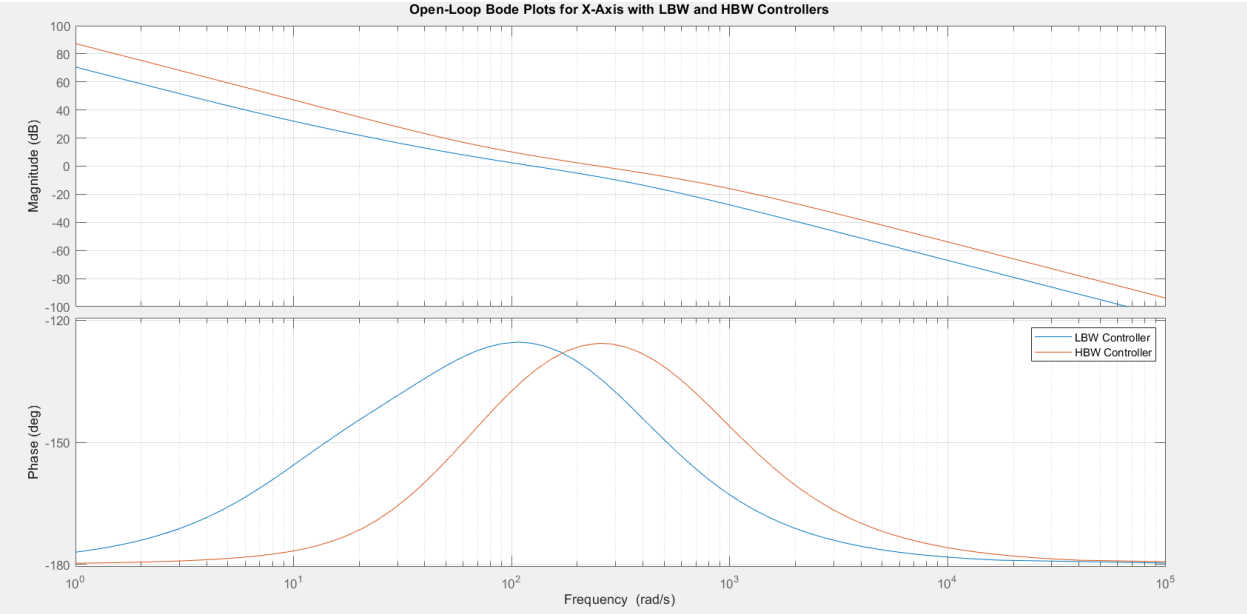


Figure 6: Open Loop Bode Plots for X-Axis Controllers

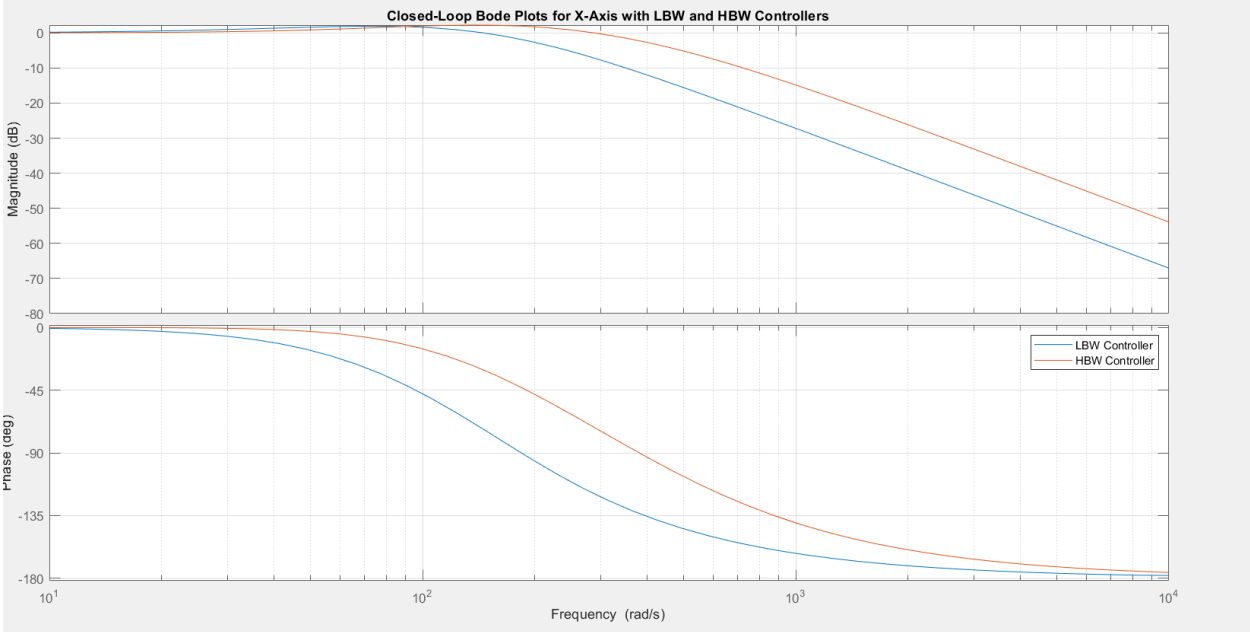


Figure 7: Closed Loop Bode Plots for X-Axis Controllers

B3) Determine the poles and zeros of the closed loop systems (both continuous (s) and discrete (z) domain), bandwidth, overshoot, and rise time for both cases of LBW and HBW controllers. Tabulate your result.

Using MATLAB, we determined important statistics and parameters of our closed-loop systems in both the discrete and continuous domains. These values can be seen in the following tables.

Table 9: Closed Loop Controller Bandwidth, Overshoot, & Rise Time

Controller Transfer Function	**Bandwidth**	**Overshoot (%)**	**Rise Time (s)**
X_LowBW_Cont	206.1122	22.1208	0.0091
Y_LowBW_Cont	206.3769	20.9087	0.0093
X_HighBW_Cont	411.3585	24.1330	0.0045
Y_HighBW_Cont	411.7196	23.2370	0.0045
X_LowBW_Dis	206.1122	22.1208	0.0091
X_LowBW_Dis	206.3769	20.9087	0.0093
X_LowBW_Dis	411.3585	24.1330	0.0045
X_LowBW_Dis	411.7196	23.2370	0.0045

Table 10: Closed Loop Controller Bandwidth, Overshoot, & Rise Time

Continuous Systems	**Poles**	**Zeros**
X_LowBW_Cont	0.9860 + 0.0078i	-1.0000

	0.9860 - 0.0078i	-1.0000
	0.9987 + 0.0000i	0.9987
	0.9925 + 0.0000i	0.9955
Y_LowBW_Cont	0.9870 + 0.0100i	-1.0000
	0.9870 - 0.0100i	-1.0000
	0.9987 + 0.0000i	0.9987
	0.9929 + 0.0000i	0.9950
X_HighBW_Cont	0.9975 + 0.0000i	-1.0000
	0.9801 + 0.0018i	-1.0000
	0.9801 - 0.0018i	0.9975
	0.9613 + 0.0000i	0.9921
Y_HighBW_Cont	0.9975 + 0.0000i	-1.0000
	0.9839 + 0.0000i	-1.0000
	0.9706 + 0.0090i	0.9975
	0.9706 - 0.0090i	0.9917
X_LowBW_Dis	0.9860 + 0.0078i	-1.0000
	0.9860 - 0.0078i	-1.0000
	0.9987 + 0.0000i	0.9987
	0.9925 + 0.0000i	0.9955
Y_LowBW_Dis	0.9870 + 0.0100i	-1.0000
	0.9870 - 0.0100i	-1.0000
	0.9987 + 0.0000i	0.9987
	0.9929 + 0.0000i	0.9950
X_HighBW_Dis	0.9975 + 0.0000i	-1.0000
	0.9801 + 0.0018i	-1.0000
	0.9801 - 0.0018i	0.9975
	0.9613 + 0.0000i	0.9921
Y_HighBW_Dis	0.9975 + 0.0000i	-1.0000
	0.9839 + 0.0000i	-1.0000
	0.9706 + 0.0090i	0.9975
	0.9706 - 0.0090i	0.9917

Prelab Part C – Contouring Performance Simulation

In this section, we will combine our previous systems to create a discrete position controller that can be used to control the 2-Axis Gantry System. Our input will be our applied system trajectories, and our output will be our system responses.

C1) Implement the discrete position controller cascaded with the continuous plant using a zero-order hold block in MATLAB/Simulink environment. Apply the generated reference trajectories in A.1 to the two-axis position control system. Verify that your trajectories and controllers result in stable contouring of the toolpath. Include the simulated path.

Our implemented discrete position controller can be seen in the following figure. We used an MATLAB script to change the position controllers accordingly for High or Low crossover frequencies automatically.

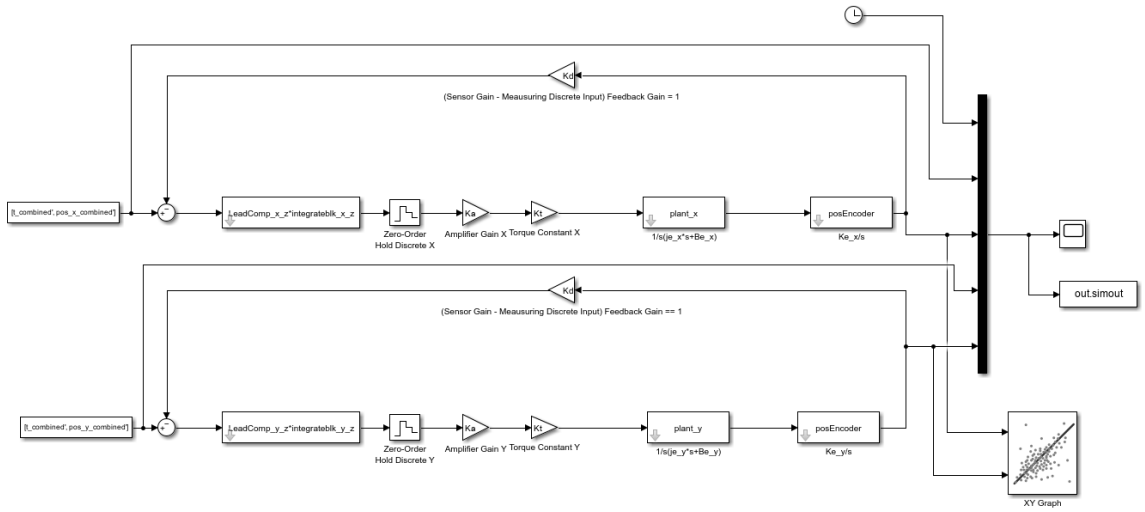


Figure 8: 2-Axis Discrete Position Simulink Controller

Below are our scope outputs and XY Graphs for the low crossover and high crossover frequency controllers.

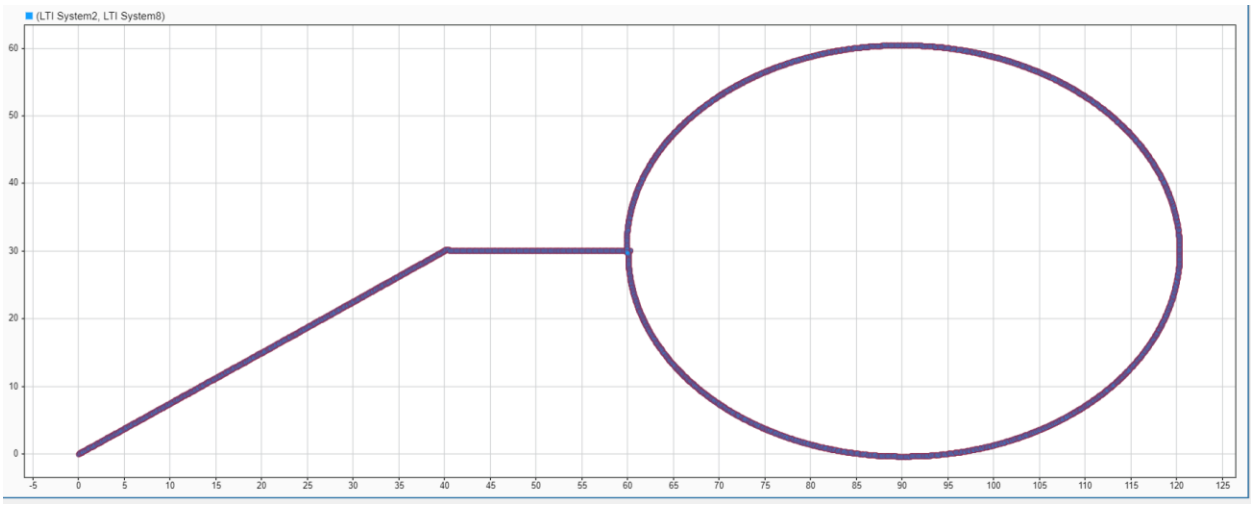


Figure 9: Low Crossover Frequency Simulink Controller XY Graph Output

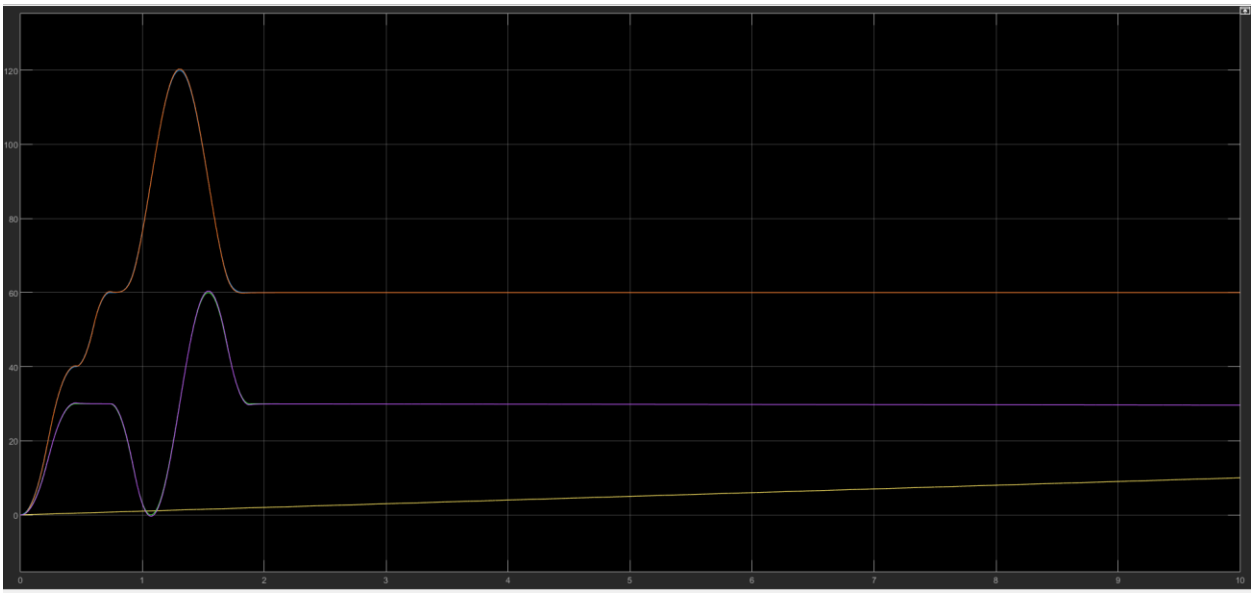


Figure 10: Low Crossover Frequency Simulink Controller XY Scope Output (Red=Y-Axis; Purple = X-Axis; Yellow = Clock)

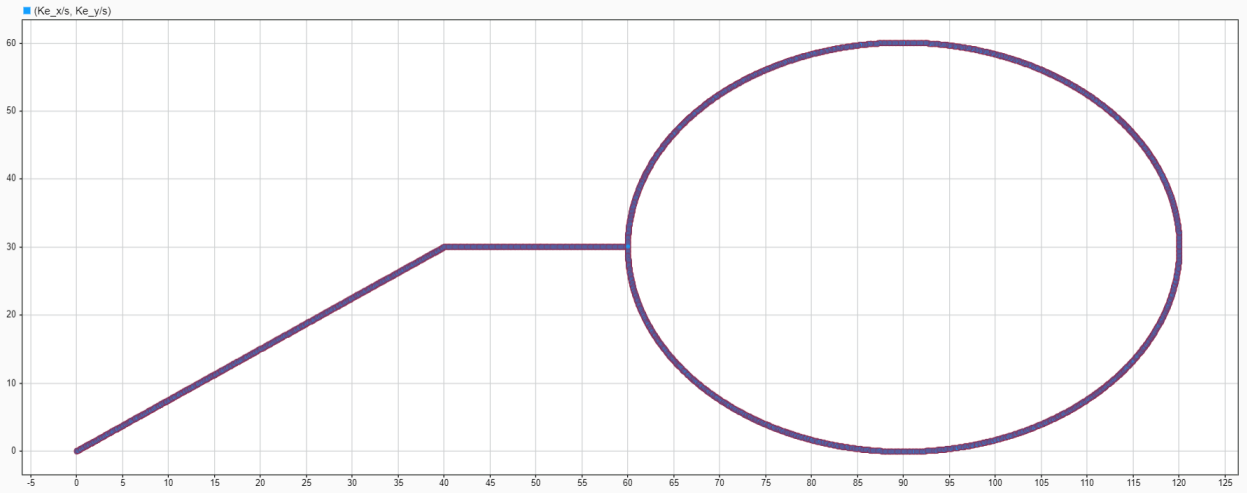


Figure 11: High Crossover Frequency Simulink Controller XY Graph Output

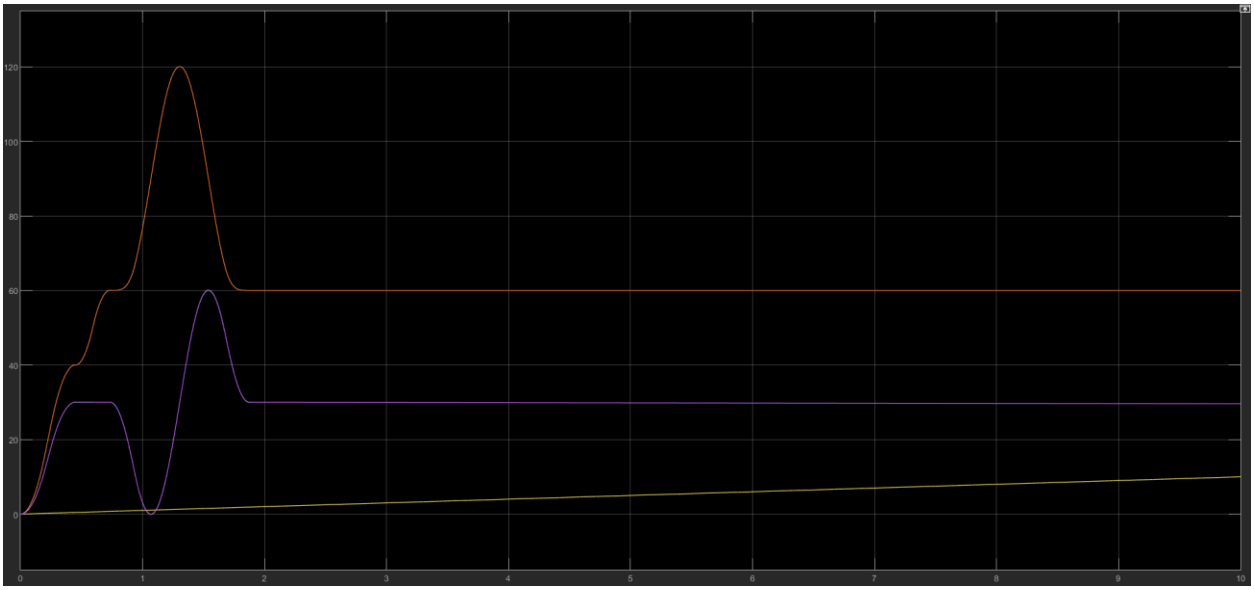


Figure 12: High Crossover Frequency Simulink Controller XY Scope Output (Red=Y-Axis; Purple = X-Axis; Yellow = Clock)

As you can see in the above figures, our plots and simulated output trajectories from our Simulink controller match our expectations.

C2) Implement your own XY Trajectory, using linear and circular interpolation techniques.

For this system, we have different system parameters compared to the earlier system interpolations.

Table 11: System Trajectory Parameters for Custom XY Trajectory

Parameter	Value
Feedrate (mm/s)	100
Tangential Acceleration (mm/s^2)	250
Tangential Deceleration (mm/s^2)	-250
Interval Time-Steps (ms)	0.1

Using the original X & Y Axis Controller parameters, I implemented the following geometry.

- P1 = [0, 0];
- P2 = [20, 50];
- P3 = [30, 0];
- P_Circle = [30, 30]; % Center of circular radius
- AngleTrav = pi/2;

The system will go from P1, to P2, to P3, then do a quarter circle angle from P3 based on the center of circle P_Circle. The simulated trajectory graphs from Q1 can be seen below.

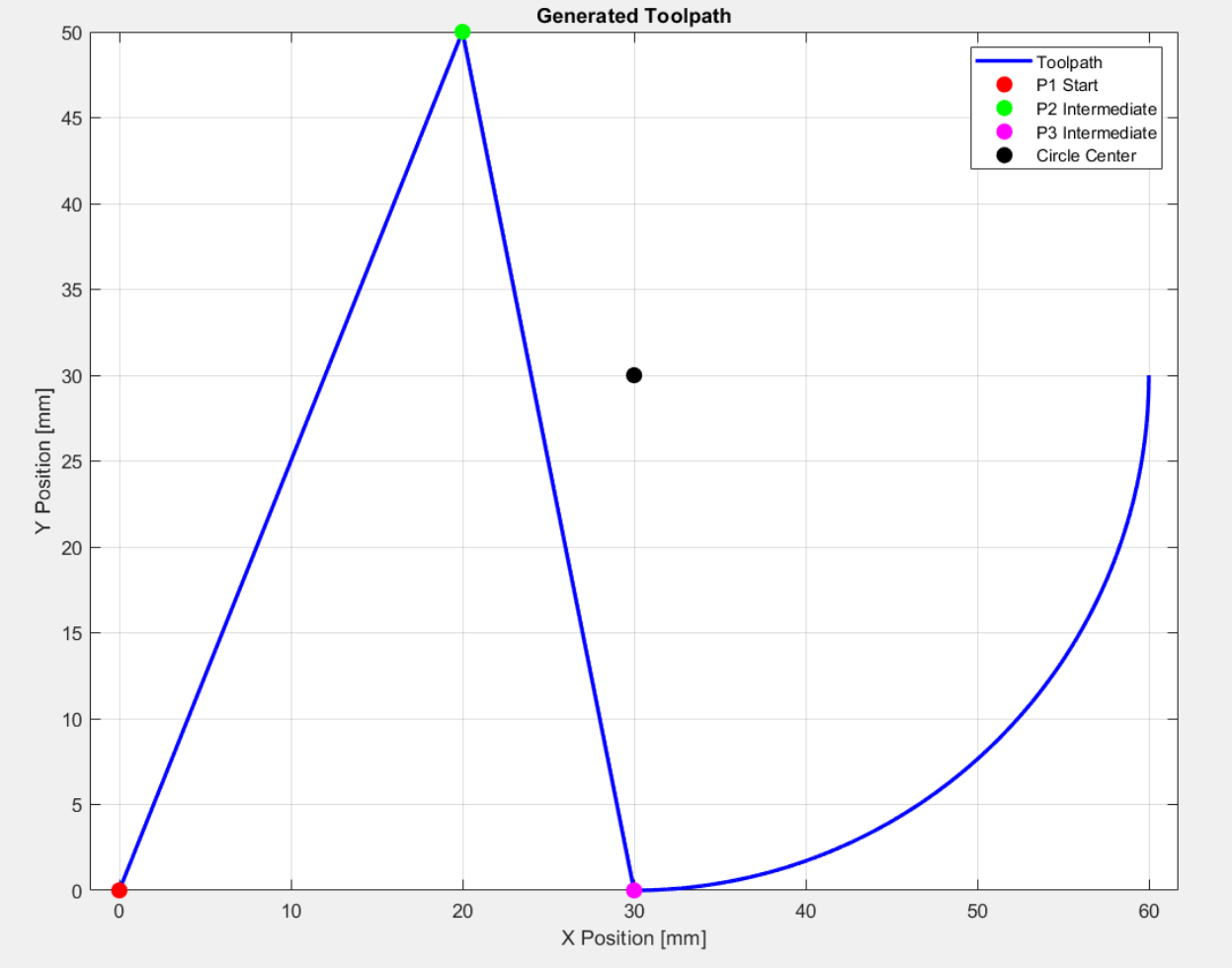


Figure 13: Generated Custom Toolpath: X-Axis Commands vs Y-Axis Commands

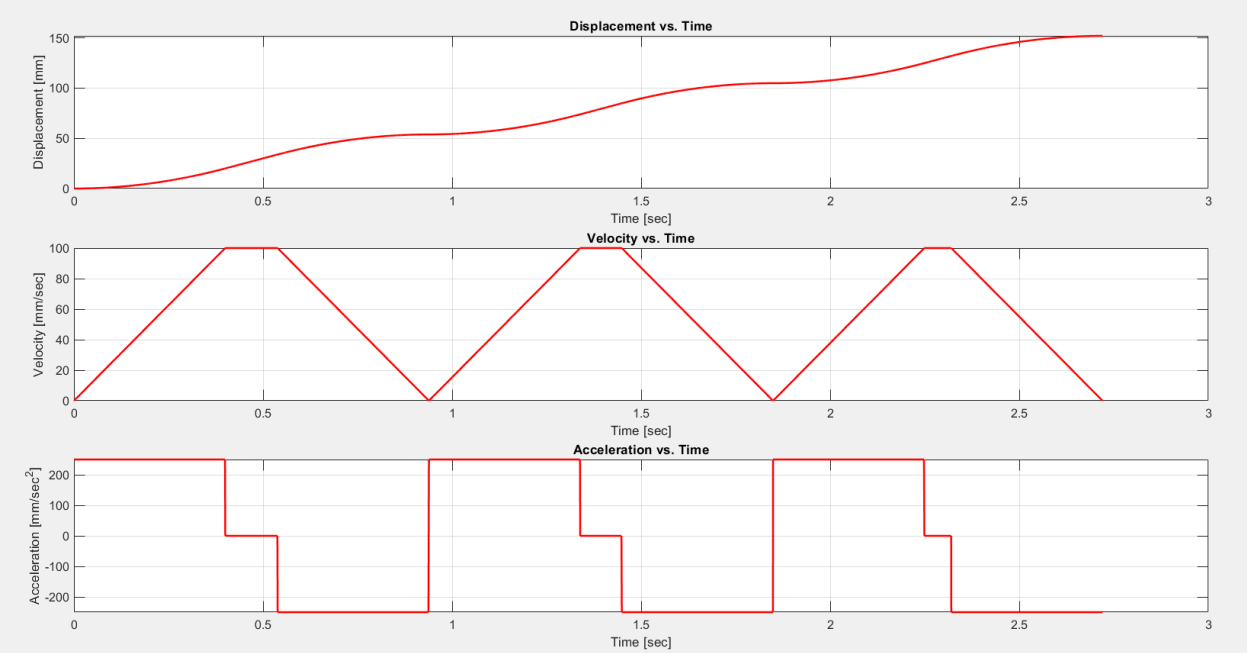


Figure 14: System Custom Motion Profiles Vs Time

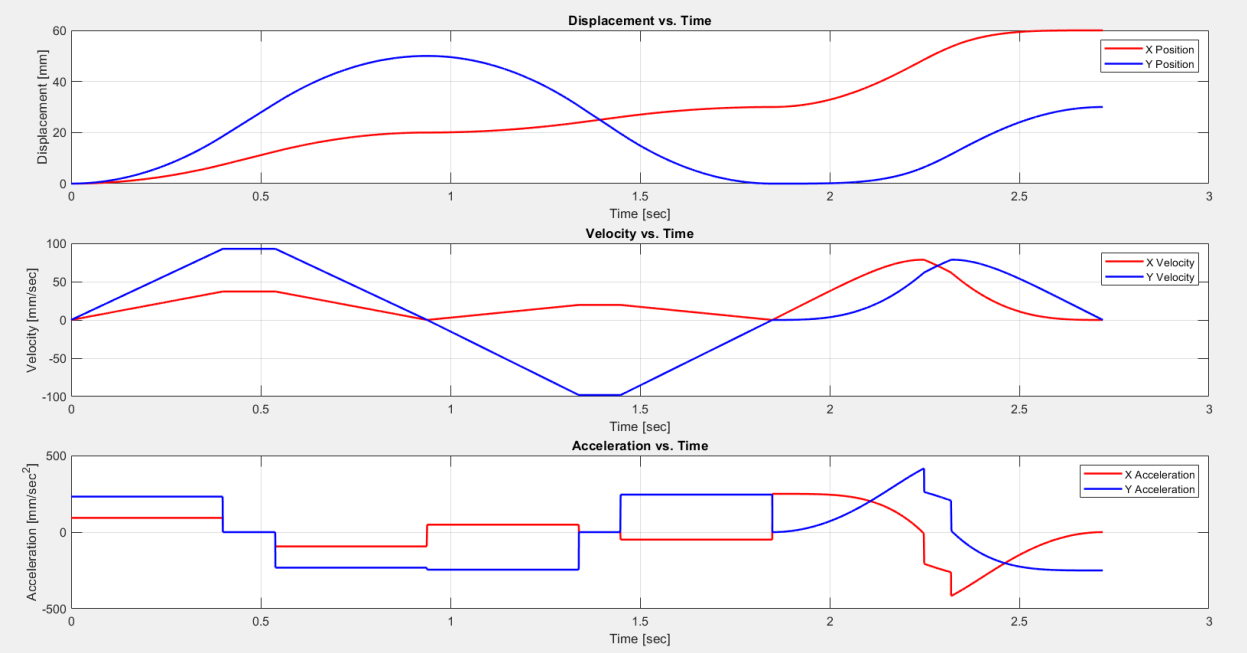


Figure 15: Custom Axis Position, Velocity, and Acceleration Vs Time

I inputted my custom trajectory values into my Simulink controller for the Low Crossover frequencies Lead Comp controllers, which can be seen in the following graph.

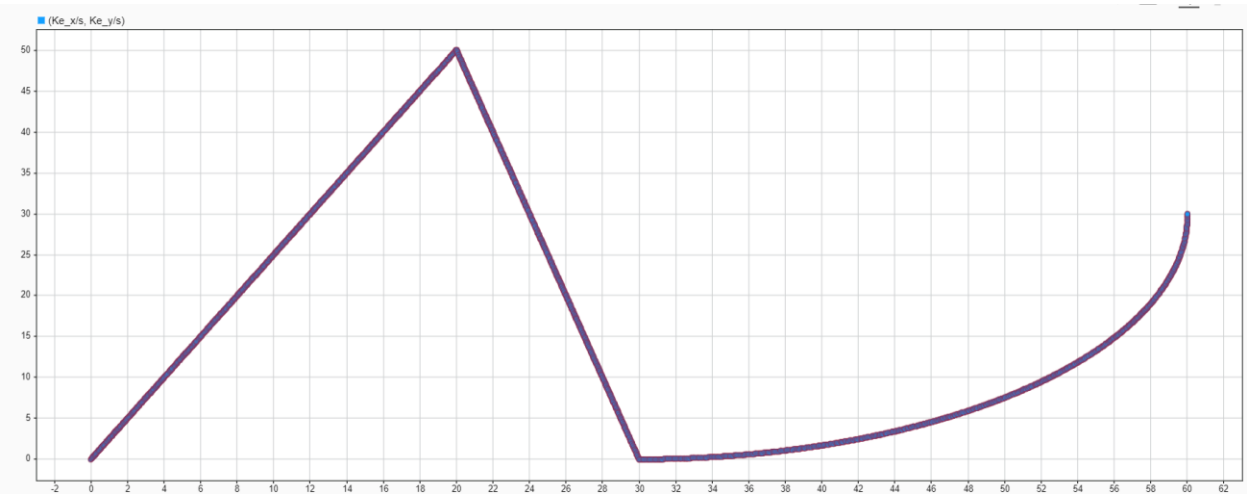


Figure 16: Low Crossover Frequency Simulink Controller XY Graph Output for Custom Trajectory

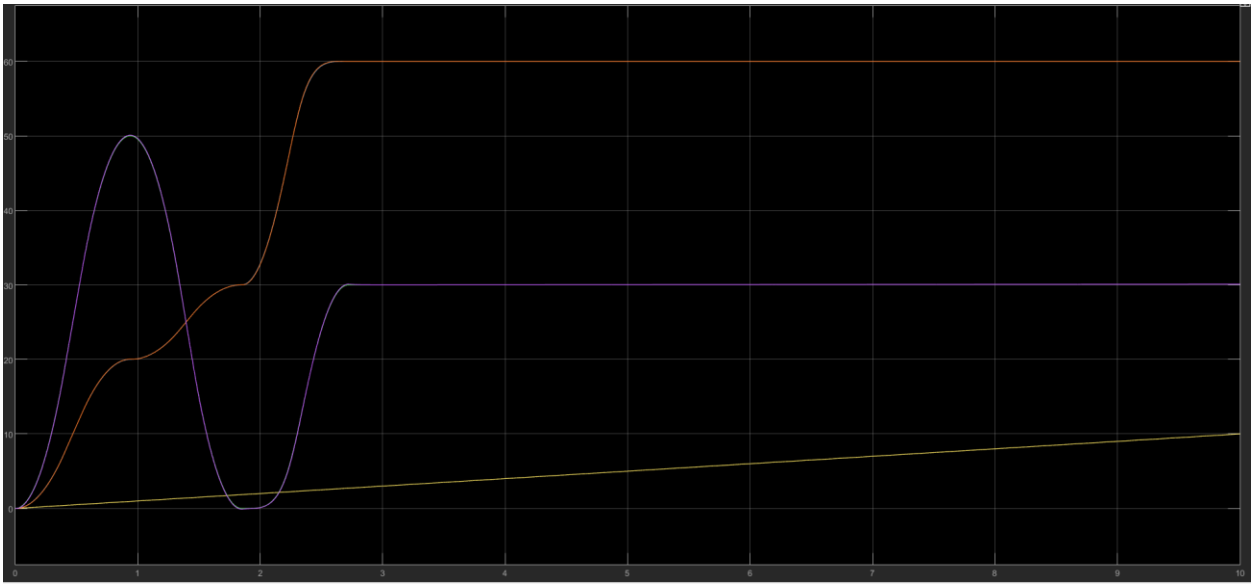


Figure 17: Low Crossover Frequency Simulink Controller XY Scope Output for Custom Trajectory (Red=Y-Axis; Purple = X-Axis; Yellow = Clock)

Lab Part E – Effect of Bandwidth

In this part of the lab analysis, we will analyze the line-circle trajectory and the effect on bandwidth due to the usage of the two different low and high bandwidth controllers.

E1) Plot tracking errors for the X & Y axis as a function of time for both LBW and HBW controllers.

The tracking errors for both axes can be seen below. We combined each Axis' low and high Bandwidth tracking together for figure plotting.

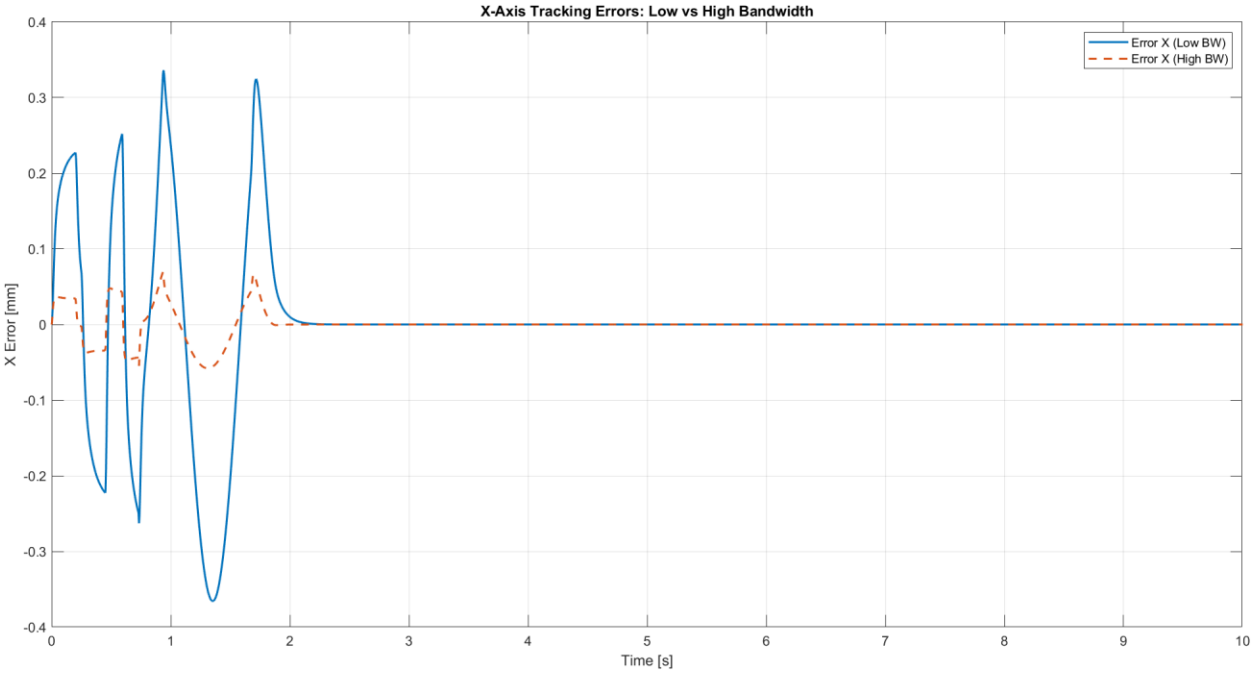


Figure 18: Axis Tracking Errors for X-Axis - Low & High Bandwidth

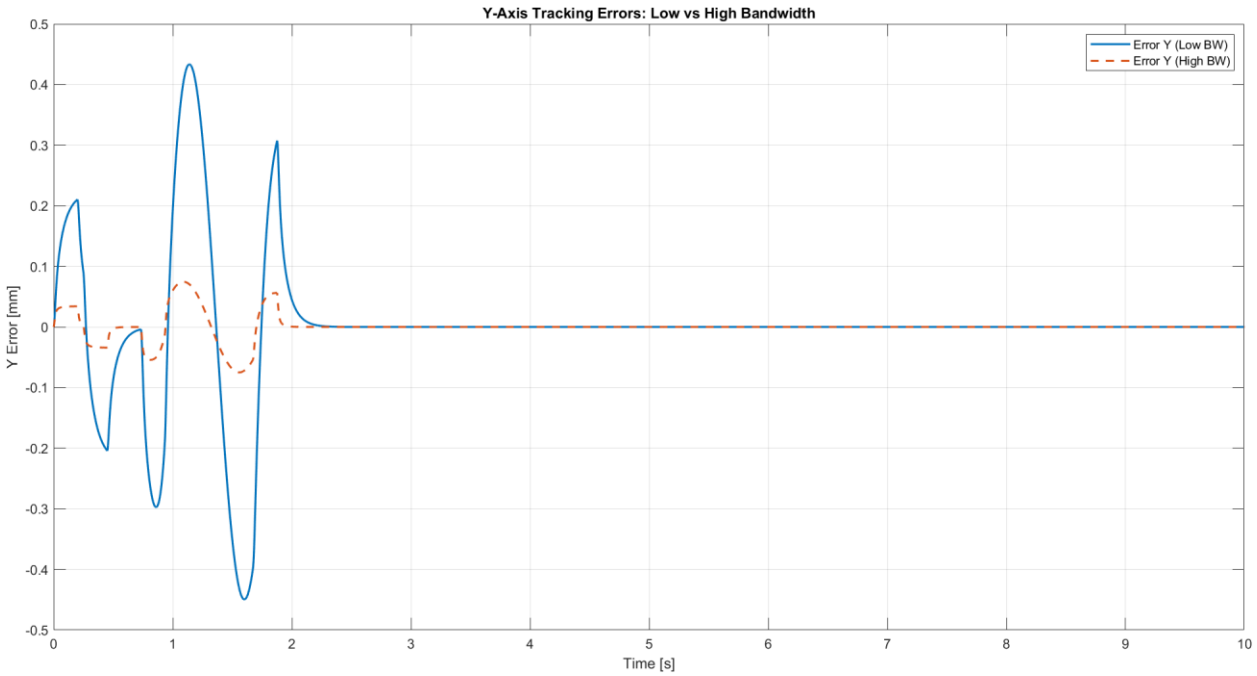


Figure 19: Axis Tracking Errors for Y-Axis - Low & High Bandwidth

As we can see in the above graphs, our low bandwidth system has a higher error rate compared to our low bandwidth error. This matches expectations, as a low bandwidth controller reacts slower than its high bandwidth counterpart. This will result with a more sluggish response and larger tracking errors, which can be seen in the above figures.

E2) Plot simulated Tool Motion for all three cases of the control system. Zoom Into 4 critical regions. Measure largest predicted contour error and comment on the effect of bandwidth on contouring error.

Our simulated toolpaths from our Simulink controller and our reference input toolpath from our trajectory planning program can be seen in the following figure. We plotted all three tool motion cases, which can be seen in the following graph.

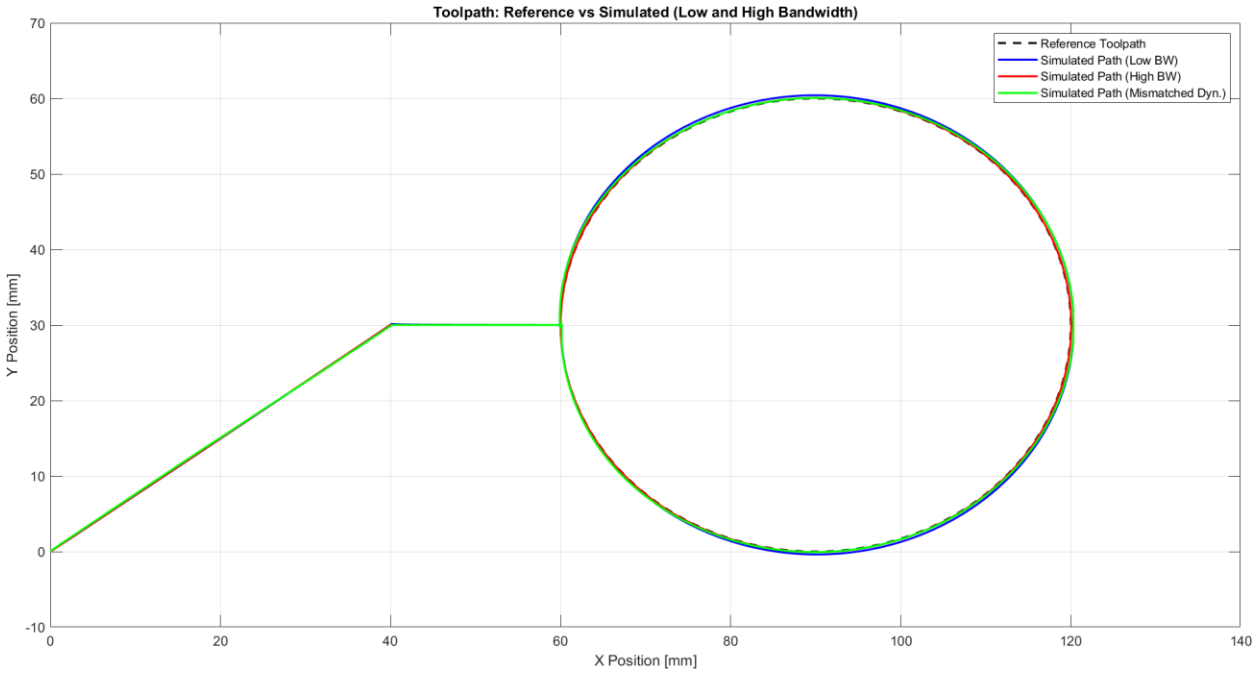


Figure 20: Reference Vs Simulated Toolpaths

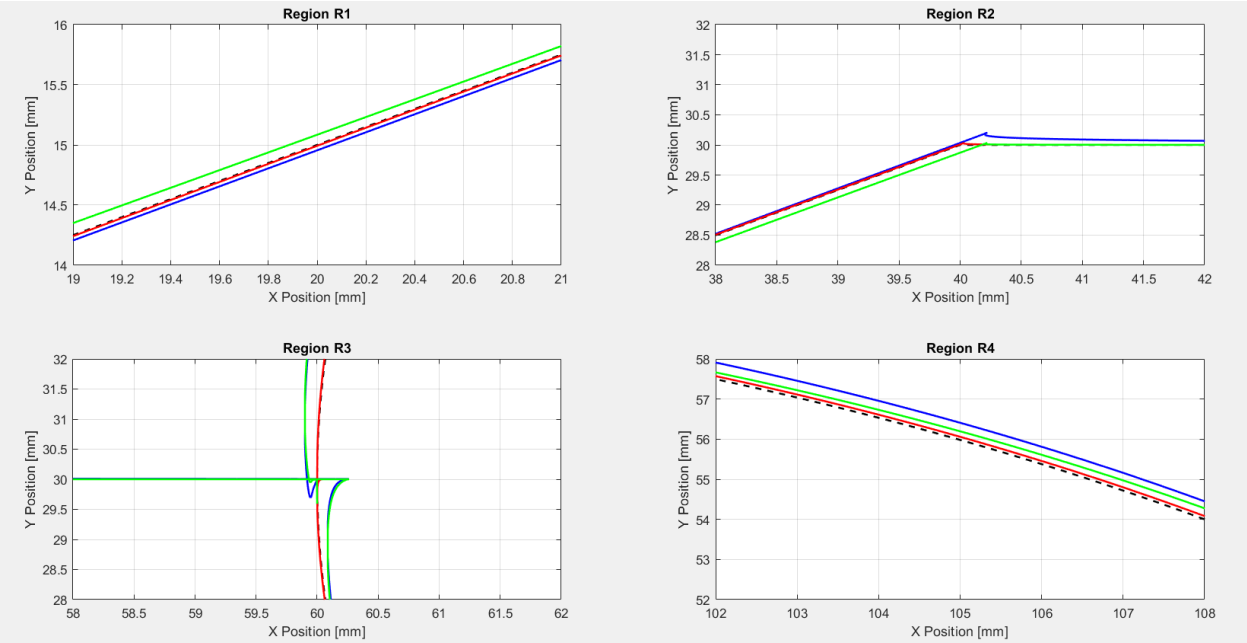


Figure 21: Reference Vs Simulated Toolpaths Zoomed on Points R1, R2, R3, R4 (Black = Reference; Red=Low BW; Blue = High BW; Green = Mismatched Dyn.)

As seen from the above figures, our mismatched dynamics system has the worse errors and incorrect trajectory pathing to all systems. In all 4 critical regions, our green path is either the farthest from our reference path (R1 & R4) or incorrectly overshot the expected path, leading to incorrect pathing (R2 & R3).

We can calculate our largest contouring error in MATLAB, which comes from our input and output trajectory values. They can be seen in the following table. Similar to our previous experiments, our low bandwidth controller has a higher error compared to our low bandwidth controller.

Table 12: Contouring Errors for Both Bandwidth Controllers

Bandwidth Controller	Contouring Error (mm)
Low Bandwidth	0.45
High Bandwidth	0.08
Mismatched Dyn	0.37

As seen in the above figures and tables, a lower bandwidth corresponds to a higher contouring error and a slower overall system, since the system responds slower to changes. In short, our systems match our expectations regarding contouring error and bandwidth.

E3) For midpoint of the first line (region R1) and Case 1 controllers, find the values of the tracking errors in the x and y axes from your simulation, and calculate the contouring error analytically using these tracking errors. Is it always possible to calculate the contouring error analytically?

Using MATLAB, we can determine the tracking error values at the midpoint of R1. Using these values, we can calculate the contouring error analytically afterwards. The tracking values can be found at the following table.

Table 13: Tracking Errors for Case 1 Controllers

Axis	Tracking Error (mm)
X-Axis	0.12
Y-Axis	0.14

We can calculate contouring error using these values by comparing our tracking error to our input values, as seen in the following calculations, as well as by using the current geometry of our trajectory during the R1 motion.

We can determine the slope of our trajectory line since R1 is linear in motion. We can assume the normal vector to this line as [-slope,1], allowing us to determine contouring error geometrically using the following equations.

Equation 9: Analytical Contouring Error Calculations

$$slope = \frac{y_2 - y_1}{x_2 - x_1} = \frac{30}{40} = \ 0.75$$

$$contouring\ error = \frac{\left|TE_x * -0.75 + TE_y * 1\right|}{\sqrt[2]{0.75^2 + 1^2}} = 0.04mm$$

Based on our system, we should be able to calculate the contouring error of any motion if we know the trajectory planning details and can calculate the normal vector to said motion. If the trajectory is complex, however, we won't be able to determine the normal vector and contouring error without significant calculations and analysis.

E4) Discussion: In Case 3, the cross-over frequencies of the X and Y axes are assigned differently. Hence, their bandwidths are different, and the two axes no longer have identical position loop dynamics. Considering the simulated toolpaths in part E.2, comment on the contouring performance in case of mismatched axis dynamics. Is it desirable to have mismatched dynamics? Why?

It is not desirable to have mismatched dynamics. This leads to different system responses and dynamics, as seen in the earlier systems and higher contouring errors. In short, mismatched dynamics will lead to a less precise and synchronous systems and a system that does not follow the given trajectory plans correctly. These unintended effects are specifically noticeable during curves and corners, where these mismatched dynamics will greatly affect our system's response and output.

Lab Part F: Effect of Maximum Feedrate

F1) Regenerate the reference trajectory using a desired feedrate of[mm/sec]250=F . Plot the new displacement, feedrate, tangential acceleration profiles, as well as axis position, velocity, and acceleration commands, all as functions of time.

New reference trajectories using a desired federate of 250mm/s, alongside motion profiles and X-Y graphs can be seen below.

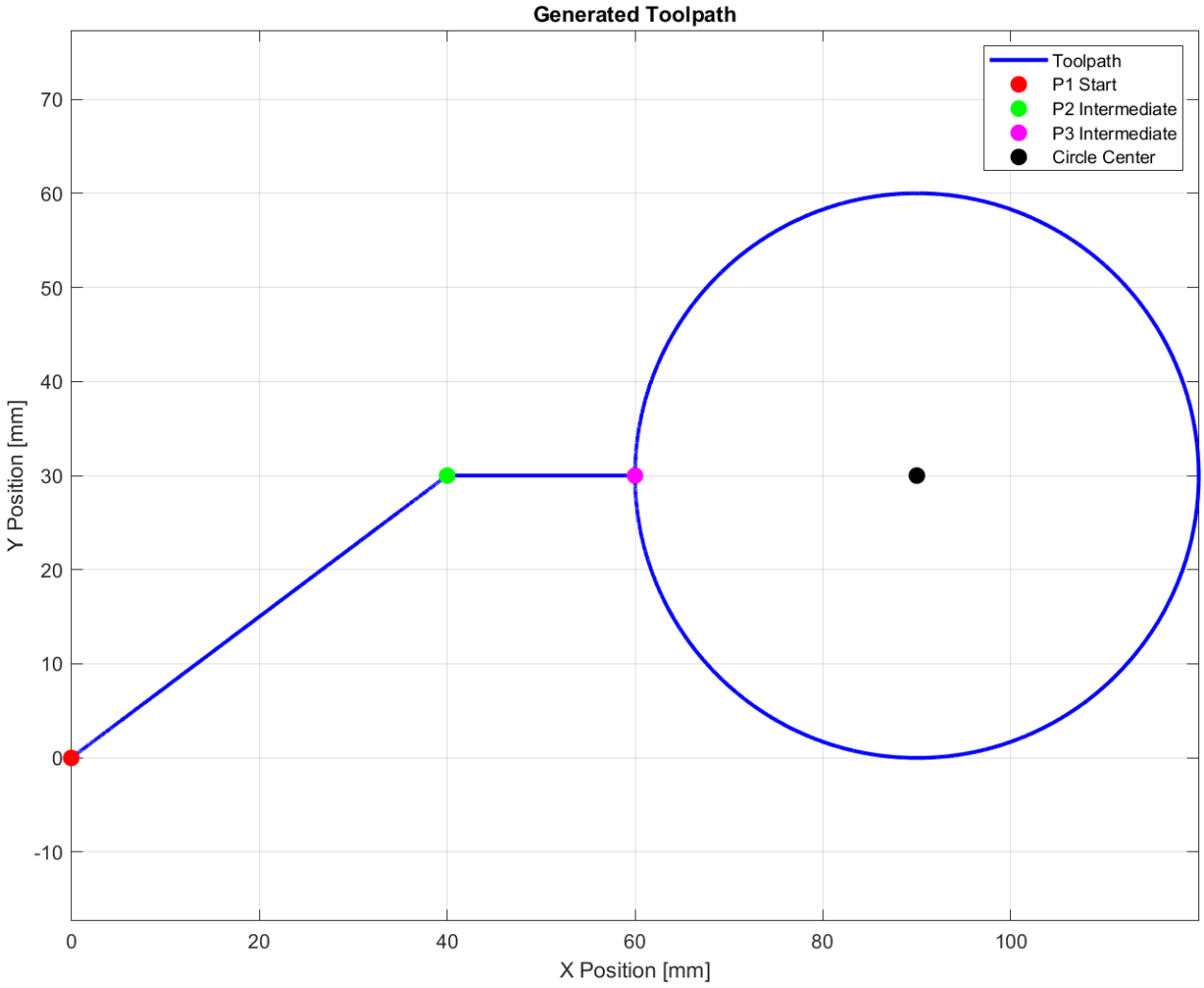


Figure 22: Generated Toolpath with F=250mm/s: X-Axis Commands vs Y-Axis Commands

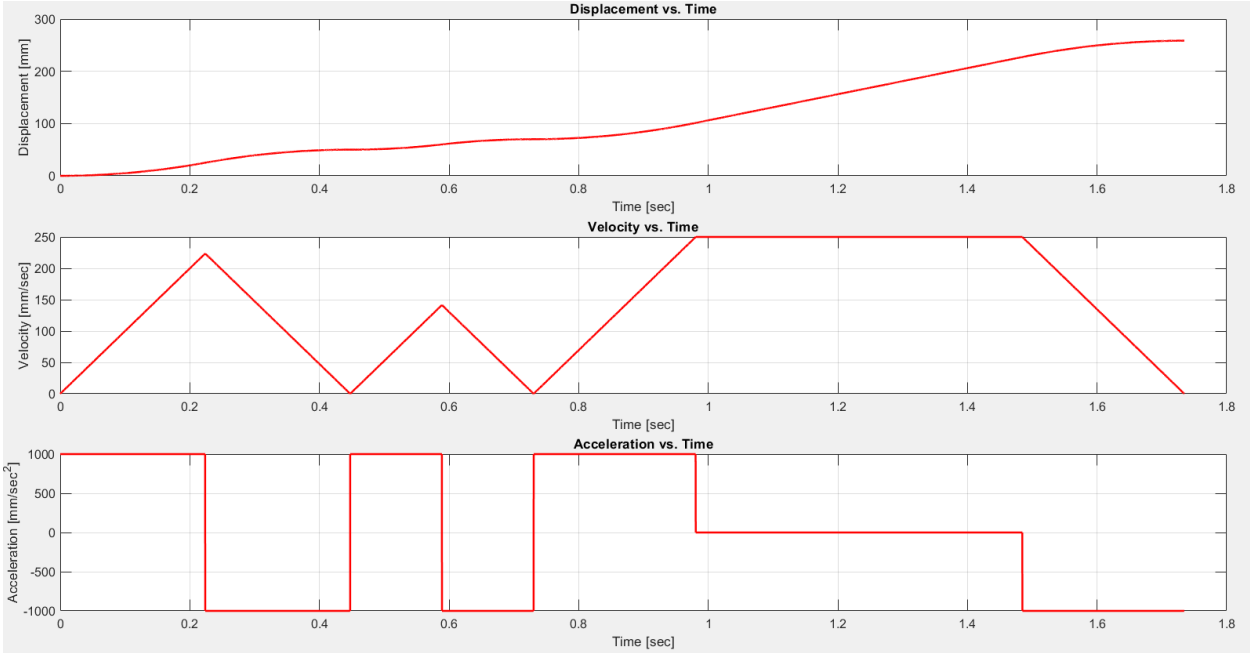


Figure 23: System Motion Profiles Vs Time with F=250mm/s

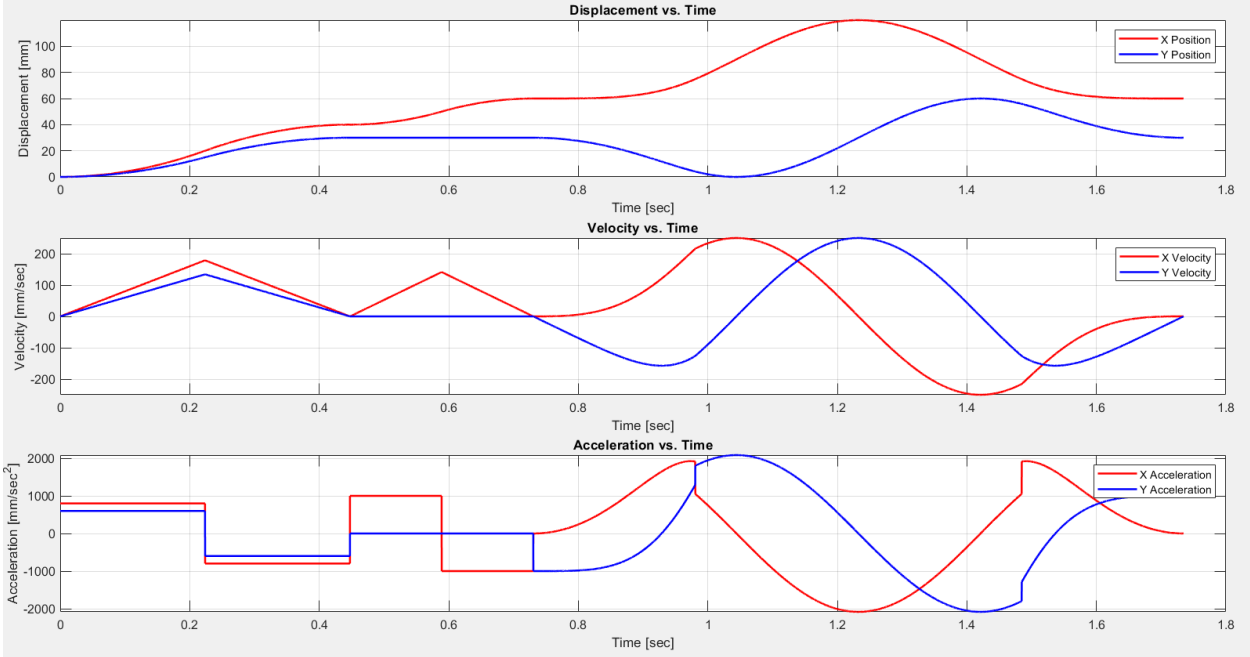


Figure 24: Axis Position, Velocity, and Acceleration Vs Time with F=250mm/s

F2) Simulate the two-axis response using the high feedrate trajectories and LBW controllers (Case 1). Plot the tracking errors vs. time (in both x and y axes) on top of the tracking errors in the case of low feedrate trajectories (part E.1). Comment on the effect of maximum feedrate on tracking errors.

Below are the tracking errors of two different feedrates in both the X and Y Axis.

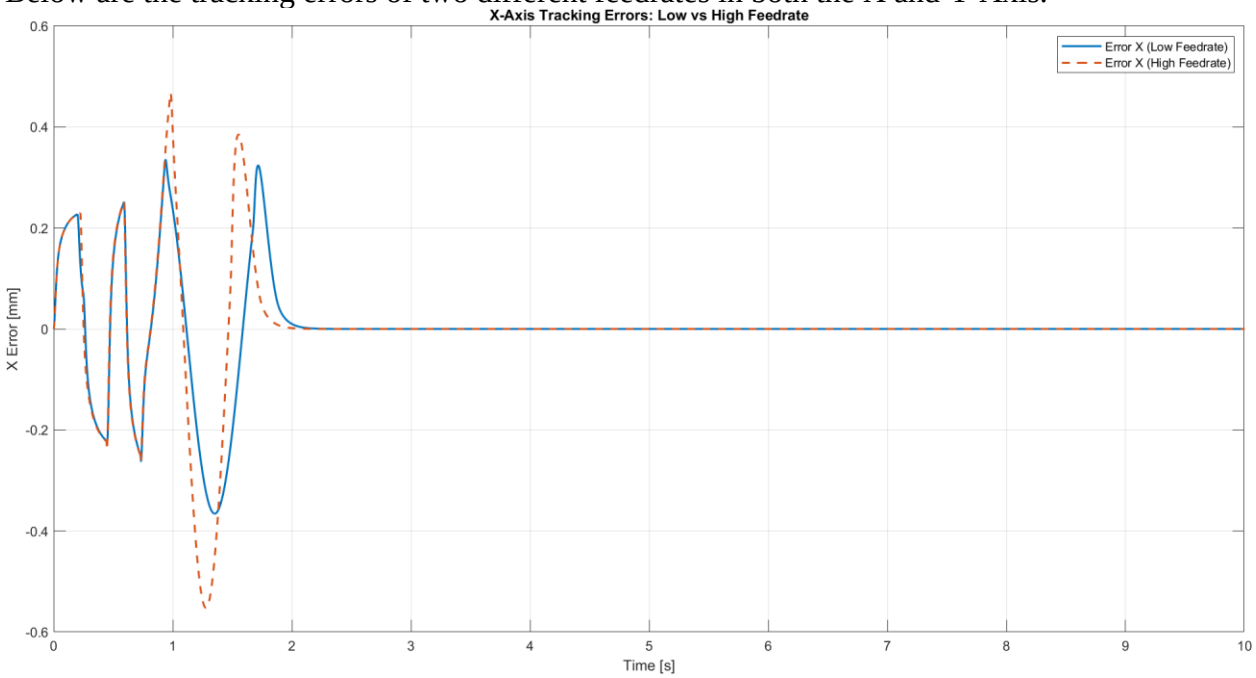


Figure 25: Axis Tracking Errors for X-Axis - Low & High Feedrates

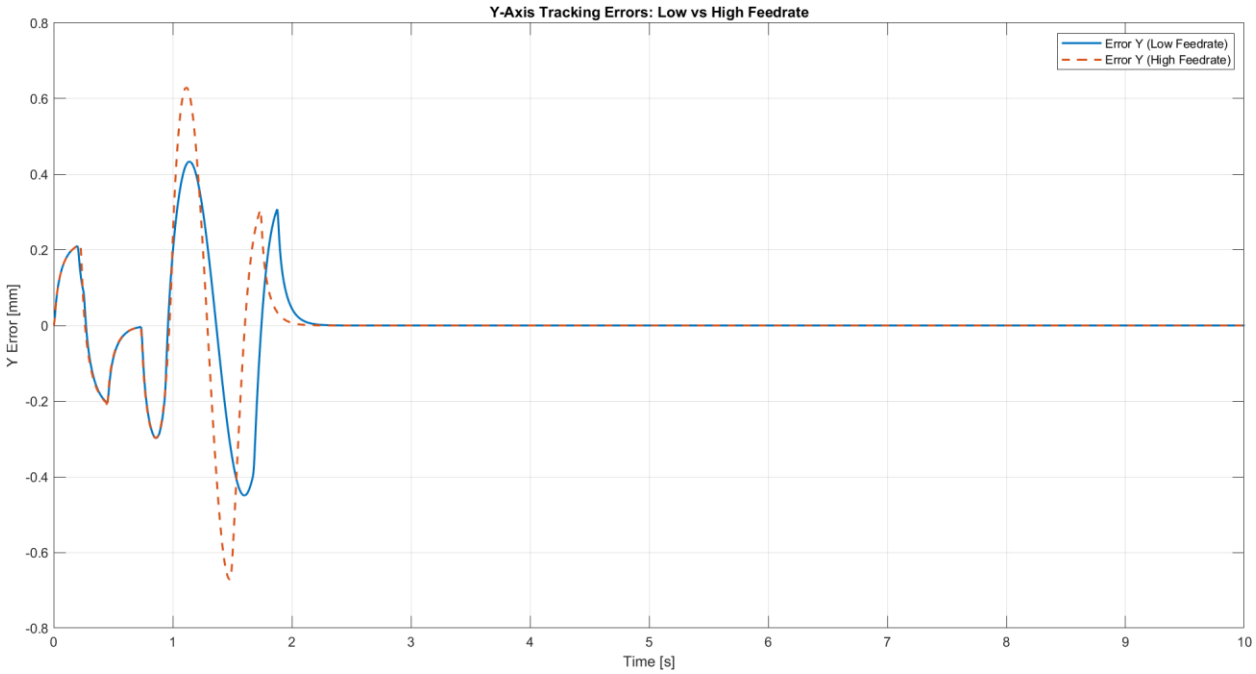


Figure 26: Axis Tracking Errors for X-Axis - Low & High Feedrates

For higher feedrates, the system is moving faster and must respond faster to trajectory changes. This can lead to higher tracking errors as the system tries to keep up with the faster feedrate, which can be seen in the above figures. As with contouring errors in the previous question, these errors will also increase dramatically for curved trajectories and sharp corners.

F3) Overlay the simulated toolpath for the low and high feedrates, zoom into the regions R1 to R4, and comment on the effect of maximum feedrate on contouring accuracy.

Simulated toolpaths for the low and high feedrates can be seen below.

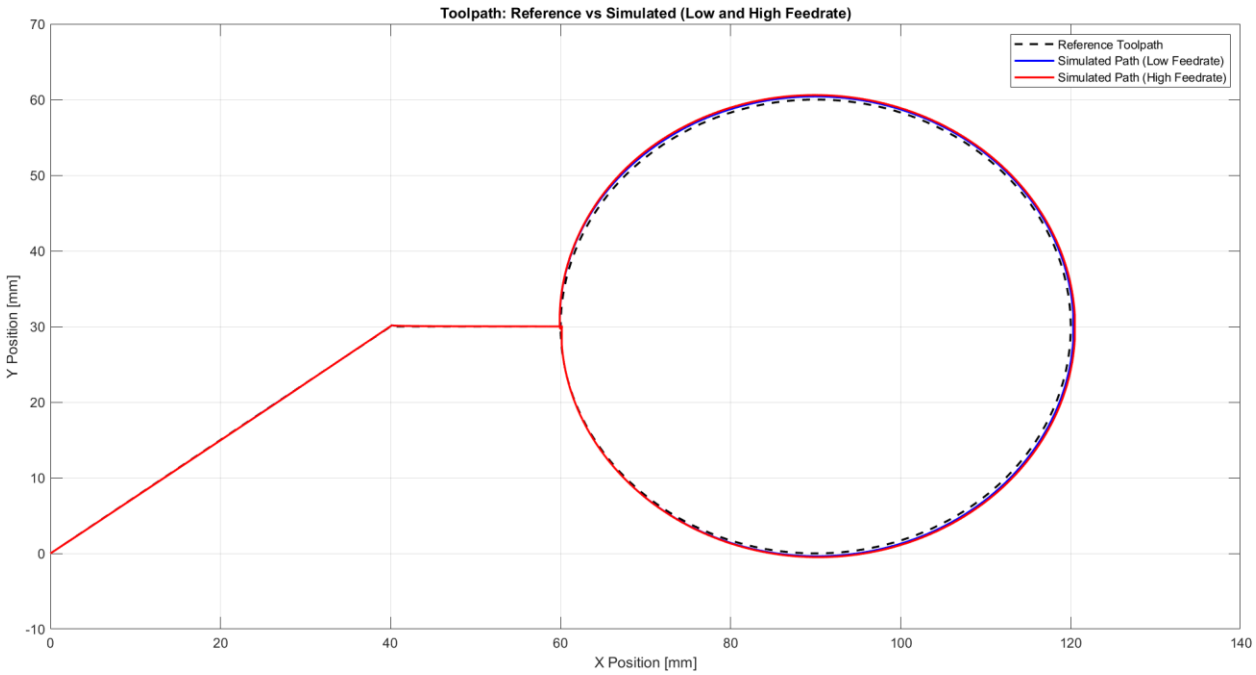


Figure 27: Reference Vs Simulated Toolpaths for Low & High Feedrates

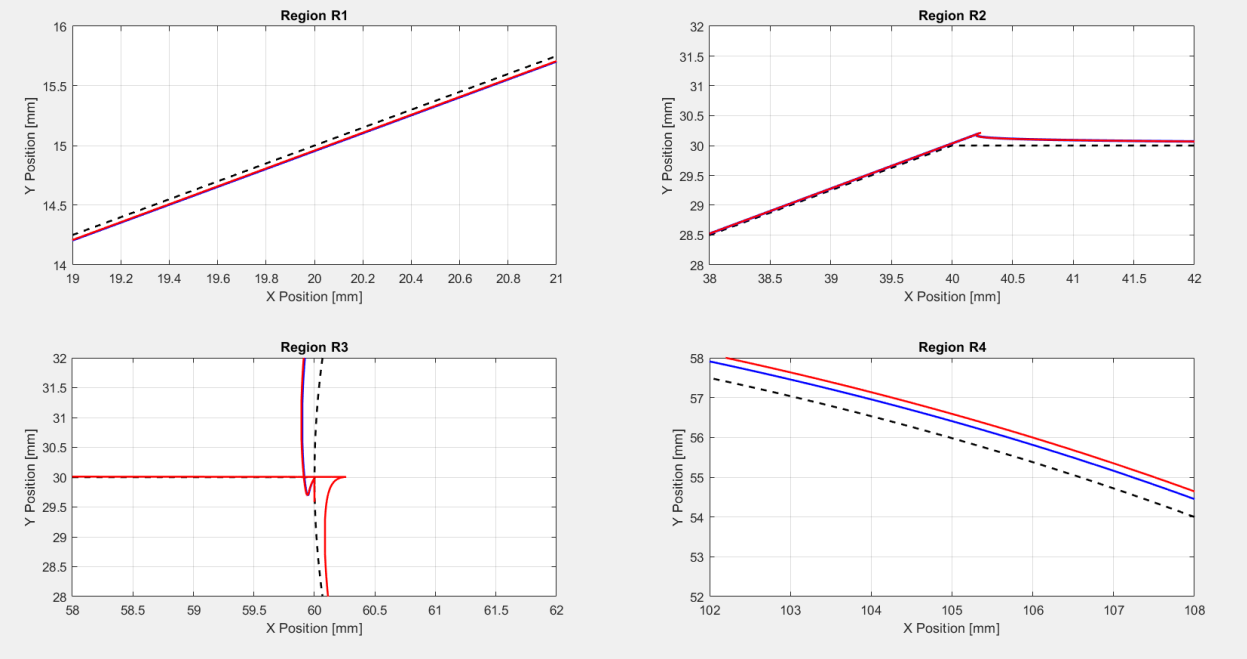


Figure 28: Reference Vs Simulated Toolpaths for Multiple Feedrates; Zoomed on Points R1, R2, R3, R4 (Black = Reference; Red=High Feedrate; Blue = Low Feedrate)

Similar to tracking errors, our contouring errors will increase as our maximum feedrate increases. Since the system moves faster at higher feedrates, it will need to respond faster to changes or system lag and inaccuracies will occur. For linear portions, such as R1, the system's contouring errors are minimal. However, as we complete sharp corners or with curved trajectories, our contouring errors increase as the system lags behind and has a worse response, which can be seen in R2 & R3.

Lab Part G – Experiment Vs Simulation

G1) For the first given trajectory & for measured axis responses, plot the experimental tracking errors for the LBW controllers. Overlay simulated & experimental toolpaths as well & zoom in on areas of interest R1 to R4. Comment on reasons for discrepancies.

G2) For the custom trajectory & its measured axis responses, Overlay simulated & experimental toolpaths as well & zoom in on areas of interest. Provide Experimental Apparatus pictures as well.

Conclusion

This project demonstrated the complete process of designing, implementing, and analyzing a two-axis machine controller for precise trajectory planning applications. By generating trajectories and implementing lead-lag controllers, the system achieved stable and accurate motion control. The study highlighted the impact of controller bandwidth and feedrate on tracking and contouring accuracy, emphasizing the importance of balanced axis dynamics. Experimental results validated the simulation findings as well. Overall, this project reinforced the critical role of trajectory design and controller optimization in achieving high-precision motion in multi-axis systems.

Appendix

Appendix 1: Prelab Q1 Matlab Code

```
% Main Script for Linear Trajectory Planning and Plotting

% Clear workspace and figures
clear;
close all;
clc;

%% Constants - Original System Parameters
% Ti = 0.0001;          % Interpolation period [sec]
% F = 200;              % Desired feedrate [mm/sec]
% A = 1000;             % Acceleration [mm/sec^2]
% D = 1000;             % Deceleration [mm/sec^2]
% P1 = [0, 0];          % Point 1
% P2 = [40, 30];        % Point 2
% P3 = [60, 30];        % Point 3
% P_Circle = [90, 30];  % Center of circular radius
% Direction = 1;        % Counter-clockwise direction
% AngleTrav = 2*pi;

%% Constants - Original System Parameters
Ti = 0.0001;          % Interpolation period [sec]
F = 100;              % Desired feedrate [mm/sec]
A = 250;             % Acceleration [mm/sec^2]
D = 250;             % Deceleration [mm/sec^2]
P1 = [0, 0];          % Point 1
P2 = [20, 50];        % Point 2
P3 = [30, 0];        % Point 3
P_Circle = [30, 30];  % Center of circular radius
Direction = 1;        % counter-clockwise direction
AngleTrav = pi/2;

%% Trajectory Planning for Segment P1 to P2
% Call the linearTraj function for P1 to P2
[t1, pos_x1, pos_y1, vel_x1, vel_y1, accel_x1, accel_y1, s1, s_dot1, s_ddot1] = ...
    linearTraj(P1, P2, F, A, D, Ti);

%% Trajectory Planning for Segment P2 to P3
% Call the linearTraj function for P2 to P3
[t2, pos_x2, pos_y2, vel_x2, vel_y2, accel_x2, accel_y2, s2, s_dot2, s_ddot2] = ...
    linearTraj(P2, P3, F, A, D, Ti);

%% Trajectory Planning for Circular Motion (P3 to P3 via circle)
[t3, pos_x3, pos_y3, vel_x3, vel_y3, accel_x3, accel_y3, s3, s_dot3, s_ddot3] = ...
    circularTraj(P3, P_Circle, F, A, D, Ti, Direction, AngleTrav);


%% Combine Trajectory Data for Continuous Plotting
% Adjust time vectors to ensure continuity
t2 = t2 + t1(end) + Ti;  % Offset the second segment's time
```

```
t3 = t3 + t2(end) + Ti;            % Offset the circular segment's time
% Offset motion profiles for continuity
s2 = s2 + s1(end);       % Offset the second segment's trajectory
s3 = s3 + s2(end);       % Offset the third segment's trajectory


%% Combine Trajectory Data for Continuous Plotting
t_combined = [t1, t2, t3];
pos_x_combined = [pos_x1, pos_x2, pos_x3];
pos_y_combined = [pos_y1, pos_y2, pos_y3];
vel_x_combined = [vel_x1, vel_x2, vel_x3];
vel_y_combined = [vel_y1, vel_y2, vel_y3];
accel_x_combined = [accel_x1, accel_x2, accel_x3];
accel_y_combined = [accel_y1, accel_y2, accel_y3];
s_combined = [s1, s2, s3];
s_dot_combined = [s_dot1, s_dot2, s_dot3];
s_ddot_combined = [s_ddot1, s_ddot2, s_ddot3];

%% Plotting the Toolpath
figure('Name', 'Toolpath');
plot(pos_x_combined, pos_y_combined, 'b-', 'LineWidth', 2);
hold on;
plot(P1(1), P1(2), 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r'); % Start Point
plot(P2(1), P2(2), 'go', 'MarkerSize', 8, 'MarkerFaceColor', 'g'); % Intermediate
Point
plot(P3(1), P3(2), 'mo', 'MarkerSize', 8, 'MarkerFaceColor', 'm'); % End Point
plot(P_Circle(1), P_Circle(2), 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k'); %
Circle Center
title('Generated Toolpath');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
legend('Toolpath', 'P1 Start', 'P2 Intermediate', 'P3 Intermediate', 'Circle
Center');
grid on;
axis equal;
hold off;

%% Plotting X & Y vs Time
% Plotting Displacement vs. Time
figure('Name', 'X & Y vs. Time');

subplot(3,1,1);
plot(t_combined, pos_x_combined, 'r-', 'LineWidth', 1.5);
hold on;
plot(t_combined, pos_y_combined, 'b-', 'LineWidth', 1.5);
title('Displacement vs. Time');
xlabel('Time [sec]');
ylabel('Displacement [mm]');
legend('X Position', 'Y Position');
grid on;
hold off;

% Plotting Velocity vs. Time
subplot(3,1,2);
plot(t_combined, vel_x_combined, 'r-', 'LineWidth', 1.5);
```

```
hold on;
plot(t_combined, vel_y_combined, 'b-', 'LineWidth', 1.5);
title('Velocity vs. Time');
xlabel('Time [sec]');
ylabel('Velocity [mm/sec]');
legend('X Velocity', 'Y Velocity');
grid on;
hold off;

% Plotting Acceleration vs. Time
subplot(3,1,3);
plot(t_combined, accel_x_combined, 'r-', 'LineWidth', 1.5);
hold on;
plot(t_combined, accel_y_combined, 'b-', 'LineWidth', 1.5);
title('Acceleration vs. Time');
xlabel('Time [sec]');
ylabel('Acceleration [mm/sec^2]');
legend('X Acceleration', 'Y Acceleration');
grid on;
hold off;

%% Plotting Profiles vs Time
% Plotting Displacement vs. Time
figure('Name', 'Profiles vs. Time');

subplot(3,1,1);
plot(t_combined, s_combined, 'r-', 'LineWidth', 1.5);
title('Displacement vs. Time');
xlabel('Time [sec]');
ylabel('Displacement [mm]');
grid on;
hold off;

% Plotting Velocity vs. Time
subplot(3,1,2);
plot(t_combined, s_dot_combined, 'r-', 'LineWidth', 1.5);
title('Velocity vs. Time');
xlabel('Time [sec]');
ylabel('Velocity [mm/sec]');
grid on;
hold off;

% Plotting Acceleration vs. Time
subplot(3,1,3);
plot(t_combined, s_ddot_combined, 'r-', 'LineWidth', 1.5);
title('Acceleration vs. Time');
xlabel('Time [sec]');
ylabel('Acceleration [mm/sec^2]');
grid on;
hold off;
```

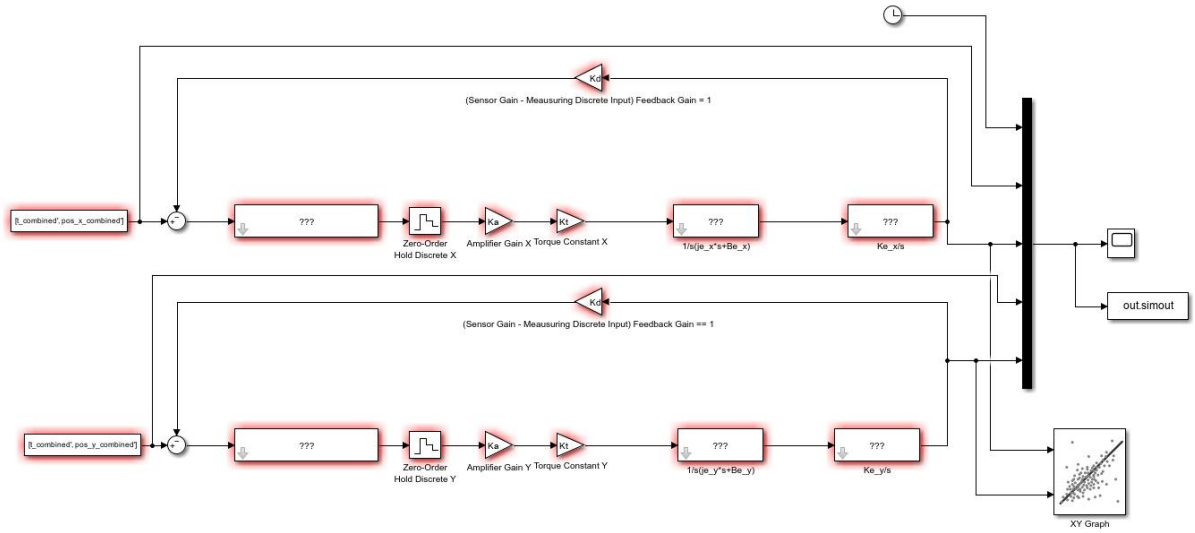
Appendix 2: Prelab Q2 Matlab Code

```
% Define system parameters
Ka = 1;  %[A/V]
Kt = 0.49;  %[Nm/A]
Ke = 1.59;  %[mm/rad]
Je_x = 4.36e-4;  %[kgm^2]
Je_y = 3e-4;  %[kgm^2]
Be_x = 0.0094;  %[Nm/rad/s]
Be_y = 0.0091;  %[Nm/rad/s]
Ts = 0.0001; %[s]
high_w_crossover = 40*2*pi; %[rad/s]
low_w_crossover = 20*2*pi; %[rad/s]
Ki_high = high_w_crossover / 10;
Ki_low = low_w_crossover / 10;

%% Determine Kp Controller For Both Axis and Both Crossover Frequencies
%X-axis
Kp_high_x = calcKp(high_w_crossover,Je_x,Be_x);
Kp_low_x = calcKp(low_w_crossover,Je_x,Be_x);
% Y-axis
Kp_high_y = calcKp(high_w_crossover,Je_y,Be_y);
Kp_low_y = calcKp(low_w_crossover,Je_y,Be_y);

%% Determine Current Phase for both axis and both Crossover Frequencies
%X-axis
currPhase_high_x = findPhaseAtCrossover(Kp_high_x, Je_x, Be_x);
currPhase_low_x = findPhaseAtCrossover(Kp_low_x, Je_x, Be_x);
% Y-axis
currPhase_high_y = findPhaseAtCrossover(Kp_high_y, Je_y, Be_y);
currPhase_low_y = findPhaseAtCrossover(Kp_low_y, Je_y, Be_y);

%% Determine Peak Phase for both axis and both frequencies
%X-Axis
peakPhase_high_x = calcPeakPhase(currPhase_high_x);
peakPhase_low_x = calcPeakPhase(currPhase_low_x);
%Y-AxiTs
peakPhase_high_y = calcPeakPhase(currPhase_high_y);
peakPhase_low_y = calcPeakPhase(currPhase_low_y);

%% Determine Lead Compensator parameters
% X-Axis
[alpha_high_x, tau_high_x] = calculateLeadCompParameters(peakPhase_high_x,
high_w_crossover);
[alpha_low_x, tau_low_x] = calculateLeadCompParameters(peakPhase_low_x,
low_w_crossover);
% Y-Axis
[alpha_high_y, tau_high_y] = calculateLeadCompParameters(peakPhase_high_y,
high_w_crossover);
[alpha_low_y, tau_low_y] = calculateLeadCompParameters(peakPhase_low_y,
low_w_crossover);

%% Determine Lead Compensator Kp
% X-Axis
```

```
Kp_high_x = calculateLeadCompKp(alpha_high_x, tau_high_x, Je_x, Be_x,
high_w_crossover);
Kp_low_x = calculateLeadCompKp(alpha_low_x, tau_low_x, Je_x, Be_x, low_w_crossover);
% Y-Axis
Kp_high_y = calculateLeadCompKp(alpha_high_y, tau_high_y, Je_y, Be_y,
high_w_crossover);
Kp_low_y = calculateLeadCompKp(alpha_low_y, tau_low_y, Je_y, Be_y, low_w_crossover);

%% Develop Open Loop Integrator-Lead-Lag Controller Transfer Function
% X-Axis
Gs_high_x = calcLeadIntegratorFunction(Kp_high_x, Ki_high, alpha_high_x, tau_high_x,
Je_x, Be_x);
Gs_low_x = calcLeadIntegratorFunction(Kp_low_x, Ki_low, alpha_low_x, tau_low_x, Je_x,
Be_x);
% Y-Axis
Gs_high_y = calcLeadIntegratorFunction(Kp_high_y, Ki_high, alpha_high_y, tau_high_y,
Je_y, Be_y);
Gs_low_y = calcLeadIntegratorFunction(Kp_low_y, Ki_low, alpha_low_y, tau_low_y, Je_y,
Be_y);

%% Develop Continous Closed Loop Transfer Functions
% X-Axis
CL_low_x = feedback(Gs_low_x, 1);
CL_high_x = feedback(Gs_high_x, 1);
% Y-Axis
CL_low_y = feedback(Gs_low_y, 1);
CL_high_y = feedback(Gs_high_y, 1);

%% Develop Discrete closed loop transfer functions
% X-axis
CL_low_x_dis = c2d(CL_low_x, Ts, 'tustin');
CL_high_x_dis = c2d(CL_high_x, Ts, 'tustin');
% y-ayis
CL_low_y_dis = c2d(CL_low_y, Ts, 'tustin');
CL_high_y_dis = c2d(CL_high_y, Ts, 'tustin');

%% Determine Poles, Zeros, bandwidth, overshoot, and rise time for both axis,
frequencies, and domains
% X-axis, continous
param_x_low_cont = analyzeTF(CL_low_x_dis);
param_x_high_cont = analyzeTF(CL_high_x_dis);
% X-axis, discrete
param_x_low_dis = analyzeTF(CL_low_x_dis);
param_x_high_dis = analyzeTF(CL_high_x_dis);
% Y-axis, continous
param_y_low_cont = analyzeTF(CL_low_y_dis);
param_y_high_cont = analyzeTF(CL_high_y_dis);
% Y-axis, discrete
param_y_low_dis = analyzeTF(CL_low_y_dis);
param_y_high_dis = analyzeTF(CL_high_y_dis);

%% Plot Bode Plots for Open-Loop Systems of X-Axis at both crossover frequencies
% Plot Open-Loop Bode Plots for X-Axis
figure;
bode(Gs_low_x);
```

```
hold on;
bode(Gs_high_x);
title('Open-Loop Bode Plots for X-Axis with LBW and HBW Controllers');
legend('LBW Controller', 'HBW Controller');
grid on;
% Plot Closed-Loop Bode Plots for X-Axis
figure;
bode(CL_low_x);
hold on;
bode(CL_high_x);
title('Closed-Loop Bode Plots for X-Axis with LBW and HBW Controllers');
legend('LBW Controller', 'HBW Controller');
grid on;
```

Appendix 3: Prelab Q3 Simulink Models



Appendix 4: Matlab Q4 Code

```
% Extract data from the low-bandwidth simulation
simDataLow = outLow.simout;
timeLow = simDataLow.Time;          % Time vector for low bandwidth
input_x_low = simDataLow.Data(:, 2); % Input x (reference trajectory r_x) - Low BW
output_x_low = simDataLow.Data(:, 3); % Output x (simulated x position) - Low BW
input_y_low = simDataLow.Data(:, 4); % Input y (reference trajectory r_y) - Low BW
output_y_low = simDataLow.Data(:, 5); % Output y (simulated y position) - Low BW

% Extract data from the high-bandwidth simulation
simDataHigh = outHigh.simout;
timeHigh = simDataHigh.Time;          % Time vector for high bandwidth
input_x_high = simDataHigh.Data(:, 2); % Input x (reference trajectory r_x) - High BW
output_x_high = simDataHigh.Data(:, 3); % Output x (simulated x position) - High BW
input_y_high = simDataHigh.Data(:, 4); % Input y (reference trajectory r_y) - High BW
output_y_high = simDataHigh.Data(:, 5); % Output y (simulated y position) - High BW

%% E1 - Plot Error graphs
% Calculate X-axis tracking errors
error_x_low = input_x_low - output_x_low; % X-axis error for low bandwidth
error_x_high = input_x_high - output_x_high; % X-axis error for high bandwidth

% Calculate Y-axis tracking errors
error_y_low = input_y_low - output_y_low; % Y-axis error for low bandwidth
error_y_high = input_y_high - output_y_high; % Y-axis error for high bandwidth

% Plot X-axis tracking errors (low vs high bandwidth)
figure;
plot(timeLow, error_x_low, 'LineWidth', 1.5, 'DisplayName', 'Error X (Low BW)');
hold on;
plot(timeHigh, error_x_high, '--', 'LineWidth', 1.5, 'DisplayName', 'Error X (High
BW)');
xlabel('Time [s]');
ylabel('X Error [mm]');
title('X-Axis Tracking Errors: Low vs High Bandwidth');
legend show;
grid on;

% Plot Y-axis tracking errors (low vs high bandwidth)
figure;
plot(timeLow, error_y_low, 'LineWidth', 1.5, 'DisplayName', 'Error Y (Low BW)');
hold on;
plot(timeHigh, error_y_high, '--', 'LineWidth', 1.5, 'DisplayName', 'Error Y (High
BW)');
xlabel('Time [s]');
ylabel('Y Error [mm]');
title('Y-Axis Tracking Errors: Low vs High Bandwidth');
legend show;
grid on;

%% E2 Code below
%% Combine and Plot Simulated Tool Motion with Reference Toolpath
% Simulated Tool Motion (Low and High Bandwidth) = Simulation Output
% Reference Toolpath = trajectory created inputs
```

```
% Full Plot: Reference Toolpath and Simulated Paths
figure;
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Low BW)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High BW)');
plot(output_x_low, output_y_high, 'g-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Mismatched Dyn.)');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
title('Toolpath: Reference vs Simulated (Low and High Bandwidth)');
legend show;
grid on;

%% Critical Regions R1, R2, R3, and R4
% Define approximate limits for critical regions
R1_limits = [19, 21, 14, 16]; % [xmin, xmax, ymin, ymax]
R2_limits = [38, 42, 28, 32];
R3_limits = [58, 62, 28, 32];
R4_limits = [102, 108, 52, 58];

% Create a 2x2 subplot for zoomed-in critical regions
figure;

subplot(2, 2, 1); % R1
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Low BW)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High BW)');
plot(output_x_low, output_y_high, 'g-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Mismatched Dyn.)');
axis(R1_limits); % Zoom to R1
title('Region R1');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

subplot(2, 2, 2); % R2
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Low BW)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High BW)');
plot(output_x_low, output_y_high, 'g-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Mismatched Dyn.)');
axis(R2_limits); % Zoom to R2
```

```
title('Region R2');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

subplot(2, 2, 3); % R3
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Low BW)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High BW)');
plot(output_x_low, output_y_high, 'g-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Mismatched Dyn.)');
axis(R3_limits); % Zoom to R3
title('Region R3');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

subplot(2, 2, 4); % R4
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Low BW)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High BW)');
plot(output_x_low, output_y_high, 'g-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Mismatched Dyn.)');
axis(R4_limits); % Zoom to R4
title('Region R4');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

% Measure the Largest Contouring Error
% Contouring error is the maximum deviation from the reference toolpath
% Calculate contouring error for each bandwidth (distance between simulated and
reference paths)

contour_error_low = sqrt((output_x_low - input_x_low).^2 + ...
                         (output_y_low - input_y_low).^2);
contour_error_high = sqrt((output_x_high - input_x_low).^2 + ...
                          (output_y_high - input_y_low).^2);

max_contour_error_low = max(contour_error_low); % Max contouring error for low BW
max_contour_error_high = max(contour_error_high); % Max contouring error for high BW

fprintf('Max Contouring Error (Low Bandwidth): %.2f mm\n', max_contour_error_low);
fprintf('Max Contouring Error (High Bandwidth): %.2f mm\n', max_contour_error_high);

% Comment on the effect of bandwidth:
% - Low bandwidth results in larger contouring error, especially in sharp turns.
```

```
% - High bandwidth minimizes contouring error but may introduce small oscillations or
overshoot.

%% Step 1: Extract Midpoint of Region R1
x_mid = mean([R1_limits(1), R1_limits(2)]); % X midpoint
y_mid = mean([R1_limits(3), R1_limits(4)]); % Y midpoint

%% Step 2: Find Closest Point to Midpoint in Reference Trajectory
% Compute distances from all reference points to the midpoint
distances = sqrt((input_x_low - x_mid).^2 + (input_y_low - y_mid).^2);
[~, idx_mid] = min(distances); % Index of the closest point

% Extract values at the midpoint
ref_x_mid = input_x_low(idx_mid);
ref_y_mid = input_y_low(idx_mid);
sim_x_mid = output_x_low(idx_mid); % Low bandwidth simulation
sim_y_mid = output_y_low(idx_mid);

% Calculate tracking errors at the midpoint
error_x_mid = ref_x_mid - sim_x_mid;
error_y_mid = ref_y_mid - sim_y_mid;

% Display results
fprintf('Tracking Error at Midpoint (X-axis): %.2f mm\n', error_x_mid);
fprintf('Tracking Error at Midpoint (Y-axis): %.2f mm\n', error_y_mid);
fprintf('Contouring Error at Midpoint: %.2f mm\n', contour_error_mid);
```

Appendix 5: Q5 Matlab Code

```
% Extract data from the low-bandwidth simulation - low feedrate
simDataLow = outLow.simout;
timeLow = simDataLow.Time;          % Time vector for low bandwidth
input_x_low = simDataLow.Data(:, 2); % Input x (reference trajectory r_x) - Low BW
output_x_low = simDataLow.Data(:, 3); % Output x (simulated x position) - Low BW
input_y_low = simDataLow.Data(:, 4); % Input y (reference trajectory r_y) - Low BW
output_y_low = simDataLow.Data(:, 5); % Output y (simulated y position) - Low BW

% Extract data from the low-bandwidth simulation - high feedrate
simDataHigh = outHighFeedrate.simout;
timeHigh = simDataHigh.Time;          % Time vector for high bandwidth
input_x_high = simDataHigh.Data(:, 2); % Input x (reference trajectory r_x) - High BW
output_x_high = simDataHigh.Data(:, 3); % Output x (simulated x position) - High BW
input_y_high = simDataHigh.Data(:, 4); % Input y (reference trajectory r_y) - High BW
output_y_high = simDataHigh.Data(:, 5); % Output y (simulated y position) - High BW

%% F1 - Plot Error graphs
% Calculate X-axis tracking errors
error_x_low = input_x_low - output_x_low; % X-axis error for low bandwidth
error_x_high = input_x_high - output_x_high; % X-axis error for high bandwidth

% Calculate Y-axis tracking errors
error_y_low = input_y_low - output_y_low; % Y-axis error for low bandwidth
error_y_high = input_y_high - output_y_high; % Y-axis error for high bandwidth

% Plot X-axis tracking errors (low vs high bandwidth)
figure;
plot(timeLow, error_x_low, 'LineWidth', 1.5, 'DisplayName', 'Error X (Low
Feedrate)');
hold on;
plot(timeHigh, error_x_high, '--', 'LineWidth', 1.5, 'DisplayName', 'Error X (High
Feedrate)');
xlabel('Time [s]');
ylabel('X Error [mm]');
title('X-Axis Tracking Errors: Low vs High Feedrate');
legend show;
grid on;

% Plot Y-axis tracking errors (low vs high bandwidth)
figure;
plot(timeLow, error_y_low, 'LineWidth', 1.5, 'DisplayName', 'Error Y (Low
Feedrate)');
hold on;
plot(timeHigh, error_y_high, '--', 'LineWidth', 1.5, 'DisplayName', 'Error Y (High
Feedrate)');
xlabel('Time [s]');
ylabel('Y Error [mm]');
title('Y-Axis Tracking Errors: Low vs High Feedrate');
legend show;
grid on;

%% F2 Code below
%% Combine and Plot Simulated Tool Motion with Reference Toolpath
```

```
% Simulated Tool Motion (Low and High Bandwidth) = Simulation Output
% Reference Toolpath = trajectory created inputs

% Full Plot: Reference Toolpath and Simulated Paths
figure;
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Low Feedrate)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High Feedrate)');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
title('Toolpath: Reference vs Simulated (Low and High Feedrate)');
legend show;
grid on;

%% Critical Regions R1, R2, R3, and R4
% Define approximate limits for critical regions
R1_limits = [19, 21, 14, 16]; % [xmin, xmax, ymin, ymax]
R2_limits = [38, 42, 28, 32];
R3_limits = [58, 62, 28, 32];
R4_limits = [102, 108, 52, 58];

% Create a 2x2 subplot for zoomed-in critical regions
figure;

subplot(2, 2, 1); % R1
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 2, 'DisplayName', 'Simulated Path
(Low Feedrate)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High Feedrate)');
axis(R1_limits); % Zoom to R1
title('Region R1');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

subplot(2, 2, 2); % R2
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 2, 'DisplayName', 'Simulated Path
(Low Feedrate)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High Feedrate)');
axis(R2_limits); % Zoom to R2
title('Region R2');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;
```

```
subplot(2, 2, 3); % R3
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Low Feedrate)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High Feedrate)');
axis(R3_limits); % Zoom to R3
title('Region R3');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

subplot(2, 2, 4); % R4
plot(input_x_low, input_y_low, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference
Toolpath');
hold on;
plot(output_x_low, output_y_low, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (Low Feedrate)');
plot(output_x_high, output_y_high, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated
Path (High Feedrate)');
axis(R4_limits); % Zoom to R4
title('Region R4');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

% Measure the Largest Contouring Error
% Contouring error is the maximum deviation from the reference toolpath
% Calculate contouring error for each bandwidth (distance between simulated and
reference paths)

contour_error_low = sqrt((output_x_low - input_x_low).^2 + ...
                         (output_y_low - input_y_low).^2);
contour_error_high = sqrt((output_x_high - input_x_low).^2 + ...
                          (output_y_high - input_y_low).^2);

max_contour_error_low = max(contour_error_low); % Max contouring error for low BW
max_contour_error_high = max(contour_error_high); % Max contouring error for high BW

fprintf('Max Contouring Error (Low Bandwidth): %.2f mm\n', max_contour_error_low);
fprintf('Max Contouring Error (High Bandwidth): %.2f mm\n', max_contour_error_high);
```

Appendix 6: Q6 Matlab Code

```
%% Load Experimental Data & Trajectory Data (Y)
dataG = readtable("standard.csv");
tMeasured = dataG.Time/1000;
xTraj = dataG.X_Traj;
yTraj = dataG.Y_Traj;
xMeasured = dataG.X_ActPos;
yMeasured = dataG.Y_ActPos;

%% Load Input (r) Data
xInput = pos_x_combined;
yInput = pos_y_combined;
tInput = t_combined;

%% Compute Errors
ErrorXInput = xInput - xTraj;
ErroryInput = yInput - yTraj;
ErrorXMeasured = xMeasured - xTraj;
ErrorYMeasured = yMeasured - yTraj;

%% G - Plot Error graphs
% Plot X-axis tracking errors (low vs high bandwidth)
figure;
plot(tInput, ErrorXInput, 'LineWidth', 1.5, 'DisplayName', 'Error X (Low BW)');
hold on;
plot(tMeasured, ErrorXMeasured, '--', 'LineWidth', 1.5, 'DisplayName', 'Error X (High
BW)');
xlabel('Time [s]');
ylabel('X Error [mm]');
title('X-Axis Tracking Errors: Simulated & Measured');
legend show;
grid on;

% Plot Y-axis tracking errors (low vs high bandwidth)
figure;
plot(tInput, ErroryInput, 'LineWidth', 1.5, 'DisplayName', 'Error X (Low BW)');
hold on;
plot(tMeasured, ErrorYMeasured, '--', 'LineWidth', 1.5, 'DisplayName', 'Error X (High
BW)');
xlabel('Time [s]');
ylabel('Y Error [mm]');
title('Y-Axis Tracking Errors: Simulated & Measured');
legend show;
grid on;

%% E2 Code below
%% Combine and Plot Simulated Tool Motion with Reference Toolpath
% Simulated Tool Motion (Low and High Bandwidth) = Simulation Output
% Reference Toolpath = trajectory created inputs

% Full Plot: Reference Toolpath and Simulated Paths
figure;
plot(xinput, xinput, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference Toolpath');
hold on;
```

```
plot(xTraj, yTraj, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path (Low BW)');
plot(xMeasured, yMeasured, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path
(High BW)');
plot(xTraj, yMeasured, 'g-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path
(Mismatched Dyn.)');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
title('Toolpath: Reference vs Simulated (Low and High Bandwidth)');
legend show;
grid on;

%% Critical Regions R1, R2, R3, and R4
% Define approximate limits for critical regions
R1_limits = [19, 21, 14, 16]; % [xmin, xmax, ymin, ymax]
R2_limits = [38, 42, 28, 32];
R3_limits = [58, 62, 28, 32];
R4_limits = [102, 108, 52, 58];

% Create a 2x2 subplot for zoomed-in critical regions
figure;

subplot(2, 2, 1); % R1
plot(xinput, yinput, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference Toolpath');
hold on;
plot(xTraj, yTraj, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path (Low BW)');
plot(xMeasured, yMeasured, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path
(High BW)');
axis(R1_limits); % Zoom to R1
title('Region R1');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

subplot(2, 2, 2); % R2
plot(xinput, xinput, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference Toolpath');
hold on;
plot(xTraj, yTraj, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path (Low BW)');
plot(xMeasured, yMeasured, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path
(High BW)');
axis(R2_limits); % Zoom to R2
title('Region R2');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;

subplot(2, 2, 3); % R3
plot(xinput, xinput, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference Toolpath');
hold on;
plot(xTraj, yTraj, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path (Low BW)');
plot(xMeasured, yMeasured, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path
(High BW)');
axis(R3_limits); % Zoom to R3
title('Region R3');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
```

```
grid on;

subplot(2, 2, 4); % R4
plot(xinput, xinput, 'k--', 'LineWidth', 1.5, 'DisplayName', 'Reference Toolpath');
hold on;
plot(xTraj, yTraj, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path (Low BW)');
plot(xMeasured, yMeasured, 'r-', 'LineWidth', 1.5, 'DisplayName', 'Simulated Path
(High BW)');
axis(R4_limits); % Zoom to R4
title('Region R4');
xlabel('X Position [mm]');
ylabel('Y Position [mm]');
grid on;
```