WORCESTER POLYTECHNIC INSTITUTE
ROBOTICS ENGINEERING PROGRAM

Lab 2 – Forward Kinematics

SUBMITTED BY
Nicolas Graham
Pranav Jain
Fiona Prendergast

Date Submitted: 09.12.2024
Date Completed: 09.12.2024
Course Instructor: Prof. Agheli
Lab Section: RBE 3001 A'24

Abstract

This Report covers the implementation of forward kinematics using Denavit-Hartenberg (DH) convention on the OpenManipulator-X robot arm using MATLAB. The objective of this lab was to use MATLAB to compute the position and orientation of the end effector (EE) in physical space. The Lab was focused on calculating forward kinematics, visualizing the robot's current configuration using a stick diagram in a 3d plot in MATLAB, and verified this model by manipulating the OpenManipulator-X robot arm to multiple different positions. These results will help in future labs and trajectory generation tasks.

Introduction

The purpose of Lab 2 was to build on the foundational concepts introduced and explored in Lab 1, where communication with the robot arm was achieved by writing methods in MATLAB to sending values to and receiving data from the robot arm. The overarching goal of the lab was to compute the forward kinematics (FK) of the robot arm.

Forward Kinematics is a method of determining the position and orientation of the end effector (the gripper in this case), called the pose, of the robotic arm relative to the base of the robot when given a set of joint angles. The Denavit-Hartenberg Method (DH Method) was used to calculate the FK of the robot. The DH method involves a series of steps to assign frames, determine the DH parameters between each link, and finally determine the final homogeneous transformation matrix to describe the pose of the end effector.

By calculating the FK of the robot, MATLAB could be used to visualize the robot's movement and verify the calculations by comparing a 3D stick model to the actual movement of the robot. By verifying and being able to visualize the movement of the robot arm, the work done in Lab 2 is an important step to next be able to work backwards to control the pose of the end effector with inverse kinematics and plan the robot motion with trajectory planning.

Methodology

The methodology for Lab 2 focused on writing the methods for and implementing the forward kinematics of the OpenManipulator-X robot arm with its 4 DoF using DH parameters. The process was divided into a few key steps:

1) DH Parameter assignment
    - The first step was to assign the proper frames to each joint of the robot arm according to the DH convention. Figure 1 shows a diagram of the robot arm with each frame and X and Z axis labeled.
    - The next step was to create a table of the DH parameters that described the robot arm's geometry. Figure 2 shows the table of DH parameters for each of the links of the robot arm.
2) Transformation Matrices
    - A method called dh2mat() was written in MATLAB to calculate the symbolic transformation matrix for each link. The method takes in the four DH parameters for a transformation between two joints (theta, d, a, and alpha) and returns the corresponding 4x4 homogeneous transformation matrix describing the link.

3) Forward Kinematics
    - The next step was to write a method to calculate the final symbolic transformation based on input of the DH table to describe the pose of the end effector in relation to the base of the robot, called dh2fk(). The method used the previous method dh2mat() and the DH table shown in figure 2 to compute the transformation matrix for each link and post multiply to find the final symbolic 4x4 homogeneous transformation matrix.
    - A method called fk3001() calculates the final numeric transformation based on input of the four joint angles to describe the pose of the end effector in relation to the base of the robot. The method used the previous method dh2fk() and the joint angles to compute the final symbolic 4x4 homogeneous transformation matrix.
    - In order to interface with the robot arm, three methods were written utilizing the fk3001() method and the methods written in Lab 1 to get the current, setpoint and goal values of the joints.
        - The measured_cp() method takes the output of measured_js() and formats it to be input for the fk3001() to return the transformation matrix describing the end effector based on the current measured joint values.
        - The setpoint_cp() method takes the output of setpoint_js() and formats it to be input for the fk3001() to return the transformation matrix describing the end effector based on the current setpoint positions.
        - The goal_cp() method takes the output of goal_js() and formats it to be input for the fk3001() to return the transformation matrix describing the end effector based on the joint goal position.
4) Error Analysis
    - In order to verify the consistency of the robot arm, a test was conducted. The robot arm was sent to an arbitrary position with angles of [-90, -45, -30, 52] and then it came back to its home configuration. This was repeated 4 more times, and the end-effector position was recorded for each iteration. The position values were then plotted along with the ideal position. Lastly, the average end-effector position was calculated and the RMS for each of the iterations were calculated.
5) Visualization of Robot Movement (3D stick model)
    - A class named Model was created that was a sub-class of handle and references the Robot class.
    - A method called plot_arm() was created in the Model class that displays the arm position based on the input of the four joint angles. It uses the dh2mat() method to calculate each of the link transformations which is then used to create the arm.
    - By continuously calling the plot_arm() and inputting the current angles of the joints using setpoint_js(), a live plot of the robotic arm is created.
6) Triangular Movement
    - To create a triangle, three arbitrary positions are selected that act as the three vertices of the triangle. These arbitrary positions have angles of [0, -30, 0, 0], [0, -30, 0, 60] and [0, 0, 0, 0]. While the robotic arm is creating the triangle, the time, joint angles and the end-effector positions are recorded and stored in a file named ‘lab2Data.csv’. These values

are used to plot multiple graphs including the joint angles against time, the X and Z values of end-effector against time, the X values against the Z values of the end-effector, and the X values against the Y values of the end-effector.

By doing each step, the forward kinematics for the robot arm were successfully calculated and verified, and the methods written to be used with data taken from the robot arm.

Results

DH Parameter Assignment

Figure 1 shows the frames that we assigned to each of the joints. Figure 2 shows the DH table. Figure 3 is the lengths of each of the links labeled in the diagram of the robot arm in Figure 1. The dia

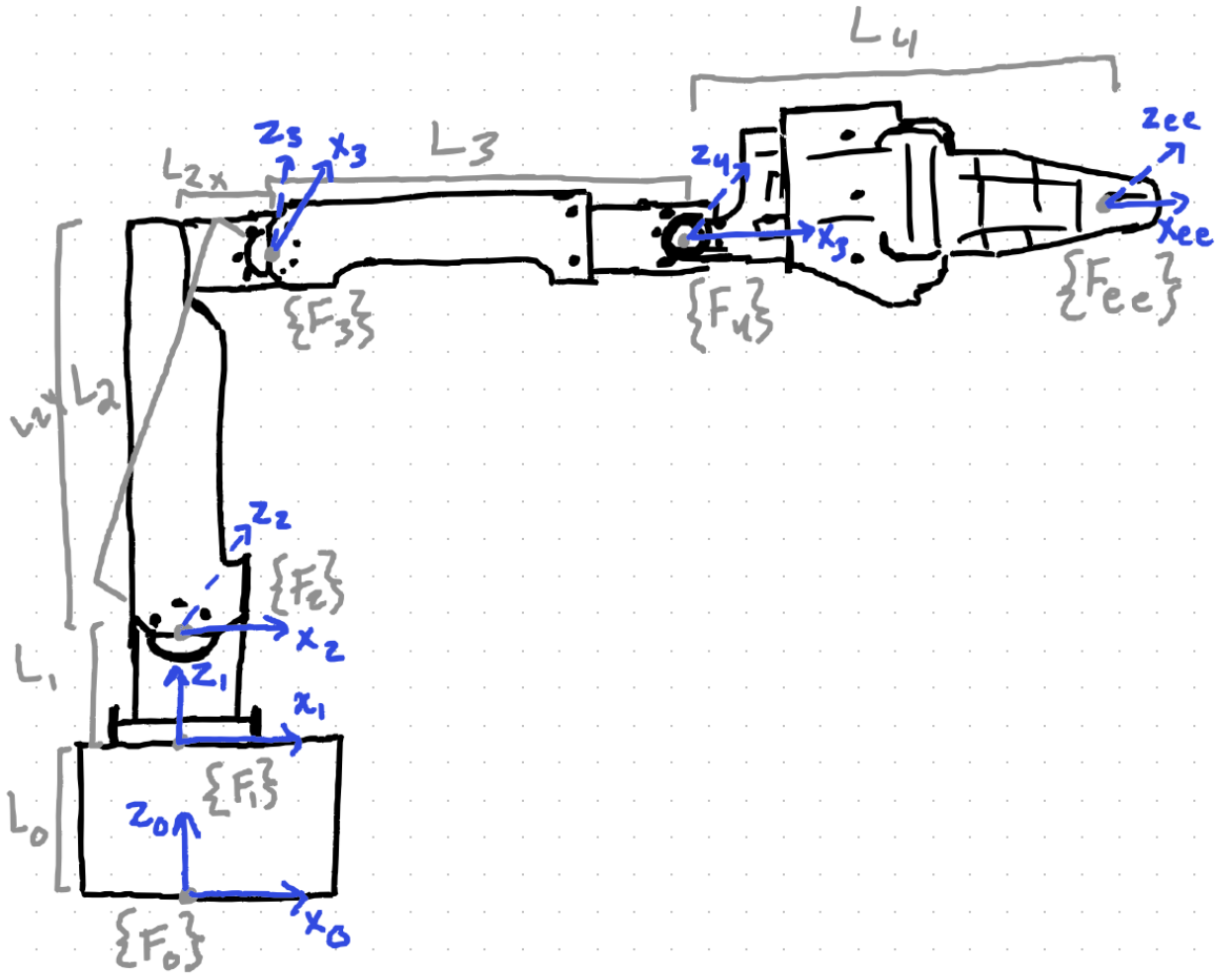

Figure 1: Diagram of the OpenManipulator-X robot arm in the home position (all of the joint angles set to 0 degrees) with the frames 0-4 and end effector frame assigned, and the link lengths labeled.

Link	theta [deg]	d [mm]	a [mm]	alpha1 [deg]
1	theta1	L1	0	-90
2	theta2 – 79.38	0	L2	0
3	theta2 + 79.38	0	L3	0
4	theta4	0	L4	0

Figure 2: Table of the DH Parameters for each link in the robot arm (not including link 0). The 'theta' and 'alpha' parameters are in degrees and the 'd' and 'a' parameters are in millimeters.

Link	Lengths [mm]
L0	36.076
L1	60.25
L2x	24.0
L2y	128.0
L2	130.231
L3	124
L4	133.4

Figure 3: Table of the lengths of each of the robot links in millimeters from the lengths given in Lab 1. L2 was calculated using L2x and L2y and the Pythagorean theorem.

Transformation Matrices

The following figures 4-8 each show the transformation matrix computed by the dh2mat() where each variable has T and the link number. The input of the function was the row in the table of dh parameters (Figure 2) corresponding to the link number. T0 was found by inspection for the variable input.

```
T0 =

[1, 0, 0,         0]
[0, 1, 0,         0]
[0, 0, 1, 9019/250]
[0, 0, 0,         1]
```

Figure 4: Screenshot of MATLAB output showing the transformation matrix computed for link 0 of the robot using the dh2mat method with an input of (0, L0, 0, 0) where L0 = 36.076.

```
T1 =

[cos((pi*theta1)/180),  0, -sin((pi*theta1)/180),     0]
[sin((pi*theta1)/180),  0,  cos((pi*theta1)/180),     0]
[                   0, -1,                     0, 241/4]
[                   0,  0,                     0,     1]
```

Figure 5: Screenshot of MATLAB output showing the transformation matrix computed for link 1 of the robot using the dh2mat method with an input of (theta1, L1, 0, -90) where L1 = 60.25.

```
T2 =

[cos((pi*(theta2 - 3969/50))/180), -sin((pi*(theta2 - 3969/50))/180), 0, (572761995187585*cos((pi*(theta2 - 3969/50))/180))/4398046511104]
[sin((pi*(theta2 - 3969/50))/180),  cos((pi*(theta2 - 3969/50))/180), 0, (572761995187585*sin((pi*(theta2 - 3969/50))/180))/4398046511104]
[                               0,                                 0, 1,                                                             0]
[                               0,                                 0, 0,                                                             1]
```

Figure 6: Screenshot of MATLAB output showing the transformation matrix computed for link 2 of the robot using the dh2mat method with an input of (theta2-79.38, 0, L2, 0) where L2 = 130.231.

```
T3 =

[cos((pi*(theta3 + 3969/50))/180), -sin((pi*(theta3 + 3969/50))/180), 0, 124*cos((pi*(theta3 + 3969/50))/180)]
[sin((pi*(theta3 + 3969/50))/180),  cos((pi*(theta3 + 3969/50))/180), 0, 124*sin((pi*(theta3 + 3969/50))/180)]
[                               0,                                 0, 1,                                   0]
[                               0,                                 0, 0,                                   1]
```

Figure 7: Screenshot of MATLAB output showing the transformation matrix computed for link 3 of the robot using the dh2mat method with an input of (theta3+79.38, 0, L3, 0) where L3 = 124.

```
T4 =

[cos((pi*theta4)/180), -sin((pi*theta4)/180), 0, (667*cos((pi*theta4)/180))/5]
[sin((pi*theta4)/180),  cos((pi*theta4)/180), 0, (667*sin((pi*theta4)/180))/5]
[                   0,                     0, 1,                           0]
[                   0,                     0, 0,                           1]
```

Figure 8: Screenshot of MATLAB output showing the transformation matrix computed for link 4 of the robot using the dh2mat method with an input of (theta4, 0, L4, 0) where L4 = 133.4.

Forward Kinematics

Post multiplying the matrices in figures 4-8, representing the 4x4 homogeneous transformation matrices for links 0-4, will result in T_0^5 which shows the transformation from the base of the robot to the end effector. The output calls the matrix T5, and has been put through a 'pretty()' function to make it easier to look at since the symbolic representation of the transformation matrix would be too long to see the output clearly. T5 was computed as output from the dh2fk() function with input from the matrix holding the values in the dh table shown in Figure 2.

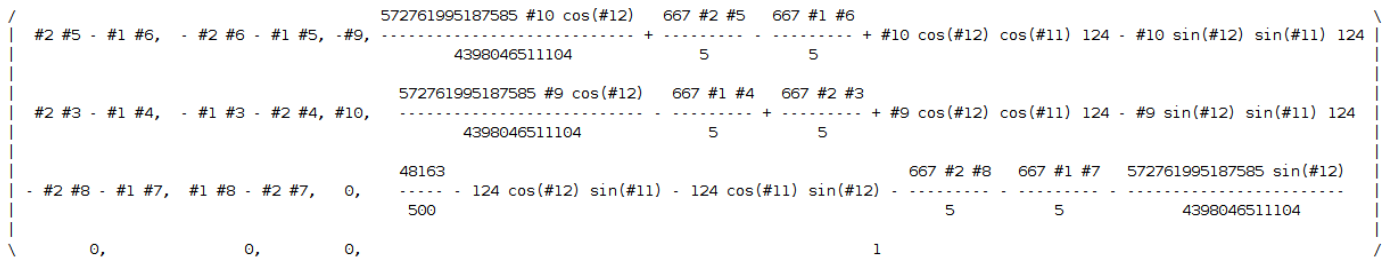

Figure 9a: Screenshot of MATLAB output of T5 showing the 4x4 homogeneous transformation matrix computed by inputting the dh table into the dh2fk() function and putting the output into pretty() function for ease of readability.

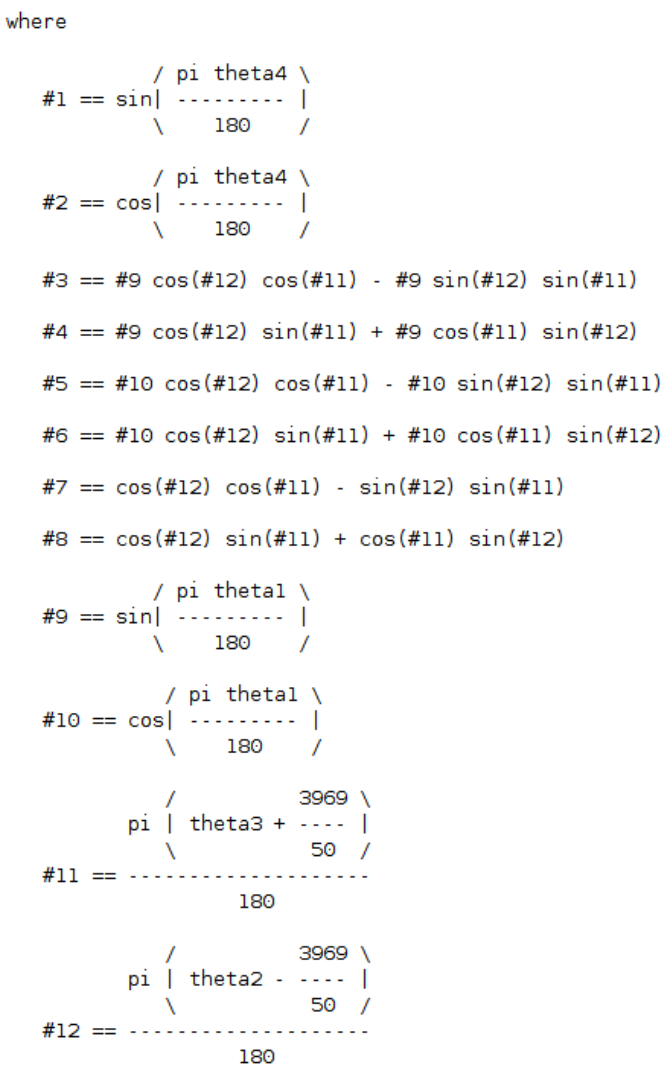

Figure 9b: Screenshot of MATLAB output of the values to be substituted into T5 (shown in figure 9a) which shows the 4x4 homogeneous transformation dh2fk() function representing the transformation from the base of the robot to the end effector.

The forward kinematics were also computed based on the joint angles based on the pose of the robot when put in various positions. The following 4 matrices shown in figures 10-13 show the 4x4 homogeneous transformation matrix computed with the fk3001() function after moving the robot to a specific pose. Figure 10 shows the transformation matrix, called T6, computed with the robot arm in the home position and the matrices T7, T8 and T9 show the transformation matrices for arbitrary poses of the robot.

The forward kinematics for joint angles [0, 0, 0, 0] is shown below in Figure 10.

```
T6 =

    1.0000         0         0  281.4009
         0         0    1.0000         0
         0   -1.0000         0  224.3263
         0         0         0    1.0000
```

Figure 10: Screenshot of MATLAB output of T6 showing the 4x4 homogeneous transformation matrix computed by the fk3001() method with the robot in the home position with joint angles of [0, 0, 0, 0] degrees.

The forward kinematics for joint angles [45, 30, 0, -20] is shown below in Figure 11.

```
T7 =

    0.6964   -0.1228   -0.7071  228.7816
    0.6964   -0.1228    0.7071  228.7816
   -0.1736   -0.9848         0  110.0124
         0         0         0    1.0000
```

Figure 11: Screenshot of MATLAB output of T7 showing the 4x4 homogeneous transformation matrix computed by the fk3001() method with the joint angles of the robot set to [45, 30, 0, -20] degrees.

The forward kinematics for joint angles [0, 30, -55, 15] is shown below in Figure 12.

```
T8 =

    0.9848    0.1736         0  328.5410
         0         0    1.0000         0
    0.1736   -0.9848         0  270.7464
         0         0         0    1.0000
```

Figure 12: Screenshot of MATLAB output of T8 showing the 4x4 homogeneous transformation matrix computed by the fk3001() method with the joint angles of the robot set to [0, 30, -55, 15] degrees.

The forward kinematics for joint angles [-90, -45, -30, 52] is shown below in Figure 13.

```
T9 =

    0.0000    0.0000    1.0000    0.0000
   -0.9205   -0.3907    0.0000  -81.3502
    0.3907   -0.9205         0  375.7054
         0         0         0    1.0000
```

Figure 13: Screenshot of MATLAB output of T9 showing the 4x4 homogeneous transformation matrix computed by the fk3001() method with the joint angles of the robot set to [-90, -45, -30, 52] degrees.

Robot Interfacing Functions

The functions measured_cp(), setpoint_cp() and goal_cp() were written in order to be able to interface forward kinematics with the measured values from the joints in the robot arm. In order to verify the functions, the following figures show the matrices computed by each of the three functions when the robotic arm was sent to its base configuration with joint angles of [0, 0, 0, 0] (and thus the goal and setpoint values were also [0, 0, 0, 0]). The measured_cp() gives M1 shown in figure 14. The setpoint_cp() gives M2 shown in figure 15. The goal_cp() gives M3 shown in figure 16.

```
M1 =

    0.9996   -0.0291         0  282.6962
         0         0    1.0000         0
   -0.0291   -0.9996         0  217.8913
         0         0         0    1.0000
```

Figure 14: Screenshot of MATLAB output of M1 showing the 4x4 homogeneous transformation matrix computed by the measured_cp() method with the robot in the home position with joint angles of [0, 0, 0, 0] degrees.

```
M2 =

    0.9996   -0.0291         0  282.6962
         0         0    1.0000         0
   -0.0291   -0.9996         0  217.8913
         0         0         0    1.0000
```

Figure 15: Screenshot of MATLAB output of M1 showing the 4x4 homogeneous transformation matrix computed by the setpoint_cp() method with the robot in the home position with joint angles of [0, 0, 0, 0] degrees.

```
M3 =

    0.9909    0.1346         0  270.0417
         0         0    1.0000         0
    0.1346   -0.9909         0  261.2599
         0         0         0    1.0000
```

Figure 16: Screenshot of MATLAB output of M1 showing the 4x4 homogeneous transformation matrix computed by the goal_cp() method with the robot in the home position with joint angles of [0, 0, 0, 0] degrees.

Error Analysis

The reliability of the fk3001 method was tested by moving the arm away and back to the home position, computing and plotting the tip positions in a 3D plot compared to the ideal position calculated with the ideal joint angles of [0, 0, 0, 0]. The plot is shown below in Figure 17. The average tip position and the root mean squared for each of the transformation matrices were also computed with the data and are listed below.

Average Tip Position: [283.0766, -0.0865, 217.2887]

RMS for P1 = 3.9424

RMS for P2 = 4.4708

RMS for P3 = 4.3163

RMS for P4 = 4.7398

RMS for P5 = 3.4543

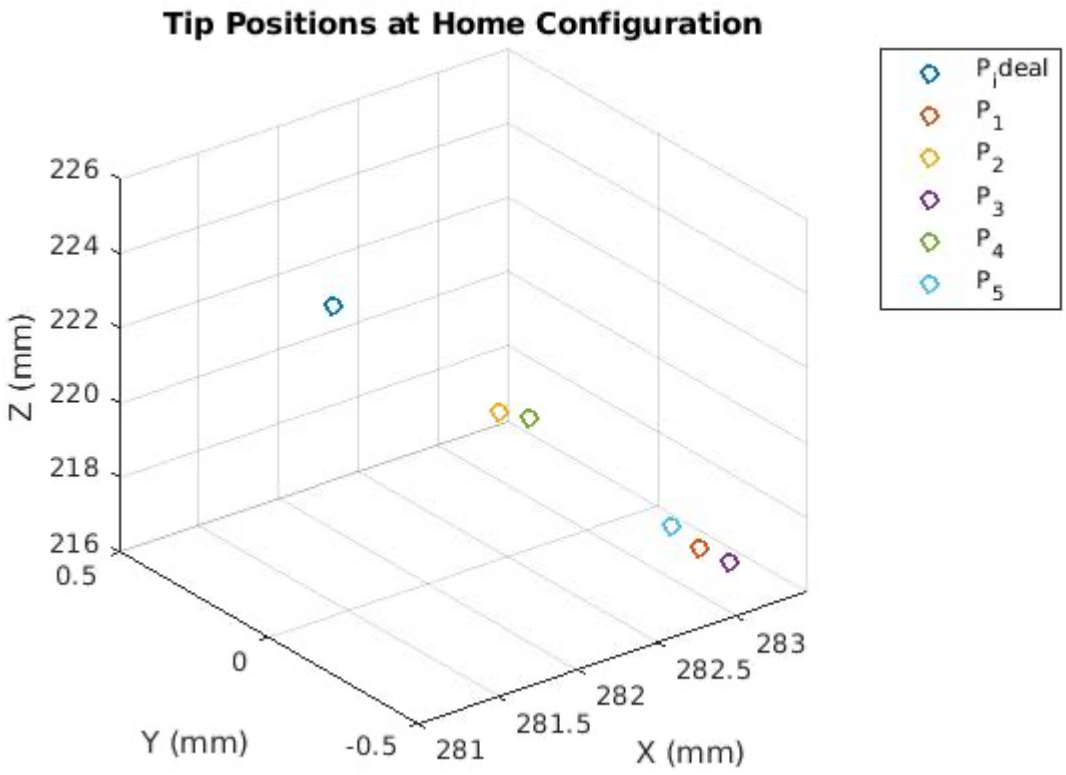

Figure 17: 3D MATLAB plot showing the position of the end effector of the robot arm computed from the fk3001 function five times compared to the ideal position of the end effector with joint angles [0, 0, 0, 0] degrees.

Visualization of Robot Movement (3D Stick Model)

A 3D stick model was also created to visualize the motion of the robot in 3 dimensions. A few joint values were tested in plot_arm() method to see whether it was working properly. The home configuration with joint angles of [0, 0, 0, 0] and two other positions with angles of [45, 30, 0, -20] and angles [0, 30, -55, 15] respectively were plotted and tested and the 3D stick plots are shown below in Figure 18, Figure 19 and Figure 20.

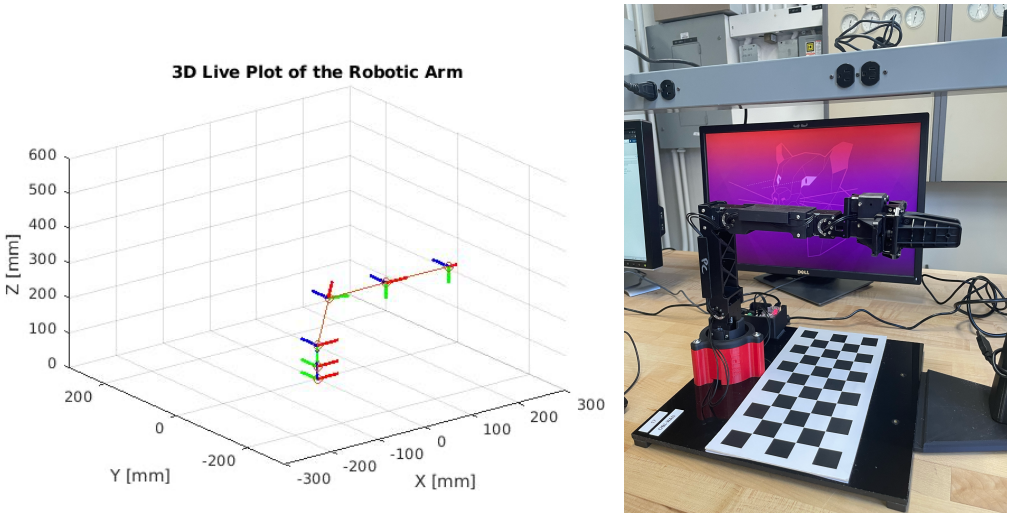

Figure 18: Home Position Stick Model and Physical image with joint angles [0, 0, 0, 0].

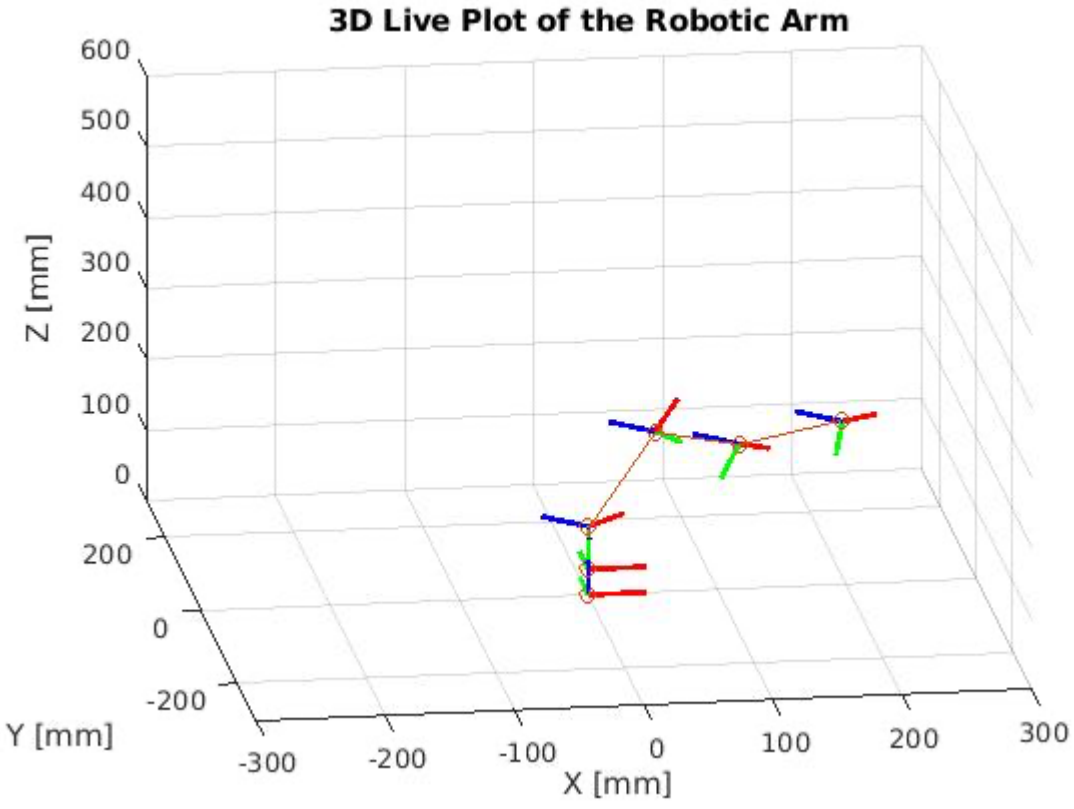

Figure 19: 3D Stick Model with joint angles [45, 30, 0, -20].

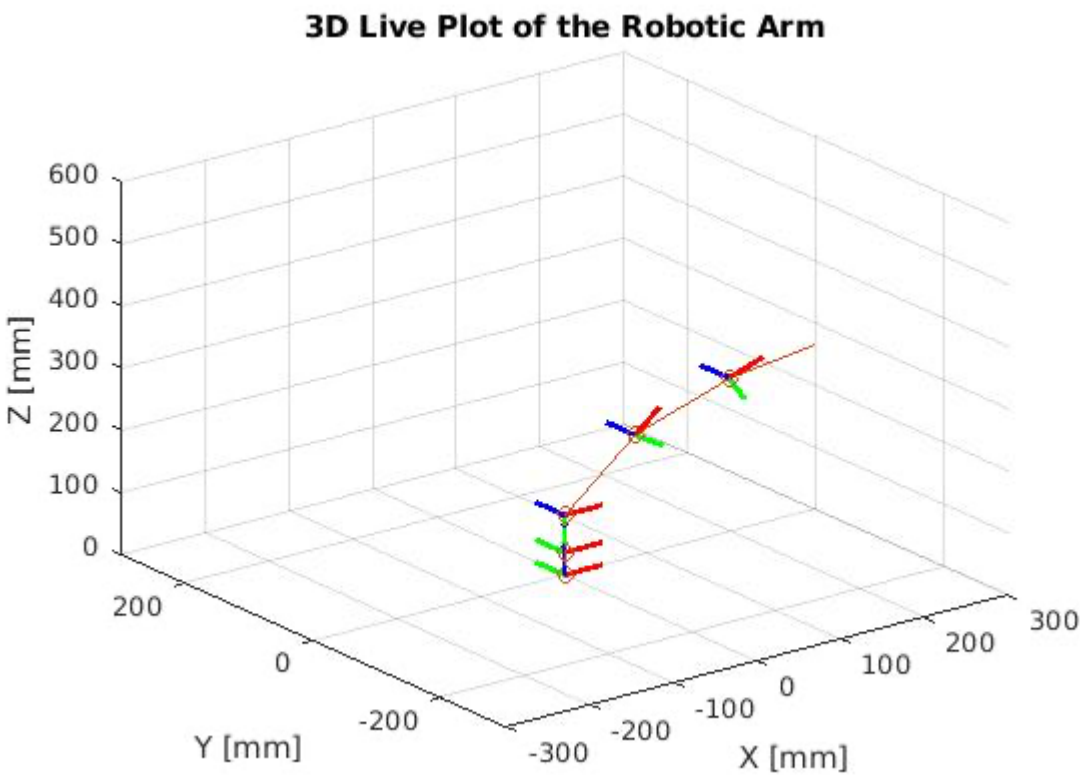

Figure 20: 3D Stick Model with joint angles [0, 30, -55, 15].

After making sure that the plot_arm() method was functioning properly, it was used to live plot the position of the robotic arm across 5 arbitrary positions for further verification that the 3D stick plot accurately modeled the robot arm. Figures 21 through 25 show the five positions of the robot arm and the corresponding 3D stick plots. The stick plot proved to be consistent with the physical position of the robot arm across all joint values and poses.

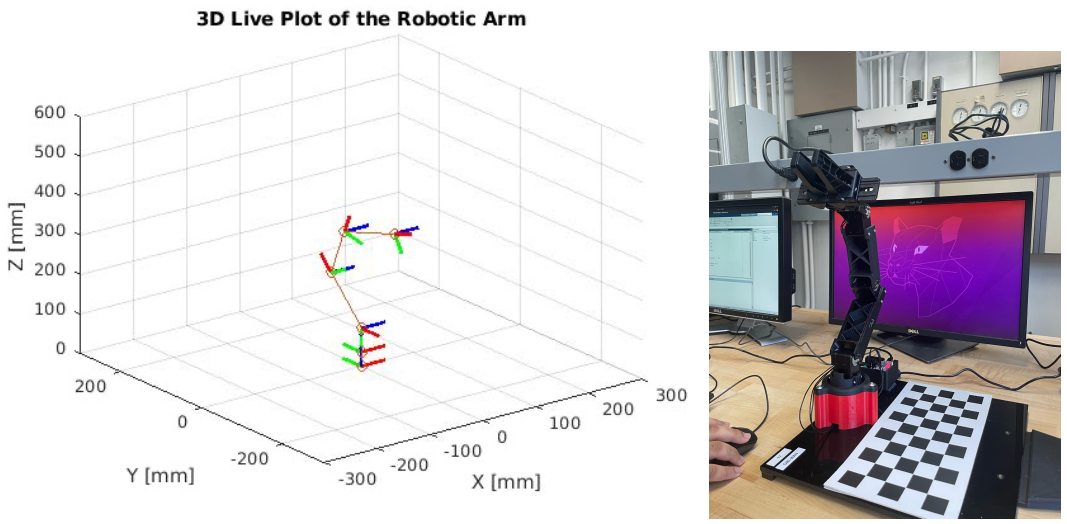

Figure 21: 3D Stick Model MATLAB plot and physical image of robot arm with joint angles [-90, -45, -30, 52].

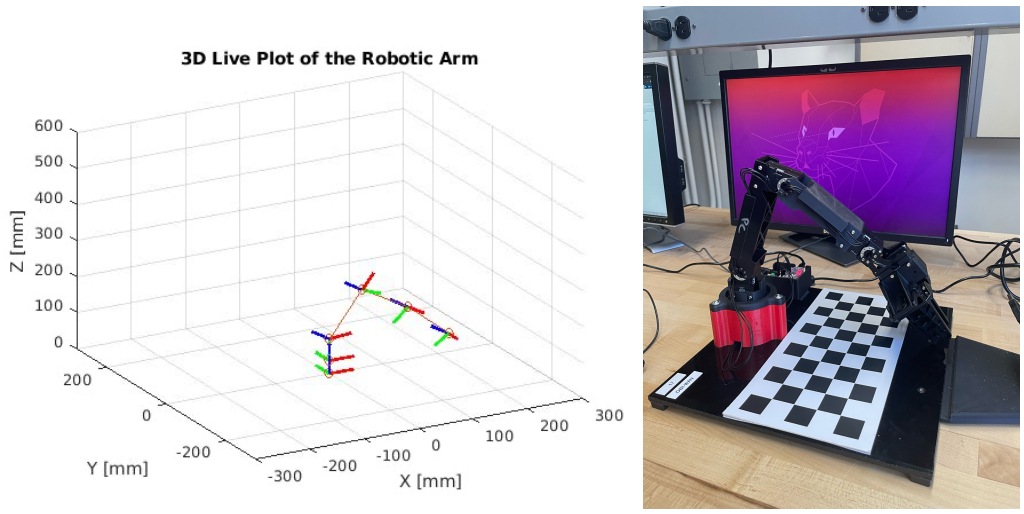

Figure 22: 3D Stick Model MATLAB plot and physical image of robot arm with joint angles [10, 20, 20, 10].

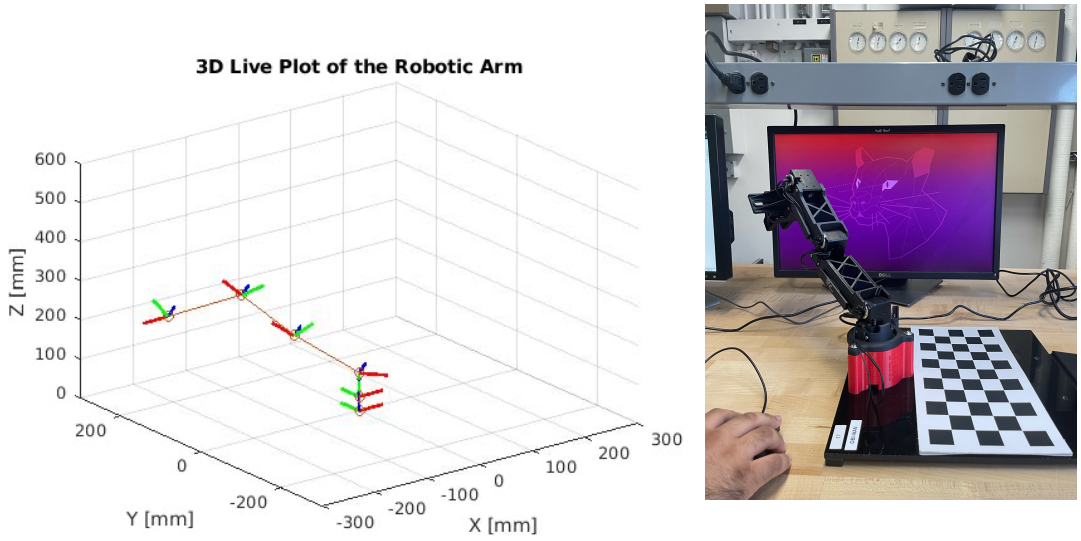

Figure 23: 3D Stick Model MATLAB plot and physical image of robot arm with joint angles [-50, -60, -70, -80].

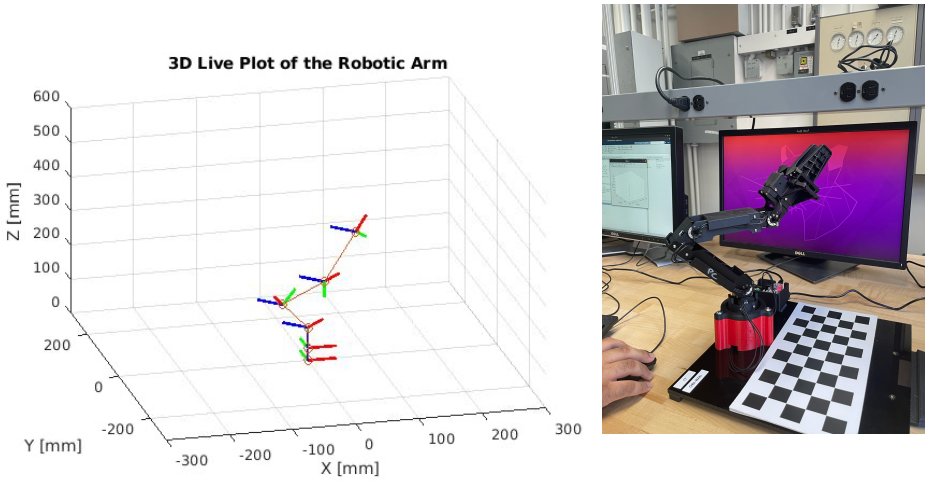

Figure 24: 3D Stick Model MATLAB plot and physical image of robot arm with joint angles [45, -45, 45, -45].

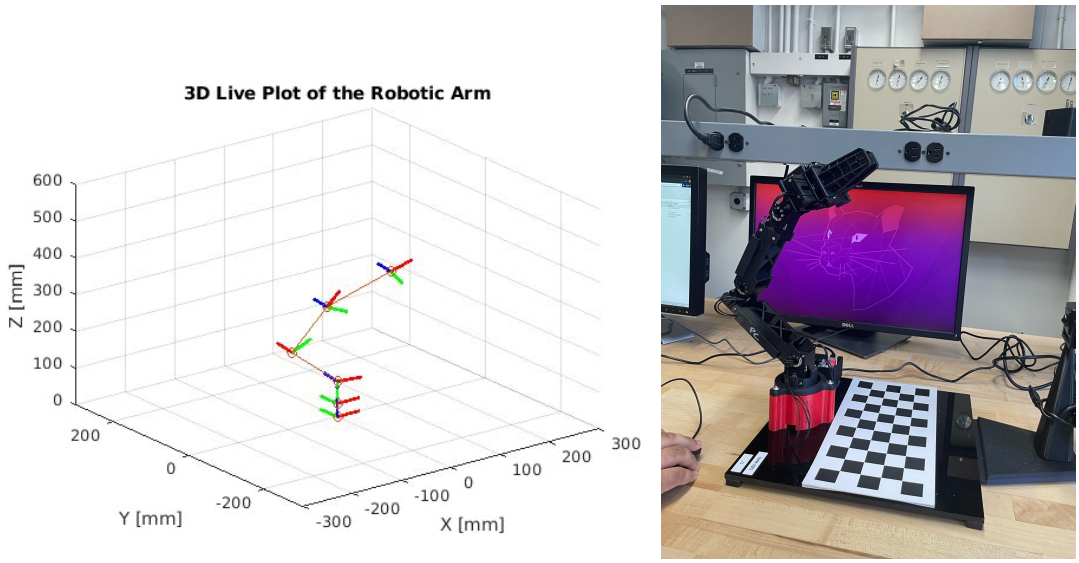

Figure 25: 3D Stick Model MATLAB plot and physical image of robot arm with joint angles [-10, -50, -10, 30].

Triangular Movement

While keeping the base angle at 0 degrees, a MATLAB script was written to move the robot arm sequentially through three vertices of a triangle in the X-Z plane while recording the joint angles and the end-effector's position.

Figure 26 shows the joint angle values plotted over time. Joint 2 (the blue line) experiences the most change over time and both joints 2 and 4 follow rather smooth and symmetrical curves over time. Figure 27 graphs the movement of the end effector in the X and Z directions over time. The graph shows that the end-effector started by moving along the X-axis, then started to return to the initial position after a peak at about 1.5 seconds before making another large movement. The Z-position followed a similar trajectory, peaking just after the X-axis value peaked, showing that the movement along the Z-axis was controlled in sync with the X-axis to form a smooth diagonal (and triangular as a result) path.

Figure 28 shows the triangular path of the end effector with each vertex of the triangle marked with a red circle. The triangle is pretty well formed with smooth sides, showing that the robot arm successfully was able to follow the triangular trajectory between the three points. The curvature on the edges suggests that there was a slight imbalance between the X and Z value timing of movements leading to a not perfectly straight line.

Finally, the drift in the Y-direction is shown in a graph in figure 29. The graph shows a slight increase in the Y position over time, even though no change is to be expected in the Y-direction since all of the arm movement was constrained to the XZ plane. There were some jumps in the Y position which could be a result of imperfections in the actuators or human error such as disrupting the robot arm in motion or the surface it was resting on.

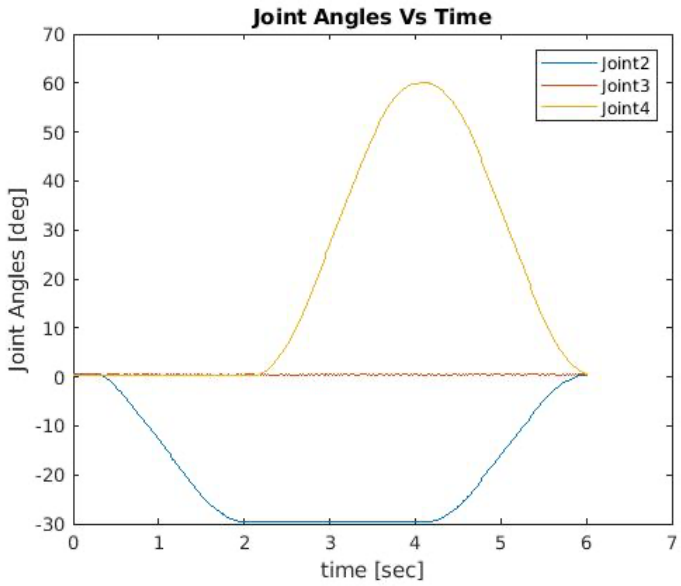

Figure 26: Plot showing the individual joint angles with time.

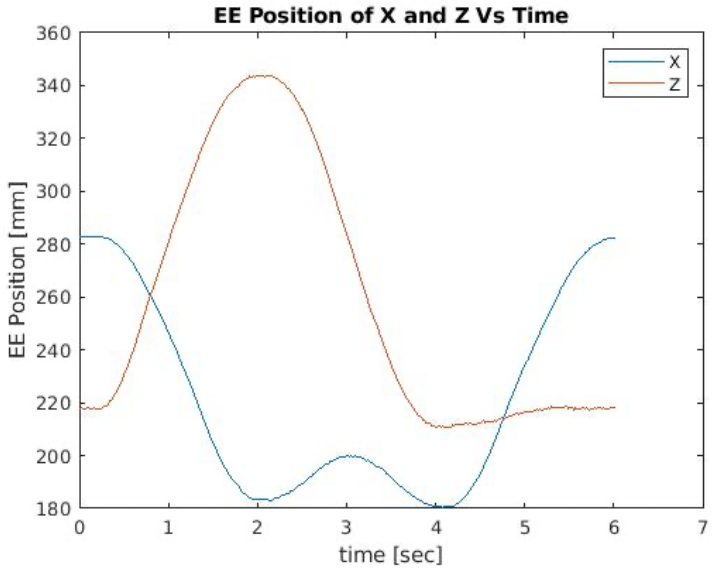

Figure 27: Plot showing the X and Z positions of the end-effector with time.

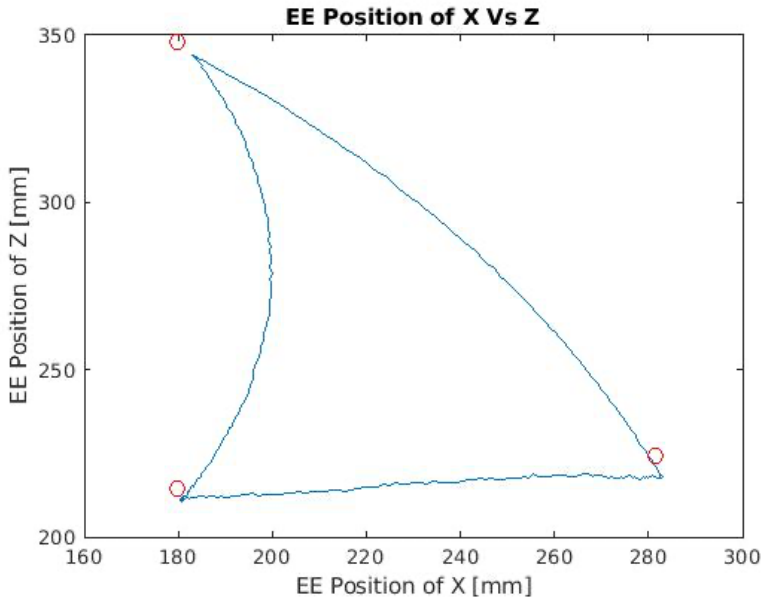

Figure 28: Plot showing the X and Z end-effector positions creating a triangle.

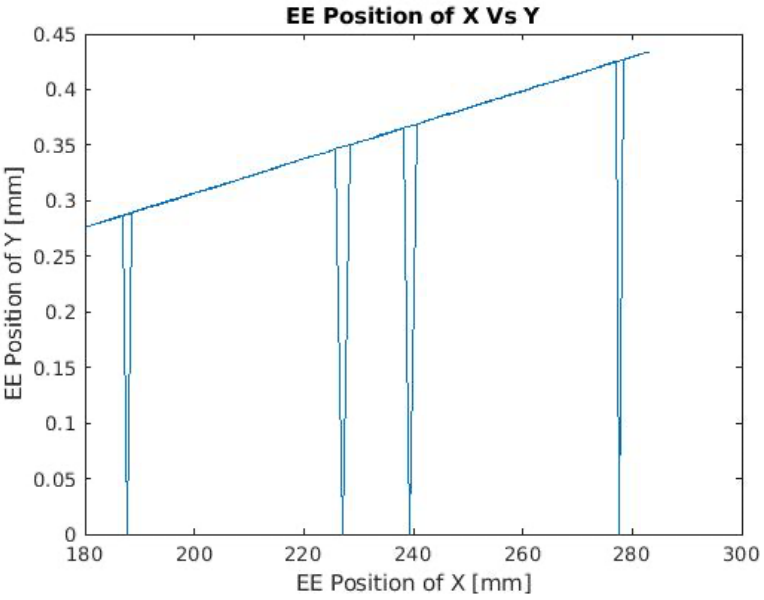

Figure 29: Plot showing the X and Y end-effector positions with some error.

Discussion

The results found in Lab 2 demonstrate successful calculation and implementation of the forward kinematics describing the 4 DOF robot arm and the MATLAB methods written to compute the transformation matrices. Each part of the lab as discussed in the methodology was an important part of the

process in computing, verifying and visualizing the forward kinematics of the robot and the key findings for each step are listed below.

DH Parameters and Forward Kinematics Implementation

- The first step of assigning frames and the dh parameters to the diagram of the robot was straightforward and went as expected. It was important to have a diagram to refer to when visualizing the robot and it also helped simplify the process of formulating the forward kinematics and writing the methods to compute the transformation matrices. The methods dh2mat and dh2fk were simple and easy to verify with the robot geometry as a result.
- The computed forward kinematics using the fk3001() function for different joint angles confirmed the accuracy of our FK implementation. The transformation matrices for the home position (Figure 10) and arbitrary joint configurations (Figures 11-13) helped verify that the function computed the transformation matrix and can generalize across different poses. The cp methods were simply an extension of the forward kinematics with a more specific interaction with the robot, which again in the home position helped to show the consistency and accuracy of the forward kinematic functions.

Error analysis and consistency

- When the robot arm was moved away from and returned to the home position, the slight discrepancies in the tip positions were observed and captured in the RMS error values. The average tip position, when compared to the ideal tip position, showed that the deviations were minimal (average tip position: [283.0766, -0.0865, 217.2887]). The RMS values ranged between 3.4543 mm and 4.7398 mm and the error in the 3D plot was also relatively small (Figure 17). These variations are likely caused by small mechanical imperfections or sensor inaccuracies in the joint actuators as seen in the error in Lab 1, but overall, the repeatability of the robot was quite reliable.
- The errors, while small, seemed random rather than systematic since there was no clear pattern to the points plotted. This suggests that the error is a result of slight inconsistencies in the hardware and could be helped by calibration but will likely never be fully resolved. This could be important to note going forward and when working backwards using inverse kinematics to control the robot arm location.

3D Stick Model and forward kinematic visualization

- The 3D stick model programmed in MATLAB successfully represented the physical position of the robot arm in 3D space. Figures 18-25 show the stick plot matched with the physical position of the robot arm and show that the stick model accurately matched the physical robot pose in each joint angle configuration. This real-time visualization served as an effective validation tool for the FK calculations, ensuring that the computed transformation matrices were correct.
- The model also allowed for clear observation of the robot's movement, which highlights the effectiveness and reliability of the MATLAB plot_arm() method to visualize the robot's joint configurations and positions in real time.

Triangular Movement

- Figure 26 clearly shows the motion of the triangle. Initially, joint 2 goes from the 0-degree angle to –30-degree angle. Then, joint 4 goes from the 0-degree angle to 60-degree angle. Finally, both of the joints return to 0 degrees. This causes the formation of the triangle. However, there are curvatures in the triangle which are shown in Figure 28, which suggest imperfections in the arm motion since straight lines between the vertices of the triangle would be expected. These curvatures are further seen in Figure 27 where changes to a joint angle 2, 3 and 4 causes both the X and Z values of the end-effector's position to change.
- In figure 28, the vertices deviate from the “actual” vertices due to random error present within the robotic arm as discussed and explored in the Error analysis and consistency section.
- The robot arm successfully followed the triangular path in the XZ plane with smooth movements throughout each motion. While the overall deviations were small, there were curvatures in the path of the end effector between the vertices and some slight drifting in the Y position. These deviations could be minimized with further refinement in trajectory planning or control algorithms, potentially by optimizing the timing between joint movements to ensure more precise transitions between points.

Conclusion

This lab demonstrated the use of forward kinematics for a 4-DOF robotic arm using matlab. The functions developed allow accurate computation of the OpenManipulator-X robot arms end effector position and orientation, providing a foundation for future labs and Future uses of Forward kinematics functions. This lays a foundation for more advanced kinematic styles and techniques and complex dynamic models that may be used in the future.

Three methods were created for forward kinematics. The first method was dh2mat() which allowed the transformation matrix for each link to be calculated symbolically. The second method was dh2fk() which allowed the transformation matrix using the whole DH table to be calculated symbolically. The final method was fk3001() which allowed the joint angles to be inputted to get the numeric transformation matrix. These methods were used by other methods in order to interface forward kinematics with the current data from the joint values by the measured_cp(), setpoint_cp() and goal_cp() methods.

A 3D stick model was created and used to better visualize the movement and positioning of the robot arm in space and the motion of the robot arm across a specific plane of motion was tested and plotted. The testing and visualization of the end effector and joint positioning is necessary insight into the architecture and behavior or the robot arm and will be useful tools alongside the forward kinematic functions to aid in future trajectory planning and inverse kinematic calculations for solving more complex problems and utilizing the full workspace and capabilities of the robot arm.