Worcester Polytechnic Institute

Robotics Engineering Program

LAB 2:SYSTEM ARCHITECTURE & JOINT SPACE CONTROL

Submitted By

Yichen Guo

Yuhan Wu

Haojun Feng

Date Submitted : 09 / 24 / 2021
Date Completed: 09 / 24 / 2021
Course Instructor: Prof. Mahdi Agheli
Lab Section: RBE 3001 A21

Abstract

In this report, we will use inverse kinematics and trajectory planning techniques to generate the actual path of the robot arm, and visualize it in MATLAB.

Introduction

In Lab 1, we communicated with the robot by sending commands to the joints through a MATLAB code, then in Lab 2, we used forward kinematics (FK) to calculate the robot's position in the task space given the joint variables, where the FK method in Robot class takes joint angles as an input; implemented real-time stick model visualization in MATLAB; and had the arm move to vertices of a triangle.

For this Lab, we will use trajectory generation techniques in joint space to generate a smooth trajectory from one vertex to another, so in order to achieve that goal, we will work on implementing inverse kinematics (IK). The IK method takes a position vector of the end effector in the task space to determine the corresponding joint angles. By sending those joint angles to the arm, the end-effector should move to the desired position.

Moreover, we will integrate IK with trajectory generation techniques to implement a smooth motion between set points in task space. Since the robot only has 3 degrees of freedom, we will focus on making the robot go to a given position in the task space, represented as (x, y, z) value, but not set an arbitrary orientation since the robot does not have sufficient degree of freedom to set both position and orientation.

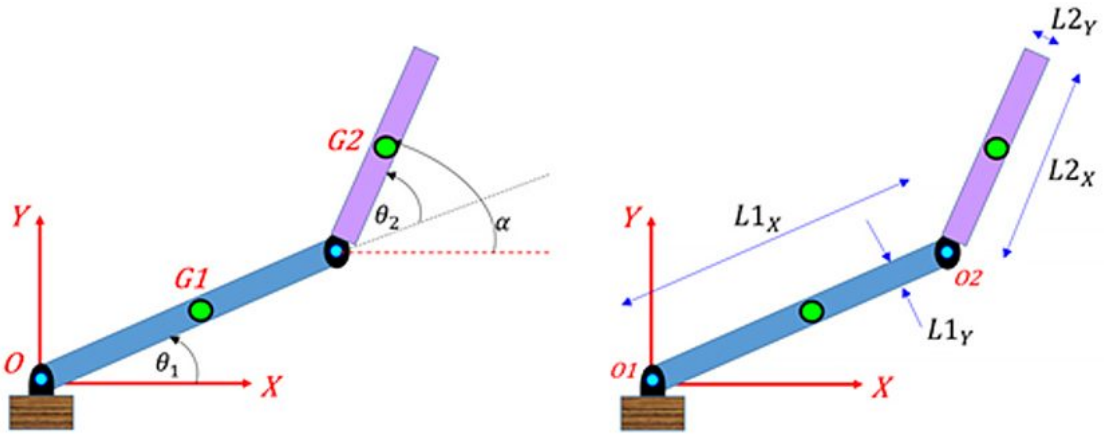


Figure.0: irrelevant decoration

Methodology

1. Formulate the inverse kinematics (IK) of the robot arm and verify it with forward kinematics (FK) if possible.

2. Create MATLAB scripts and functions to implement your IK calculations while interacting with the robot. Create ik3001 method, which takes a 3x1 task space position vector (i.e. x, y, z components of vector p that is the robot end-effector's position w.r.t the base frame) as the input and returns a set of corresponding joint angles (i.e. q1, q2, q3) that would make the robot's end-effector move to that target position.

3. Perform trajectory planning in joint space using cubic polynomials.

4. Perform trajectory planning in task space using linear interpolation and cubic polynomial.

5. Perform trajectory planning with a quintic polynomial.

6. Visualize and characterize joint-space and task-space level motion trajectories.

7. Extra: Plot 3D shape: pyramid

Results

Questions to address:

- **Calculate the inverse kinematics of the 3-DOF robot arm**

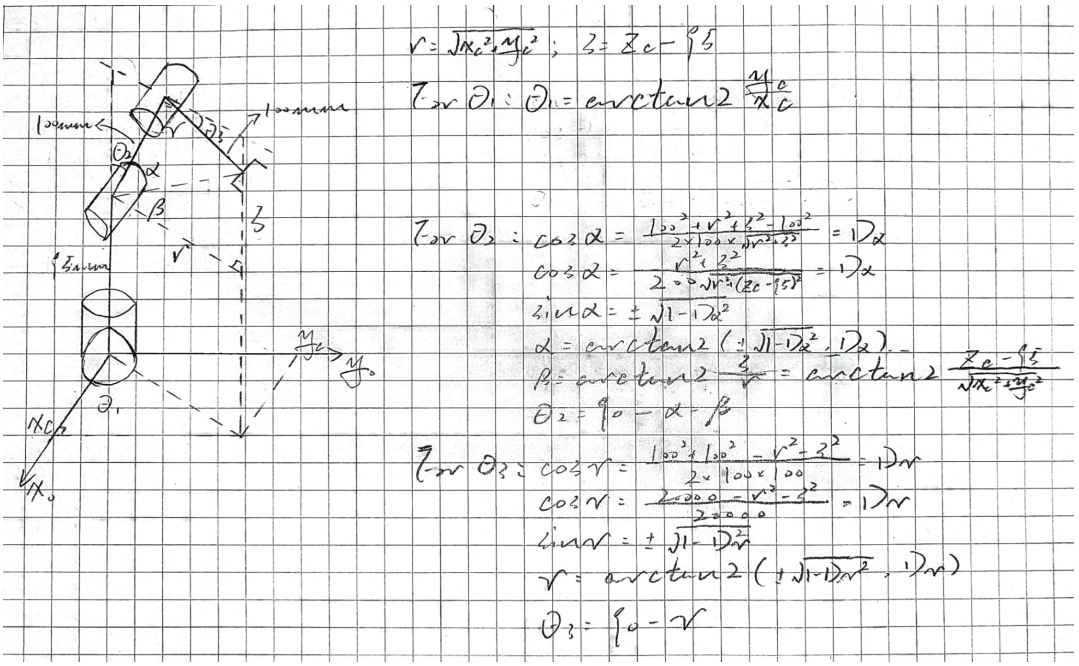


Figure.1: manual solution for inverse kinematics

- **Implement the inverse kinematics in MATLAB.**

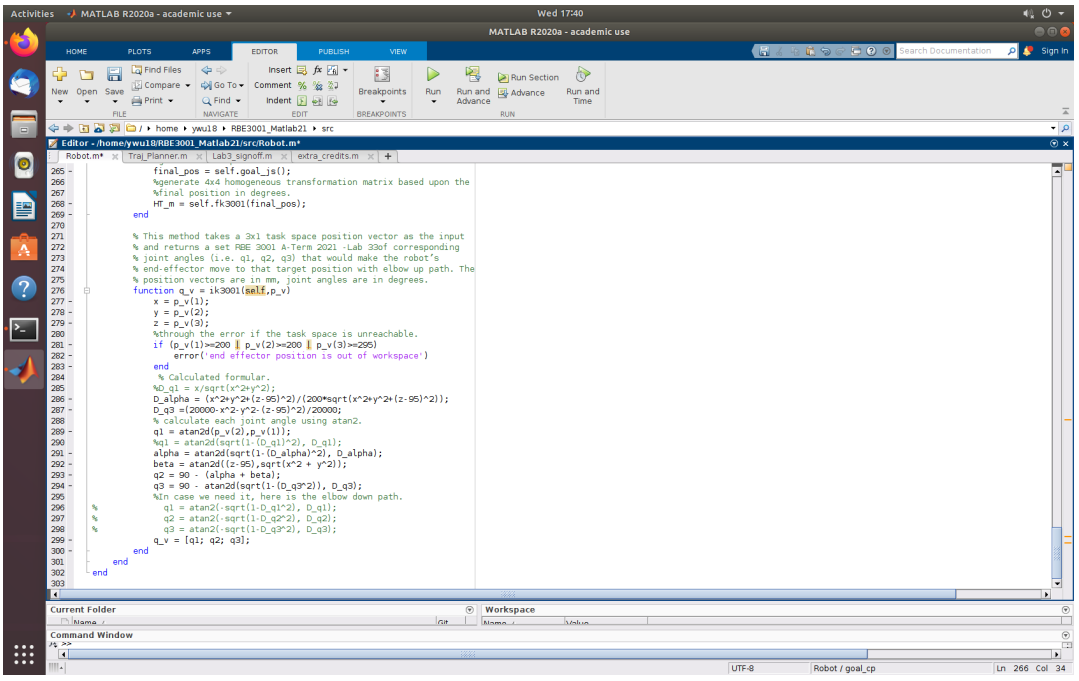


Figure.2: matlab inverse kinematics calculation

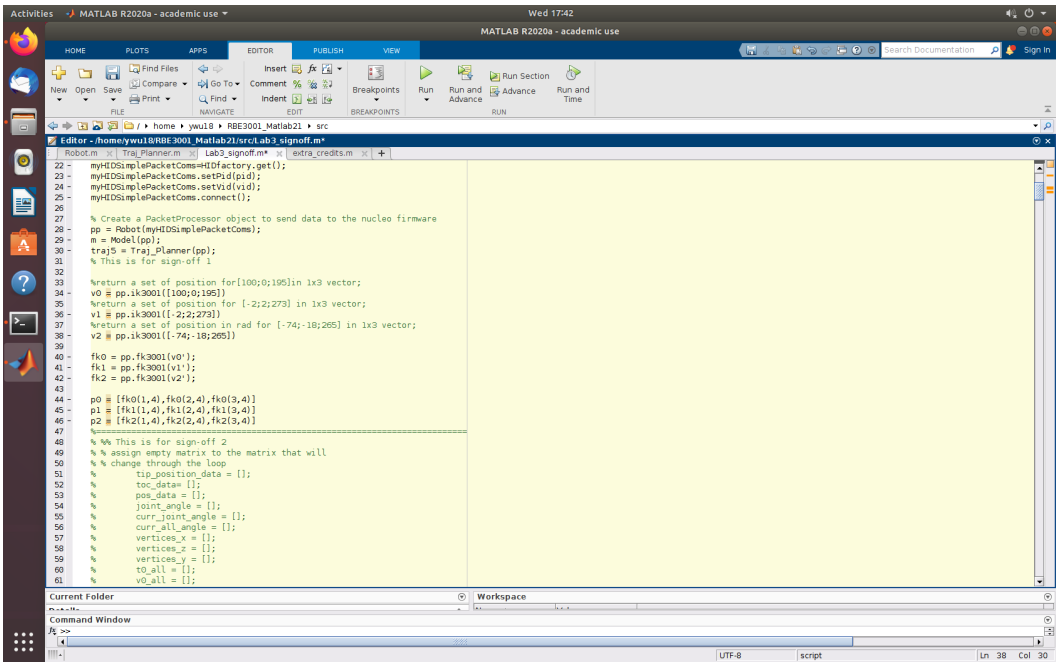


Figure.3: matlab inverse kinematics method

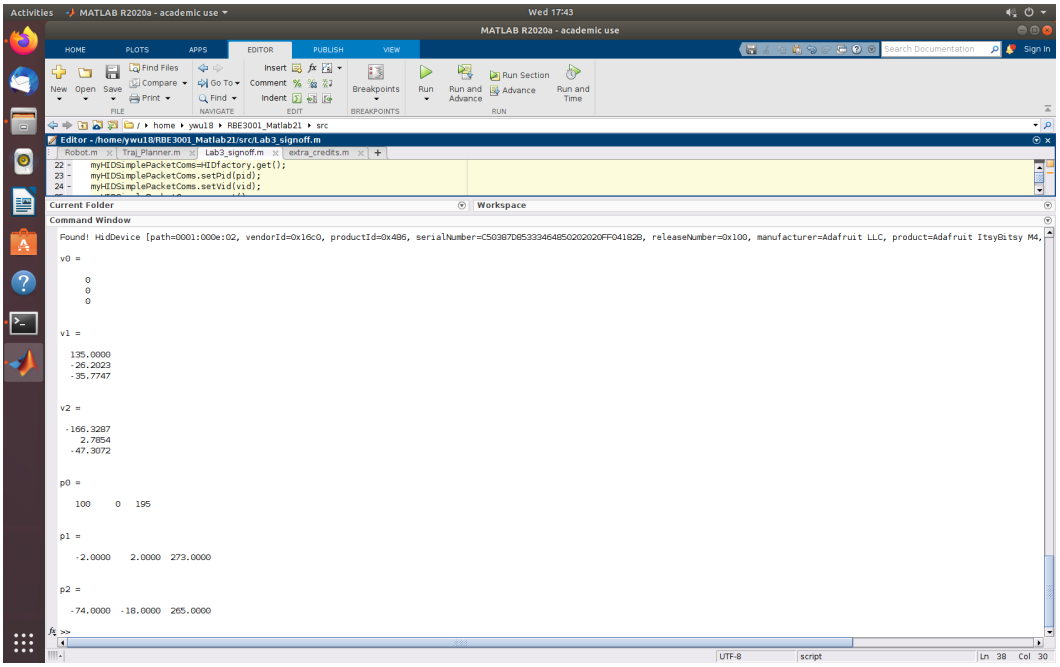
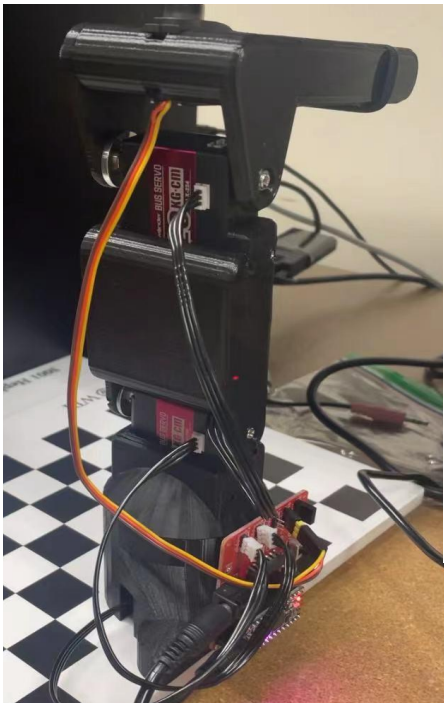
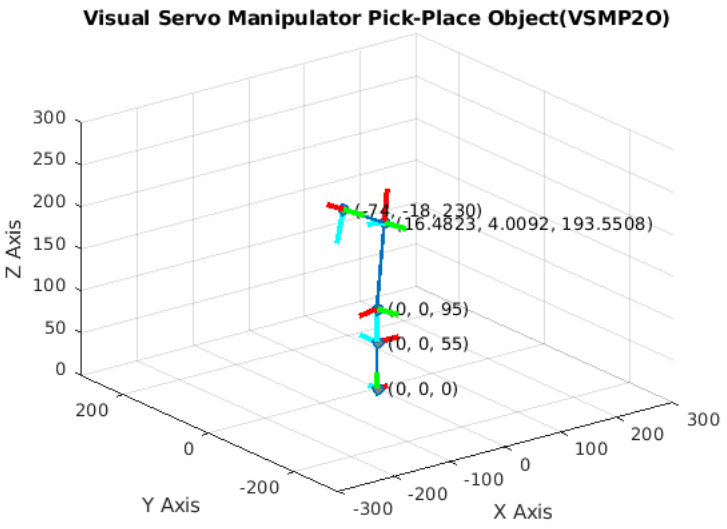


Figure.4: matlab result

Since the demo code had already handled the quadrant problem, we do not need to consider the positive or negative square root answers in our equation. Figure.4 is our final solution given by MATLAB and we checked that with our manual solution, obviously, they correspond to each other.

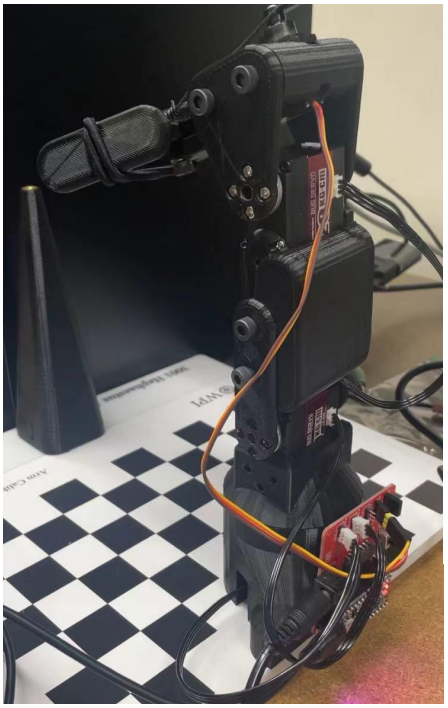
- **Validate your IK solution with the robot**

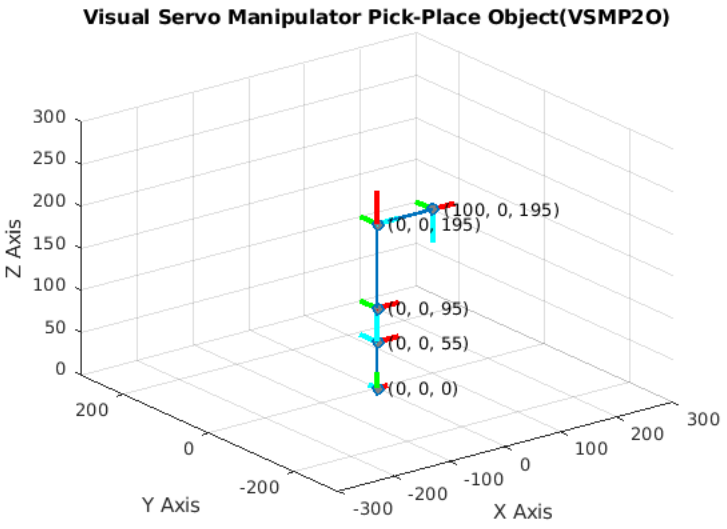




↑ Figure.5: MATLAB graph of the first configuration

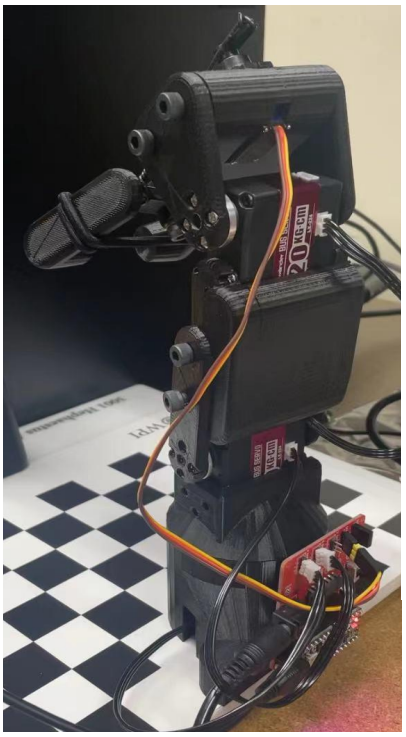
← Figure.6: Real life picture of the first configuration

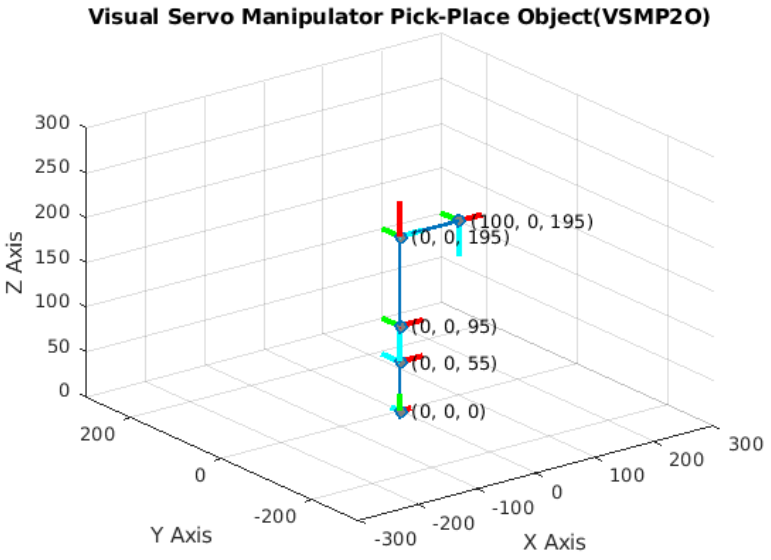




↑ Figure.7: MATLAB graph of the second configuration

← Figure.8: Real life picture of the second configuration





↑ Figure.9: MATLAB graph of the third configuration

← Figure.10: Real life picture of the third configuration

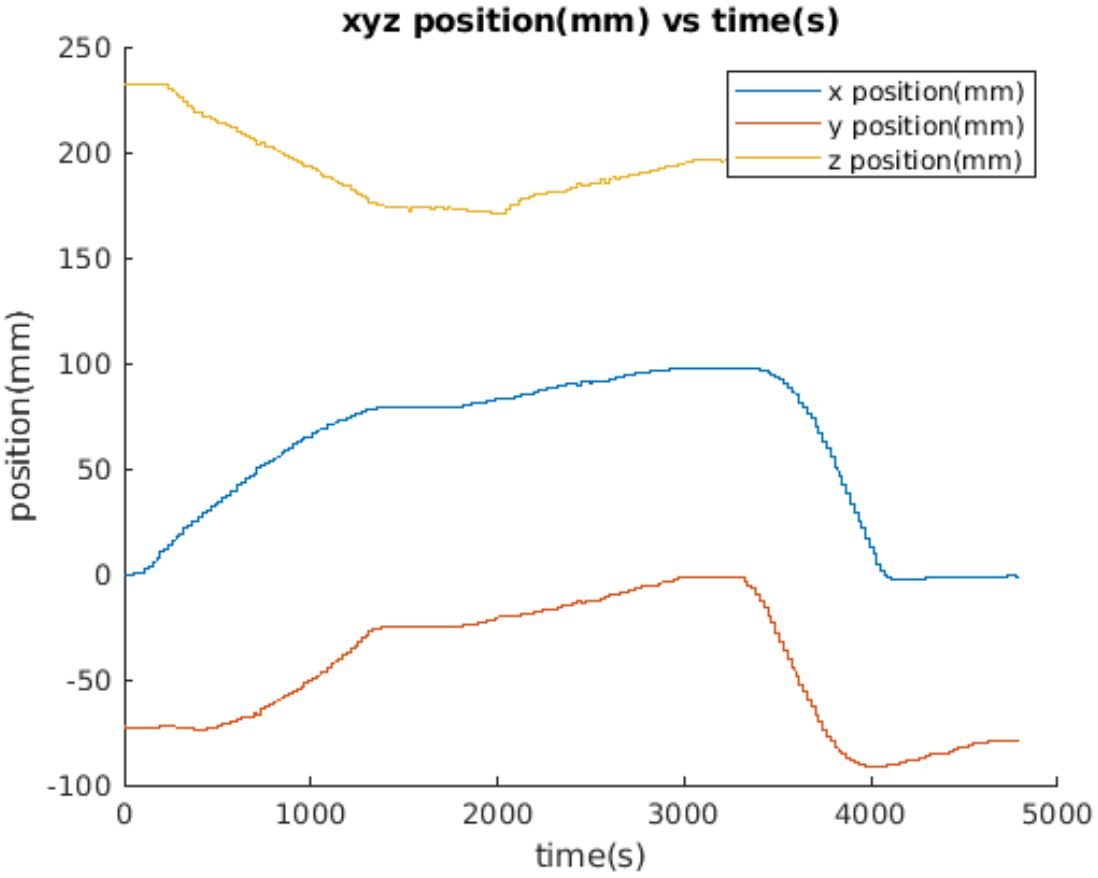


Figure.11: xyz position graph of the same point

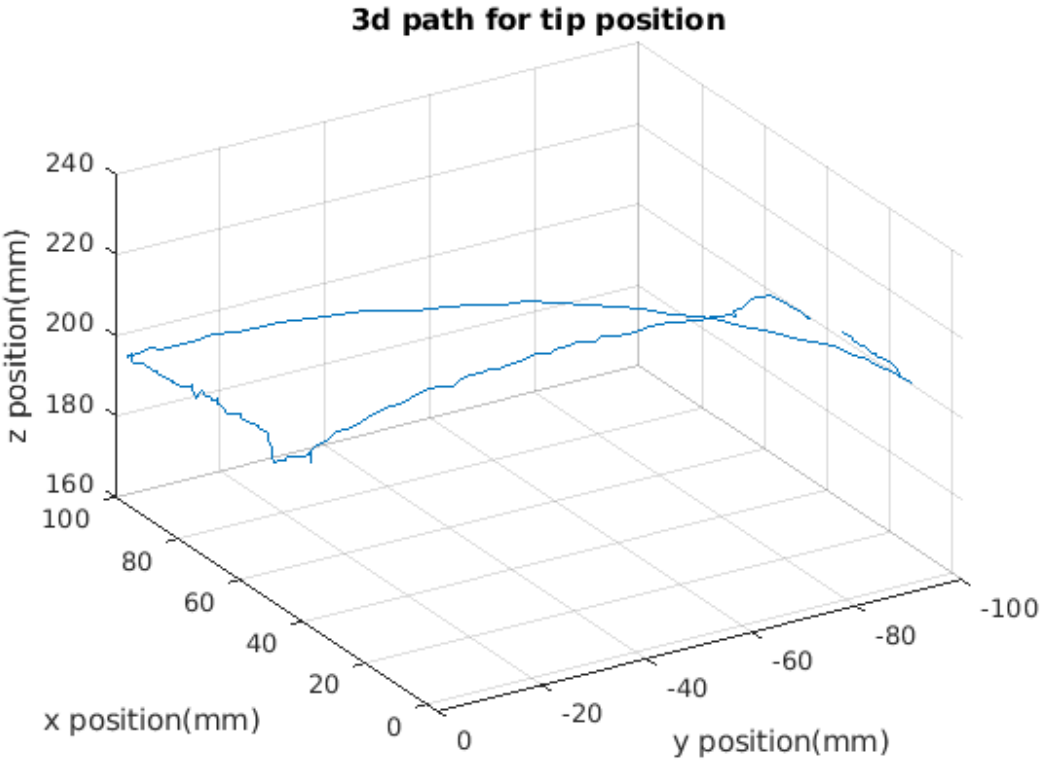


Figure.12: 3D path graph of the same point

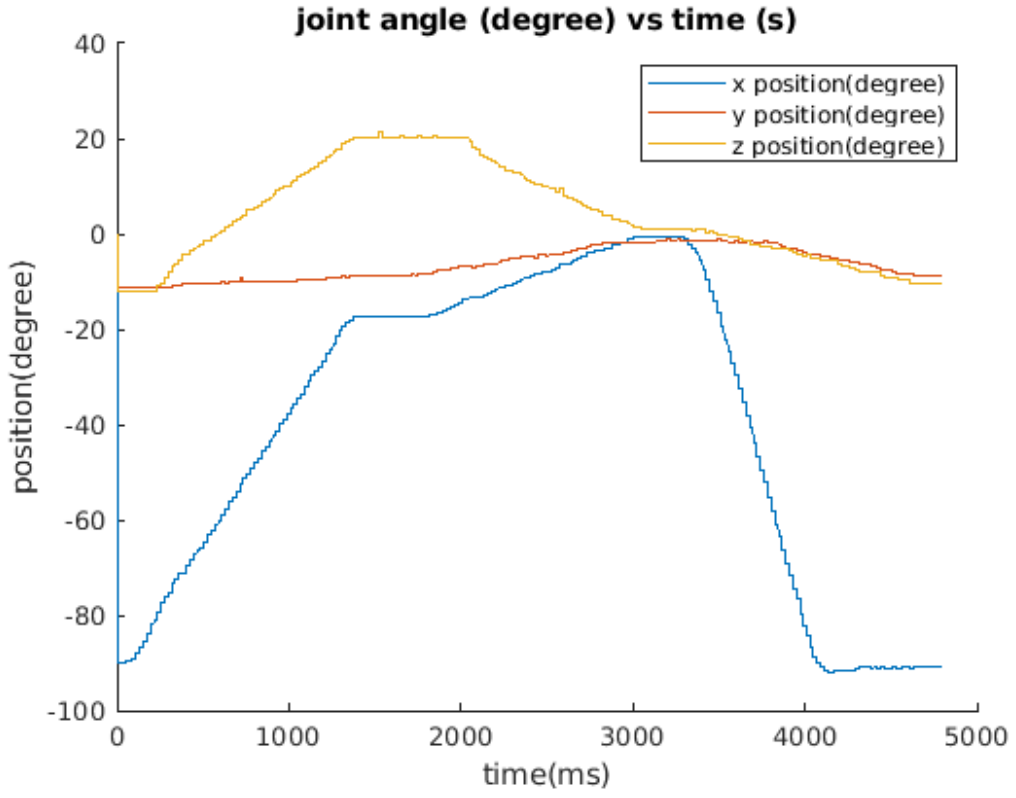
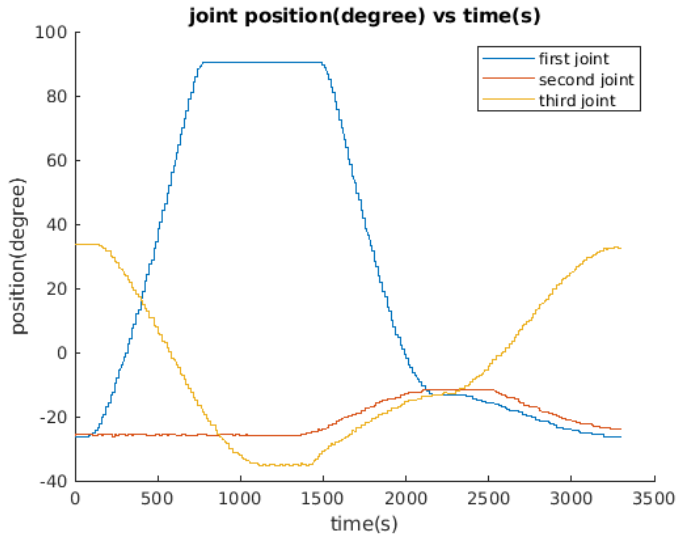
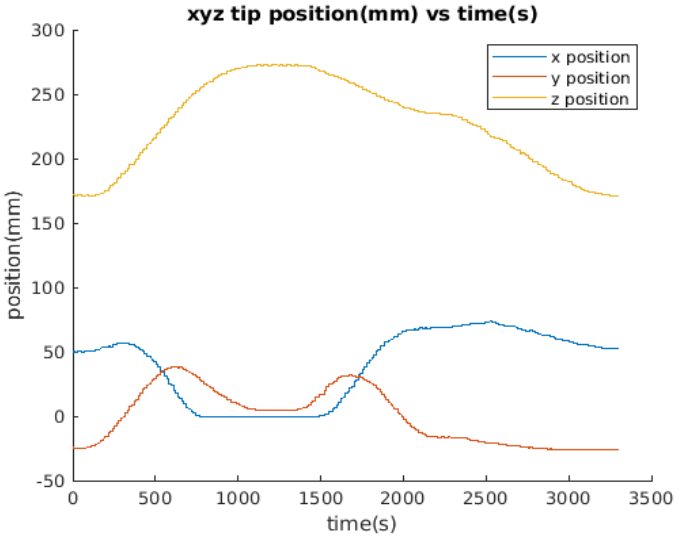


Figure.13: x, y, z position graph of the same point

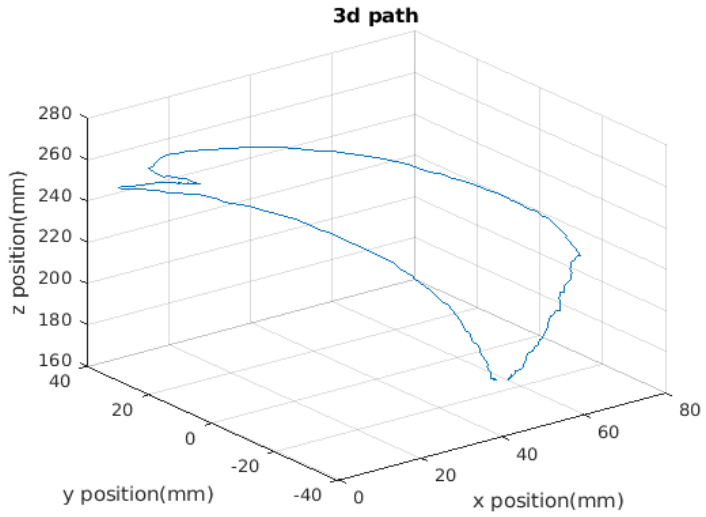
- **Trajectory planning in Joint Space**



←Figure.14: joint motion of each joint



→Figure.15: task space motion of each joint



←Figure.16: 3D plot in task space

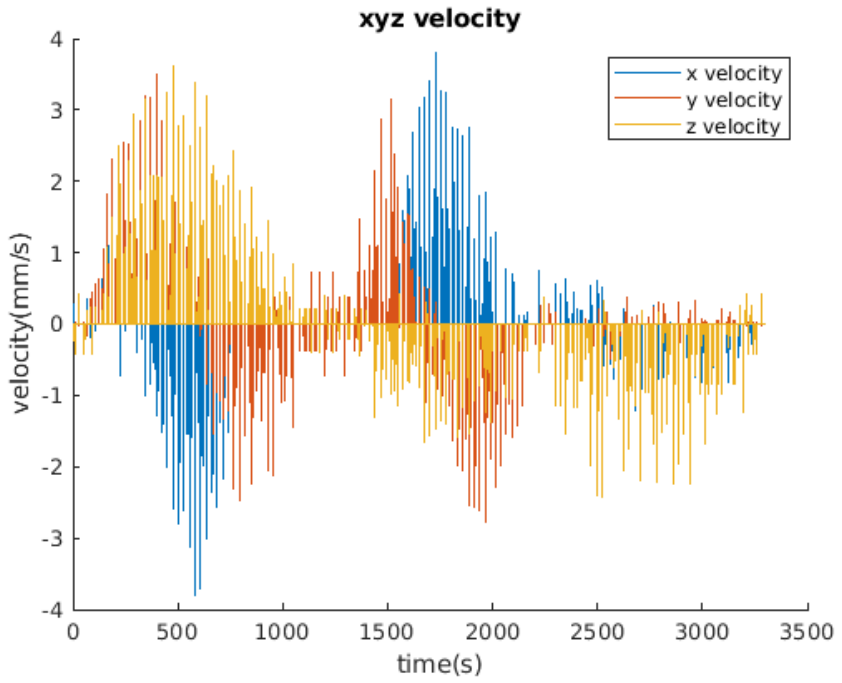


Figure.17: x, y, z position velocity

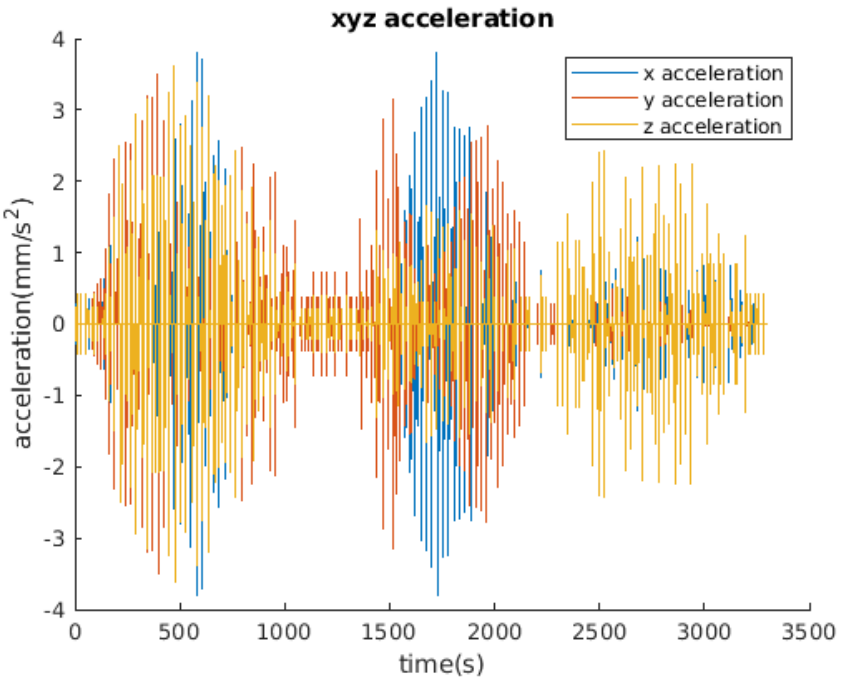


Figure.18: x, y, z position acceleration

We took the derivative of position of all the x, y, z positions with respect to time to get the velocity, then took the derivative velocity again with respect to time to get acceleration.

- **Trajectory planning in Task Space**

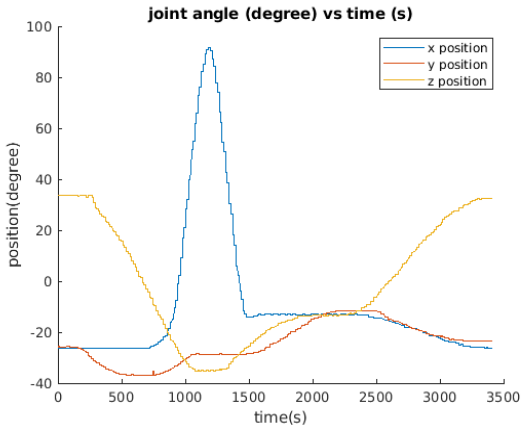
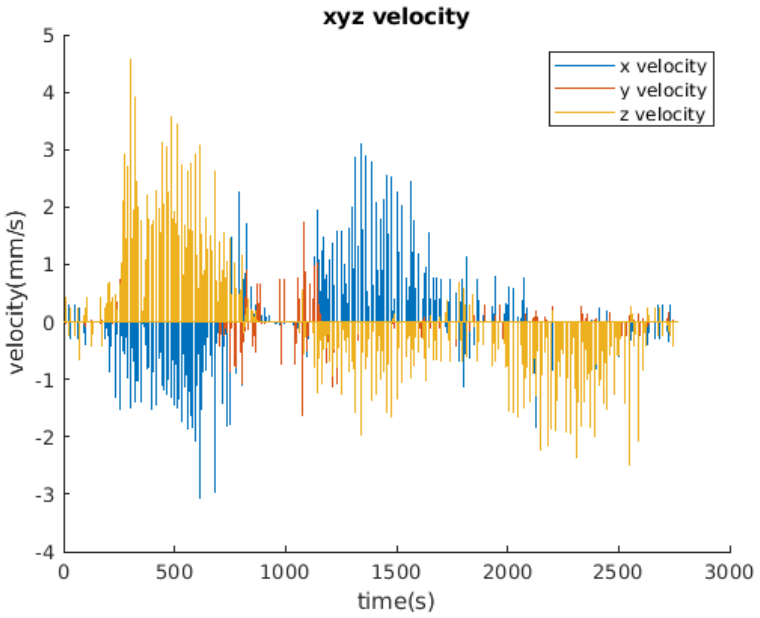
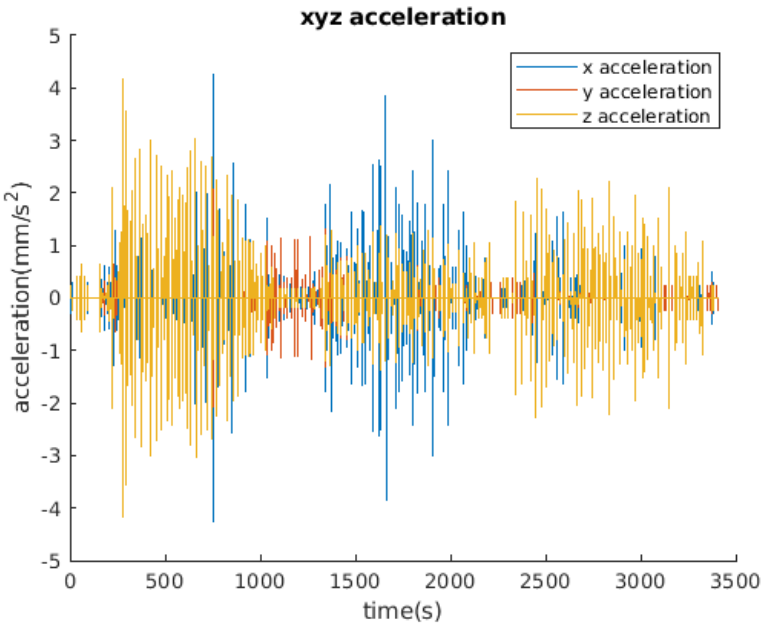


Figure.19: joint motion of each joint

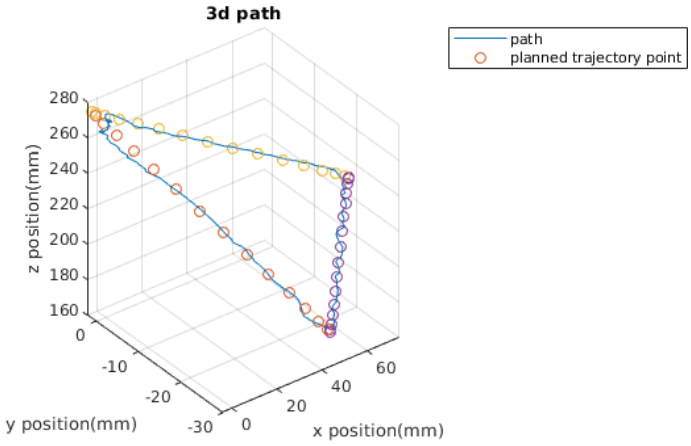
Figure.20: x, y, z position value over time



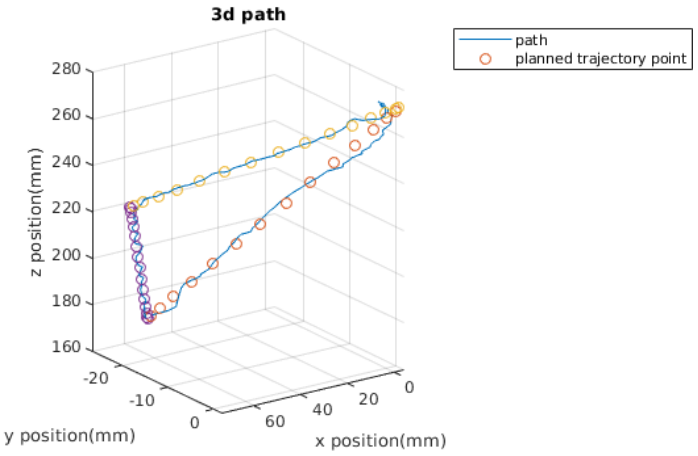
←Figure.21: x, y, z velocity over time



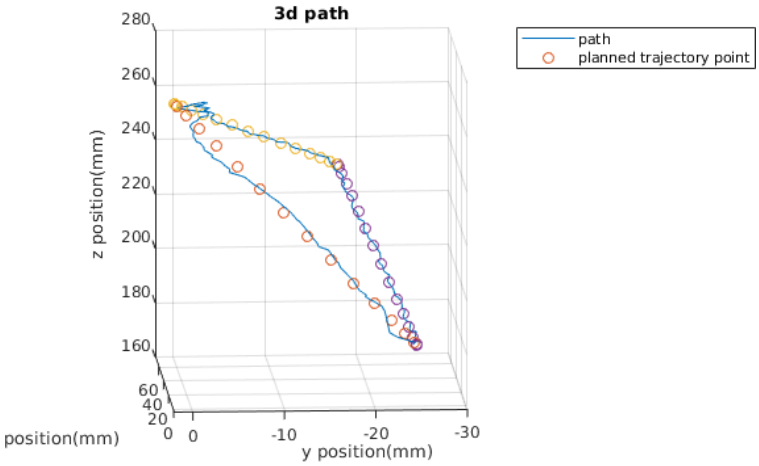
→Figure.22: x, y, z acceleration over time



←Figure.23: perpendicular view



→Figure.24: random view 1



←Figure.25: random view 2

- **Quintic trajectory planning in Task Space**

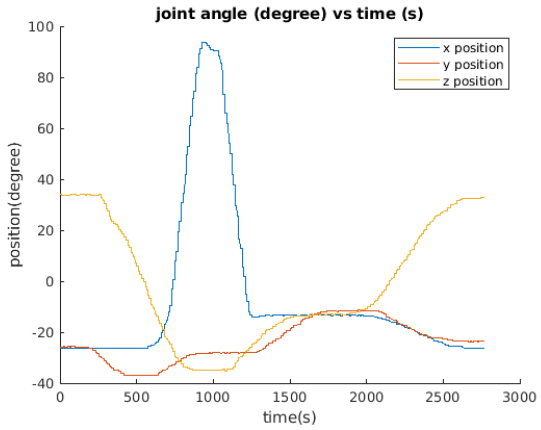
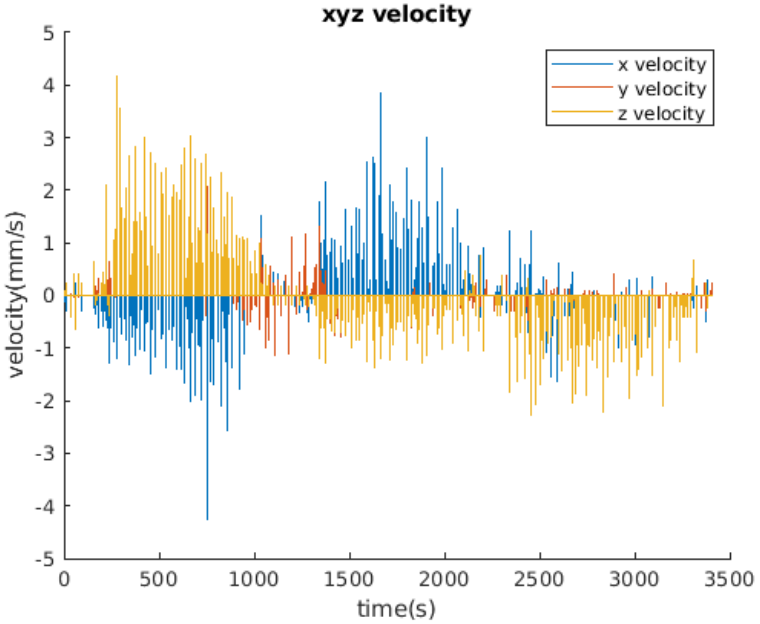
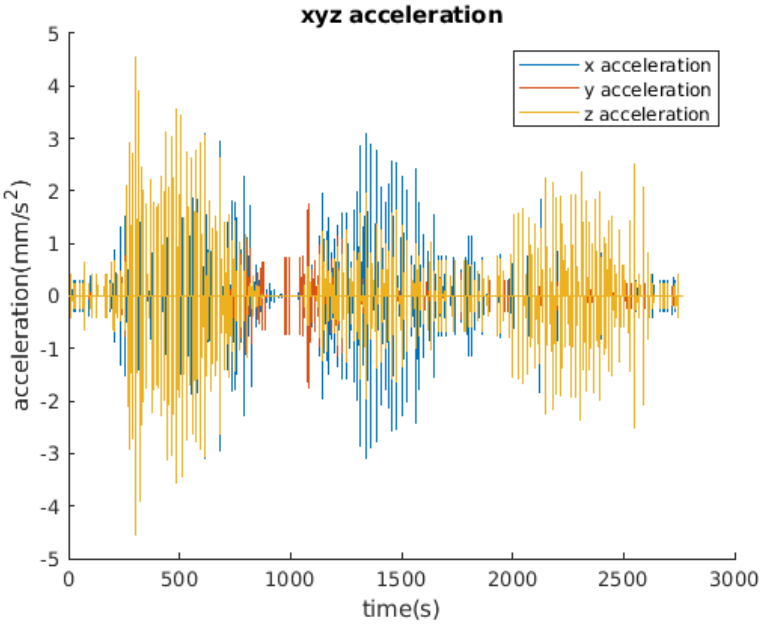


Figure.26: joint motion of each joint

Figure.27: x, y, z position value over time



←Figure.28: x, y, z velocity over time



→Figure.29: x, y, z acceleration over time

- Extra: Plot 3D shape: pyramid

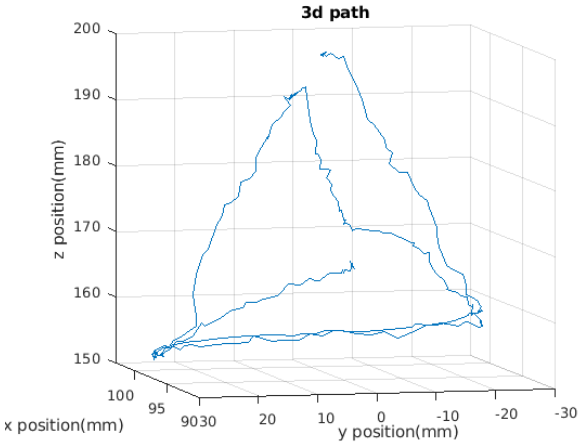


Figure.30: pyramid 3D plot 1

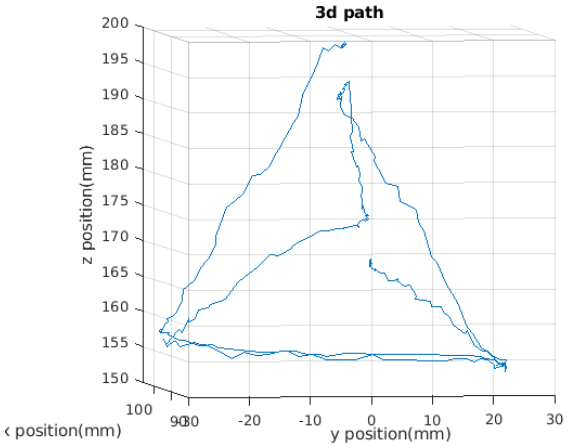
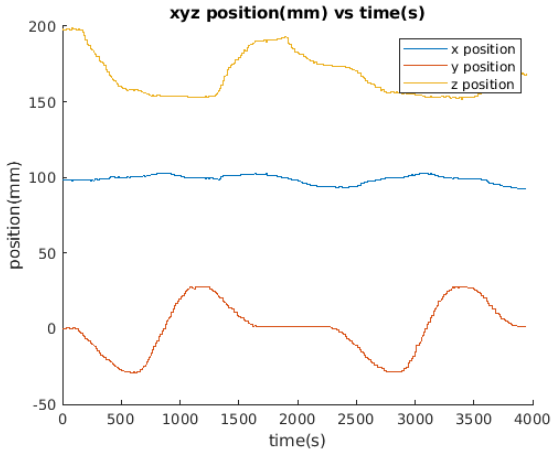
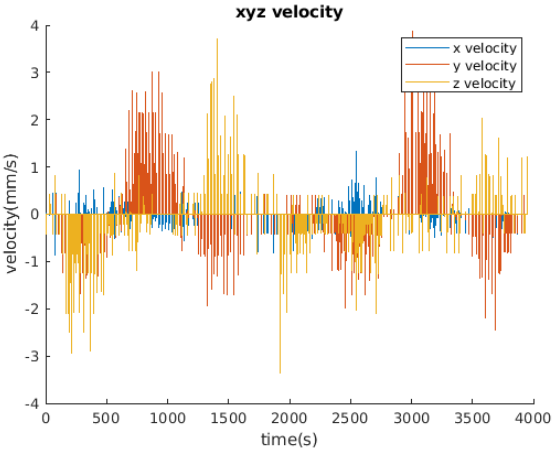


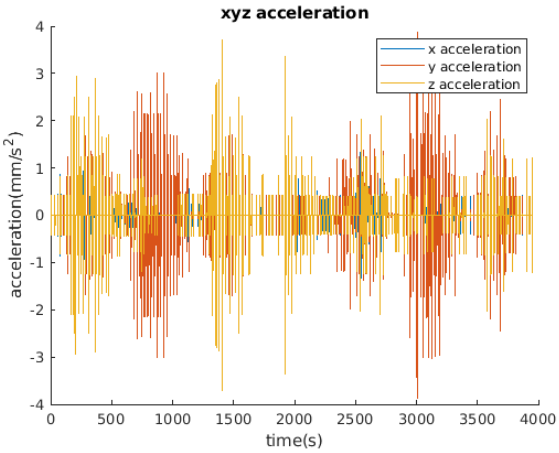
Figure.31: pyramid 3D plot 2



←Figure.32: x, y, z position value over time

→Figure.33: x, y, z velocity value over time





←Figure.34: x, y, z acceleration value over time

Discussion

- **Calculate the inverse kinematics of the 3-DOF robot arm / Implement the inverse kinematics in MATLAB.**

For our manual calculation, we chose geometric method instead of algebraic method, because it is more convenient and approachable on the graph, at least for our team. We can label everything on the graph and there are no prismatic joints so that we can use the cosine rules to solve them. After solving for the equations on the paper, we directly typed in them into MATLAB to help 'ik3001' to solve for the inverse kinematics.

- **Validate your IK solution with the robot**

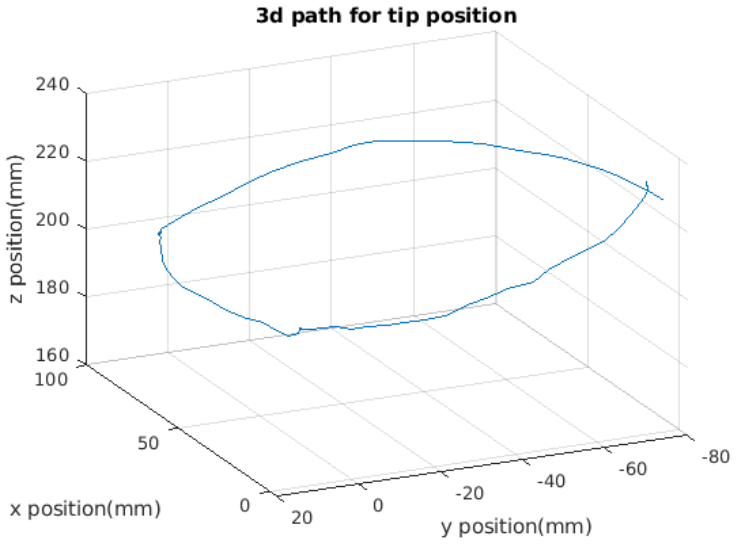


Figure.35: 3D path graph with non-interpolate method

Compare Figure.12 and Figure.35, we can find out that when implementing interpolation on motor control, there will be small fluctuations during the path, that is to say, the path is smoother for non-interpolate method, but the path is more accurate than non-interpolate methods. Fluctuations are due to the relatively long interpolation time we gave, but they still exist though we set longer interpolation time, because while using the interpolation method, the motor will vibrate more so that it works more accurately to the destination.

- **Create a trajectory generator function / Trajectory planning in Joint Space**

Compare what we have in part 3 and 5, we can find that the trajectory we have in part 5 is much more smooth, but a little curvy than what we have in part 3. The interpolation also causes fluctuations, but since we pick a smaller interpolation time, this time the fluctuation only occurs when the trajectory is short.

- **Trajectory planning in Task Space**

Compared with the previous part, we are able to tell that when planning trajectory in task space, the path is much more straight, because we directly define the path with displacement of each joint, without bothering to convert the values from joint values.
As for the 3D plot, we have an oblique triangle, so in the perpendicular view, the triangle is nearly perfectly flat, but in the side view, the triangle looks twisted a little bit. This is due to the observing reference frame. When observing in the perpendicular view, we only care about the x, y plane, but when in the side view, we care about the x, z plane.

- **Quintic trajectory planning in Task Space**

For cubic trajectory planning, there are fewer terms, so that the whole system is less accurate than the quintic trajectory planning when implementing smooth motion. In Figure.20 and Figure.27, we can find out that quintic trajectory corresponds to the ideal trajectory more and in Figure.21, 22, 28 and 29, we can find that there is less error for quintic polynomials. However, due to complexity, quintic polynomials are seldom used.

- **Extra: Plot 3D shape: pyramid**

For the sake of convenience, we used the cubic trajectory method to draw a pyramid, but since we have a short path for the sides, there are relatively many fluctuations on the graph, and it could be solved to reduce the interpolation time as well.

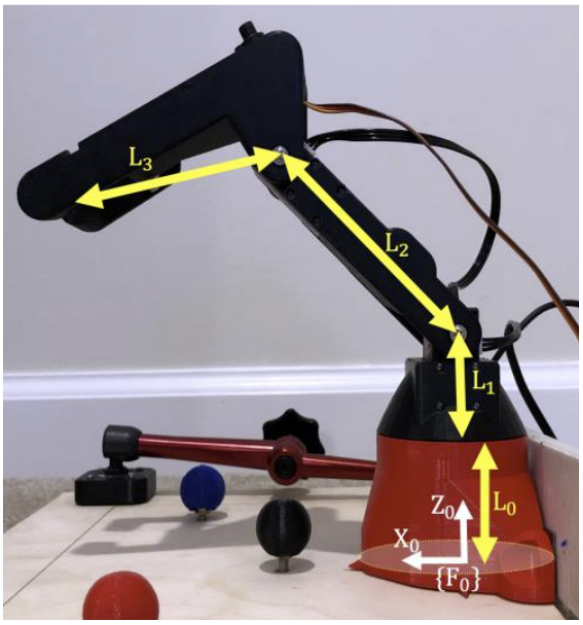
Conclusion

In this lab we successfully use MATLAB to verify inverse kinematics based on the end points we entered with various functions. Then we successfully turned these matrices into real-time plots, by representing our robot arm with a few sticks. In order to find out if our system was accurate enough, we ran a few tests and the results turned out to be pleasing.

After that, we compared the trajectory planning in the joint space and task space with both cubic and quintic trajectory planning polynomials and we found that when converting data between these two spaces, the motion of the robot arm is not affected a lot, but will end up with graphs with small differences. Also, both cubic and quintic polynomials are accurate enough to generate a smooth path, but due to the complexity, we prefer cubic polynomials more.

Luckily, we still had some time after we finished every essential sign-off, so we got our hands on the workspace pyramid 3D plot, and it correctly showed the cubic polynomials' accuracy.

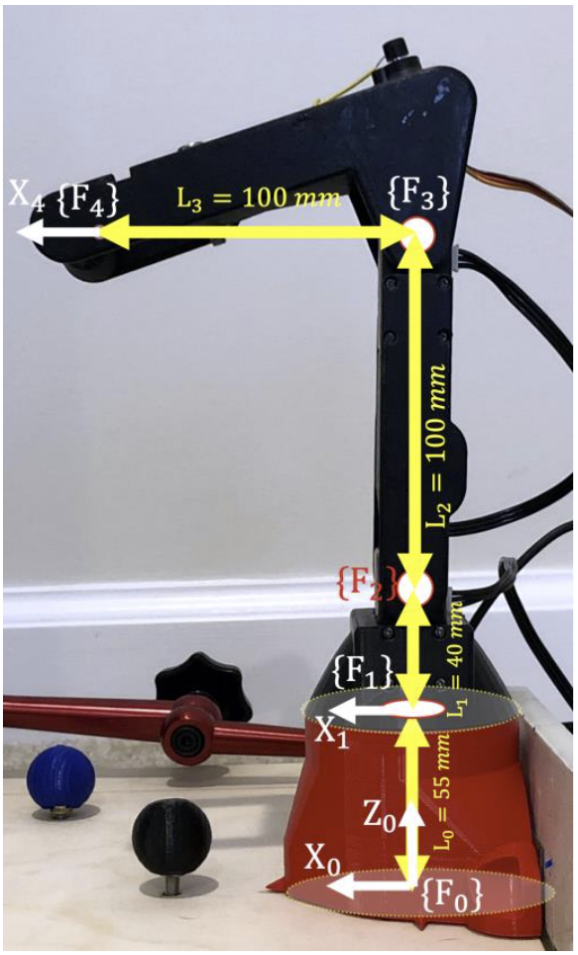
Appendix A: ARM CONFIGURATION



Left is the view of the arm showing the base frame {F0}. The link lengths are as follows:

- L0 = 55mm
- L1 = 40 mm
- L2 = 100 mm
- L3 = 100 mm

Right is the view of the arm shown in its home position with pre-assigned axes.
To clarify, Z4 is coming out of the plane parallel to Z2 and Z3 while Z1 is upward.



Appendix B: Authorship

Section	Author
Introduction	Yichen Guo
Methodology	Yichen Guo
Results	Haojun Feng
Discussion	Haojun Feng
Conclusion	Yuhan Wu