Worcester Polytechnic Institute

Robotics Engineering Program

LAB 2:SYSTEM ARCHITECTURE & JOINT SPACE CONTROL

Submitted By

Yichen Guo

Yuhan Wu

Haojun Feng

Date Submitted : 09 / 15 / 2021
Date Completed: 09 / 14 / 2021
Course Instructor: Prof. Mahdi Agheli
Lab Section: RBE 3001 A21

Abstract

In this report, we will use forward kinematics to calculate the position and orientation of the end effector in the physical space, and visualize it in MATLAB.

Introduction

In Lab 1, we communicated with the robot by sending commands to the joints through MATLAB code, as for lab 2, we mostly focus on calculating the forward kinematics of the arm, then try to use it to determine the position and orientation of the end effector in the physical space, plus, visualize it in MATLAB. Later, this will enable us to implement methods to generate smooth robot motions in future labs. More specifically, in this lab, we will write a function in MATLAB which takes the joint angles as input and outputs the end effector's position and orientation, and this is referred to the forward kinematics (FK function) of the robot arm.

Moreover, we have to visualize a MATLAB stick model of the robot and to validate our forward kinematics from previous steps, we need to manually twist the robot arm, accordingly, the stick model should move to the correct position and orientation.

Finally, we will define an arbitrary triangle within the arm's workspace, find joint configurations corresponding to the three vertices or three points in the joint space), then have the robot arm move to each vertex, one after another.

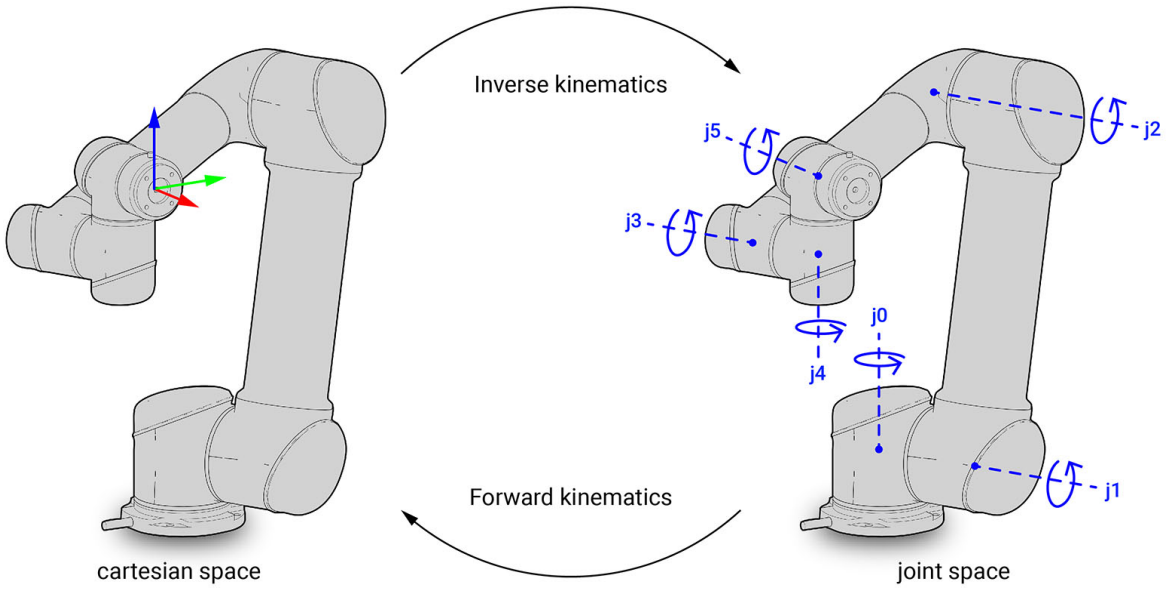


Figure.0: irrelevant decoration

Methodology

- Firstlt, we need to calculate the forward kinematics of the 3-DOF robot arm manually and prove it with matlab.
  In order to reach the goal, we need to write 3 new functions:
  - dh2mat()
    This method takes in a 1x4 array corresponding to a row of the DH parameter table for a given link.
  - dh2fk()
    This method utilizes dh2mat(), takes in an nx4 array corresponding to the n rows of the full DH parameter table. It then generates a corresponding symbolic 4x4 homogeneous transformation matrix for the composite transformation.
  - fk3001()
    This method takes n joint configurations as inputs in the form of an (n x 1) vector. It should return a 4x4 homogeneous transformation matrix representing the position and orientation of the tip frame with respect to the base frame.

- Create multiple MATLAB scripts and functions to implement our forward kinematics calculations based on last week's revised methods and previous steps.

- Implement point-to-point motion between setpoints in the joint space. Starting with checking the accuracy of the arm in the 3-D workspace in section 4.

- Visualize the arm by creating a 3D stick model in MATLAB.

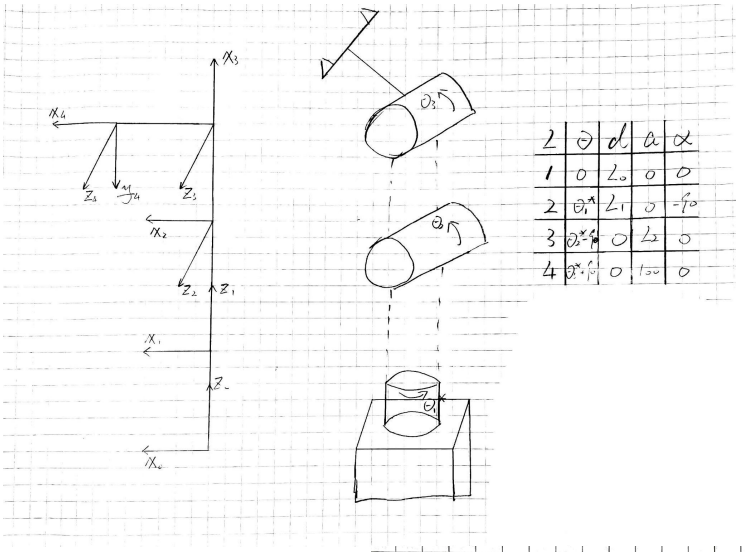
- Do the motion planning in the 3D workspace we created in MATLAB.

- Extra step 1: Plot out the workspace of the arm (workspace is defined as all the points in physical space that could be reached by the arm).

Results

Questions to address:

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



←Figure.1: manually-made robot arm configuration and DH table

Figure.2: Transfer matrix →

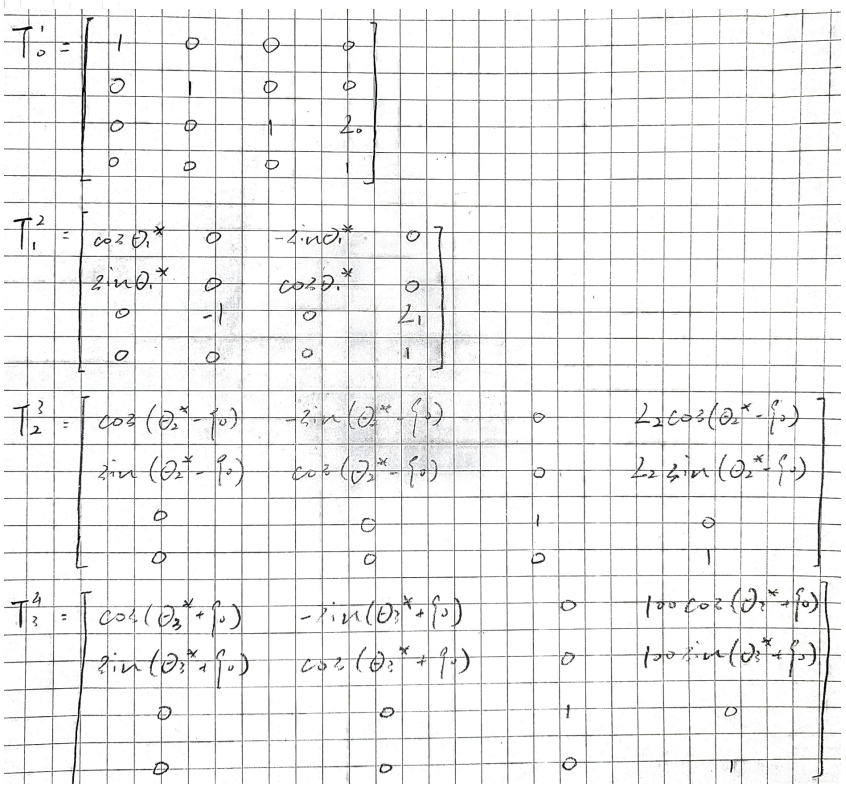


Figure.1 is the assign frames and their x, y, z perspective

Figure.2 is the raw transfer matrix before putting the number into it. According to our DH table, we typed in the elements into the matrix and used the method ‘dh2mat’ to get the transfer matrix from frame 0 to 1, 1 to 2, 2 to 3 and 3 to 4 respectively.

- **Implement the forward kinematics in MATLAB**

```
q1 = [0,0,0]';
full_dh_table_1=[0,55,0,0;
                 q1(1,1),40,0,-90;
                 q1(2,1)-90,0,100,0;
                 q1(3,1)+90,0,100,0;];

q2 = [-90.00, 86.14, 33.71]';
full_dh_table_2=[0,55,0,0;
                 q2(1,1),40,0,-90;
                 q2(2,1)-90,0,100,0;
                 q2(3,1)+90,0,100,0;];
q3 = [0,90,-90]';
full_dh_table_3=[0,55,0,0;
                 q3(1,1),40,0,-90;
                 q3(2,1)-90,0,100,0;
                 q3(3,1)+90,0,100,0;];
q4 = [90,0,0]';
full_dh_table_4=[0,55,0,0;
                 q4(1,1),40,0,-90;
                 q4(2,1)-90,0,100,0;
                 q4(3,1)+90,0,100,0;];

Q1_T01 = pp.dh2mat(full_dh_table_1(1,:))
Q1_T12 = pp.dh2mat(full_dh_table_1(2,:))
Q1_T23 = pp.dh2mat(full_dh_table_1(3,:))
Q1_T34 = pp.dh2mat(full_dh_table_1(4,:))

Q2_T01 = pp.dh2mat(full_dh_table_2(1,:))
Q2_T12 = pp.dh2mat(full_dh_table_2(2,:))
Q2_T23 = pp.dh2mat(full_dh_table_2(3,:))
Q2_T34 = pp.dh2mat(full_dh_table_2(4,:))

Q3_T01 = pp.dh2mat(full_dh_table_3(1,:))
Q3_T12 = pp.dh2mat(full_dh_table_3(2,:))
Q3_T23 = pp.dh2mat(full_dh_table_3(3,:))
Q3_T34 = pp.dh2mat(full_dh_table_3(4,:))

Q4_T01 = pp.dh2mat(full_dh_table_4(1,:))
Q4_T12 = pp.dh2mat(full_dh_table_4(2,:))
Q4_T23 = pp.dh2mat(full_dh_table_4(3,:))
Q4_T34 = pp.dh2mat(full_dh_table_4(4,:))
```

←Figure.3: code settings for 'dh2mat'

The 2 arbitrary setpoints beside (0, 0, 0) and (-90, 86.14, 33.71) are (0, 90, -90) and (90, 0, 0). By defining them with q1, q2, q3, we are able to use 'dh2mat' to find out the transfer matrices respecting each subsequent frame. We used the answers downwards to verify the manual solutions we had in part one and they are similar enough.

Command Window

```
Q2_T01 =

    1    0    0    0
    0    1    0    0
    0    0    1   55
    0    0    0    1


Q2_T12 =

    0    0    1    0
   -1    0    0    0
    0   -1    0   40
    0    0    0    1


Q2_T23 =

    0.9977    0.0673         0   99.7732
   -0.0673    0.9977         0   -6.7319
         0         0    1.0000         0
         0         0         0    1.0000


Q2_T34 =

   -0.5550   -0.8319         0  -55.4990
    0.8319   -0.5550         0   83.1857
         0         0    1.0000         0
         0         0         0    1.0000


Q3_T01 =

    1    0    0    0
    0    1    0    0
    0    0    1   55
    0    0    0    1


Q3_T12 =
```

←Figure.4: 'dh2mat' (0, 0, 0)

Figure.5: 'dh2mat' (-90, 86.14, 33.71)→

```
Q3_T01 =

     1     0     0     0
     0     1     0     0
     0     0     1    55
     0     0     0     1


Q3_T12 =

     1     0     0     0
     0     0     1     0
     0    -1     0    40
     0     0     0     1


Q3_T23 =

     1     0     0   100
     0     1     0     0
     0     0     1     0
     0     0     0     1


Q3_T34 =

     1     0     0   100
     0     1     0     0
     0     0     1     0
     0     0     0     1


Q4_T01 =

     1     0     0     0
     0     1     0     0
     0     0     1    55
     0     0     0     1
```

```
Command Window
  Q3_T34 =

       1     0     0   100
       0     1     0     0
       0     0     1     0
       0     0     0     1


  Q4_T01 =

       1     0     0     0
       0     1     0     0
       0     0     1    55
       0     0     0     1


  Q4_T12 =

       0     0    -1     0
       1     0     0     0
       0    -1     0    40
       0     0     0     1


  Q4_T23 =

       0     1     0     0
      -1     0     0  -100
       0     0     1     0
       0     0     0     1


  Q4_T34 =

       0    -1     0     0
       1     0     0   100
       0     0     1     0
       0     0     0     1

  HID device clean shutdown
  SimplePacketComs disconnect
fx >>
```

←Figure.6: ‘dh2mat’ (0, 90, -90)

Figure.7: ‘dh2mat’ (90, 0, 0)→

```
Q1_T01 =

     1     0     0     0
     0     1     0     0
     0     0     1    55
     0     0     0     1


Q1_T12 =

     1     0     0     0
     0     0     1     0
     0    -1     0    40
     0     0     0     1


Q1_T23 =

     0     1     0     0
    -1     0     0  -100
     0     0     1     0
     0     0     0     1


Q1_T34 =

     0    -1     0     0
     1     0     0   100
     0     0     1     0
     0     0     0     1
fx
```

Editor - /home/ywu18/RBE3001_Matlab21/src/Lab2_Test.m

Lab2_main.m | Lab2_Test.m | Robot.m | Model.m

```
31 -  import org.hid4java.*;
32 -  version -java
33 -  myHIDSimplePacketComs=HIDfactory.get();
34 -  myHIDSimplePacketComs.setPid(pid);
35 -  myHIDSimplePacketComs.setVid(vid);
36 -  myHIDSimplePacketComs.connect();
37
38    % Create a PacketProcessor object to send data to the nucleo firmware
39 -  pp = Robot(myHIDSimplePacketComs);
40 -  try
41    %   This is for signoff 1
42        %set the joint variable to 0,0,0
43 -      v0 = [0,0,0];
44        %calculate the trasformation matrix using the given joint configuration
45 -      a0 = pp.fk3001(v0)
46        %set the joint variable to calibration pose
47 -      v1 = [-90.00, 86.14, 33.71];
48        %calculate the trasformation matrix using the given joint configuration
49 -      a1 = pp.fk3001(v1)
50        %set the joint variable to 0,90,-90
51 -      v2 = [0,90,-90];
52        %calculate the trasformation matrix using the given joint configuration
53 -      a2 = pp.fk3001(v2)
54        %set the joint variable to 90,0,0
55 -      v3 = [90,0,0];
56        %calculate the trasformation matrix using the given joint configuration
57 -      a3 = pp.fk3001(v3)
58
59    %     full_dh_table=[0,55,0,0;
60    %                    q(1,1),40,0,-90;
61    %                    q(2,1)-90,0,100,0;
62    %                    q(3,1)+90,0,100,0;];
63    %     pp.dh2fk
64    %=========================================================================
65    %     This is for signoff 2
66    %       v1 = [0,0,0];
67    %       %make the robot arm move
68    %       pp.servo_jp(v1);
```

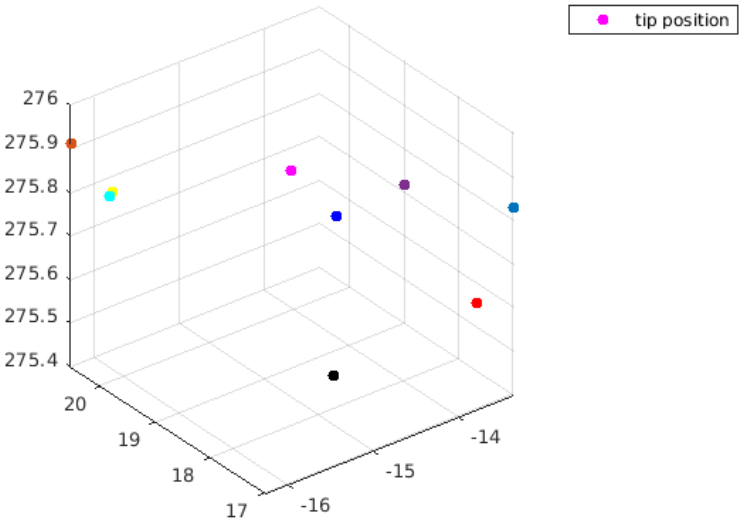
Figure.8: code settings for ‘fk3001’

```
a0 =

     1     0     0   100
     0     0     1     0
     0    -1     0   195
     0     0     0     1


a1 =

         0         0    1.0000         0
    0.4977    0.8673         0  -50.0000
   -0.8673    0.4977         0   14.9987
         0         0         0    1.0000


a2 =

     1     0     0   200
     0     0     1     0
     0    -1     0    95
     0     0     0     1


a3 =

     0     0    -1     0
     1     0     0   100
     0    -1     0   195
     0     0     0     1

HID device clean shutdown
SimplePacketComs disconnect
>>
```

Figure.9: results of transfer matrices

Based on the answers we got from 'dh2mat', we can put them into 'fk3001' to reach the final general form of transfer matrix. a0, a1, a2 and a3 are our final answers.

- **Home Position to arbitrary position**



←Figure.10: 10 times tip positions

The mean tip position we got is (-14.8663, 18.9676, 275.6862) while the root mean square is just (14.8717, 18.9741, 275.6863), which means that there is only (0.0054 0.0065 0.0001) difference. The error is small enough to say that the system is accurate and consistent enough. In addition, that could also be derived from the graph, because there are only eight points due to the overlap of two relatively identical points (usually there are 6 to 9 points on the plot). The ideal setpoint is (-11, 13, 275), so there are relatively large errors on the x and y axises, but that mostly caused by the building error for each joint and since the data is in 'mm', the error is small enough to be neglected in real life. We tried to use different interpolation time and speed control to approach the ideal setpoint, but they did not work out.

- **Visualize the arm by creating a 3D stick model in MATLAB**

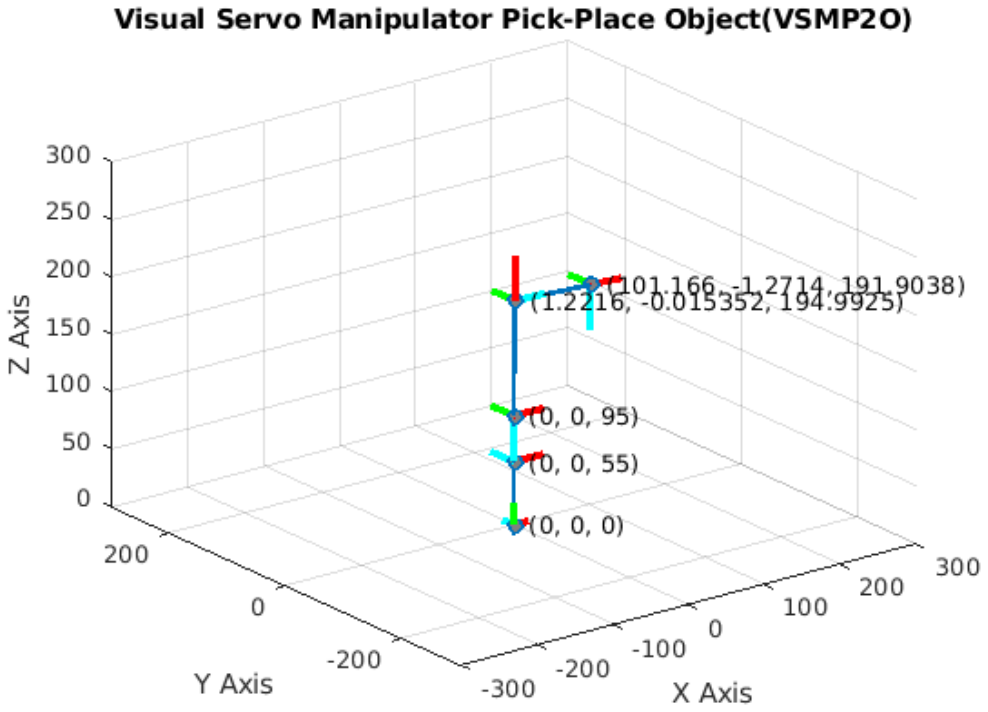


Figure.11: (0, 0, 0) position

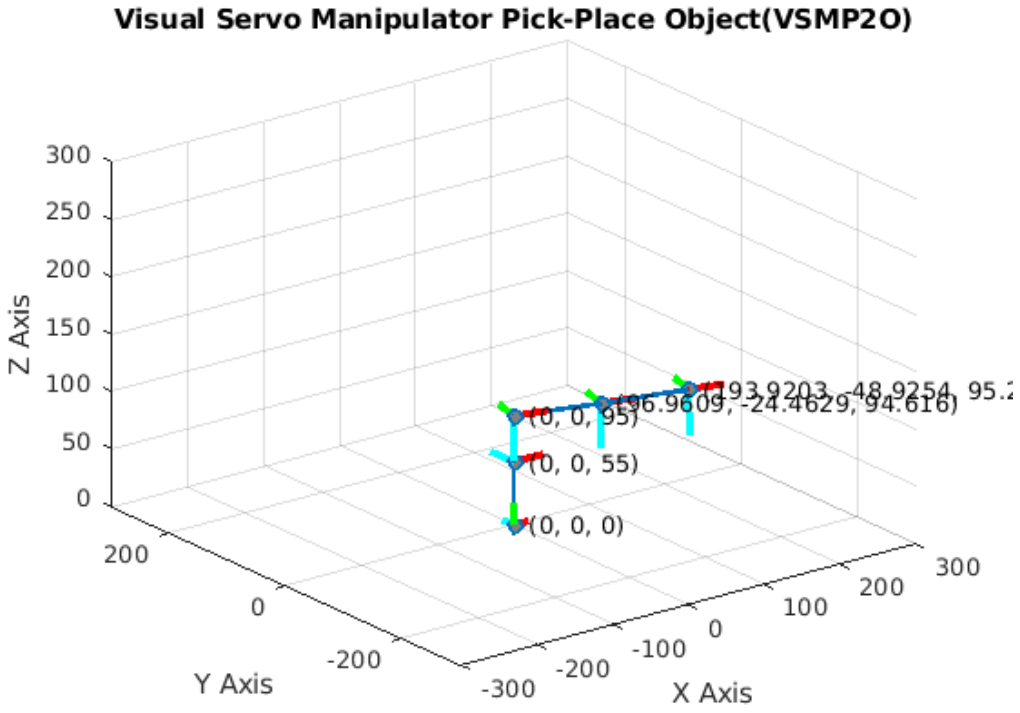


Figure.12: (0, 90, -90) position

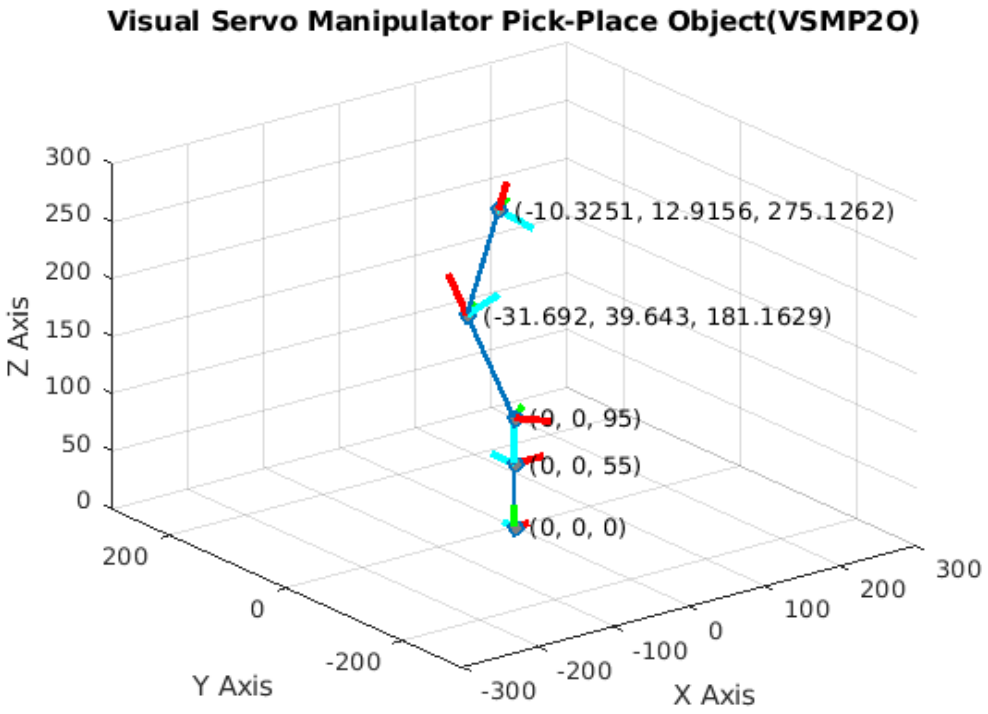


Figure.13: (50, 30, 40) position

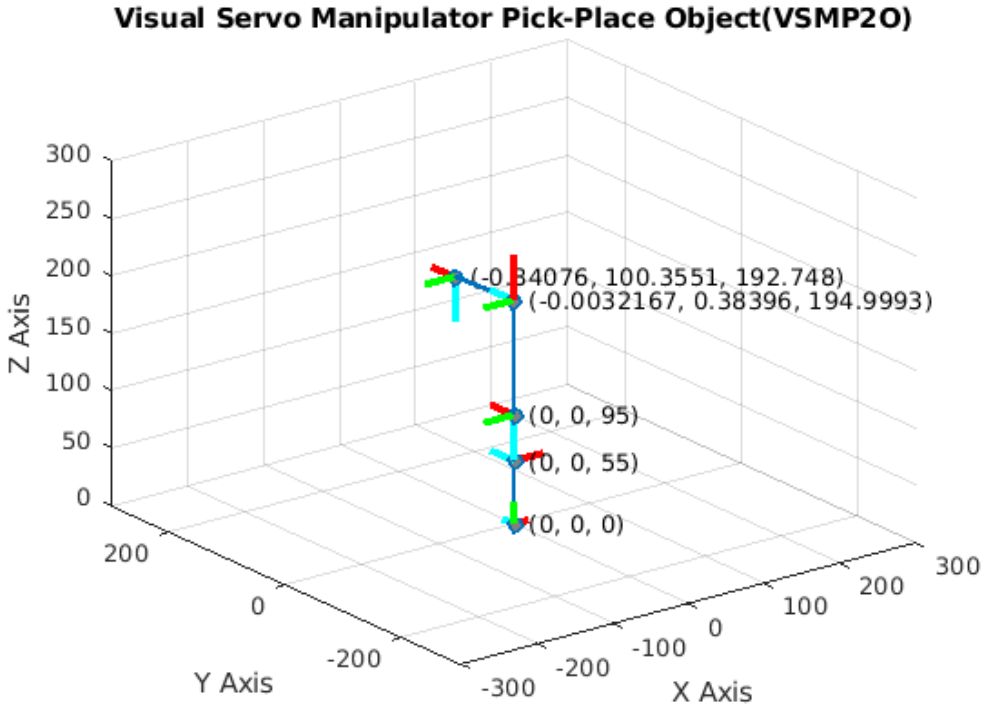


Figure.14: (90, 0, 0) position

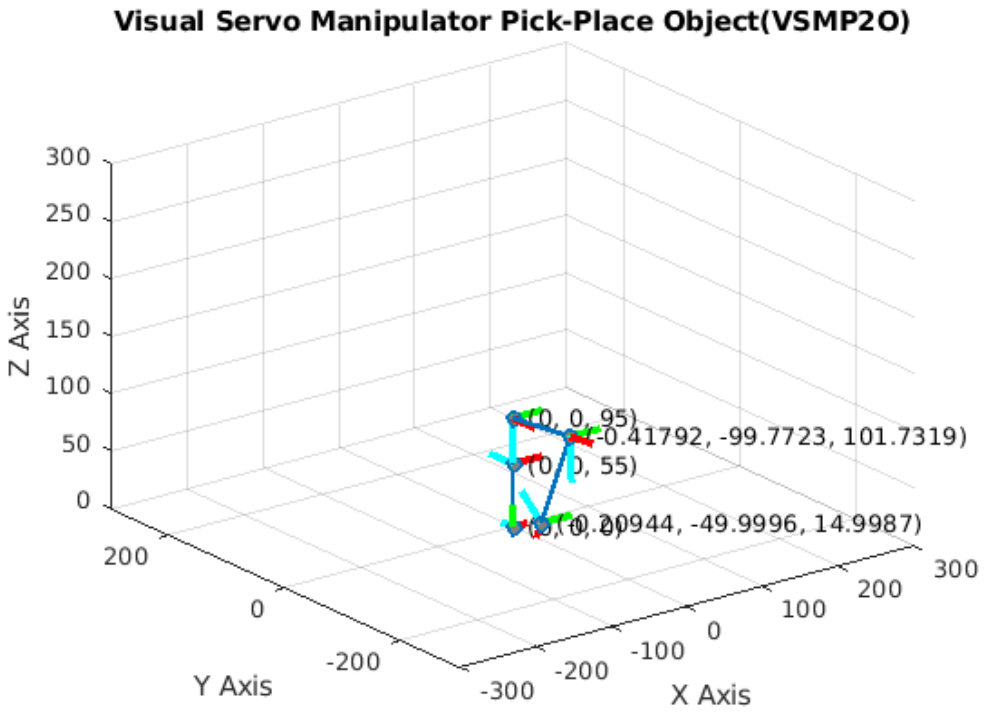


Figure.15: (-90, 86.14, 33.71) position

The point (0, 90, -90) we chose before had a conflict with the pyramid, so we switched the angle to (50, 30, 40). The final gestures we got corresponded to the passive, manual verification. Plus, all the x, y, z axises were correct regarding their frames and our calculation.

- **3D plot of the robot in MATLAB (passive)**

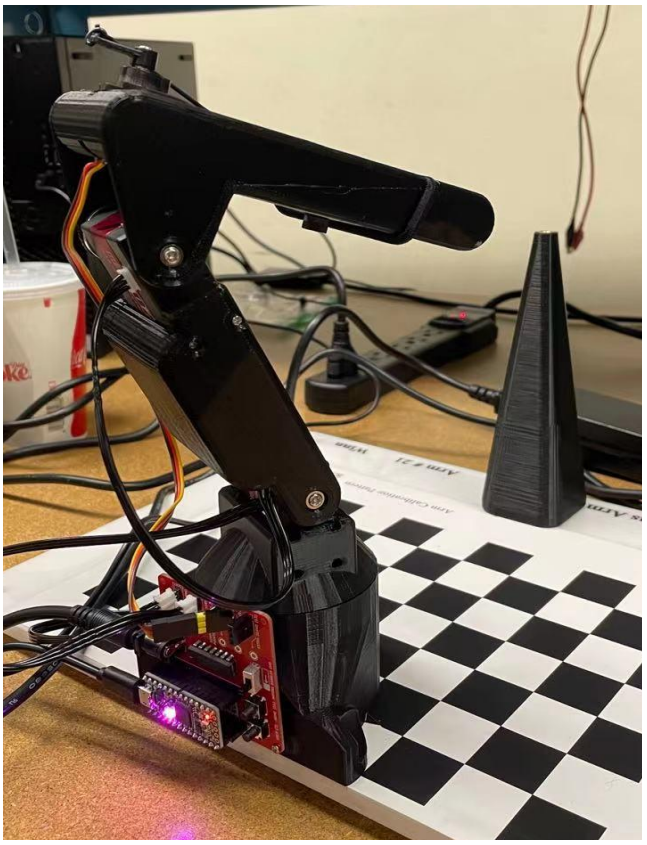


Figure.16: random manual-set gesture

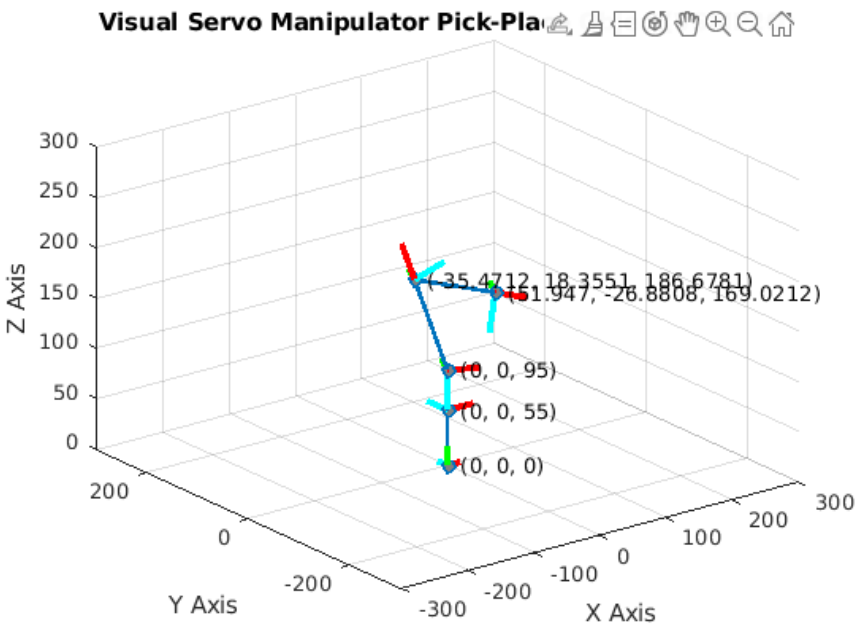


Figure.17: 3D graph for the gesture

The 3D graph matches the manual-set gesture, so our functions work just fine.

- **Live plot of the robot in MATLAB (active)**

  → (-50, -30, -40)

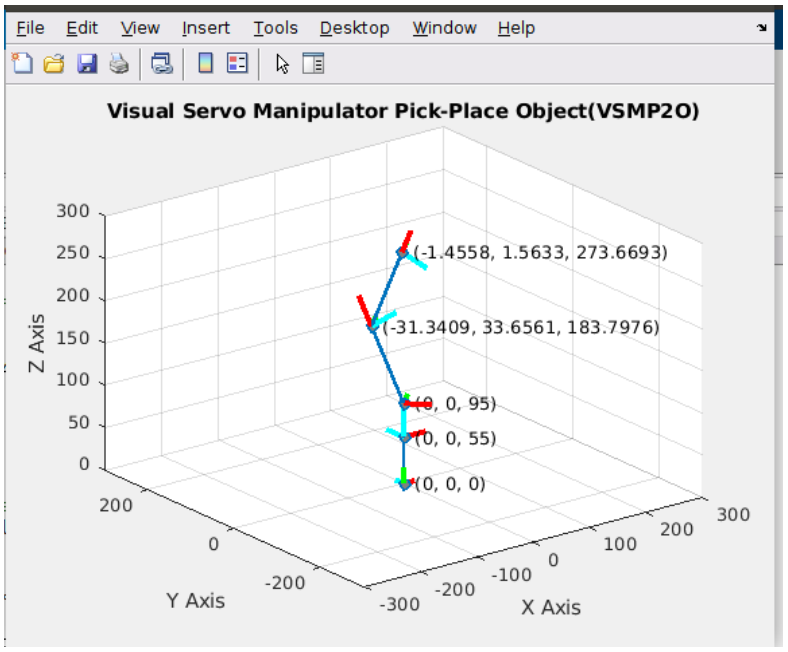


Figure.18: 3D graph 7.1.1

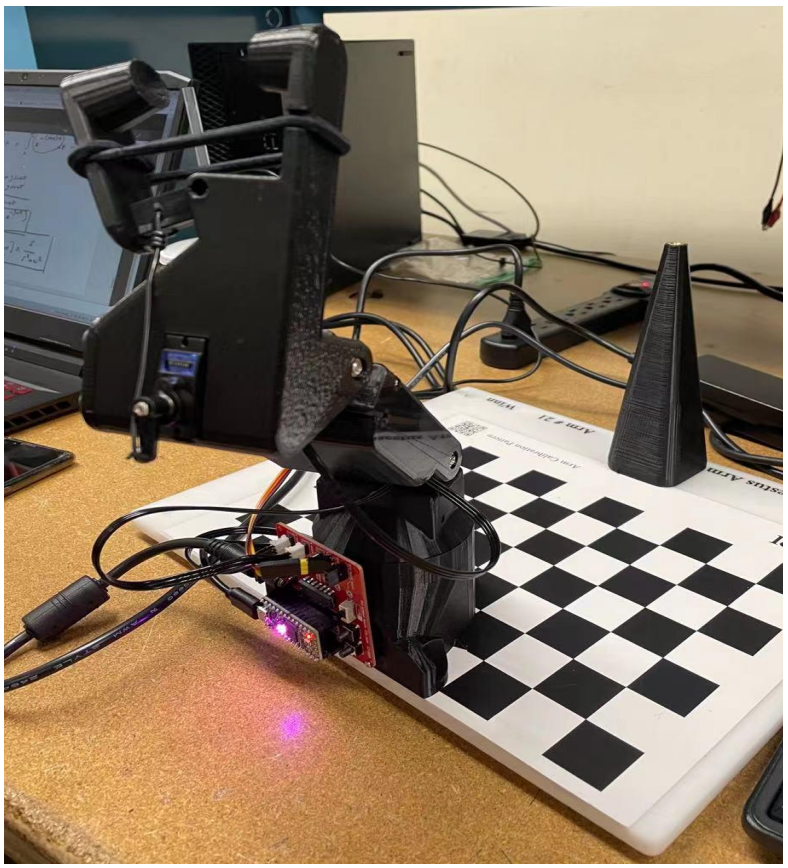


Figure.19: real graph 7.1.2

joint 1	joint 2	joint 3
-49.6800	-29.5400	-39.4900

forward kinematics: [0.2341 0.6032 0.7624 -8.4922
-0.2758 -0.7108 0.6471 10.0066
0.9323 -0.3618 0 275.2272
0 0 0 1.0000]

The recorded joint values and the translation vector are both close to the ideal situation.

The 3D graph is similar to the real life robot arm from frame 0 to 3, but there is still some offset in the last frame and this could be due to the gravity of the robot arm since systematic error exists while linking the joints. For the rest of the joints, since they are almost perpendicular to the ground, gravity influence is relatively small.

→ **(20, -70, -50)**

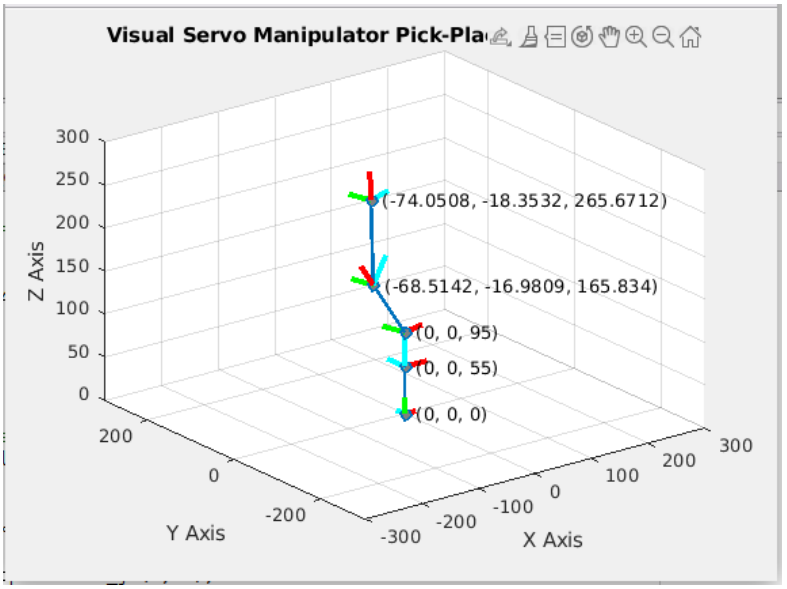


Figure.20: 3D graph 7.2.1

Figure.21: real graph 7.2.2

joint 1	joint 2	joint 3
18.7200	-44.9000	-48.8500

forward kinematics: [-0.0736 0.9415 -0.3289 -74.0210
-0.0256 0.3279 0.9444 -25.7768
0.9970 0.0779 0 265.5298
0 0 0 1.0000]

The recorded joint values are still close to the ideal value we set.

Similar to previous cases, there are gravity concerns in this temptation, so there are some offsets in the setpoint.

→ (80 -60 -20)

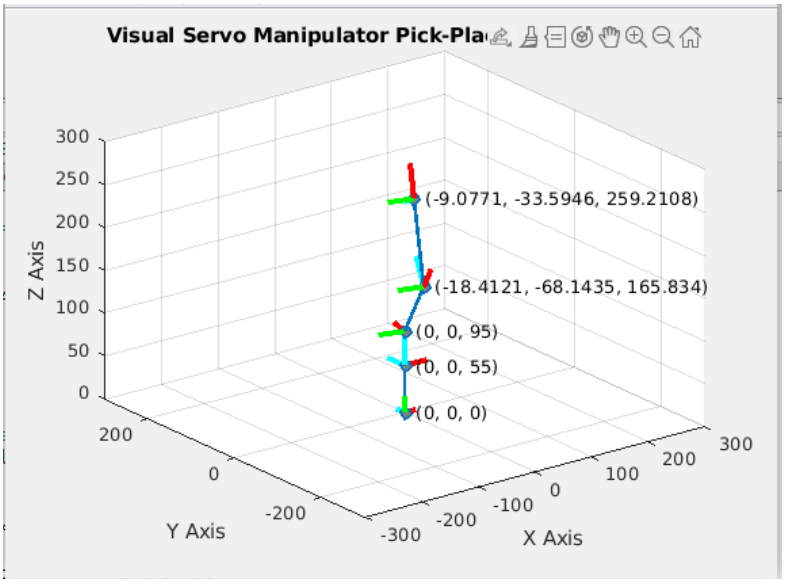


Figure.22: 3D graph 7.3.1

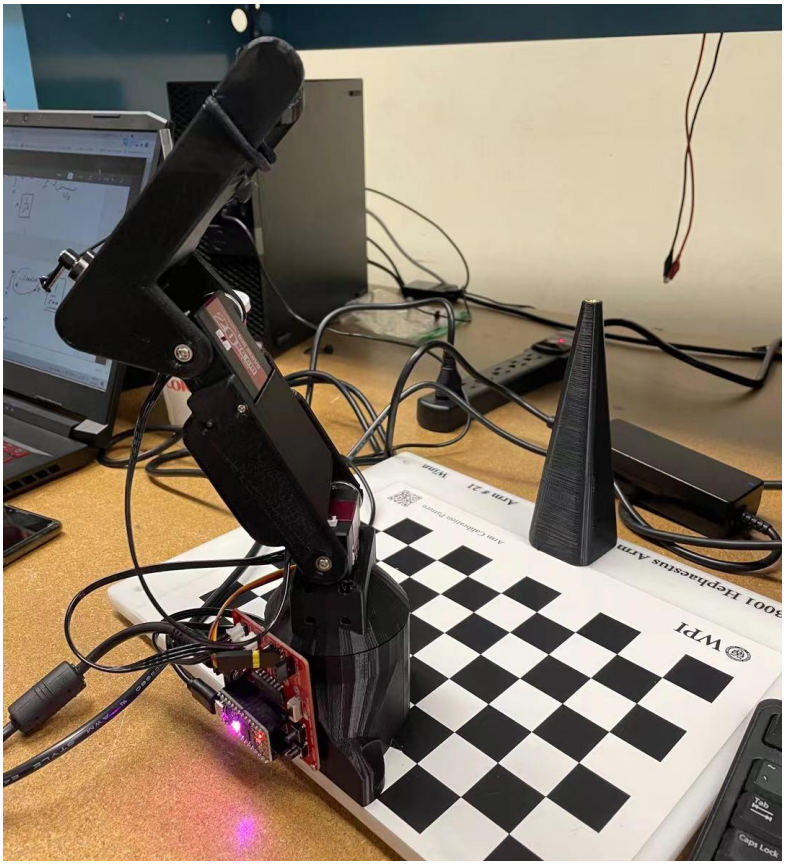


Figure.23: real graph 7.3.2

joint 1	joint 2	joint 3
78.9600	-44.9000	-21.4900

forward kinematics: [0.0767 0.1755 -0.9815 -5.8475
0.3931 0.8993 0.1915 -29.9711
0.9163 -0.4005 0 257.4633
0 0 0 1.0000]

The recorded joint values are still close to the ideal value we set.

Similar to previous cases, there are gravity concerns in this temptation, so there are some offsets in the setpoint, mostly in the x-axis direction; the y direction is less affected due to its relative perpendicular position.

→ (-70 -10 -40)

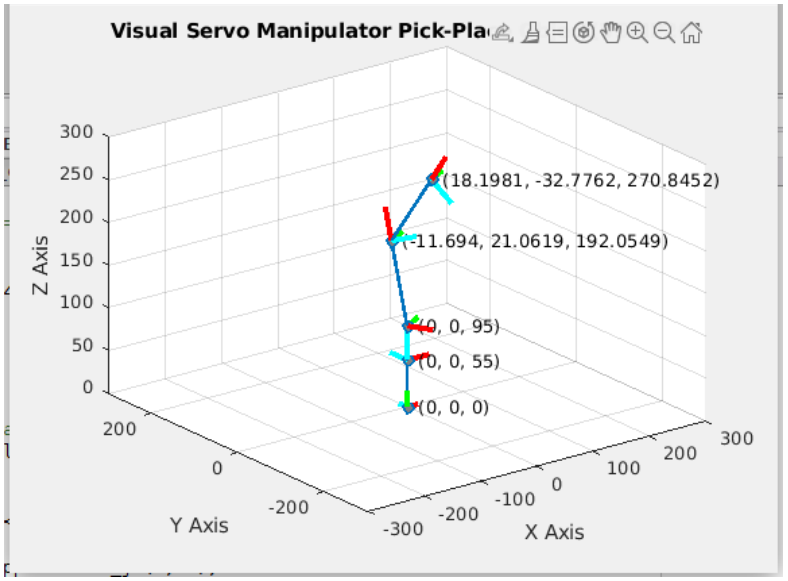


Figure.24: 3D graph 7.4.1

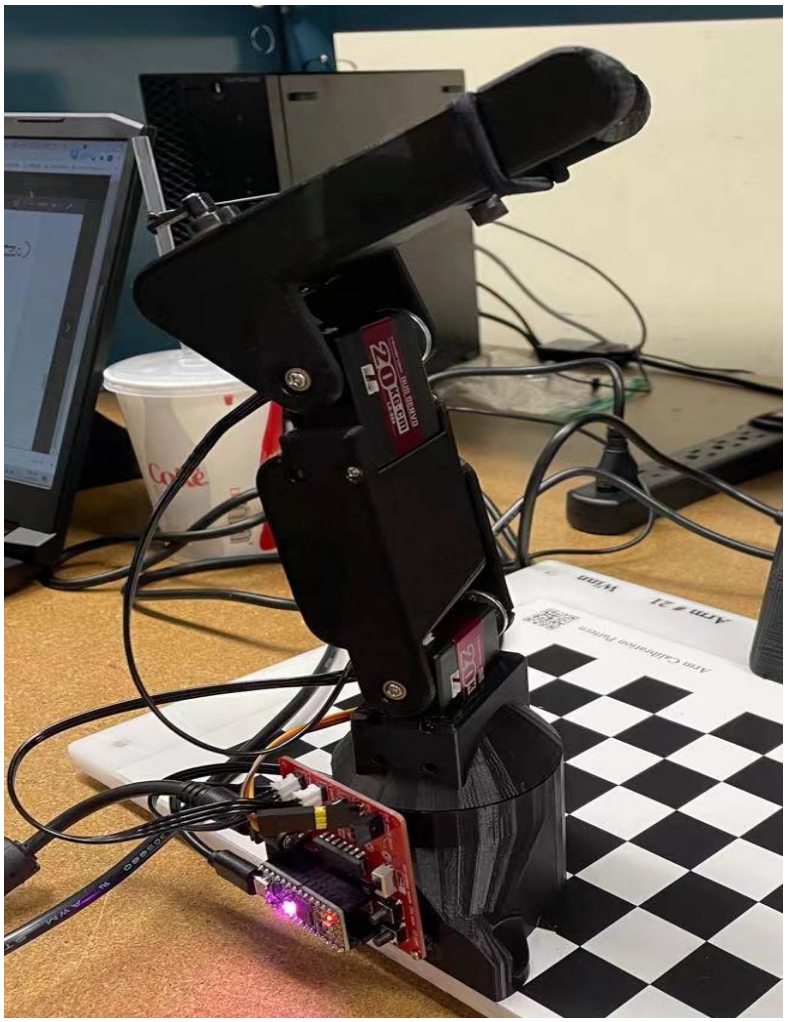


Figure.25: real graph 7.4.2

joint 1	joint 2	joint 3
-70.8000	-11.5400	-39.4900

forward kinematics: [0.2079 0.2548 0.9444 14.2107
0.5970 -0.7317 0.3289 -40.8077
0.7748 -0.6322 0 270.4619
0 0 0 1.0000]

The recorded joint values are still close to the ideal value we set.

Similar to previous cases, there are gravity concerns in this temptation, so there are some offsets in the setpoint.

→ (50, -70, -80)

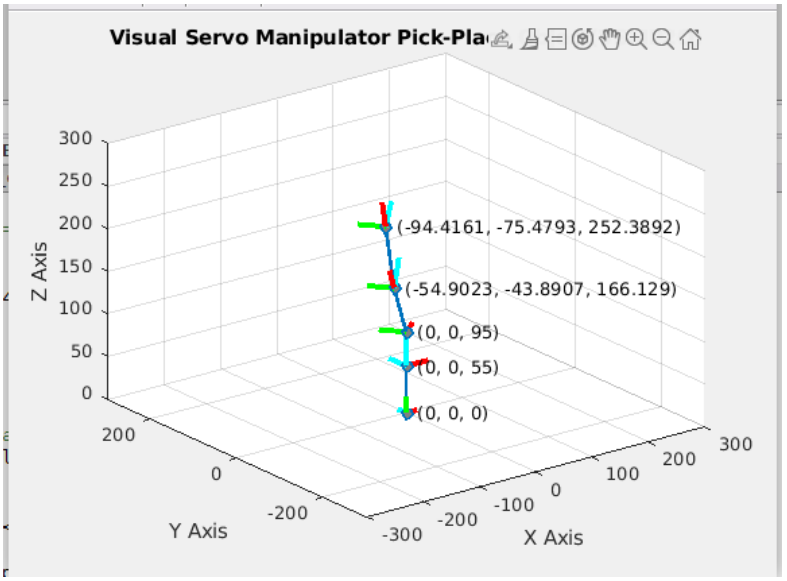


Figure.25: 3D graph 7.5.1

Figure.26: real graph 7.4.2

joint 1	joint 2	joint 3
49.4400	-45.1400	-78.8500

forward kinematics: [-0.3635 0.5391 -0.7597 -82.0573
-0.4247 0.6299 0.6502 -95.8733
0.8291 0.5590 0 249.0425
0 0 0 1.0000]

The recorded joint values are still close to the ideal value we set.

Similar to previous cases, there are gravity concerns in this temptation, so there are some offsets in the setpoint.

→ Overall

Basically, the robot arm behaves as what we thought in the real-life perspective. Human's eyes are not that accurate to figure out the exact errors, especially when we calculate datas in 'mm'. However, there are relatively large offsets in the data output compared to our ideal calculation, but that is mostly due to the system error, because z-axis data are always accurate and the angle outputs are accurate as well. If we move the arm to a more perpendicular position relative to the ground, offsets will reduce.

- **Motion Planning in Joint Space -Waypoint-based 2D motion planning**

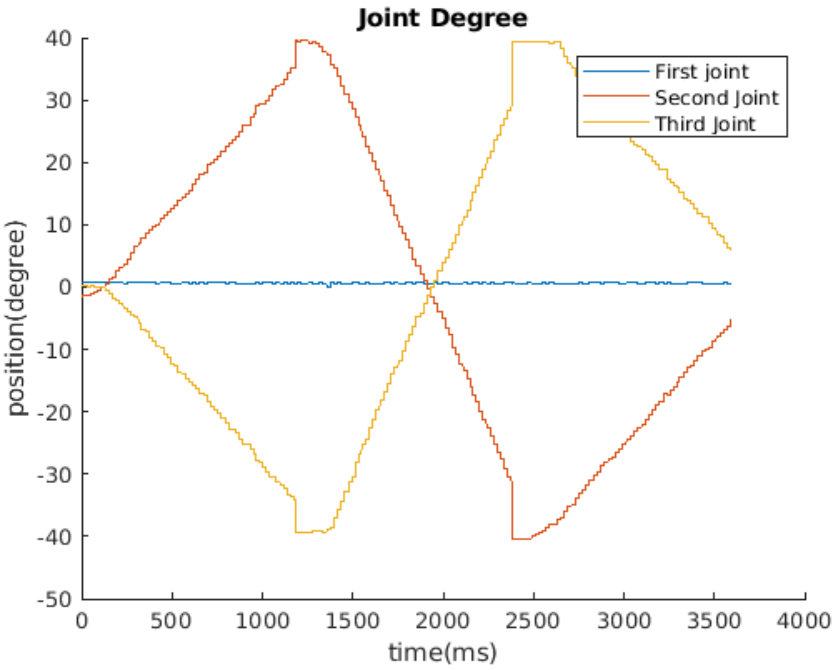


Figure.27: joint angles vs time

The shapes are almost triangles, with small deviation on the peak point. Also, small fluctuations existed, so we may apply PID control on it to smoothen the trajectory and kind of solve the overshoot problem.

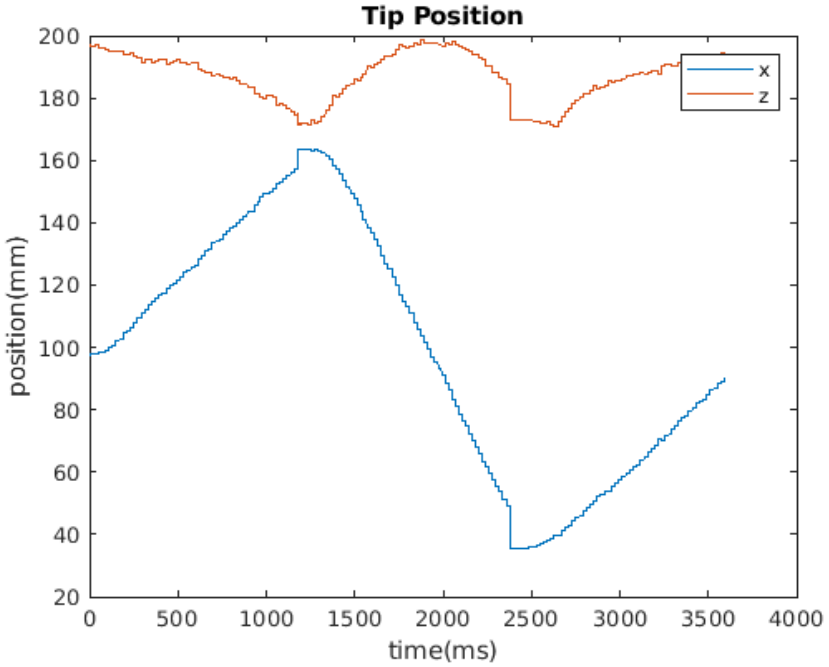


Figure.28: tip positions in x and z vs time

Both graphs look like sin or cos waves, but z is more smooth than x. Also both are relative triangle shapes.

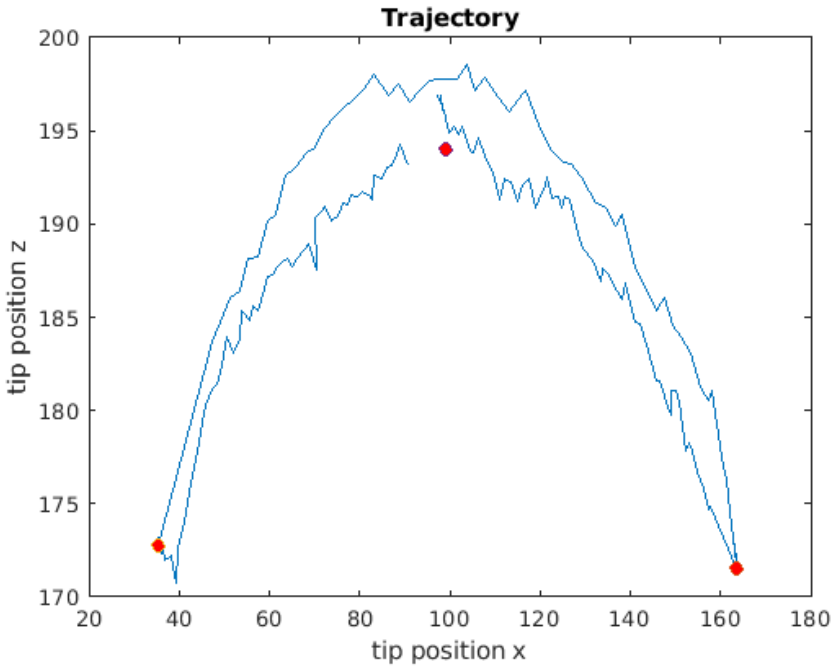


Figure.29: trajectory in the x-z plane

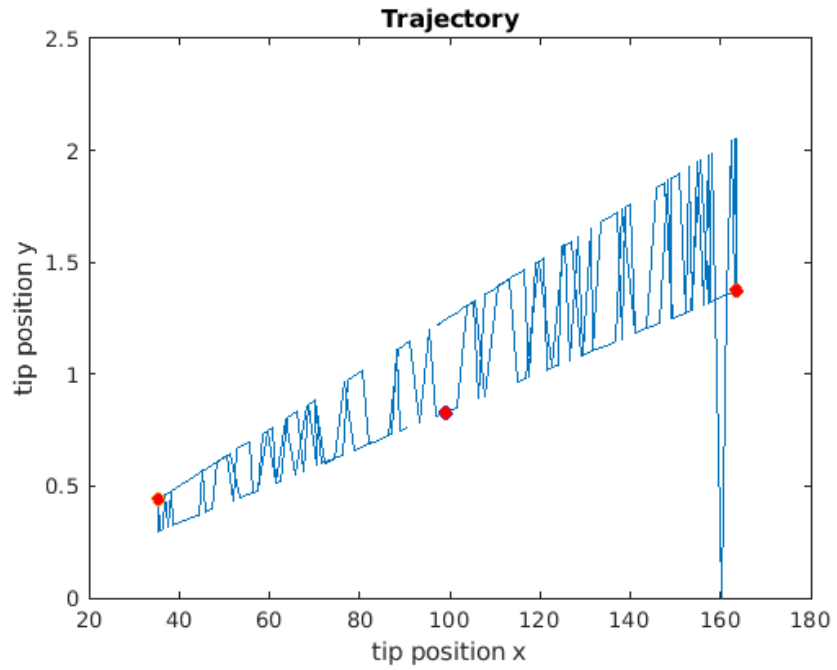


Figure.30: trajectory in the x-y plane

The trajectory in the x-z plane is in triangle shape, but in the x-y plane, it looks like a linear relationship with more fluctuations.

Both in the x-z and x-y plane, our trajectories went through the triangle vertices labelled with points of a different color, which means that our system worked pretty well.

Still, there are fluctuations that exist, we may also use PID control to reduce that.

- **Extra step 1: Plot out the workspace of the arm**

Three assumptions needed to be set up before we start to draw the workspace.

1: We do not care about the space outside of the board.

2: We neglect the obstacle on the board.

3: We set out workspace with respect to frame 1, not frame 0.

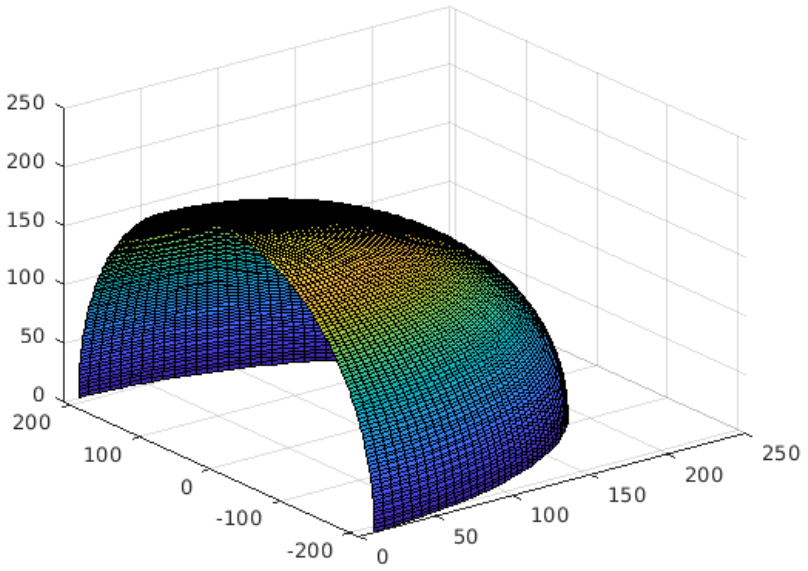


Figure.31: workspace graph

Since we define the workspace with respect to frame 1, the shape of the workspace will be a sphere, not an oval, because the farthest distance we can reach depends only on the length of the link, which are the same in all three directions and that is why we choose to use sphere equations rather than oval calculations so that the code can be more clear and relatively less complicated. As a result, it reduces its chance of causing mistakes and saves time.

```
theta = atan(y/x);
x_y = sqrt(x^2 + y^2);
phi = atan(x_y/z);
R=sqrt(x_y^2 + z^2);
Phi=linspace(0,phi);
Theta=linspace(-theta,theta);
[Phi,Theta]=meshgrid(Phi,Theta);
[X,Y,Z]=sph2cart(Theta,Phi,R);

surf(X,Y,Z);
```

This is how we define the system in the code, by plotting all the x, y and z coordinates with function surf in MATLAB.

Discussion

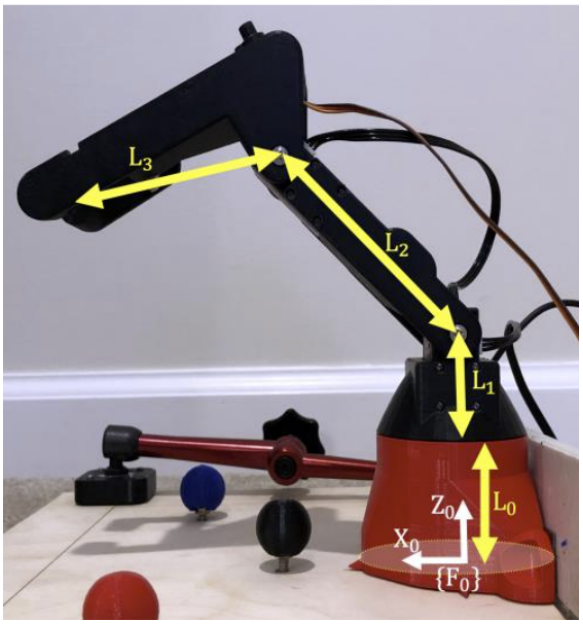
In this lab, we mostly focus on how to verify concepts of forward kinematics and do trajectory planning. We worked pretty efficiently in the past week and we are looking forward to doing better afterwards. (more to see in result sections and conclusion)

Conclusion

In this lab we successfully use MATLAB to verify forward kinematics based on the angles 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 used ‘plot_arm’ to turn all our matrices and vectors into a few sticks in the 3D workspace. The system behaved well enough for us to collect data and do the trajectory planning afterwards. At last, we successfully plotted setpoints for the tip in various 2D dimensions.

Luckily, we still had some time after we finished every essential sign-off, so we got our hands on the workspace 3D plot, and based on the assumptions we made, we successfully reached a reasonable result based on the data and eye-observation.

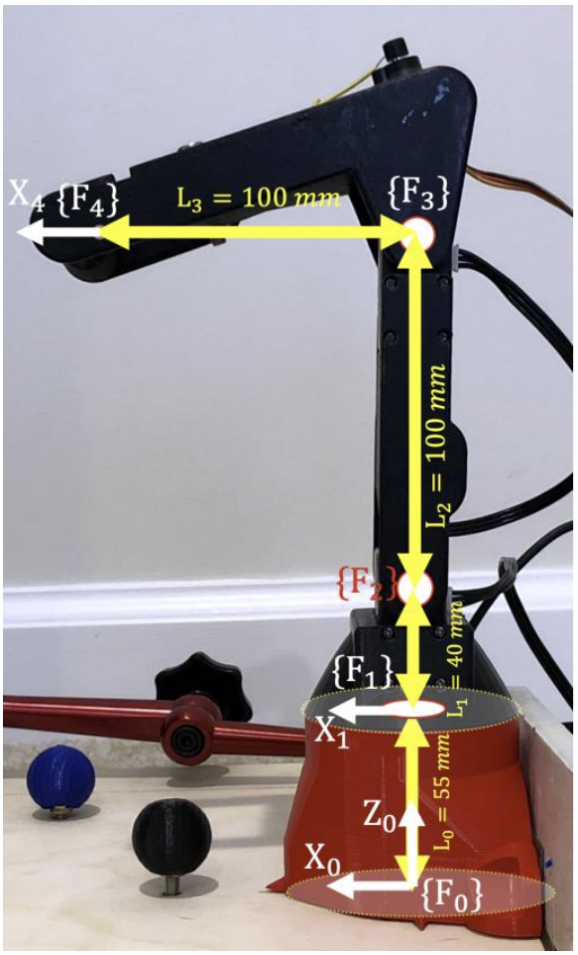
Appendix A: ARM CONFIGURATION



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