Worcester Polytechnic Institute

Robotics Engineering Program

LAB 4:VELOCITY KINEMATICS

Submitted By

Yichen Guo

Yuhan Wu

Haojun Feng

Date Submitted : 10 / 01 / 2021
Date Completed: 09 / 30 / 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. In Lab 2, we used forward kinematics 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. In Lab 3, we implemented inverse kinematics and performed trajectory generation techniques in joint space and in task space to generate a smooth trajectory from one vertex to another.

For this lab, we will implement velocity kinematics, which involves calculating the Jacobian matrix of the robot arm and using it to calculate task-space velocities for the robot, identify its singular configurations, and implement a numerical algorithm to solve the inverse kinematics problem in an iterative manner. We will then compare trajectory generation using this approach to that of polynomial trajectories.

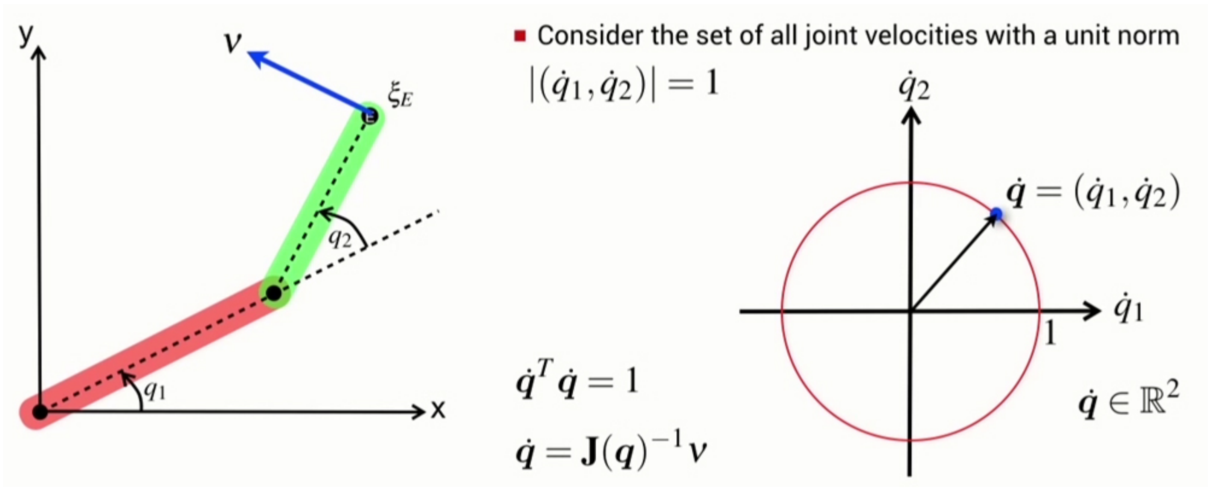

y
v
ξ_E
q_2
q_1
x
Consider the set of all joint velocities with a unit norm
$|(\dot{q}_1, \dot{q}_2)| = 1$
$\dot{q}_2$
$\dot{\boldsymbol{q}} = (\dot{q}_1, \dot{q}_2)$
$\dot{q}_1$
1
$\dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}} = 1$
$\dot{\boldsymbol{q}} = \mathbf{J}(\boldsymbol{q})^{-1} \boldsymbol{v}$
$\dot{\boldsymbol{q}} \in \mathbb{R}^2$

Figure.0: irrelevant decoration

Methodology

1. Calculate the forward velocity kinematics of the 3-DOF robot arm, both manually and by MATLAB

2. Validate your Jacobian calculation, by using the ‘jacob3001’ function. Also, find its determinant, which could tell us the singularity and prevent it from happening.

3. Calculate the forward velocity kinematics in MATLAB, by using the differential method from the position jacobian.

4. Live plot of the task-space velocity vector. Have the robot execute the trajectory and plot the stick model of the arm in real-time as the robot moves along this trajectory drawing out a triangle. Continuously read joint angles and joint velocities as it moves and use these data to continuously calculate the resulting end effector velocity vector for each time step where data is received using the ‘fdk3001’ method.

5. Discover and avoid singularities. We should implement an emergency stop that prevents the robot from reaching a singular configuration. Write a program that deliberately attempts to send the robot into a kinematic singularity while it continuously calculates the determinant of the Jacobian as the configuration changes.

6. Extra 1: Velocity-based motion planning in task space, which implements essentially the same task performed in Part 6 of Lab 3, but instead of creating a polynomial trajectory, we will command the robot to move in the direction of the target point at a set speed.

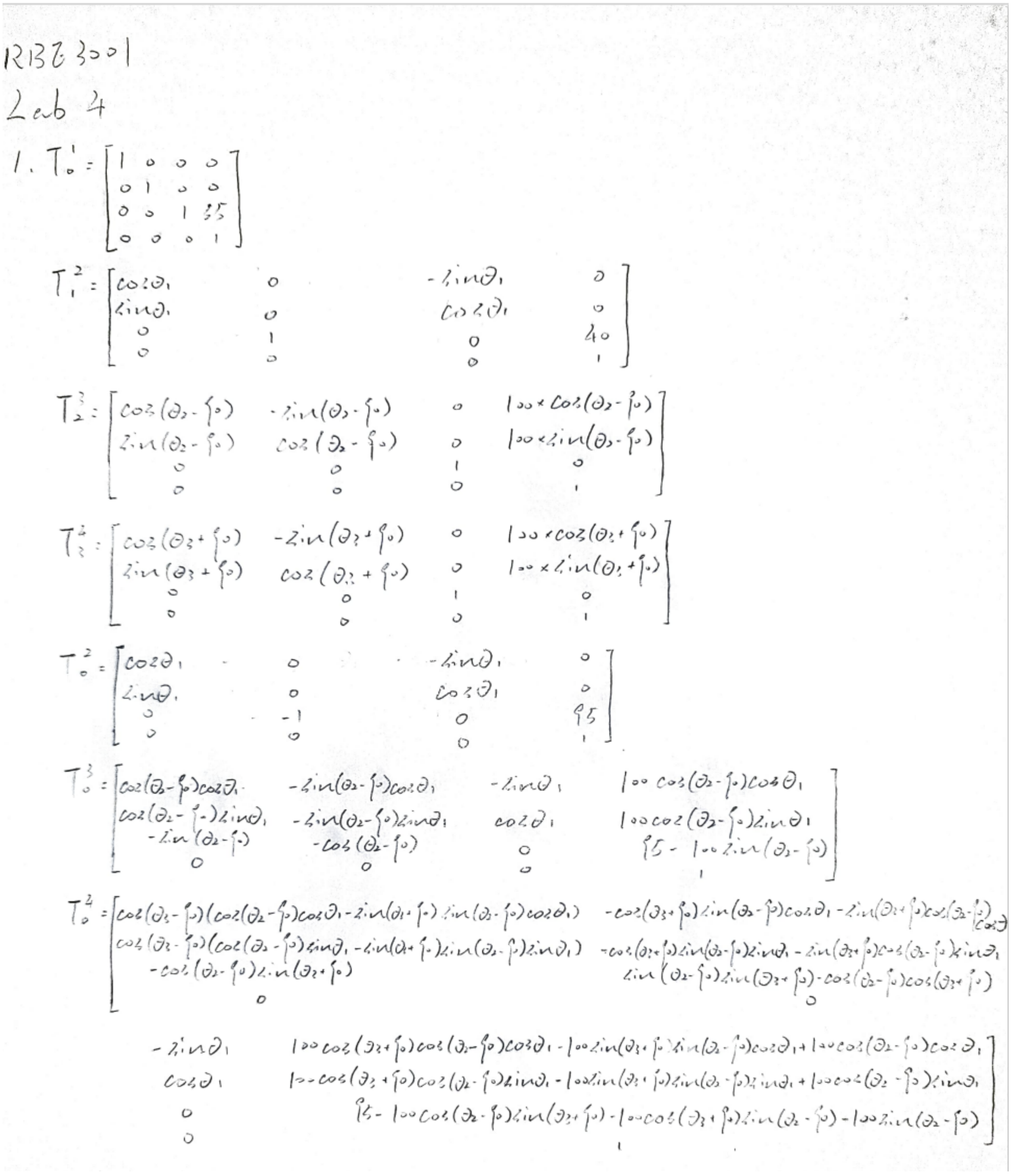
7. Extra 2: Numerical Inverse Kinematics in which you will implement a solution to the inverse kinematics problem based on velocity kinematics. This allows an iterative approach to a numeric solution without explicitly needing to determine a closed-form inverse kinematics solution and this is so called the third method of calculating Inverse Kinematics.

8. Extra 3: Power up the robot arm and use the iterative IK algorithm which has been validated in simulation to make the arm move to a predefined target position.

Results

Questions to address:

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



RBE 3001

Lab 4

1. $$T_0^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 55 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^2 = \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & 0 \\ \sin\theta_1 & 0 & \cos\theta_1 & 0 \\ 0 & 1 & 0 & 40 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^3 = \begin{bmatrix} \cos(\theta_2-90) & -\sin(\theta_2-90) & 0 & 100\times\cos(\theta_2-90) \\ \sin(\theta_2-90) & \cos(\theta_2-90) & 0 & 100\times\sin(\theta_2-90) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^4 = \begin{bmatrix} \cos(\theta_3+90) & -\sin(\theta_3+90) & 0 & 100\times\cos(\theta_3+90) \\ \sin(\theta_3+90) & \cos(\theta_3+90) & 0 & 100\times\sin(\theta_3+90) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^2 = \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & 0 \\ \sin\theta_1 & 0 & \cos\theta_1 & 0 \\ 0 & -1 & 0 & 95 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^3 = \begin{bmatrix} \cos(\theta_2-90)\cos\theta_1 & -\sin(\theta_2-90)\cos\theta_1 & -\sin\theta_1 & 100\cos(\theta_2-90)\cos\theta_1 \\ \cos(\theta_2-90)\sin\theta_1 & -\sin(\theta_2-90)\sin\theta_1 & \cos\theta_1 & 100\cos(\theta_2-90)\sin\theta_1 \\ -\sin(\theta_2-90) & -\cos(\theta_2-90) & 0 & 95-100\sin(\theta_2-90) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^4 = \left[\begin{matrix} \cos(\theta_3-90)(\cos(\theta_2-90)\cos\theta_1-\sin(\theta_3+90)\sin(\theta_2-90)\cos\theta_1) & -\cos(\theta_3+90)\sin(\theta_2-90)\cos\theta_1-\sin(\theta_3+90)\cos(\theta_2-90)\cos\theta_1 \\ \cos(\theta_3-90)(\cos(\theta_2-90)\sin\theta_1-\sin(\theta_3+90)\sin(\theta_2-90)\sin\theta_1) & -\cos(\theta_3+90)\sin(\theta_2-90)\sin\theta_1-\sin(\theta_3+90)\cos(\theta_2-90)\sin\theta_1 \\ -\cos(\theta_2-90)\sin(\theta_3+90) & \sin(\theta_2-90)\sin(\theta_3+90)-\cos(\theta_2-90)\cos(\theta_3+90) \\ 0 & 0 \end{matrix}\right.$$

$$\left.\begin{matrix} -\sin\theta_1 & 100\cos(\theta_3+90)\cos(\theta_2-90)\cos\theta_1-100\sin(\theta_3+90)\sin(\theta_2-90)\cos\theta_1+100\cos(\theta_2-90)\cos\theta_1 \\ \cos\theta_1 & 100\cos(\theta_3+90)\cos(\theta_2-90)\sin\theta_1-100\sin(\theta_3+90)\sin(\theta_2-90)\sin\theta_1+100\cos(\theta_2-90)\sin\theta_1 \\ 0 & 95-100\cos(\theta_2-90)\sin(\theta_3+90)-100\cos(\theta_3+90)\sin(\theta_2-90)-100\sin(\theta_2-90) \\ 0 & 1 \end{matrix}\right]$$

Figure.1: manual solution for Jacobian 1

$$\vec{p} = \begin{bmatrix} -100\cos\theta_1\sin\theta_2\sin\theta_3 + 100\cos\theta_1\cos\theta_2\cos\theta_3 - 100\cos\theta_1\sin\theta_2 \\ -100\sin\theta_1\sin\theta_2\sin\theta_3 + 100\sin\theta_1\cos\theta_2\cos\theta_3 - 100\sin\theta_1\sin\theta_2 \\ 95 + 100\sin\theta_2\cos\theta_3 + 100\cos\theta_2\sin\theta_3 + 100\cos\theta_2 \end{bmatrix}$$

$$J_{p1} = \begin{bmatrix} 100\sin\theta_1\sin\theta_2\sin\theta_3 - 100\sin\theta_1\cos\theta_2\cos\theta_3 + 100\sin\theta_1\sin\theta_2 \\ -100\cos\theta_1\sin\theta_2\sin\theta_3 + 100\cos\theta_1\cos\theta_2\cos\theta_3 - 100\cos\theta_1\sin\theta_2 \\ 0 \end{bmatrix}$$

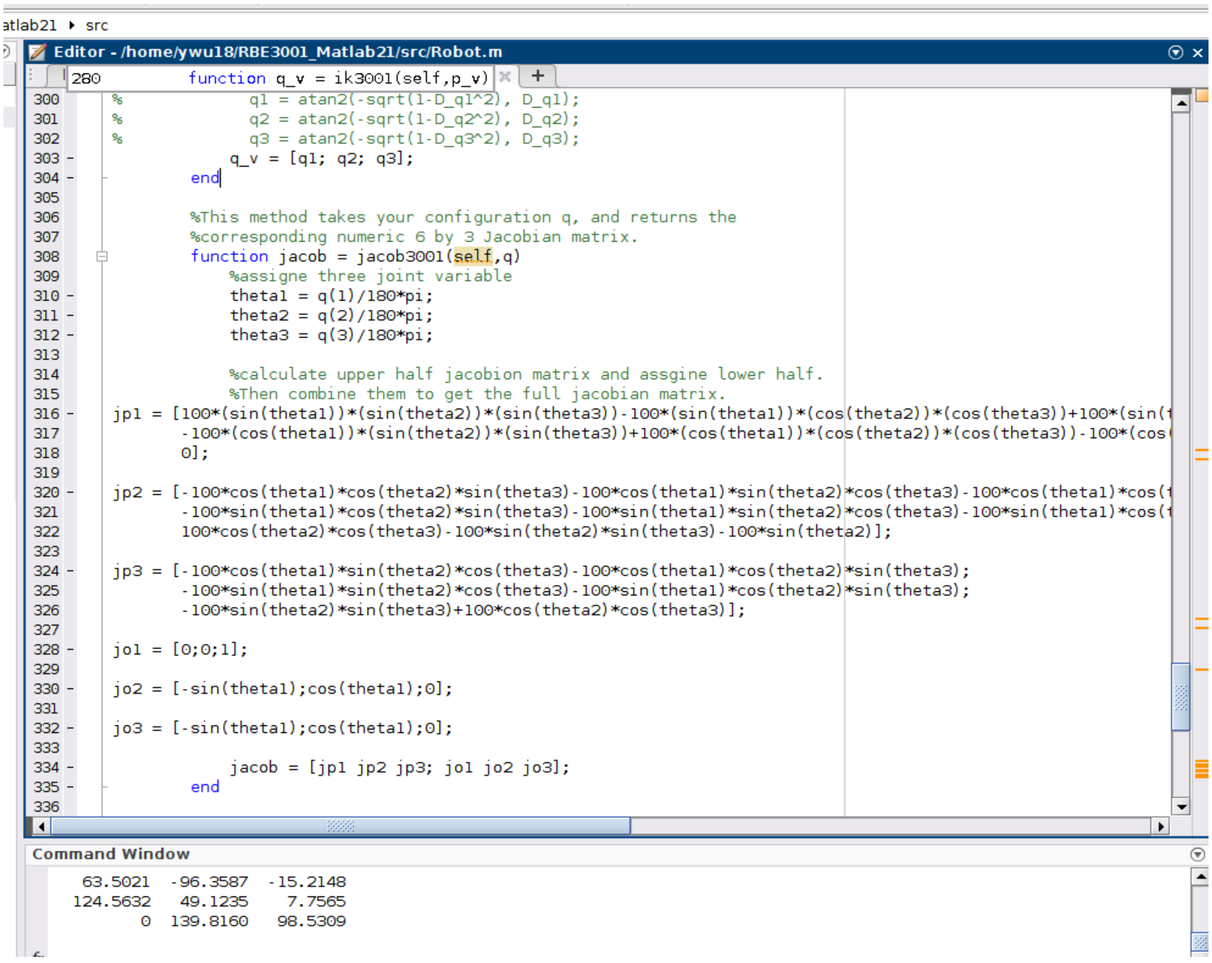
$$J_{p2} = \begin{bmatrix} -100\cos\theta_1\cos\theta_2\sin\theta_3 - 100\cos\theta_1\sin\theta_2\cos\theta_3 - 100\cos\theta_1\cos\theta_2 \\ -100\sin\theta_1\cos\theta_2\sin\theta_3 - 100\sin\theta_1\sin\theta_2\cos\theta_3 - 100\sin\theta_1\cos\theta_2 \\ 100\cos\theta_2\cos\theta_3 - 100\sin\theta_2\sin\theta_3 - 100\sin\theta_2 \end{bmatrix}$$

$$J_{p3} = \begin{bmatrix} -100\cos\theta_1\sin\theta_2\cos\theta_3 - 100\cos\theta_1\cos\theta_2\sin\theta_3 \\ -100\sin\theta_1\sin\theta_2\cos\theta_3 - 100\sin\theta_1\cos\theta_2\sin\theta_3 \\ -100\sin\theta_2\sin\theta_3 + 100\cos\theta_2\cos\theta_3 \end{bmatrix}$$

$$J(q) = \left[\begin{array}{ccc} J_{p1} & J_{p2} & J_{p3} \\ \hline 0 & -\sin\theta_1 & -\sin\theta_1 \\ 0 & \cos\theta_1 & \cos\theta_1 \\ 1 & 0 & 0 \end{array}\right]$$

Figure.2: manual solution for Jacobian 2

- **Implement the forward velocity kinematics in MATLAB.**

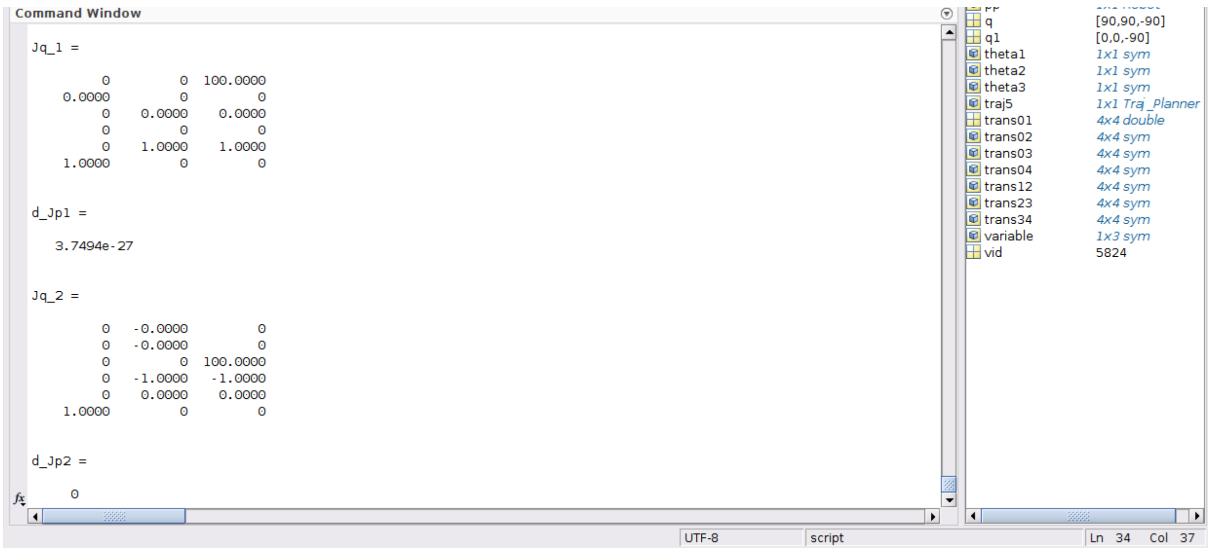


```
atlab21 ▸ src
Editor - /home/ywu18/RBE3001_Matlab21/src/Robot.m
280          function q_v = ik3001(self,p_v)
%             q1 = atan2(-sqrt(1-D_q1^2), D_q1);
%             q2 = atan2(-sqrt(1-D_q2^2), D_q2);
%             q3 = atan2(-sqrt(1-D_q3^2), D_q3);
            q_v = [q1; q2; q3];
        end

        %This method takes your configuration q, and returns the
        %corresponding numeric 6 by 3 Jacobian matrix.
        function jacob = jacob3001(self,q)
            %assigne three joint variable
            theta1 = q(1)/180*pi;
            theta2 = q(2)/180*pi;
            theta3 = q(3)/180*pi;

            %calculate upper half jacobion matrix and assgine lower half.
            %Then combine them to get the full jacobian matrix.
jp1 = [100*(sin(theta1))*(sin(theta2))*(sin(theta3))-100*(sin(theta1))*(cos(theta2))*(cos(theta3))+100*(sin(t
       -100*(cos(theta1))*(sin(theta2))*(sin(theta3))+100*(cos(theta1))*(cos(theta2))*(cos(theta3))-100*(cos(
       0];

jp2 = [-100*cos(theta1)*cos(theta2)*sin(theta3)-100*cos(theta1)*sin(theta2)*cos(theta3)-100*cos(theta1)*cos(t
       -100*sin(theta1)*cos(theta2)*sin(theta3)-100*sin(theta1)*sin(theta2)*cos(theta3)-100*sin(theta1)*cos(t
       100*cos(theta2)*cos(theta3)-100*sin(theta2)*sin(theta3)-100*sin(theta2)];

jp3 = [-100*cos(theta1)*sin(theta2)*cos(theta3)-100*cos(theta1)*cos(theta2)*sin(theta3);
       -100*sin(theta1)*sin(theta2)*cos(theta3)-100*sin(theta1)*cos(theta2)*sin(theta3);
       -100*sin(theta2)*sin(theta3)+100*cos(theta2)*cos(theta3)];

jo1 = [0;0;1];

jo2 = [-sin(theta1);cos(theta1);0];

jo3 = [-sin(theta1);cos(theta1);0];

            jacob = [jp1 jp2 jp3; jo1 jo2 jo3];
        end
```

```
Command Window
    63.5021  -96.3587  -15.2148
   124.5632   49.1235    7.7565
          0  139.8160   98.5309
```

Figure.3: MATLAB calculation set up for Jacobian

- **Validate your Jacobian calculation**



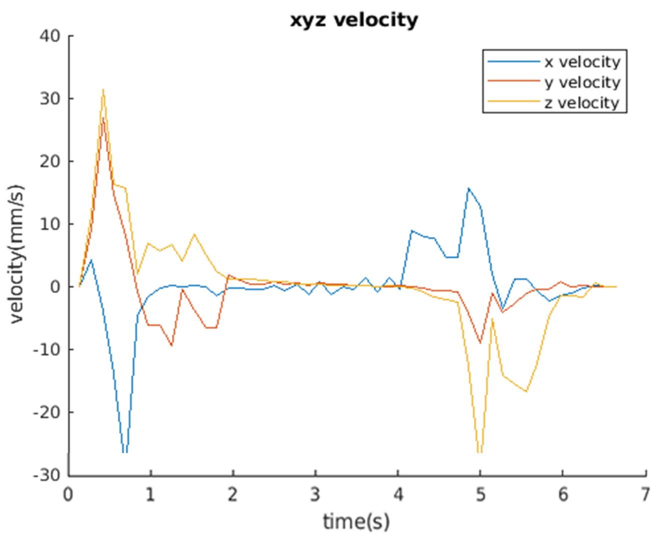
```
Command Window

Jq_1 =

         0         0  100.0000
    0.0000         0         0
         0    0.0000    0.0000
         0         0         0
         0    1.0000    1.0000
    1.0000         0         0


d_Jp1 =

   3.7494e-27


Jq_2 =

         0   -0.0000         0
         0   -0.0000         0
         0         0  100.0000
         0   -1.0000   -1.0000
         0    0.0000    0.0000
    1.0000         0         0


d_Jp2 =

     0
```

q	[90,90,-90]
q1	[0,0,-90]
theta1	1x1 sym
theta2	1x1 sym
theta3	1x1 sym
traj5	1x1 Traj_Planner
trans01	4x4 double
trans02	4x4 sym
trans03	4x4 sym
trans04	4x4 sym
trans12	4x4 sym
trans23	4x4 sym
trans34	4x4 sym
variable	1x3 sym
vid	5824

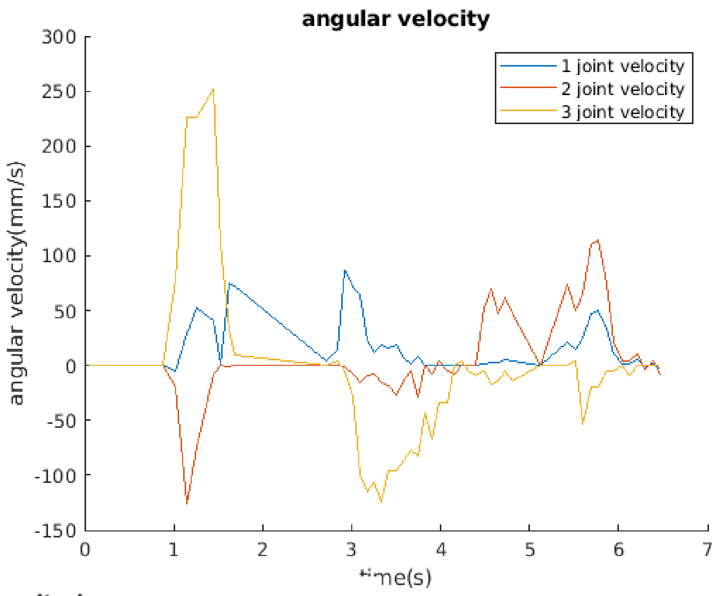
UTF-8 script Ln 34 Col 37

Figure.4: MATLAB calculation result for Jacobian

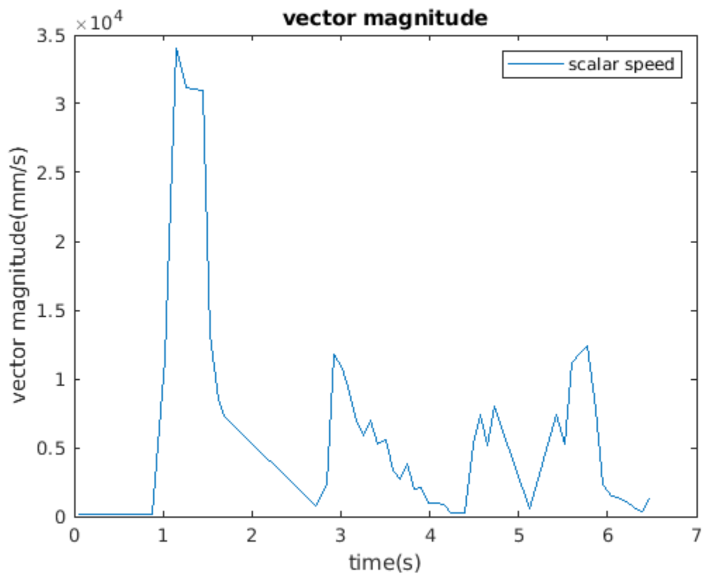
- **Live plot of the task-space velocity vector**



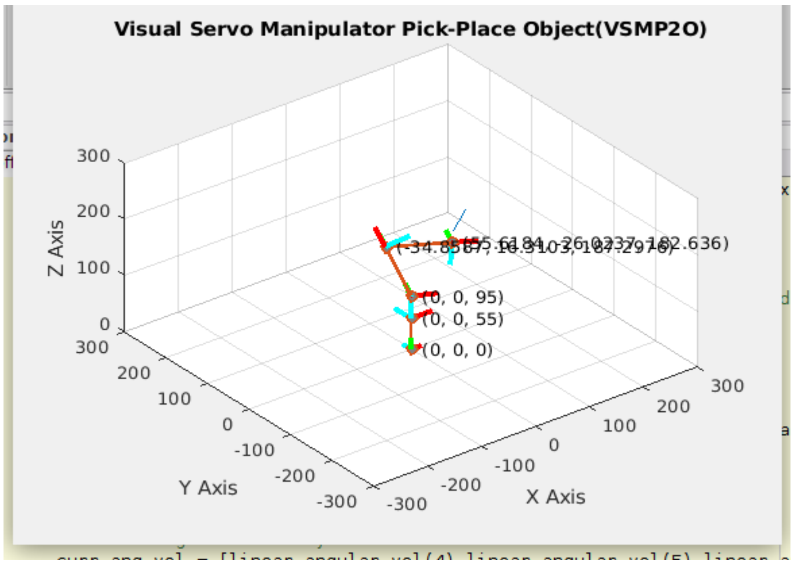
←Figure.5: xyz velocity during trajectory



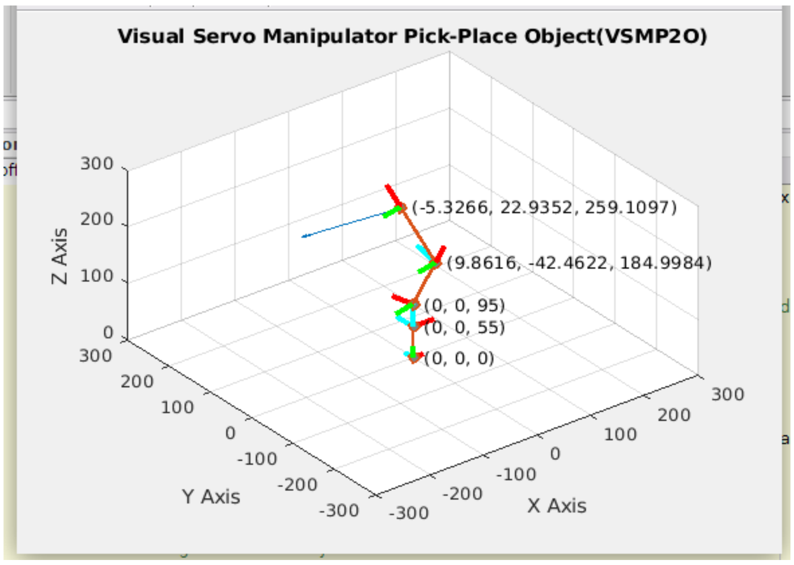
→Figure.6: angular velocity of each joint during trajectory



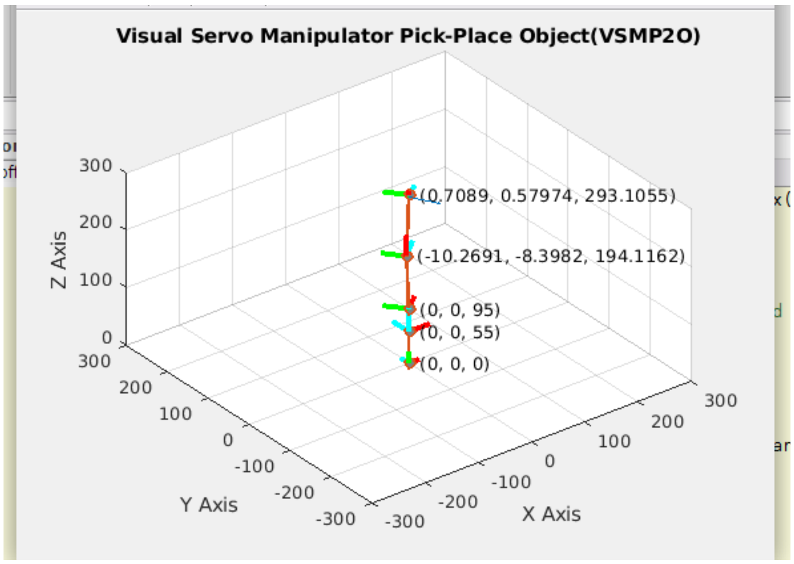
←Figure.7: vector magnitude during trajectory



←Figure.8: robot arm process 1

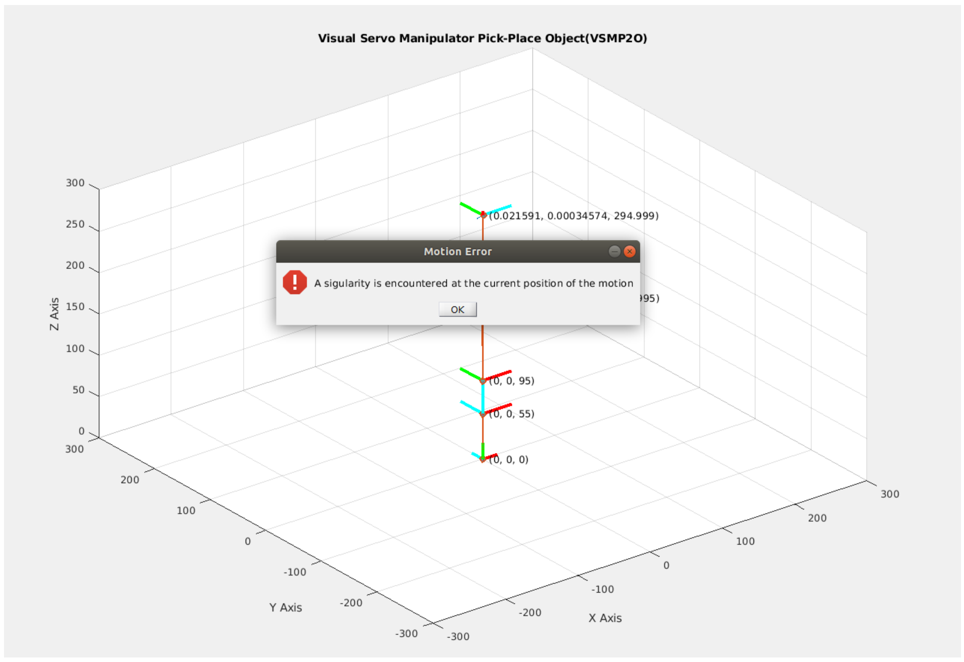


→Figure.9: robot arm process 2



←Figure.10: robot arm process 3

- Discover and avoid singularities



A singularity is encountered at the current position of the motion

Figure.11: error notification when singularity happens

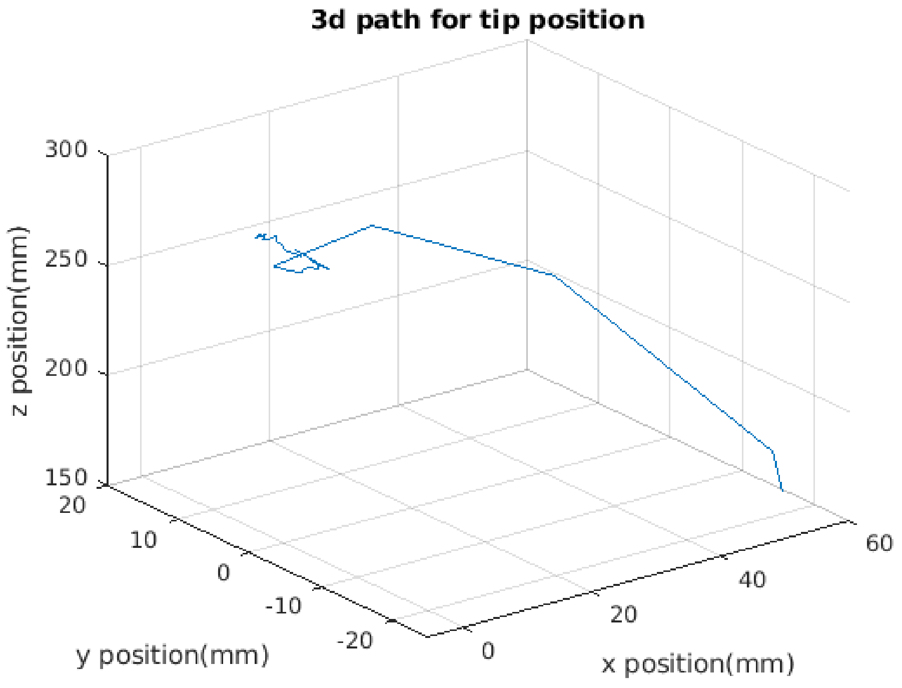


Figure.12: 3D plot tracing of the tip of the robot

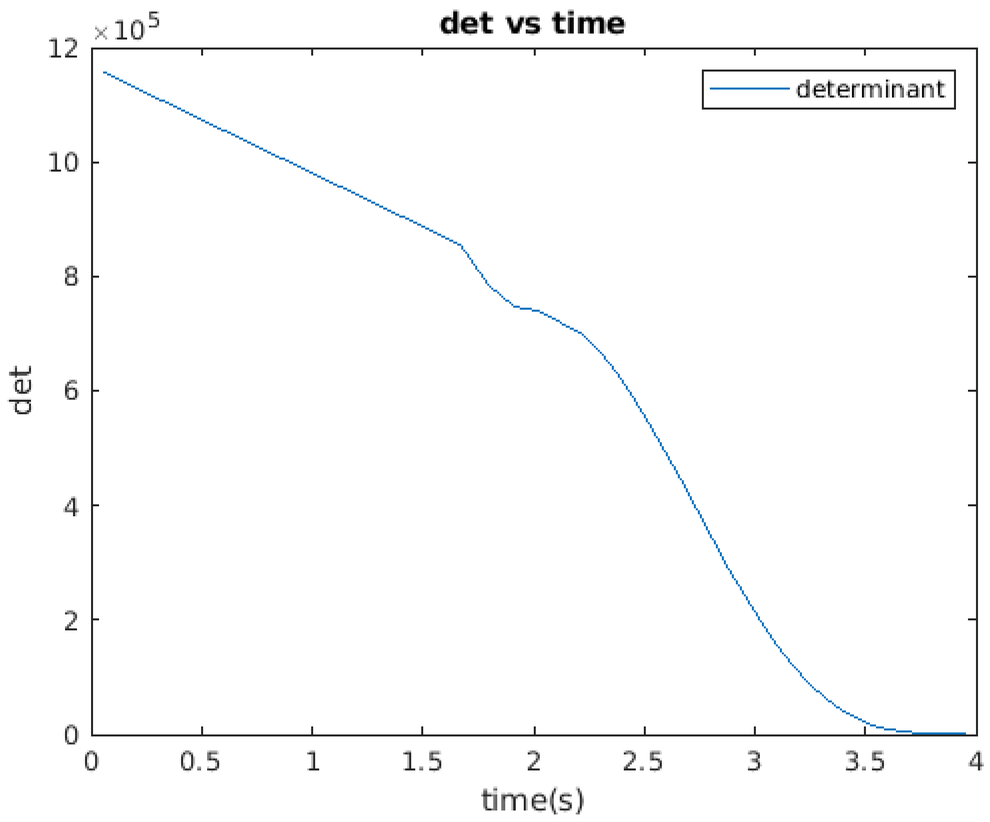


Figure.13: determinant of Jacobian v.s time

- **Extra 1: Velocity-based motion planning in task space**

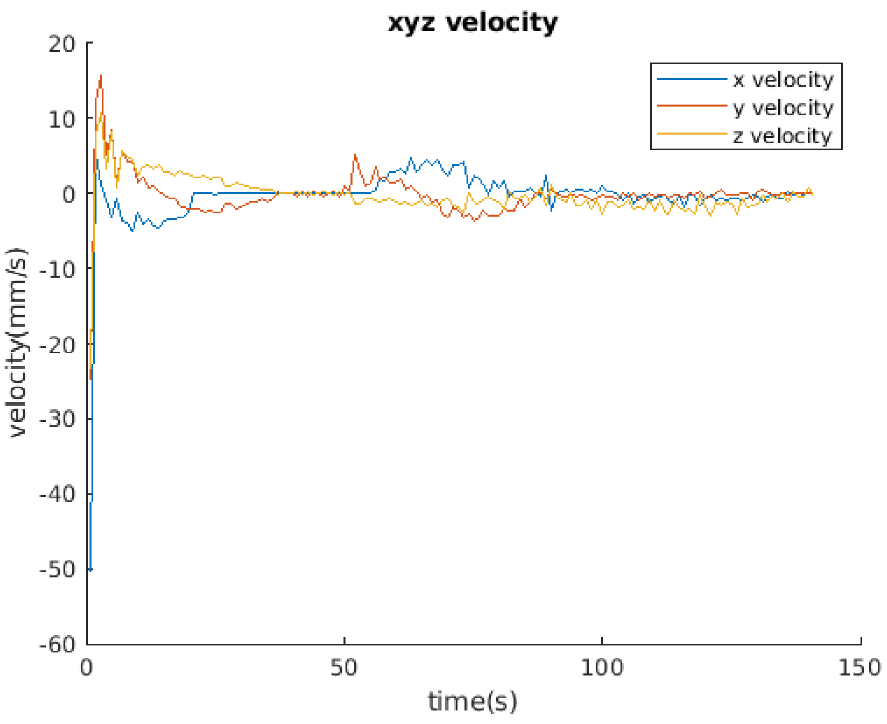


Figure.14: xyz velocity of task space trajectory

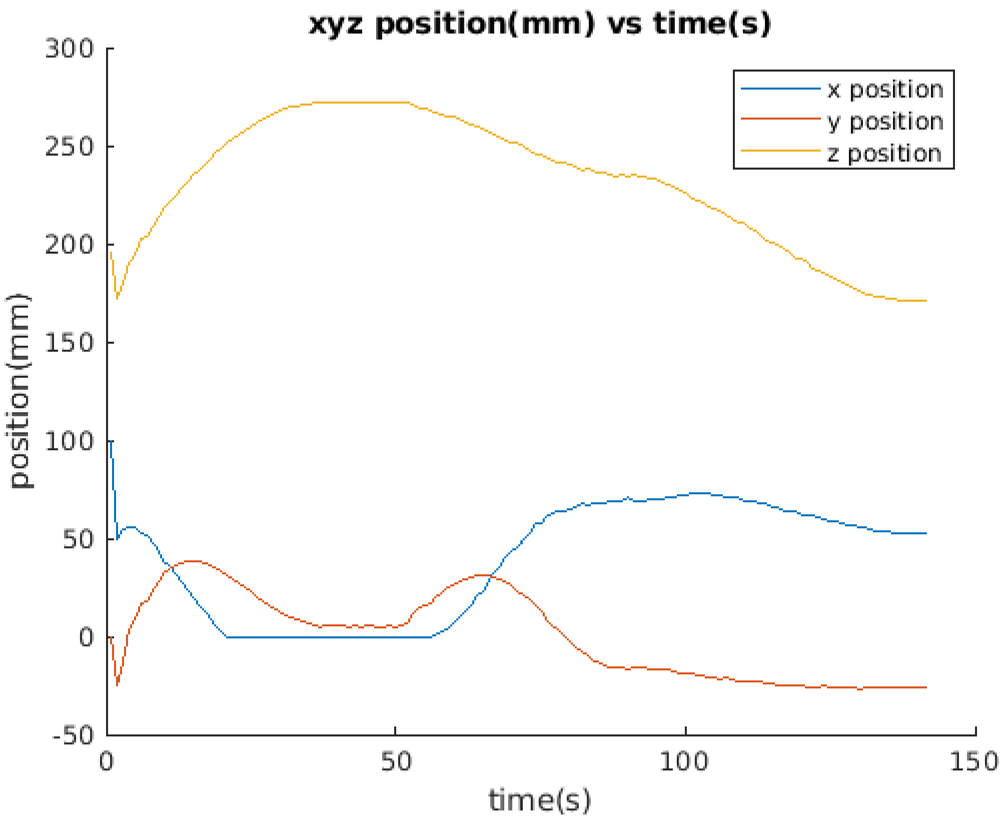


Figure.15: xyz position of task space trajectory

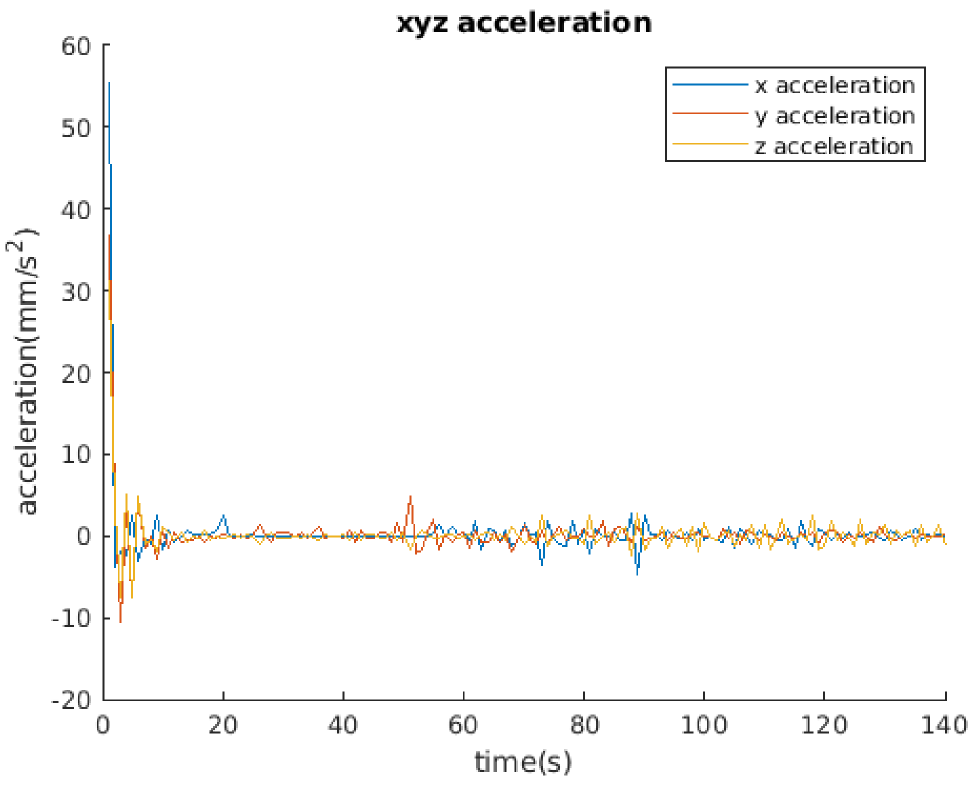


Figure.16: xyz acceleration of task space trajectory

- **Extra 2: Numerical Inverse Kinematics**
    - For point 1

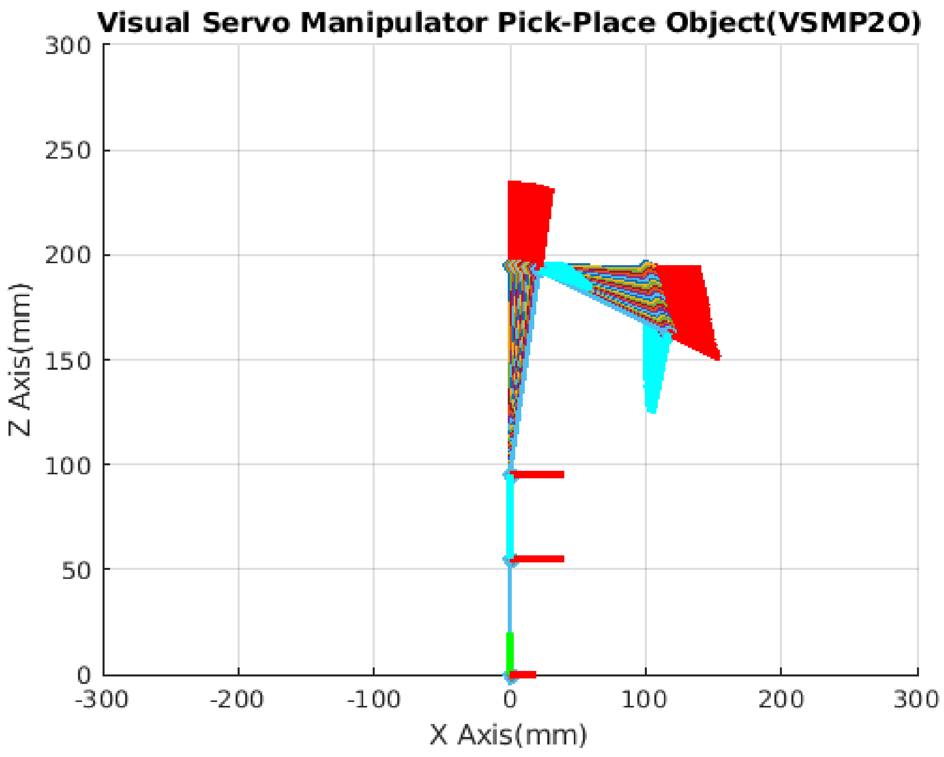
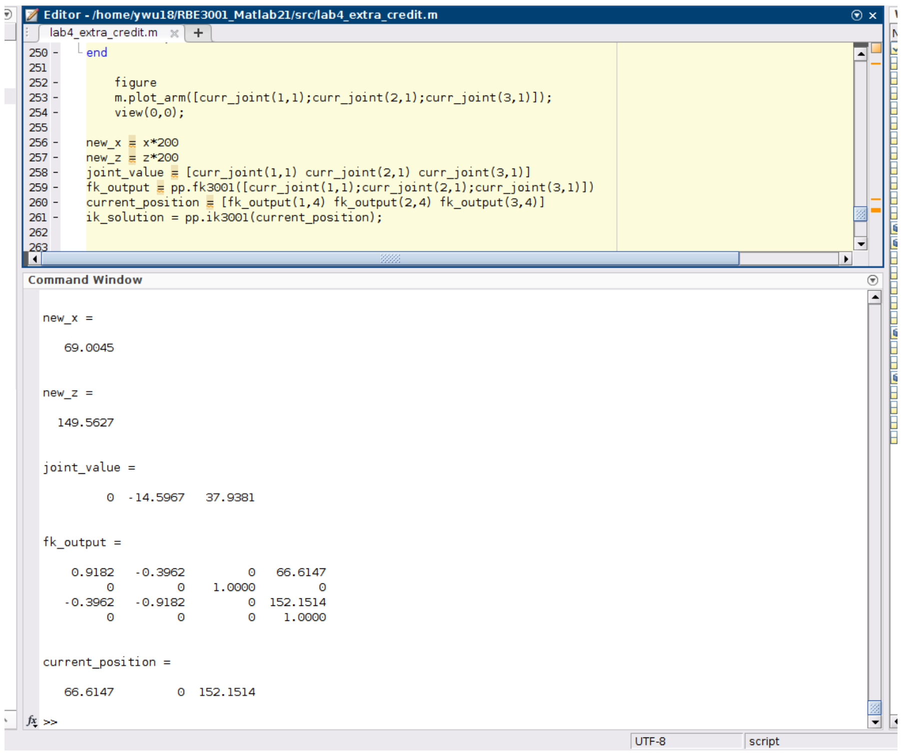


Figure.17: point 1 visual servo plot


Editor - /home/ywu18/RBE3001_Matlab21/src/lab4_extra_credit.m
lab4_extra_credit.m
Command Window
UTF-8
script

```
end

    figure
    m.plot_arm([curr_joint(1,1);curr_joint(2,1);curr_joint(3,1)]);
    view(0,0);

new_x = x*200
new_z = z*200
joint_value = [curr_joint(1,1) curr_joint(2,1) curr_joint(3,1)]
fk_output = pp.fk3001([curr_joint(1,1);curr_joint(2,1);curr_joint(3,1)])
current_position = [fk_output(1,4) fk_output(2,4) fk_output(3,4)]
ik_solution = pp.ik3001(current_position);
```

```
new_x =

   69.0045


new_z =

  149.5627


joint_value =

         0  -14.5967   37.9381


fk_output =

    0.9182   -0.3962         0   66.6147
         0         0    1.0000         0
   -0.3962   -0.9182         0  152.1514
         0         0         0    1.0000


current_position =

   66.6147         0  152.1514

>>
```

Figure.18: point 1 results

- For point 2

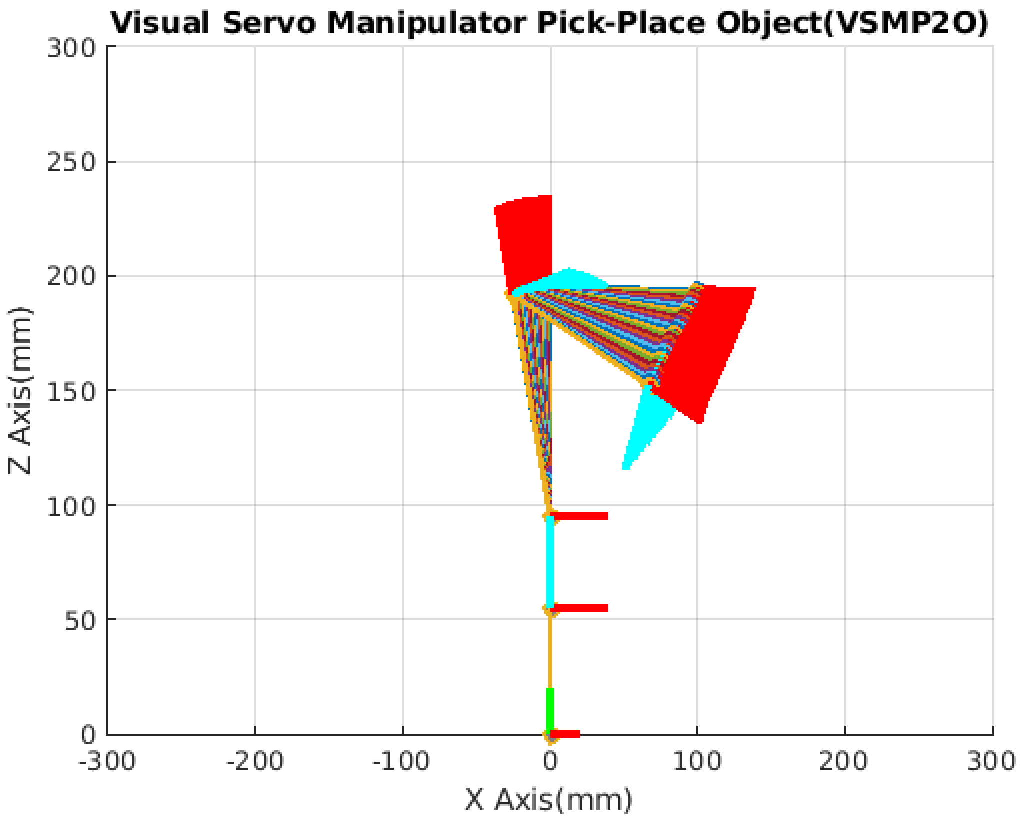
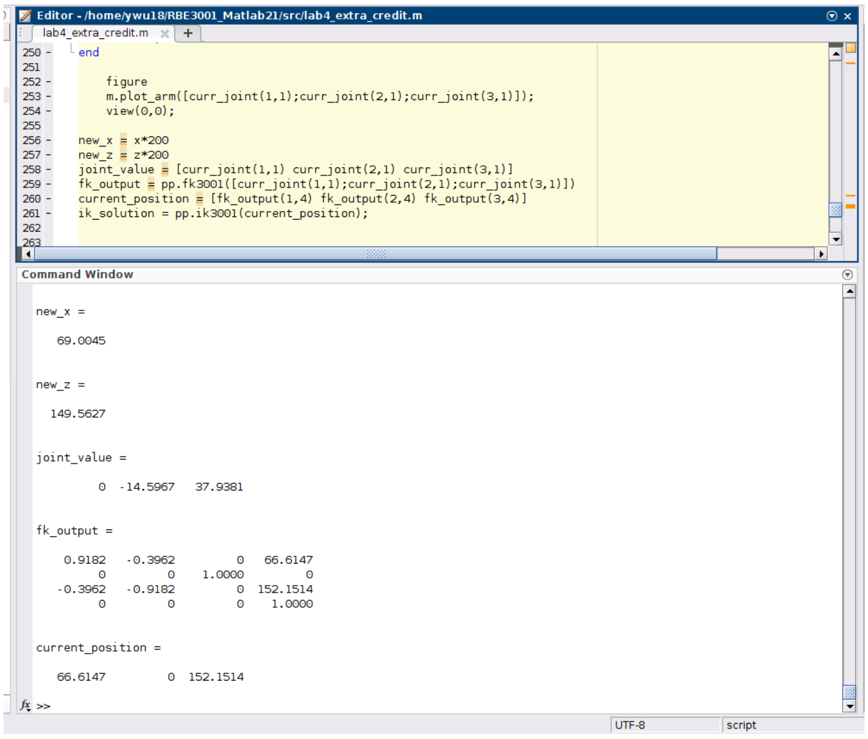


Figure.19: point 2 visual servo plot


Editor - /home/ywu18/RBE3001_Matlab21/src/lab4_extra_credit.m
lab4_extra_credit.m
Command Window
UTF-8
script

```
  end

      figure
      m.plot_arm([curr_joint(1,1);curr_joint(2,1);curr_joint(3,1)]);
      view(0,0);

  new_x = x*200
  new_z = z*200
  joint_value = [curr_joint(1,1) curr_joint(2,1) curr_joint(3,1)]
  fk_output = pp.fk3001([curr_joint(1,1);curr_joint(2,1);curr_joint(3,1)])
  current_position = [fk_output(1,4) fk_output(2,4) fk_output(3,4)]
  ik_solution = pp.ik3001(current_position);
```

```
new_x =

   69.0045


new_z =

  149.5627


joint_value =

         0  -14.5967   37.9381


fk_output =

    0.9182   -0.3962         0   66.6147
         0         0    1.0000         0
   -0.3962   -0.9182         0  152.1514
         0         0         0    1.0000


current_position =

   66.6147         0  152.1514

>>
```

Figure.20: point 2 results

- For point 3

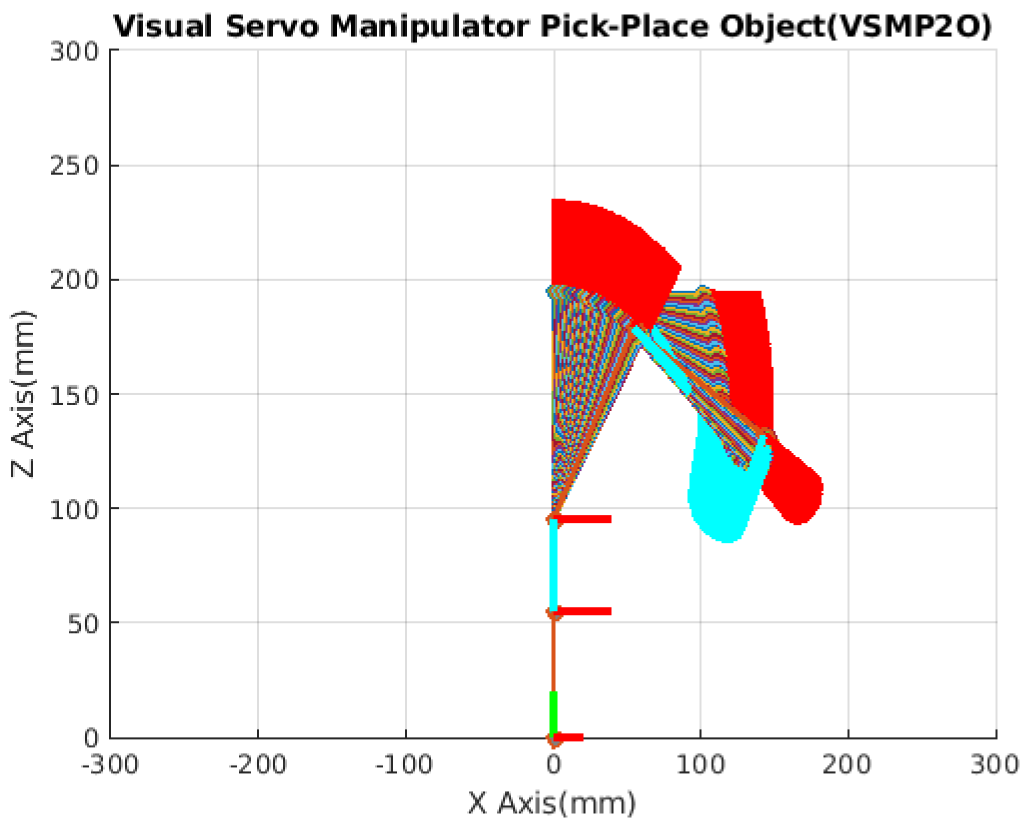
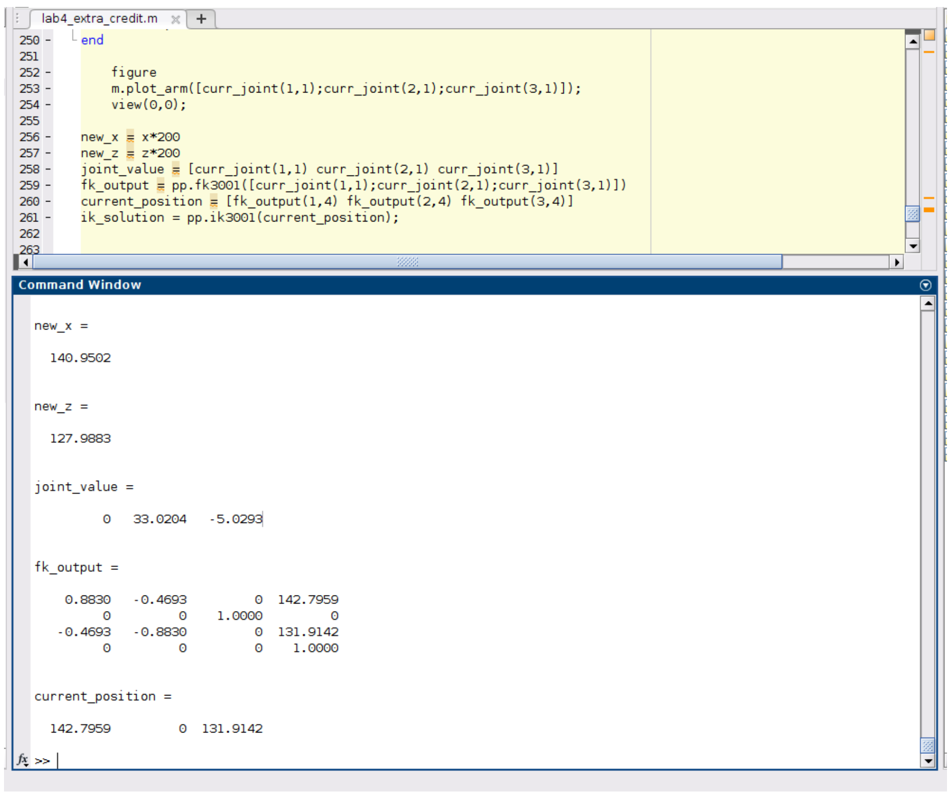


Figure.21: point 3 visual servo plot



```
end

    figure
    m.plot_arm([curr_joint(1,1);curr_joint(2,1);curr_joint(3,1)]);
    view(0,0);

new_x = x*200
new_z = z*200
joint_value = [curr_joint(1,1) curr_joint(2,1) curr_joint(3,1)]
fk_output = pp.fk3001([curr_joint(1,1);curr_joint(2,1);curr_joint(3,1)])
current_position = [fk_output(1,4) fk_output(2,4) fk_output(3,4)]
ik_solution = pp.ik3001(current_position);
```

```
new_x =

  140.9502


new_z =

  127.9883


joint_value =

         0   33.0204   -5.0293


fk_output =

    0.8830   -0.4693         0  142.7959
         0         0    1.0000         0
   -0.4693   -0.8830         0  131.9142
         0         0         0    1.0000


current_position =

  142.7959         0  131.9142

>>
```

Figure.22: point 3 result

Discussion

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

Instead of using the jacobian commend in the MATLAB toolbox, we choose to use our manual solution, because there was some error showing up when we tried to use the commend. In order to make the calculation more clear, we choose the first approach to calculate jacobian, which is the differential approach over the cross product approach. We took the partial derivative with respect to each joint angle for the upper Jacobian and found the third column of the rotation matrix with descending frames to calculate downward Jacobian. We then compare the result with previous labs' methods and the answers were close enough to consider it to be correct.

- **Validate your Jacobian calculation**

The first column of the Jp were all zeros, which meant that the coefficient of the first joint was zero. In other words, if there was any solution at all, then there must be an infinite number of them. (rank is reduced) The determinant zero represents the existence of singularity, which could lead to the previous analysis on the all zeros first column.

In Figure.4, with another singularity occasion, the first column still contained all zeros and it physically meant that MATLAB could not find the one only solution to the matrix. There were infinite many trajectories that could lead to that position.

- **Live plot of the task-space velocity vector**

Obviously, we use derivatives to find velocities and accelerations and that was why the pattern of the plots looked relatively similar, but sometimes pointing in the opposite direction.

For the 3D plot, the length of the vector is determined by the velocity when it approached the target position, the velocity decreased to nearly zero and correspondingly, the length decreased to nearly zero.

- **Discover and avoid singularities**

In our MATLAB code, since the determinants were relatively large, we set the threshold to be 100 so that the program was stopped when the determinant was smaller than 100 just as Figure.11 showed.

We need to prevent singularity, because when it happens, there are infinitely many solutions so that no exact solution could be found. If we do not do that, the robot arm may not always follow the trajectory as you planned and prevent the possible tragedy that could happen.

- **Extra 1: Velocity-based motion planning in task space**

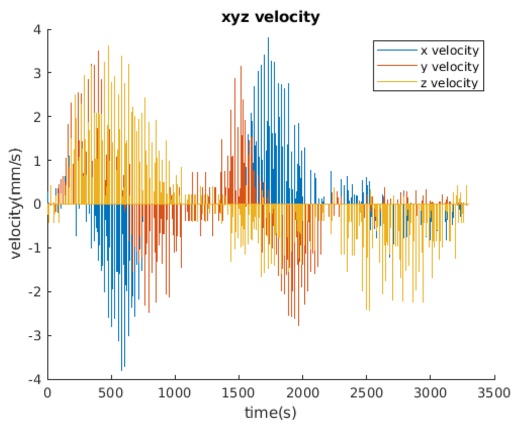


Figure.23: x, y, z position velocity

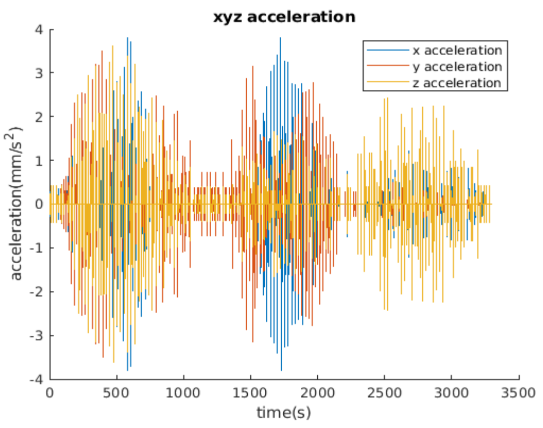


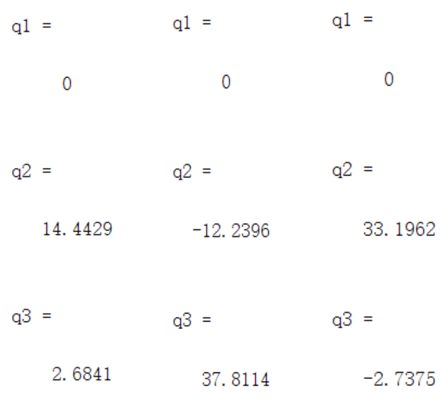
Figure.24: x, y, z position acceleration

Compare to the result from lab 3, there was not much change in the end point, because our velocity was not too large so that overshoot did not happen that much, but for the velocity and acceleration Figures shown above and compared to Figure.14 and 16, we can find that there was relatively small fluctuation this time, because the trajectory plan was based on velocity control in this lab and we wanted the velocities to approach to zero near end points and correspondingly the accelerations.

- **Extra 2: Numerical Inverse Kinematics**

In our MATLAB code, we set up a 5mm threshold of the x, z coordinates, so there could be some error occured in our results, but overall, the results were accurate enough compared to the ideal setpoints we gave. The threshold could not be eliminated, because when the velocities were relatively big, overshoot will likely happen at endpoints.

Also, based on the result from our code in lab 3 below, we can find that the difference between inverse kinematics was also small, which meant that our system was accurate. For point 1, 2 and 3, the answers in previous lab's code are shown below:



```
q1 =                q1 =                q1 =

       0                   0                   0


q2 =                q2 =                q2 =

    14.4429            -12.2396             33.1962


q3 =                q3 =                q3 =

     2.6841             37.8114             -2.7375
```

All the differences are within a few millimeters. The new method should be less accurate than the old one, because it is derived from the motion of the robot arm, but since we did not have a large system and a large velocity, the results are relatively consistent.

Figure.25: ik results from lab3 code

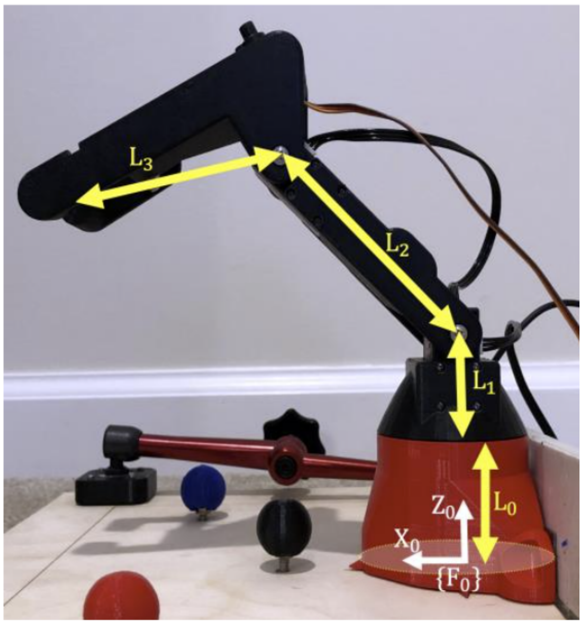
Conclusion

In this lab we successfully use MATLAB to verify Jacobian matrices 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. Then we tried to find out the forward velocity kinematics to make our system more accurate, because when reaching the target point, the velocity should approach 0 so that the overshoot can be reduced.

Finally, we use the determinant of Jacobian to prevent our system from reaching singularity. When the determinant is small enough, we consider it to be close to singularity and as a result, stop the program to prevent any tragedy from happening.

Luckily, we still had some time after we finished every essential sign-off, so we got our hands on the third method of calculating inverse kinematics. This approach relied on velocity kinematics rather than end point and the result was more accurate compared to previous lab, because we could also control the velocity when reaching end points so that we can reduce the errors.

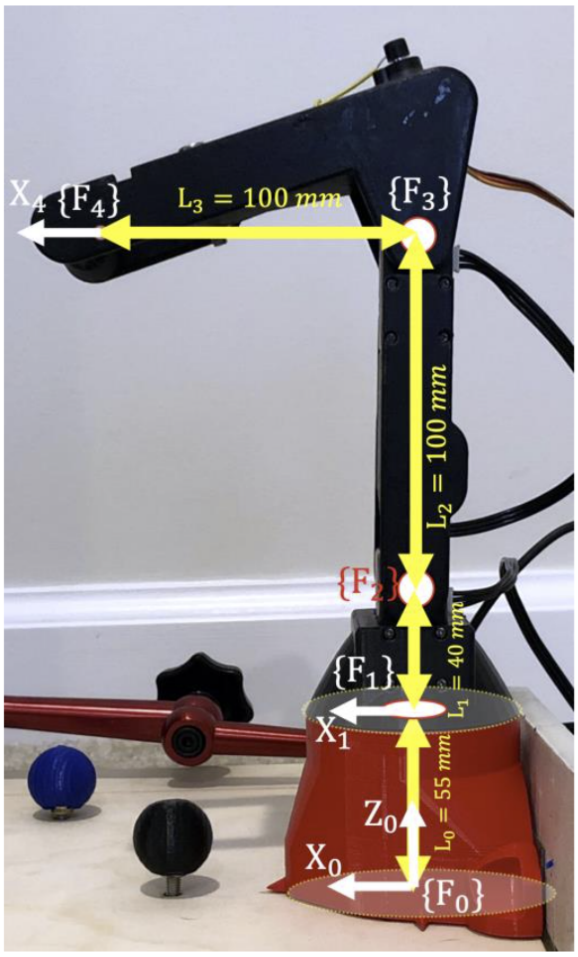
Appendix A: ARM CONFIGURATION



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