ENPM 662 – Introduction to Robot Modeling

Project – 2

Title: Robot for sorting surgical tools

Report by

1. **Dayanidhi Kandade - 120426711 (Team Leader)**
2. **Pon Aswin Sankaralingam – 121322517**
3. **Tirth Sadaria – 121322876**
4. **Anish Gupta - 121326812**

1.Introduction

Effective management of surgical instruments is crucial in orthopedic operating rooms. These procedures often rely on a diverse array of specialized tools to function. The traditional manual methods for sorting and sterilization are limited in speed, accuracy, and hygiene. Such constraints impact the workflow efficiency and introduce potential risks to patient safety.

To address these challenges, this project proposes a robotic system for the sorting and sterilization of surgical instruments. By utilizing the precision of a robotic manipulator and the advanced capabilities of computer vision. The system streamlines surgical instrument management and improves accuracy. The system also maintains strict hygiene standards to ensure a safer and more efficient surgical environment with the help of computer vision.

2. Application

The proposed system is designed to optimize surgical tool sorting in an orthopedic operating room. The system can be applied to:

1. **Automated Instrument Sorting**
   The robotic manipulator accurately identifies and categorizes surgical tools based on their unique requirements of the operator with the help of computer vision. This reduces reliance on manual intervention and minimizes errors. Thus, improving the overall preparation efficiency.
2. **Integrated Sterilization Workflow**
   UV panels can be easily integrated in the system to effortlessly sterilize the instruments. Also, the system ensures the cleanliness of the instruments by analyzing the data from its cameras. The system effectively reduces the risk of cross-contamination by minimizing manual handling of instruments. This ensures fulfilling the healthcare safety protocols and promoting patient safety.
3. **Adaptability Across Industrial cases**
   The versatility of this system enables it to be reconfigured according to the needs of unique industries. For example, with a few changes, the same system can be deployed to sort the ripe and unripe fruits.

3.Robot Type

For our project, we customized an industrial robotic arm, the UR10, for sorting surgical tools. The arm is securely mounted on a fixed cylindrical base, providing stability during operation. A camera is attached to the end effector to enable visual inspection and sorting of tools based on their types. Additionally, the modified UR10 is equipped with a vacuum gripper, allowing it to efficiently handle and organize surgical instruments.

4.DOFs and Dimensions

Degree of freedom – 6 (**6 revolute joints**)

The 6 DOF of the UR10 robotic arm makes it a highly capable and efficient solution for sorting surgical tools, offering the precision, flexibility, and dexterity necessary to meet the demands of this specialized task.

Dimensions:

The **base** in which the robot is mounted has a height of 1000mm and diameter of 500mm.

Robot arm:

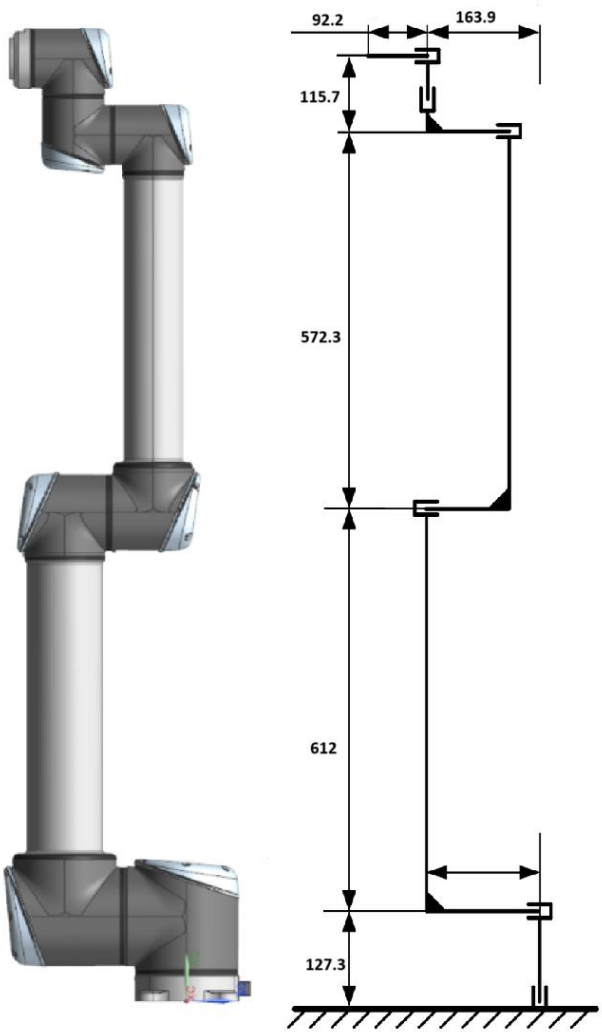


Fig 1: Dimension of robot arm

5.CAD Models

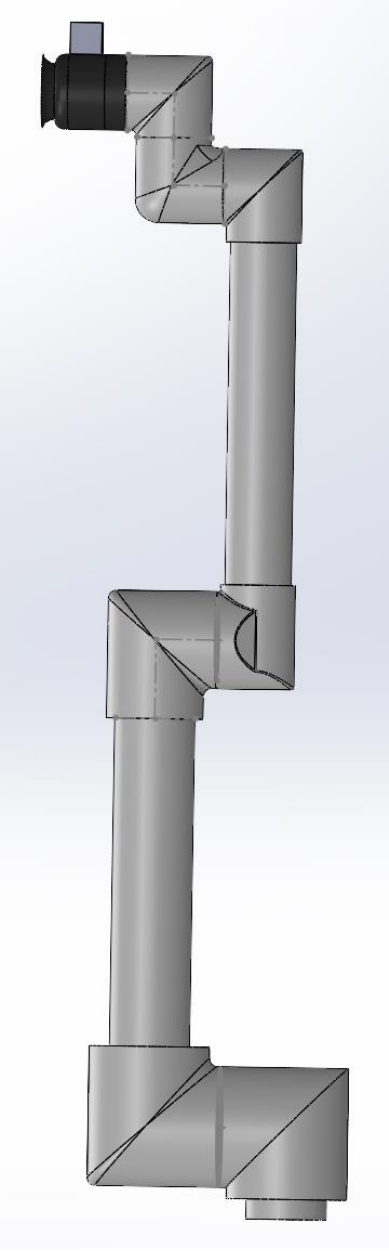


Fig 2: Customized Robot arm

Fig 3: Robot Base

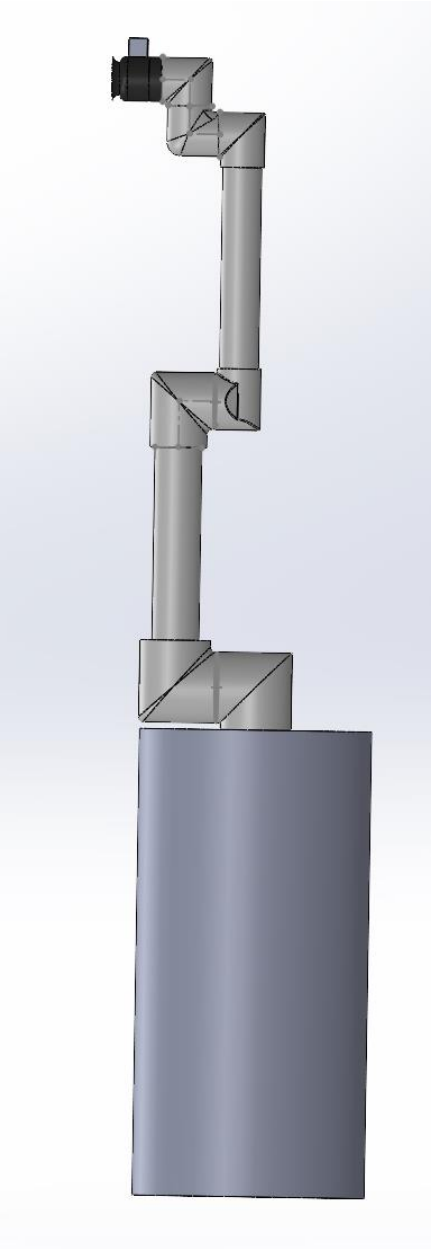


Fig 4: Assembly of robot arm with base and camera

6.Frame Assignment

The frames are assigned following the Denavit-Hartenberg (DH) convention to determine the DH parameters.

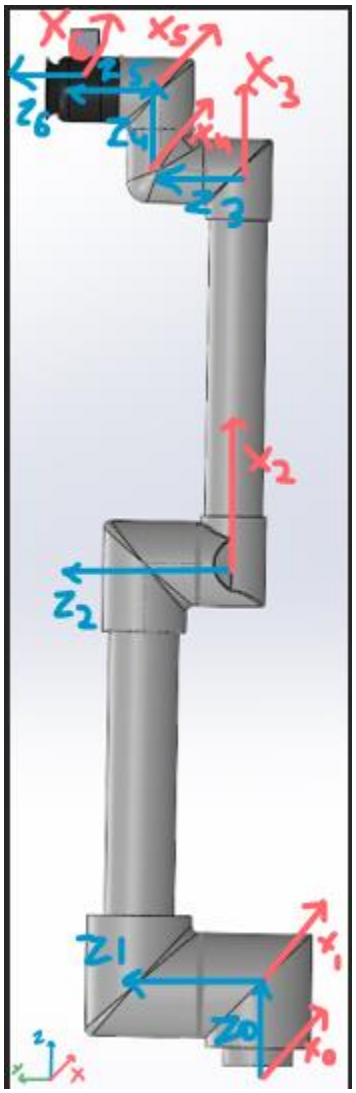


Fig 5: Frame assignment for robot arm

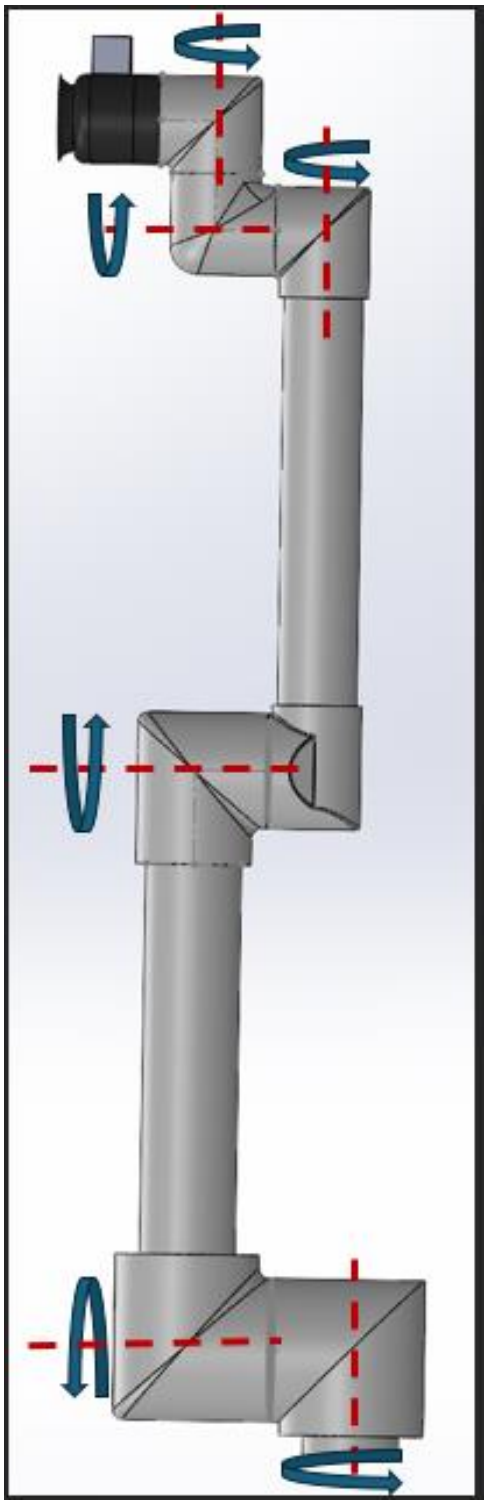


Fig 6: Joint rotations and directions for robot arm

7. D-H Parameters

For this project we have used the following D-H parameters as per our model's design:

UR10					
Kinematics	theta [rad]	a [m]	d [m]	alpha [rad]	**Dynamics**
Joint 1	0	0	0.1273	π/2	Link 1
Joint 2	0	-0.612	0	0	Link 2
Joint 3	0	-0.5723	0	0	Link 3
Joint 4	0	0	0.163941	π/2	Link 4
Joint 5	0	0	0.1157	-π/2	Link 5
Joint 6	0	0	0.0922	0	Link 6

7) Forward Kinematics

The objective of using forward kinematics is to position the end-effectors of the robotic manipulator. The process starts with the following steps:

1. Frame Assignment: The position and the orientation of each link with respect to the base frame can change due to the robot's movement. So, we need to define reference frames at each joint.

2. Denavit-Hartenberg (D-H parameters) Table: The D-H parameters must be assigned by the following rules to ensure the correct representation of actual robot.

    - The link length, or the distance measured along the X_n axis, is represented by the parameter 'a'.
    - The parameter 'd' represents the Zn-1 axis distance, also known as the Link, offset.
    - The Link twist around the X_n axis needed to align Z_{n-1} with Z_n, is represented by the parameter 'α'.

- The parameter" θ" represents the Joint angle, which is the angle of rotation around the Z_{n-1} axis that lines up X_{n-1} and X_n.

3. Transformation Matrix Calculation: The transformation matrices are computed using the DH parameters. The changes between one joint frame and the next are represented by these matrices. The overall transformation from the base to the end effector frame is obtained by multiplying these matrices. This provides the exact position and orientation of the end effector.

$$1T6 = 0T1 \cdot 1T2 \cdot 2T3 \cdot 3T4 \cdot 4T5 \cdot 5T6$$

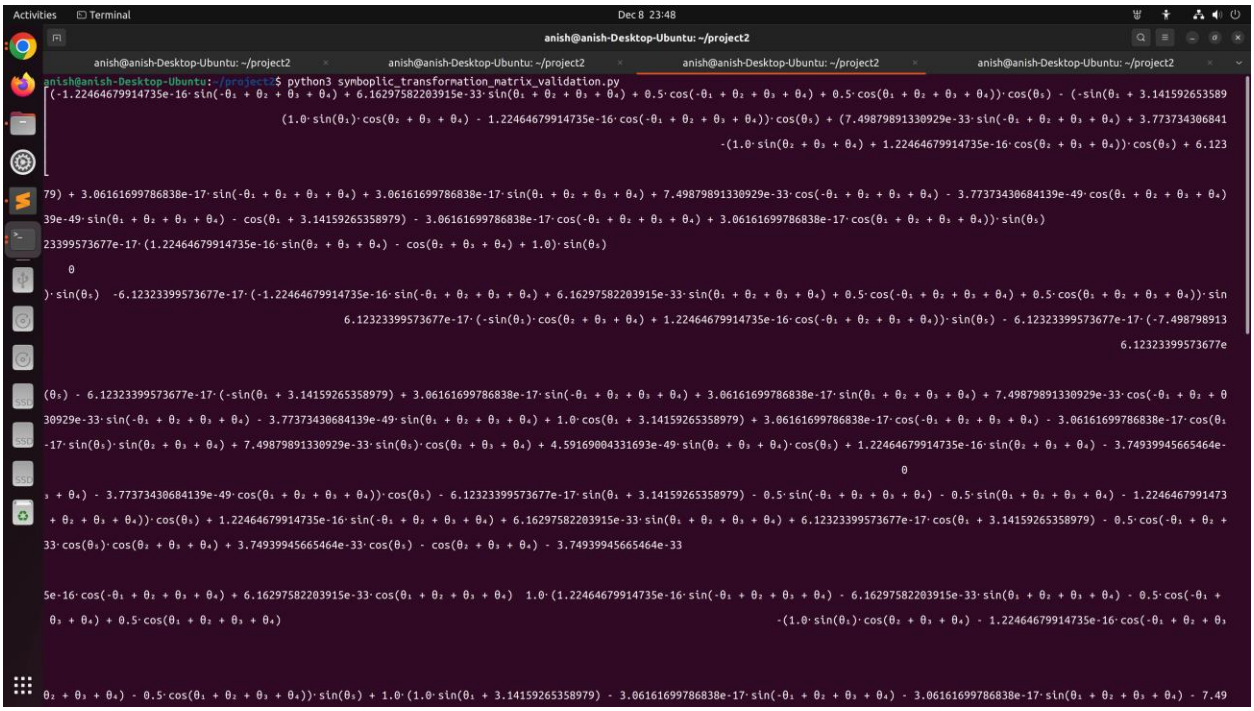
The transformation matrix output for our robot:

For all theta=0:

```
------------------------------------------------------------
------------------------------------------------------------
Transformation Matrix from frame 0 to 6 for all theta=0
[[ 1.    -0.     0.     0.   ]
 [-0.     0.     1.     0.256]
 [-0.    -1.     0.     1.427]
 [ 0.     0.     0.     1.   ]]
------------------------------------------------------------
------------------------------------------------------------
```

The parametric form can be obtained by running:

symboplic_transformation_matrix_validation.py



8.Inverse Kinematics

Inverse kinematics is like solving a puzzle for robots. In forward kinematics we provide the joint angles for the robot to move. For Inverse kinematics, the end-effector position is provided, and the program does the required joint angles calculations. The Jacobian matrix is like a tool that helps us solve this problem. It does this by looking at how changes in the joint angles affect the position and direction of the hand. So, by using the Jacobian matrix, we can calculate the right angles to make the robot's hand go where we want it to. The Jacobian matrix for a robotic system can be represented by:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \frac{\partial f_2}{\partial q_1} & \frac{\partial f_2}{\partial q_2} & \cdots & \frac{\partial f_2}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial q_1} & \frac{\partial f_m}{\partial q_2} & \cdots & \frac{\partial f_m}{\partial q_n} \end{bmatrix}$$

Where:

J : Jacobian matrix

f_i : Component function of the end effector

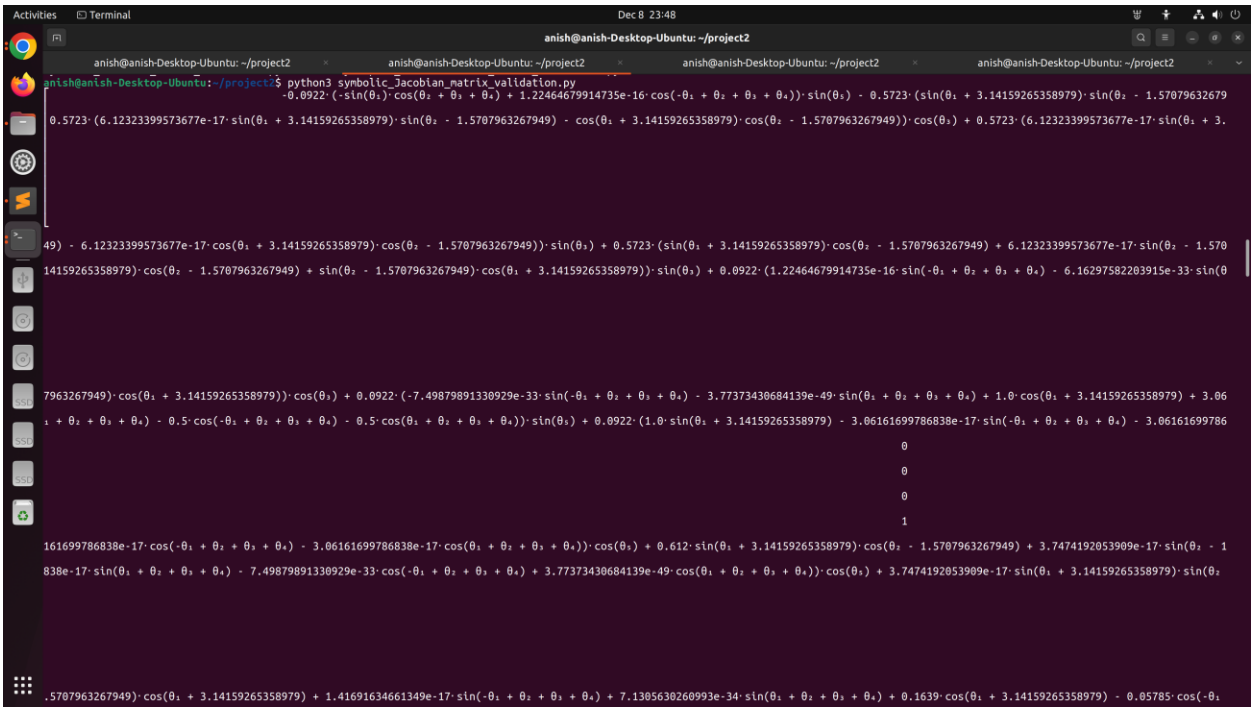
q_i : Joint variables

Our Jacobian matrix output:

For all theta = 0

```
anish@anish-Desktop-Ubuntu:~$ python3 Jacobian_matrix_validation.py
Jacobian Matrix:
[[-0.2561  1.3     0.688   0.1157 -0.0922 -0.    ]
 [ 0.     -0.     -0.     -0.      0.      0.    ]
 [ 0.     -0.     -0.     -0.      0.      0.    ]
 [ 0.      0.      0.      0.      0.      0.    ]
 [ 0.      1.      1.      1.     -0.      1.    ]
 [ 1.      0.      0.      0.      1.      0.    ]]
------------------------------------------------------------
------------------------------------------------------------
anish@anish-Desktop-Ubuntu:~$
```

The parametric form can be obtained by running symbolic_Jacobian_matrix_validation.py



Once we have Jacobian Matrix, we use the following steps for inverse kinematics calculations:

1. The Jacobian relates the end-effector velocity ($\dot{x}$) to the joint velocities ($\dot{\theta}$):

$$\dot{x} = J(\theta) \cdot \dot{\theta}$$

2. Using the inverse or pseudo-inverse of the Jacobian, solve for the joint velocities:

$$\dot{\theta} = J(\theta)^{\dagger} \cdot \dot{x}$$

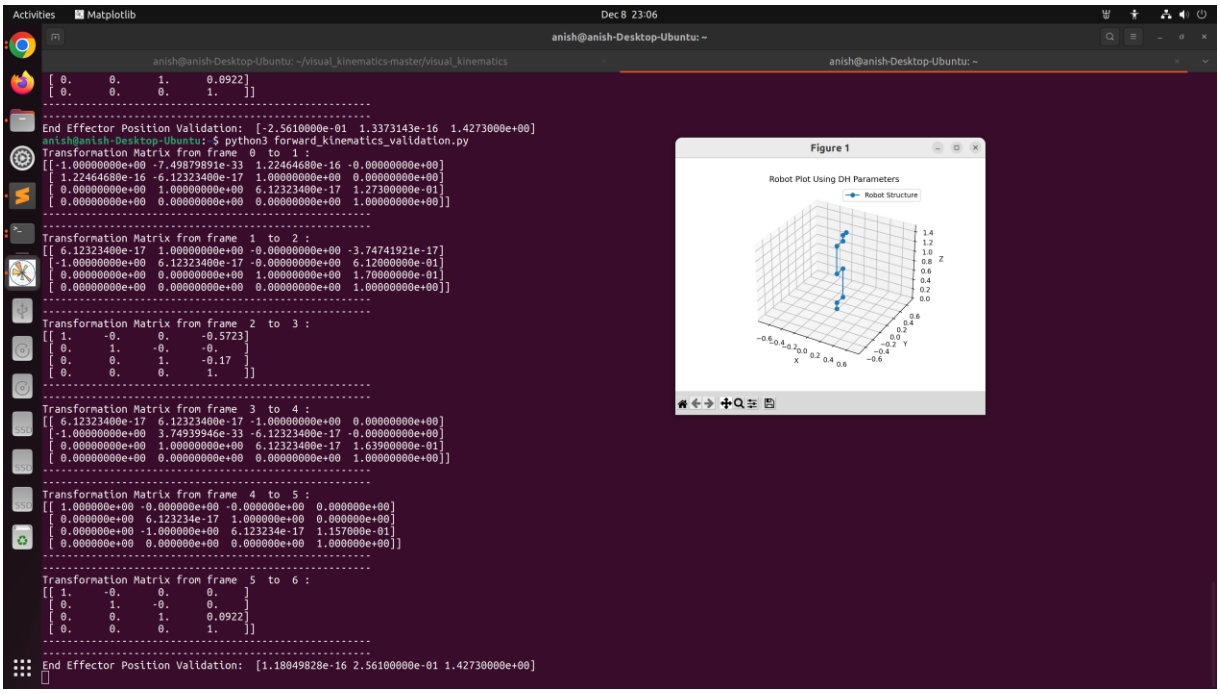
3. Multiply it by the desired velocity column vector

$$\theta(t + \Delta t) = \theta(t) + \dot{\theta}(t) \cdot \Delta t$$

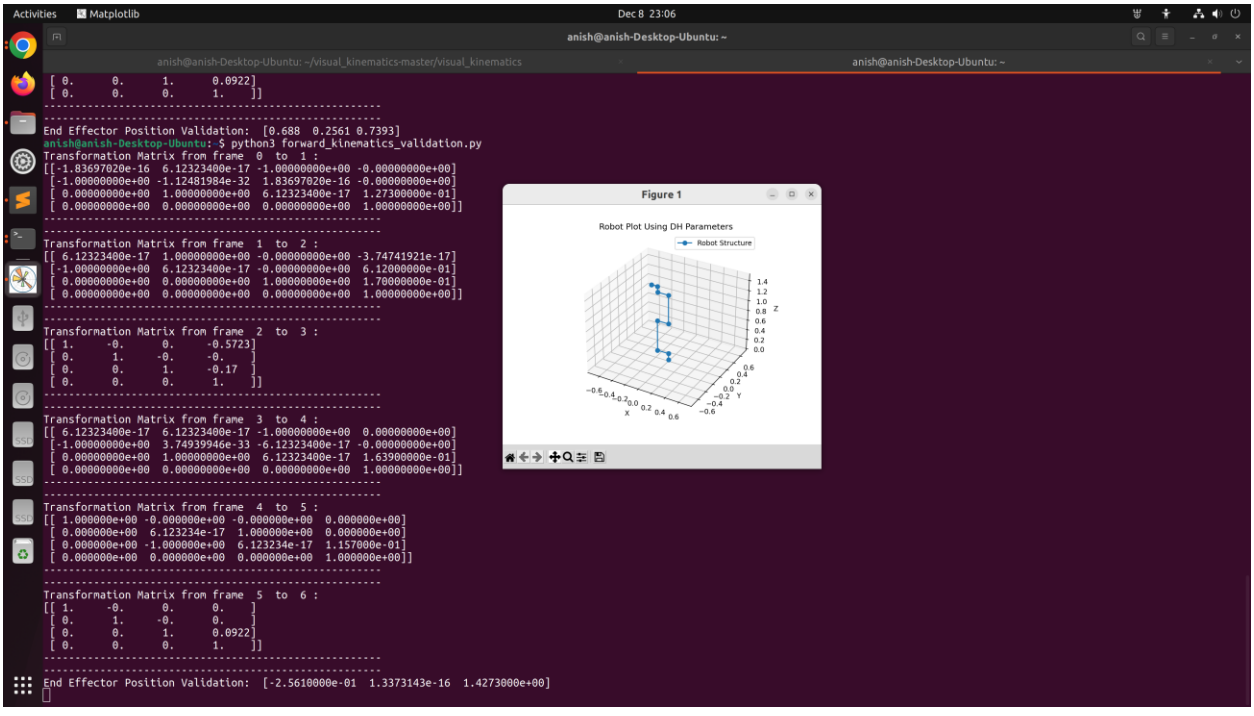
4. Get the desired trajectory.

9. Forward Kinematics Validation

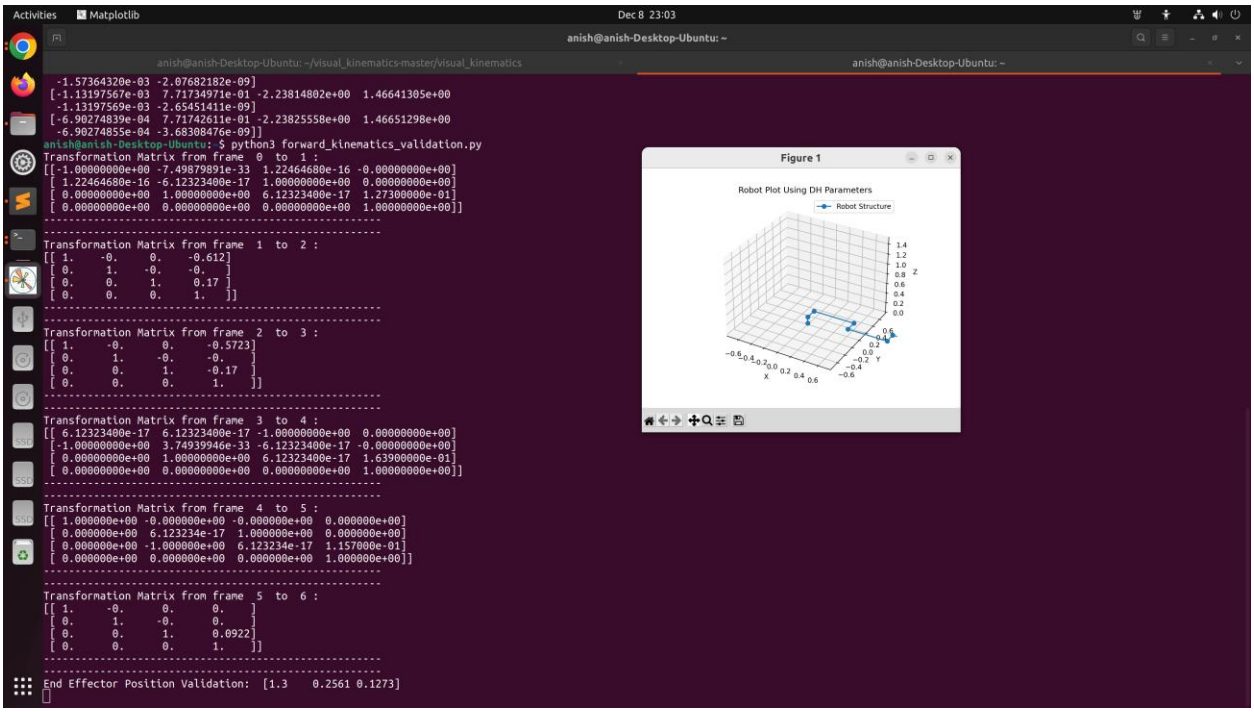
Case 1: All theta=0



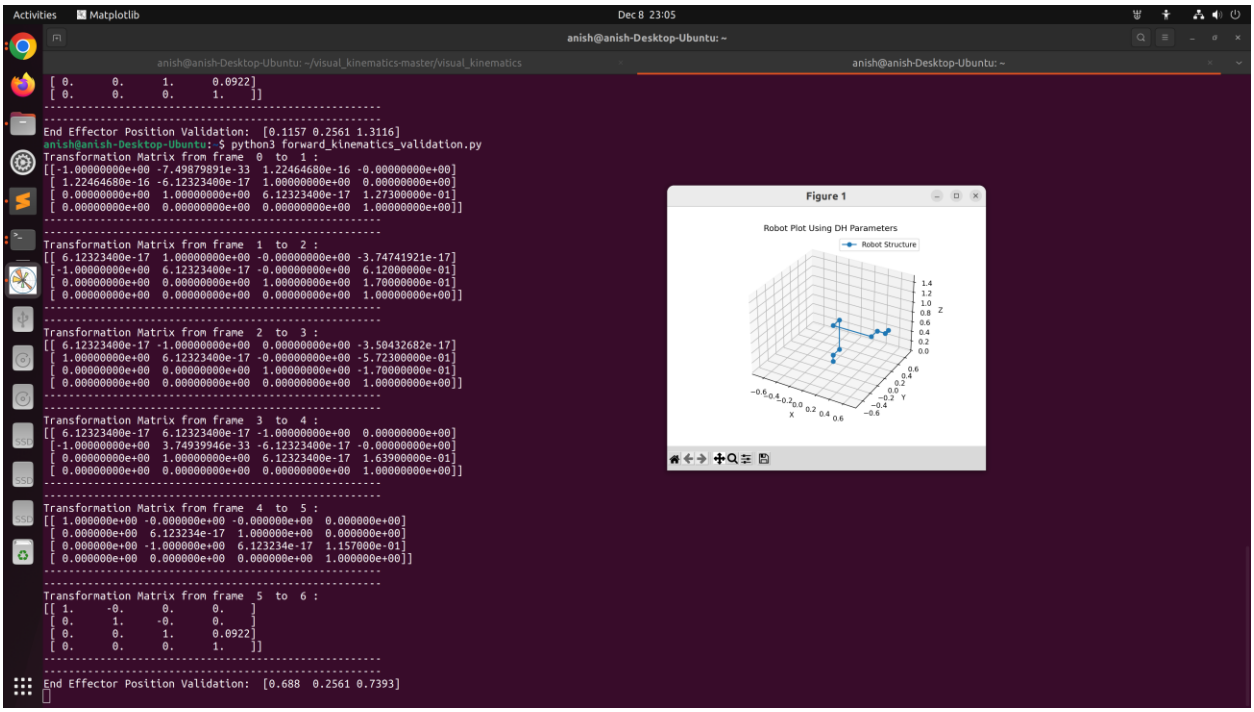
Case 2: Theta1=90



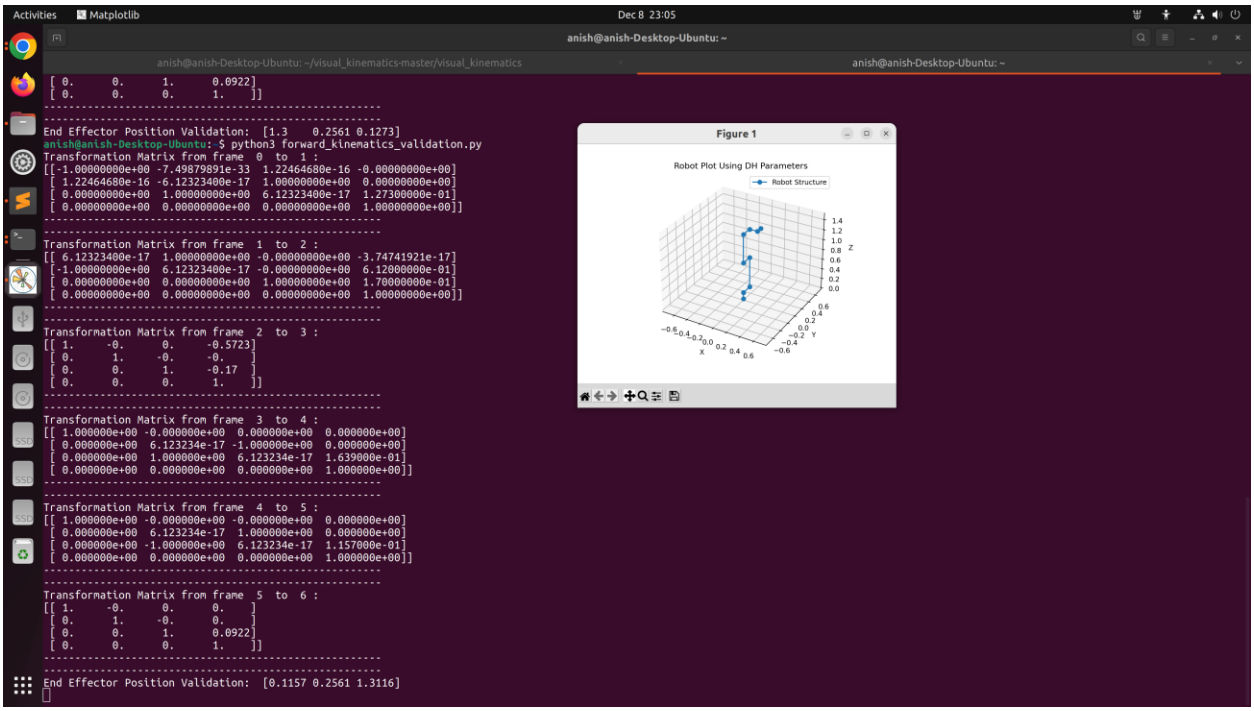
Case 3: Theta2=90



Case 4: Theta 3 =90

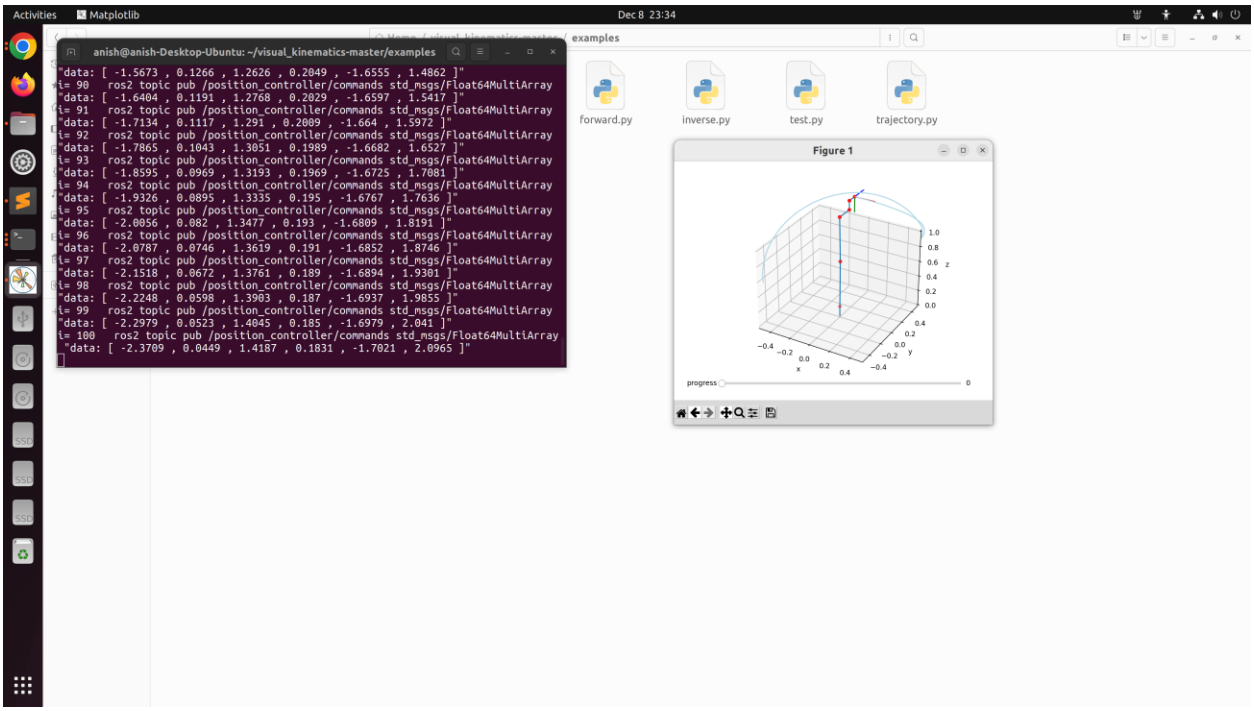


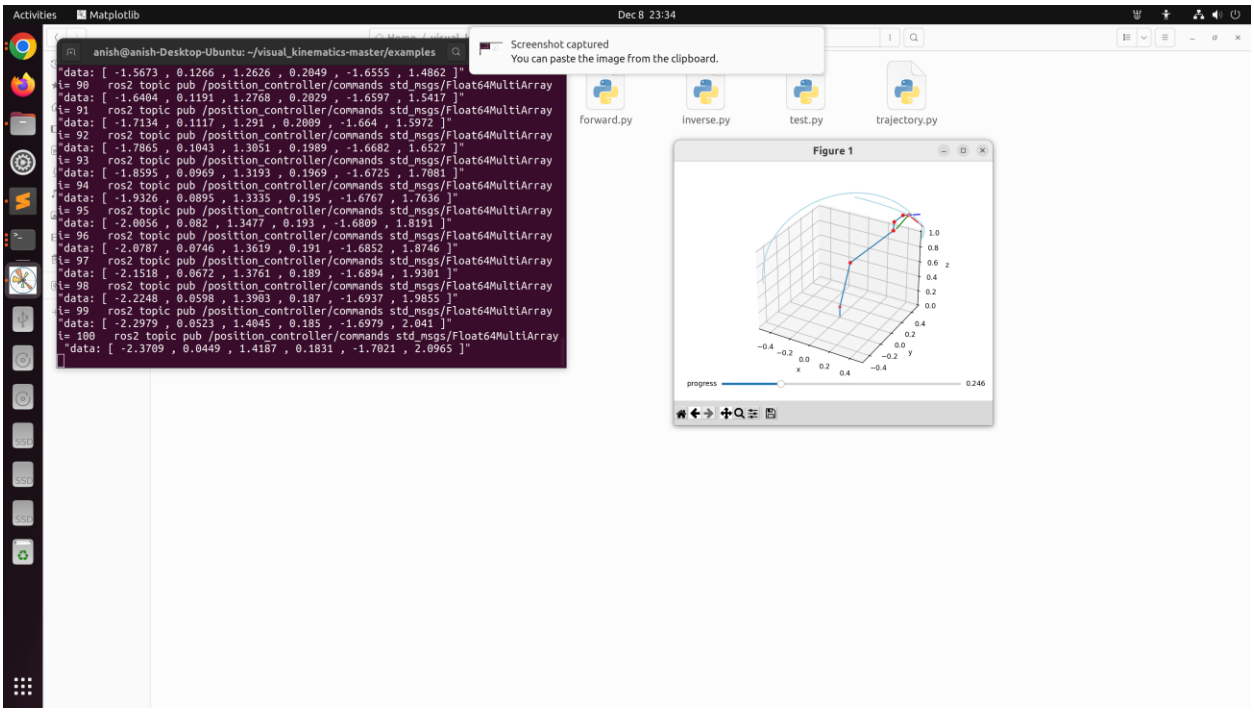
Case 5: Theta4 = 90

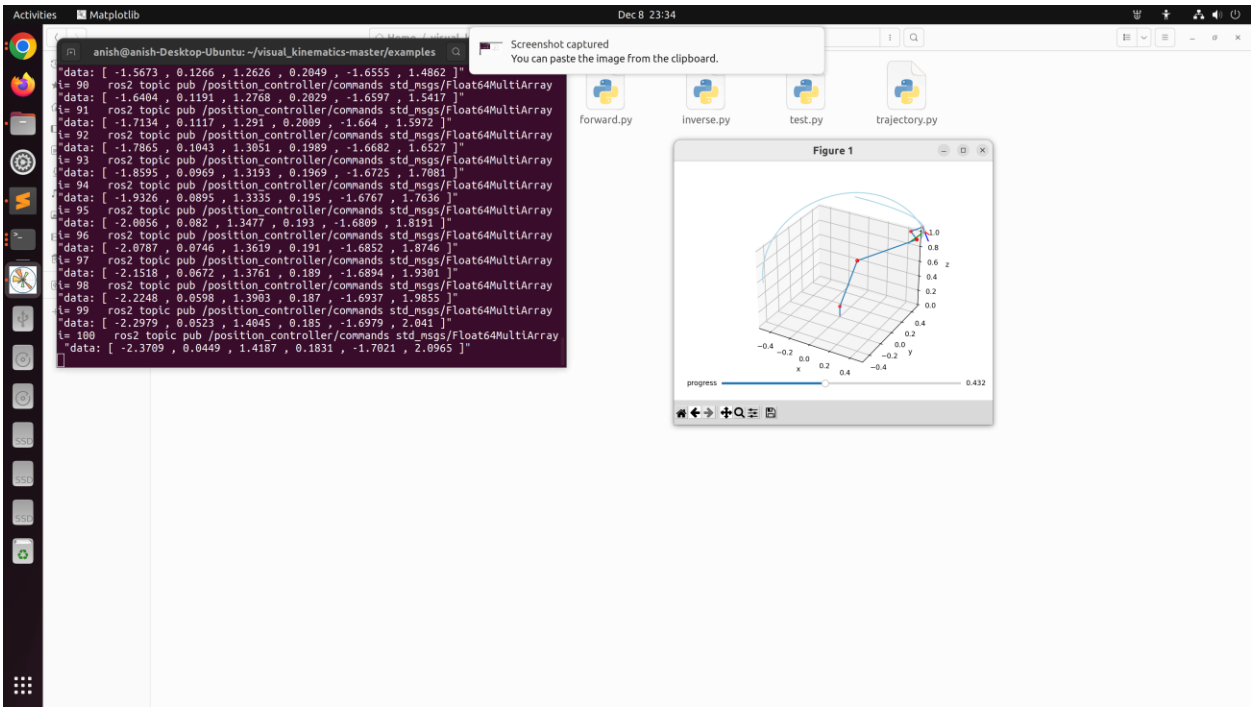


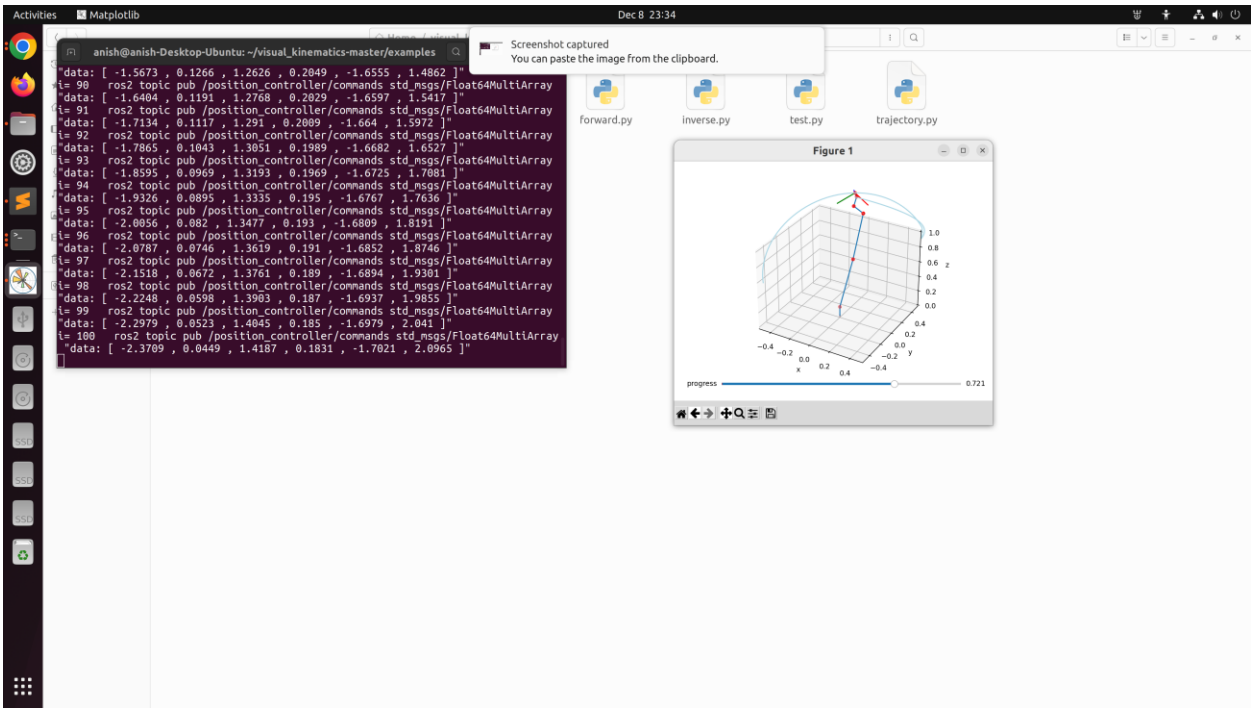
10. Inverse Kinematics Validation:

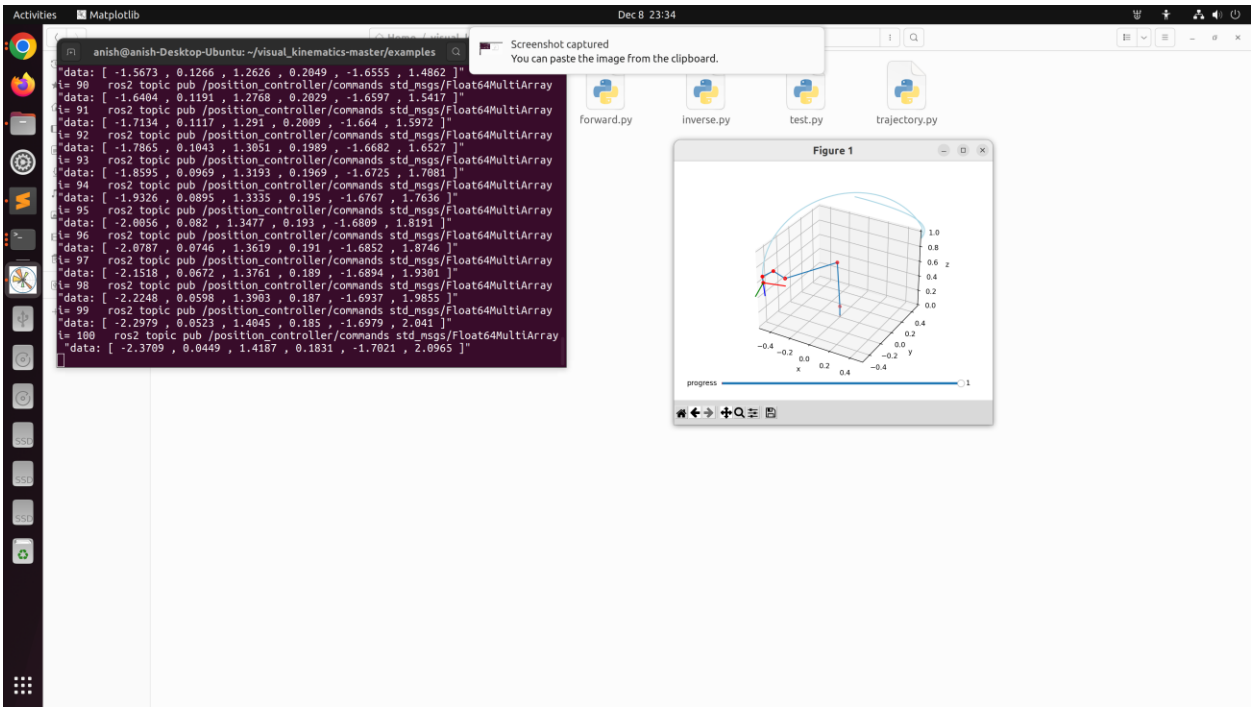
To validate the inverse kinematics, we are trying to move along the blue line in the trajectory below

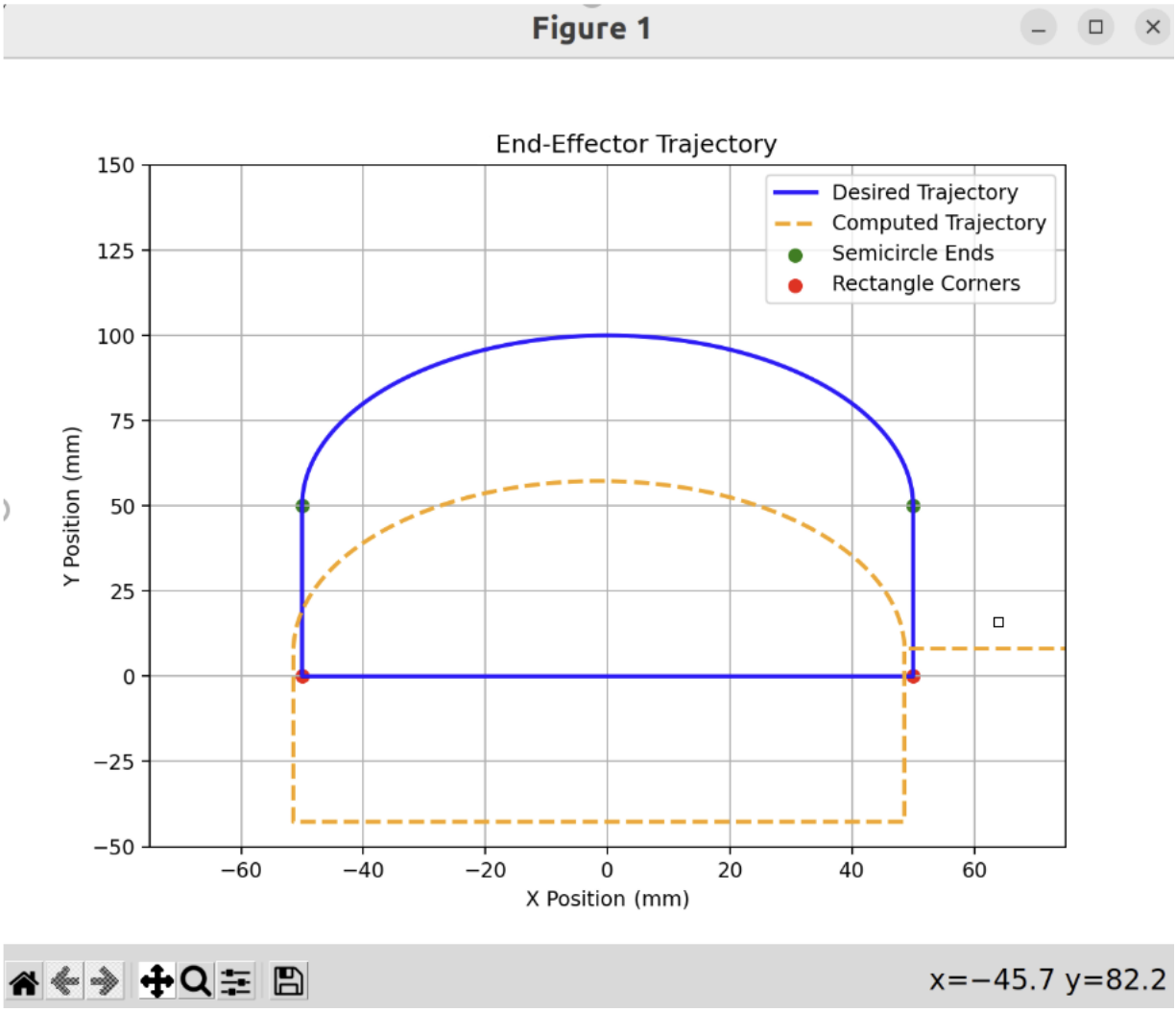

Activities
Matplotlib
Dec 8 23:34
anish@anish-Desktop-Ubuntu: ~/visual_kinematics-master/examples
"data: [-1.5673 , 0.1266 , 1.2626 , 0.2049 , -1.6555 , 1.4862]"
i= 90 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.6404 , 0.1191 , 1.2768 , 0.2029 , -1.6597 , 1.5417]"
i= 91 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7134 , 0.1117 , 1.291 , 0.2009 , -1.664 , 1.5972]"
i= 92 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7865 , 0.1043 , 1.3051 , 0.1989 , -1.6682 , 1.6527]"
i= 93 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.8595 , 0.0969 , 1.3193 , 0.1969 , -1.6725 , 1.7081]"
i= 94 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.9326 , 0.0895 , 1.3335 , 0.195 , -1.6767 , 1.7636]"
i= 95 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0056 , 0.082 , 1.3477 , 0.193 , -1.6809 , 1.8191]"
i= 96 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0787 , 0.0746 , 1.3619 , 0.191 , -1.6852 , 1.8746]"
i= 97 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.1518 , 0.0672 , 1.3761 , 0.189 , -1.6894 , 1.9301]"
i= 98 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2248 , 0.0598 , 1.3903 , 0.187 , -1.6937 , 1.9855]"
i= 99 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2979 , 0.0523 , 1.4045 , 0.185 , -1.6979 , 2.041]"
i= 100 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.3709 , 0.0449 , 1.4187 , 0.1831 , -1.7021 , 2.0965]"
examples
forward.py
inverse.py
test.py
trajectory.py
Figure 1
progress
0


Activities
Matplotlib
Dec 8 23:34
Screenshot captured
You can paste the image from the clipboard.
anish@anish-Desktop-Ubuntu: ~/visual_kinematics-master/examples
"data: [-1.5673 , 0.1266 , 1.2626 , 0.2049 , -1.6555 , 1.4862]"
i= 90 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.6404 , 0.1191 , 1.2768 , 0.2029 , -1.6597 , 1.5417]"
i= 91 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7134 , 0.1117 , 1.291 , 0.2009 , -1.664 , 1.5972]"
i= 92 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7865 , 0.1043 , 1.3051 , 0.1989 , -1.6682 , 1.6527]"
i= 93 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.8595 , 0.0969 , 1.3193 , 0.1969 , -1.6725 , 1.7081]"
i= 94 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.9326 , 0.0895 , 1.3335 , 0.195 , -1.6767 , 1.7636]"
i= 95 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0056 , 0.082 , 1.3477 , 0.193 , -1.6809 , 1.8191]"
i= 96 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0787 , 0.0746 , 1.3619 , 0.191 , -1.6852 , 1.8746]"
i= 97 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.1518 , 0.0672 , 1.3761 , 0.189 , -1.6894 , 1.9301]"
i= 98 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2248 , 0.0598 , 1.3903 , 0.187 , -1.6937 , 1.9855]"
i= 99 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2979 , 0.0523 , 1.4045 , 0.185 , -1.6979 , 2.041]"
i= 100 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.3709 , 0.0449 , 1.4187 , 0.1831 , -1.7021 , 2.0965]"
forward.py
inverse.py
test.py
trajectory.py
Figure 1
progress
0.246


Activities
Matplotlib
Dec 8 23:34
anish@anish-Desktop-Ubuntu: ~/visual_kinematics-master/examples
"data: [-1.5673 , 0.1266 , 1.2626 , 0.2049 , -1.6555 , 1.4862]"
i= 90 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.6404 , 0.1191 , 1.2768 , 0.2029 , -1.6597 , 1.5417]"
i= 91 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7134 , 0.1117 , 1.291 , 0.2009 , -1.664 , 1.5972]"
i= 92 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7865 , 0.1043 , 1.3051 , 0.1989 , -1.6682 , 1.6527]"
i= 93 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.8595 , 0.0969 , 1.3193 , 0.1969 , -1.6725 , 1.7081]"
i= 94 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.9326 , 0.0895 , 1.3335 , 0.195 , -1.6767 , 1.7636]"
i= 95 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0056 , 0.082 , 1.3477 , 0.193 , -1.6809 , 1.8191]"
i= 96 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0787 , 0.0746 , 1.3619 , 0.191 , -1.6852 , 1.8746]"
i= 97 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.1518 , 0.0672 , 1.3761 , 0.189 , -1.6894 , 1.9301]"
i= 98 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2248 , 0.0598 , 1.3903 , 0.187 , -1.6937 , 1.9855]"
i= 99 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2979 , 0.0523 , 1.4045 , 0.185 , -1.6979 , 2.041]"
i= 100 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.3709 , 0.0449 , 1.4187 , 0.1831 , -1.7021 , 2.0965]"
Screenshot captured
You can paste the image from the clipboard.
forward.py
inverse.py
test.py
trajectory.py
Figure 1
progress
0.432

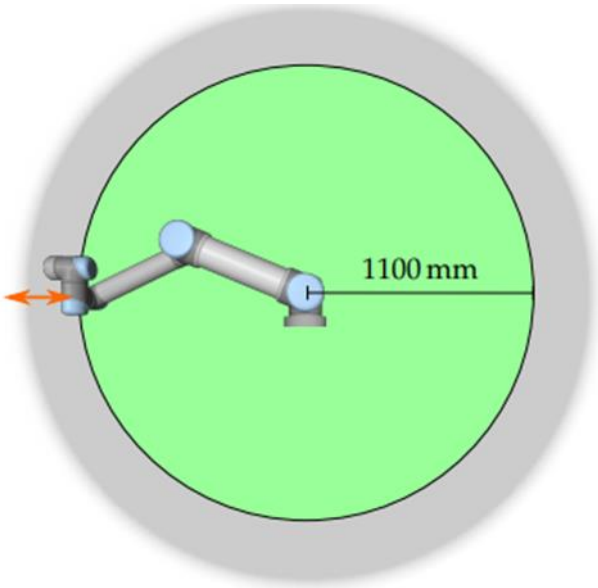

Activities
Matplotlib
Dec 8 23:34
anish@anish-Desktop-Ubuntu: ~/visual_kinematics-master/examples
"data: [-1.5673 , 0.1266 , 1.2626 , 0.2049 , -1.6555 , 1.4862]"
i= 90 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.6404 , 0.1191 , 1.2768 , 0.2029 , -1.6597 , 1.5417]"
i= 91 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7134 , 0.1117 , 1.291 , 0.2009 , -1.664 , 1.5972]"
i= 92 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7865 , 0.1043 , 1.3051 , 0.1989 , -1.6682 , 1.6527]"
i= 93 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.8595 , 0.0969 , 1.3193 , 0.1969 , -1.6725 , 1.7081]"
i= 94 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.9326 , 0.0895 , 1.3335 , 0.195 , -1.6767 , 1.7636]"
i= 95 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0056 , 0.082 , 1.3477 , 0.193 , -1.6809 , 1.8191]"
i= 96 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0787 , 0.0746 , 1.3619 , 0.191 , -1.6852 , 1.8746]"
i= 97 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.1518 , 0.0672 , 1.3761 , 0.189 , -1.6894 , 1.9301]"
i= 98 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2248 , 0.0598 , 1.3903 , 0.187 , -1.6937 , 1.9855]"
i= 99 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2979 , 0.0523 , 1.4045 , 0.185 , -1.6979 , 2.041]"
i= 100 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.3709 , 0.0449 , 1.4187 , 0.1831 , -1.7021 , 2.0965]"
Screenshot captured
You can paste the image from the clipboard.
forward.py
inverse.py
test.py
trajectory.py
Figure 1
progress
0.721


Activities
Matplotlib
Dec 8 23:34
Screenshot captured
You can paste the image from the clipboard.
anish@anish-Desktop-Ubuntu: ~/visual_kinematics-master/examples
"data: [-1.5673 , 0.1266 , 1.2626 , 0.2049 , -1.6555 , 1.4862]"
i= 90 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.6404 , 0.1191 , 1.2768 , 0.2029 , -1.6597 , 1.5417]"
i= 91 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7134 , 0.1117 , 1.291 , 0.2009 , -1.664 , 1.5972]"
i= 92 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.7865 , 0.1043 , 1.3051 , 0.1989 , -1.6682 , 1.6527]"
i= 93 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.8595 , 0.0969 , 1.3193 , 0.1969 , -1.6725 , 1.7081]"
i= 94 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-1.9326 , 0.0895 , 1.3335 , 0.195 , -1.6767 , 1.7636]"
i= 95 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0056 , 0.082 , 1.3477 , 0.193 , -1.6809 , 1.8191]"
i= 96 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.0787 , 0.0746 , 1.3619 , 0.191 , -1.6852 , 1.8746]"
i= 97 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.1518 , 0.0672 , 1.3761 , 0.189 , -1.6894 , 1.9301]"
i= 98 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2248 , 0.0598 , 1.3903 , 0.187 , -1.6937 , 1.9855]"
i= 99 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.2979 , 0.0523 , 1.4045 , 0.185 , -1.6979 , 2.041]"
i= 100 ros2 topic pub /position_controller/commands std_msgs/Float64MultiArray
"data: [-2.3709 , 0.0449 , 1.4187 , 0.1831 , -1.7021 , 2.0965]"
forward.py
inverse.py
test.py
trajectory.py
Figure 1
progress



11. Workspace Study

The advantage of using a commercially available robot is that the workspace is predefined by the company. For this project we used UR10 robot which has a spherical workspace but for our project we used only the upper hemisphere. Th UR10 has a maximum reach of 1100mm. We also verified the workspace and D-H parameters.



12) Assumptions

We have used the following assumptions for our project:

- The air resistance and friction are considered negligible.
- All the instruments to be sorted are properly recognizable.
- The room is well lit for efficient working of the camera.
- Pick and place locations are obstruction free and ideally predefined.

13. Control Method

1**. Segment Length Calculation**:

- The lengths of trajectory segments are calculated by combining translational and rotational components:

2. **Interpolation**:

- The interpolate function generates intermediate points along the trajectory using **linear interpolation** between two consecutive frames.

$$\text{len\_segs} = \text{len\_segs\_tran} + \frac{\text{len\_segs\_rot} \cdot \text{rot\_tran\_ratio}}{2\pi}$$

For point-to-point:

$$\text{interpolated values} = (1 - p_{\text{temp}}) \cdot \text{start\_values} + p_{\text{temp}} \cdot \text{end\_values}$$

For linear (lin) motion:

- The Cartesian coordinates and orientation are interpolated linearly.
- The robot's inverse kinematics is then applied to map the interpolated pose to joint angles.

3. **Time Scaling**:

- The interpolation is scaled across the trajectory's total length and optionally adjusts based on provided time_points.

4. **Numerical Integration**:

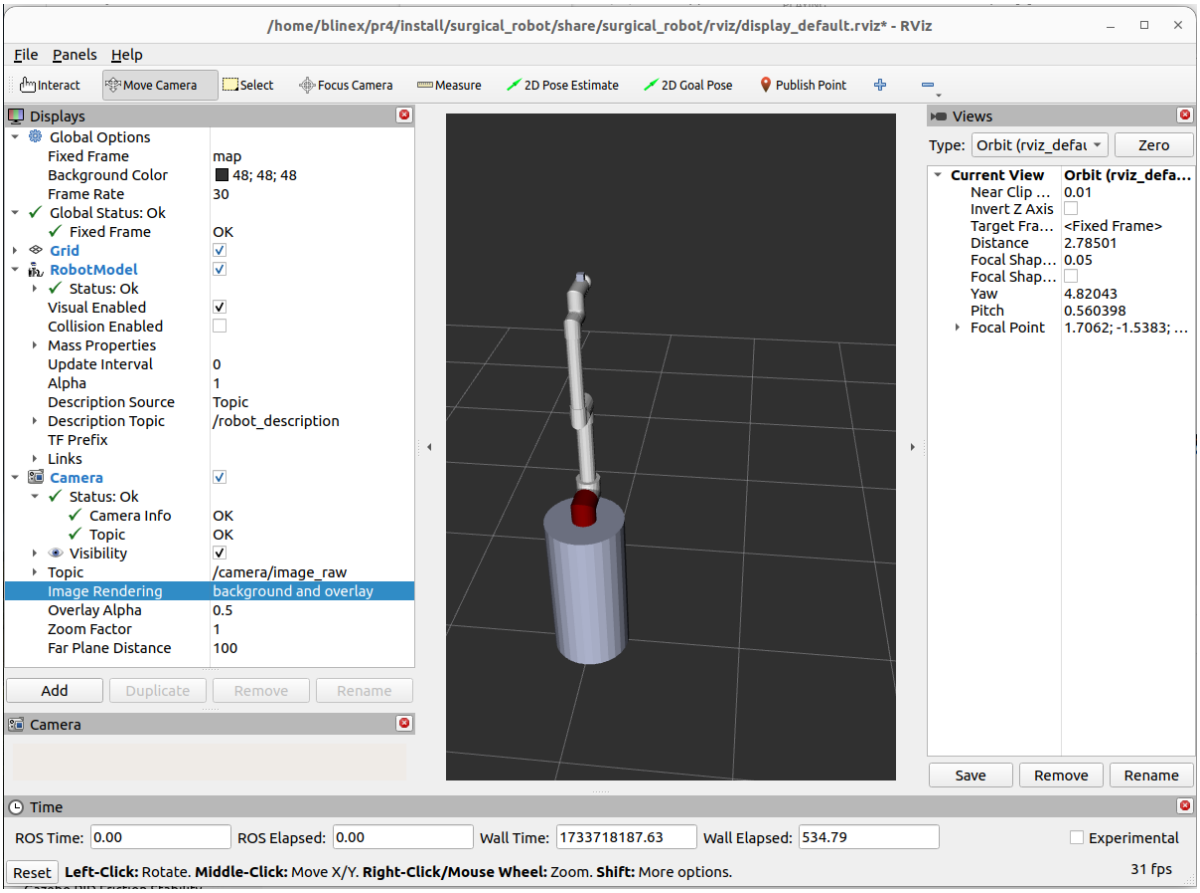
- The code iteratively interpolates over num_segs, creating a smooth sequence of intermediate values along the trajectory

14.Gazebo/RViz Visualization

Please follow this link to see the gazebo simulation of our project:

https://drive.google.com/file/d/1_iLr0wzLZfR-Hlb7MDlvRRr9vcySXbf1/view?usp=sharing

- **Rviz Visualization**



15.Problems Faced

Our group faced several hurdles while completing this project. The correct frame assignment and URDF generation proved to be a challenge. We also faced several issues importing our project in the gazebo initially. Both the problems were solved using help from internet forums. Using vacuum gripper and camera correctly also required quite a lot of troubleshooting for it to work correctly.

16.Lessons Learned

The project provided a great insight into modelling and simulation of robots using ROS2 and gazebo. We were able to apply the inverse kinematic techniques taught in class directly to our robot. This helped us bridge the gap between theory and practical work. We also learned to overcome countless hurdles and challenges while completing the project.

17.Conclusions

The goal of our project was to use a robot to sort and place the tools in a surgical environment. To achieve this, we designed a robot with vacuum gripper and integrated camera. We started with designing the robot, we used the UR10 robot for our project. Then we used D-H parameters to calculate the required forward and inverse kinematic equations. Inverse kinematics provided the trajectory for the robot to follow. Camera feed was used along with OpenCV to classify the tools based on the predefined criterion. We used ROS2 and gazebo to simulate the robot's movements. The robot successfully completed all the tasks it was designed for and can be made industry ready with some changes. The performance of this robot can be improved by further testing and refinement of the model.

18. Future Work

This project can be further developed to use advanced and state of art AI technology to identify and sort the instrument better. This project can be modified with various attachments to cater to the demands of multiple industries.

19.Links to GitHub Repository and demonstration videos

Video: https://drive.google.com/file/d/1_iLr0wzLZfR-Hlb7MDlvRRr9vcySXbf1/view?usp=sharing

Github: https://github.com/Kandade-Dayanidhi/ENPM662_Project_2.git

20.References

1. https://enpm-662introduction-to-robotmodelling.readthedocs.io/en/latest/index.html

2. Robot modeling and control - Mark W. Spong, Seth Hutchinson, and M. Vidyasagar

3. https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/