

# **EE430 Robot Dynamics and Control**

**Report of Project**

**(RRRR) Pick and Place Robot**

**By**

**Harsh Nahata(201EE123)**

**Sumukh C Prakash(201EE160)**

**Vinamra Parakh(201EE169)**

**7<sup>th</sup> Semester, BTech Student**

**Department of Electrical and Electronics  
Engineering**



**NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,**

**SURATHKAL, MANGALORE – 575025.**

# Acknowledgement

We extend our profound gratitude to Professor CMC Krishna for their expert guidance and mentorship throughout our project on robot dynamics and control. Their insightful inputs and continuous support have been pivotal in shaping our understanding of this complex field. A special note of appreciation goes to our invaluable lab guide, Aadarsh, whose exceptional dedication, extensive assistance, and hands-on support played a pivotal role in the success of our RRRR robot project. Aadarsh's guidance in navigating challenges, clarifying concepts, and offering practical solutions has been instrumental in achieving our project milestones. We are deeply thankful for Aadarsh's commitment, patience, and unwavering encouragement, which greatly enhanced our learning experience and significantly contributed to the successful execution of this endeavor. We express our gratitude to all individuals who have directly or indirectly contributed to the fruition of this project.

# 1. INTRODUCTION

This report introduces a robotic system specifically designed for assembly and material handling tasks. The robot's design incorporates an R base joint and three planar R joints, which are pivotal in optimizing its structural design. This configuration is chosen to enhance precision and control in the robot's movements, a crucial aspect for tasks that require high levels of accuracy.

Following the introduction of the robot's basic structure, the report delves into the detailed aspects of its 4-degree-of-freedom (DOF) architecture. This section focuses on the joint configurations and the overall design rationale, emphasizing the project's commitment to developing a functionally efficient and technically advanced robotic system

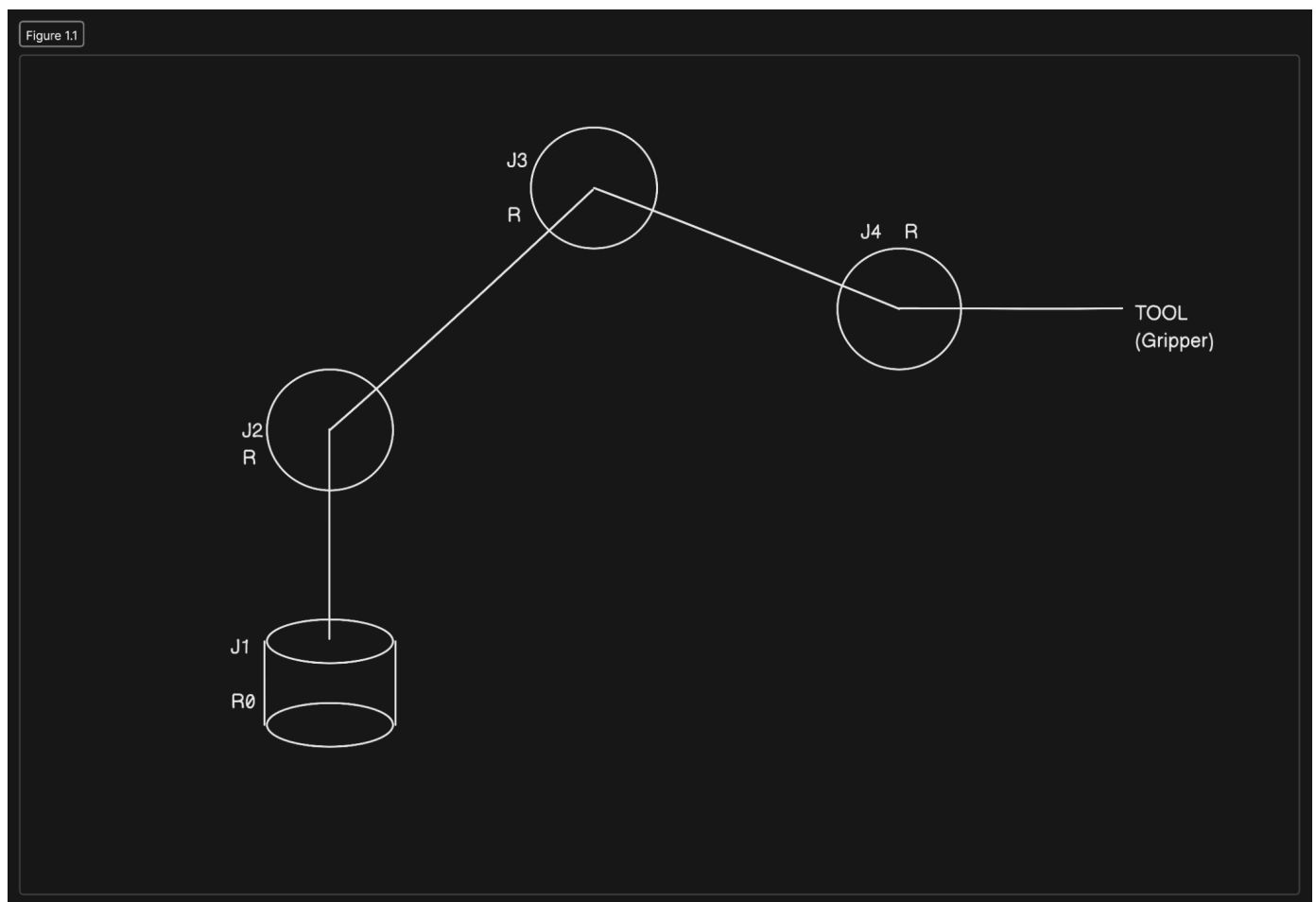


Figure 1.1

## a. Application

### 1. Assembly Line Operations:

- The robot, with its planar R joints, is applicable for precise assembly tasks on manufacturing lines, particularly in the automated handling and placement of components.

### 2. Electronic Circuit Board Assembly:

- The robot, featuring planar R joints and an R base joint, demonstrates suitability for navigating and accurately placing electronic components onto circuit boards, thereby contributing to the automation of electronics manufacturing.

### 3. Medical Equipment Manufacturing:

- In the production of medical devices, the robot's precision, facilitated by its planar R joints, ensures accurate assembly of intricate components, contributing to the quality and reliability of medical equipment.

### 4. Food Packaging and Processing:

- The robot's versatility in movement, owing to its planar R joints, is suitable for tasks in food processing and packaging, where controlled motions are essential for product quality and safety.

### 5. 3D Printing Support:

- The robot's planar R joints are advantageous in supporting 3D printing processes by precisely handling and placing objects during the additive manufacturing of complex structures.

## **b. Robot Design**

### Joint Types:

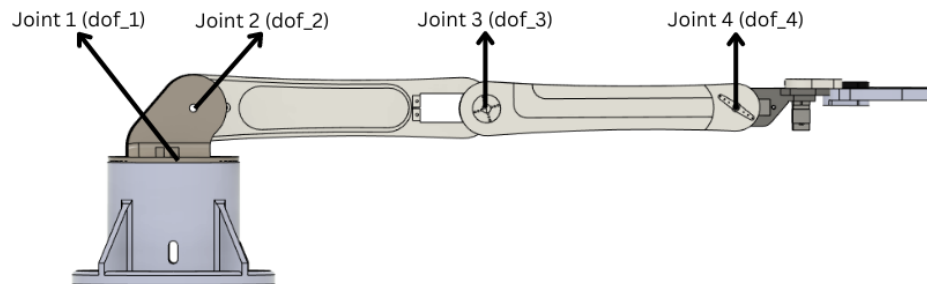
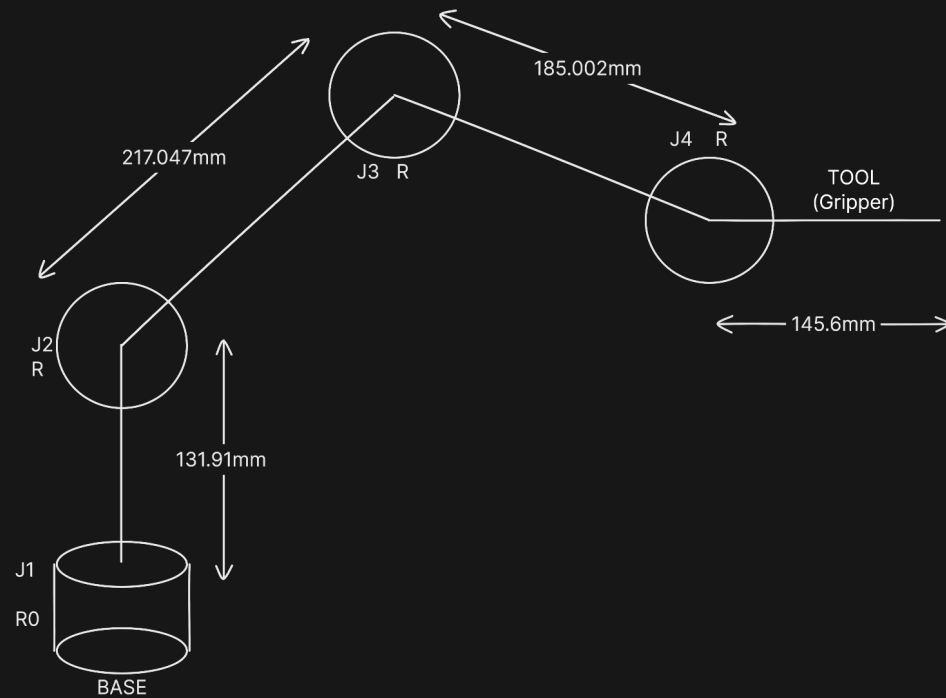
The robot incorporates a combination of joint types to facilitate specific movements and achieve the required degrees of freedom (DOFs). The base joint, denoted by the symbol "R0" signifies a revolute joint, allowing rotational motion around a single axis. The subsequent three joints are configured as planar R joints, enabling translational and rotational motion within a plane. This joint arrangement is purposefully selected as shown in Figure 1.1.

### Degrees of Freedom (DOFs):

The robot possesses a total of four degrees of freedom, distributed among the joints to enable the necessary range of motion for its designated tasks. The base joint contributes one DOF, and each of the three planar R joints adds one DOF to give a total of 4 DOF.

The robot adheres to a predefined geometric configuration to ensure compatibility with its intended operational environment. The lengths of the links and the distances between joints are determined to optimize the robot's reach and workspace. These dimensions are carefully considered to enhance precision and efficiency, contributing to the overall effectiveness of the robot in executing designated tasks.

Figure 1



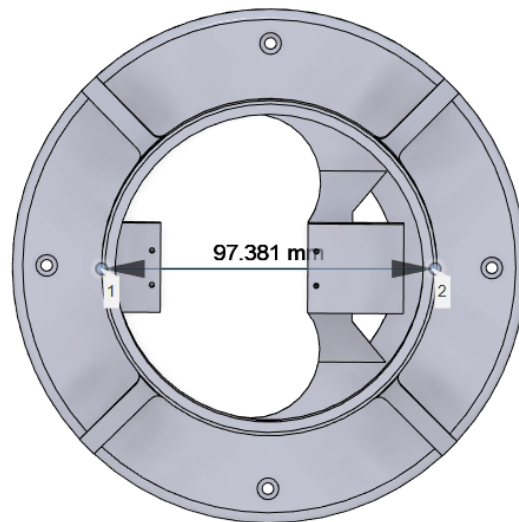
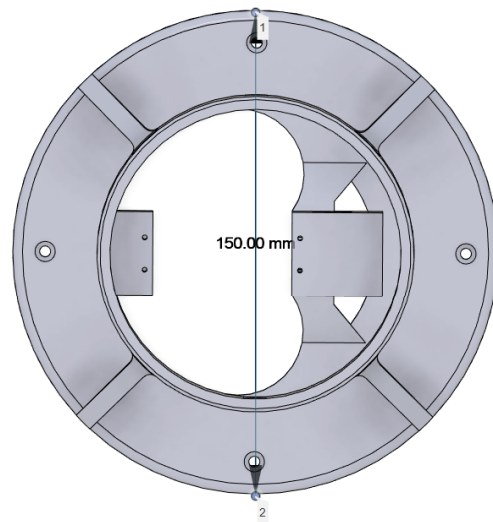
## 2. CAD Design

A .

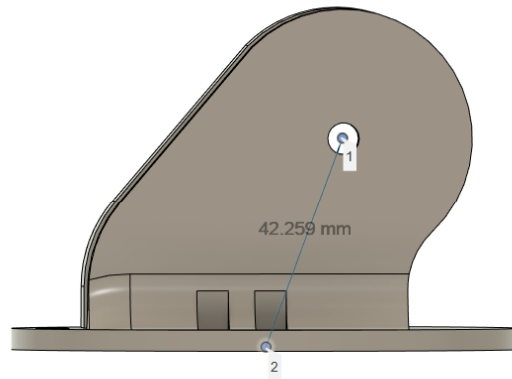
1. The model, consisting of individual components and links necessary for assembly, was sourced from [GrabCAD](#).
2. This model was not pre-assembled, requiring the team to manually assemble all the links and their corresponding joints to construct the complete robot.
3. The assembly process included connecting various parts such as the base\_link, link\_2, link\_3, and link\_4, as well as components of the gripper, labeled g\_part\_1 to g\_part\_12

## Joints

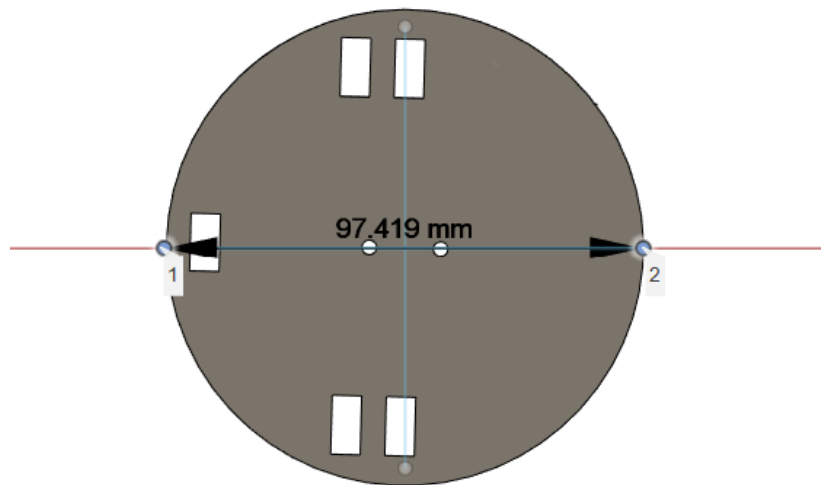
- a. dof\_1 - Revolute
  - i. Parent - base\_link
  - ii. Child - link\_2
- b. dof\_2 - Revolute
  - i. Parent - link\_2
  - ii. Child - link\_3
- c. dof\_3 - Revolute
  - i. Parent - link\_3
  - ii. Child - link\_4
- d. dof\_4 - Revolute
  - i. Parent - link\_4
  - ii. Child - g\_part\_1
- e. Rigid joints are used to assemble the gripper



**Base link Top View**



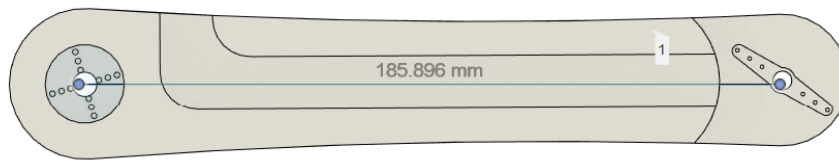
**link\_2 Front View**



**link\_2 Bottom View**



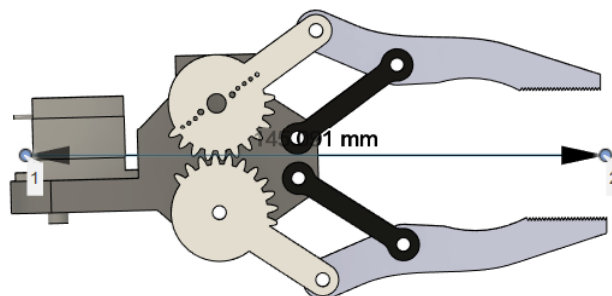
**link\_3 Front View**



**link\_4 Front View**

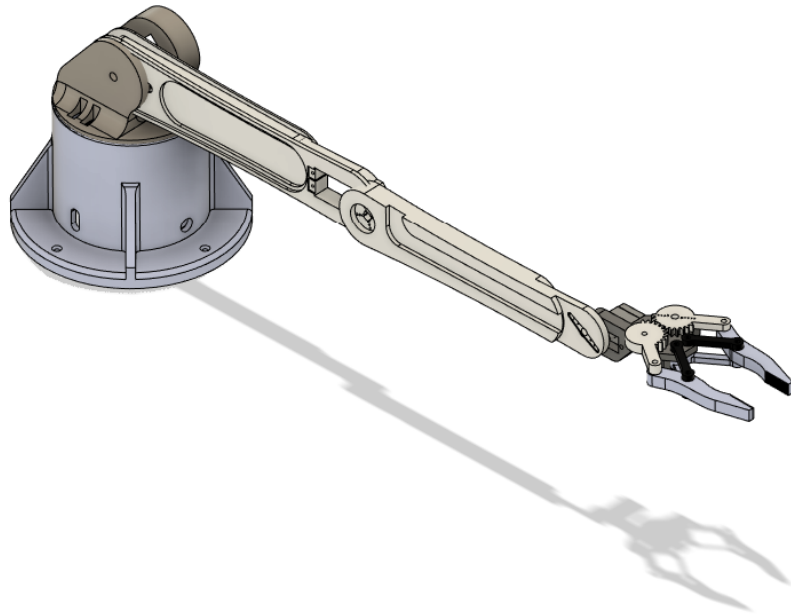


**g\_part\_1 to g\_part\_12 assembled (Gripper Front View)**

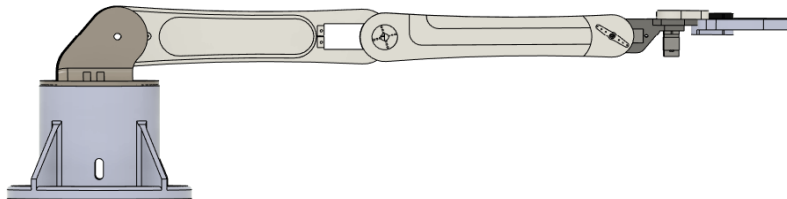


**g\_part\_1 to g\_part\_12 assembled (Gripper Top View)**

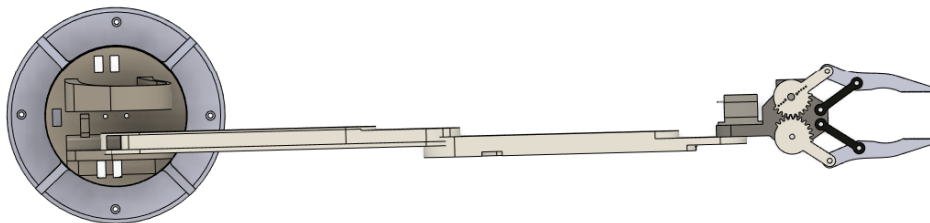




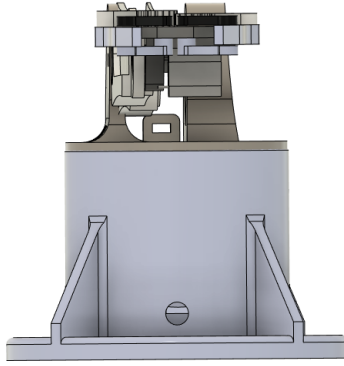
**Robot Home View**



**Robot Front View**



**Robot Top View**



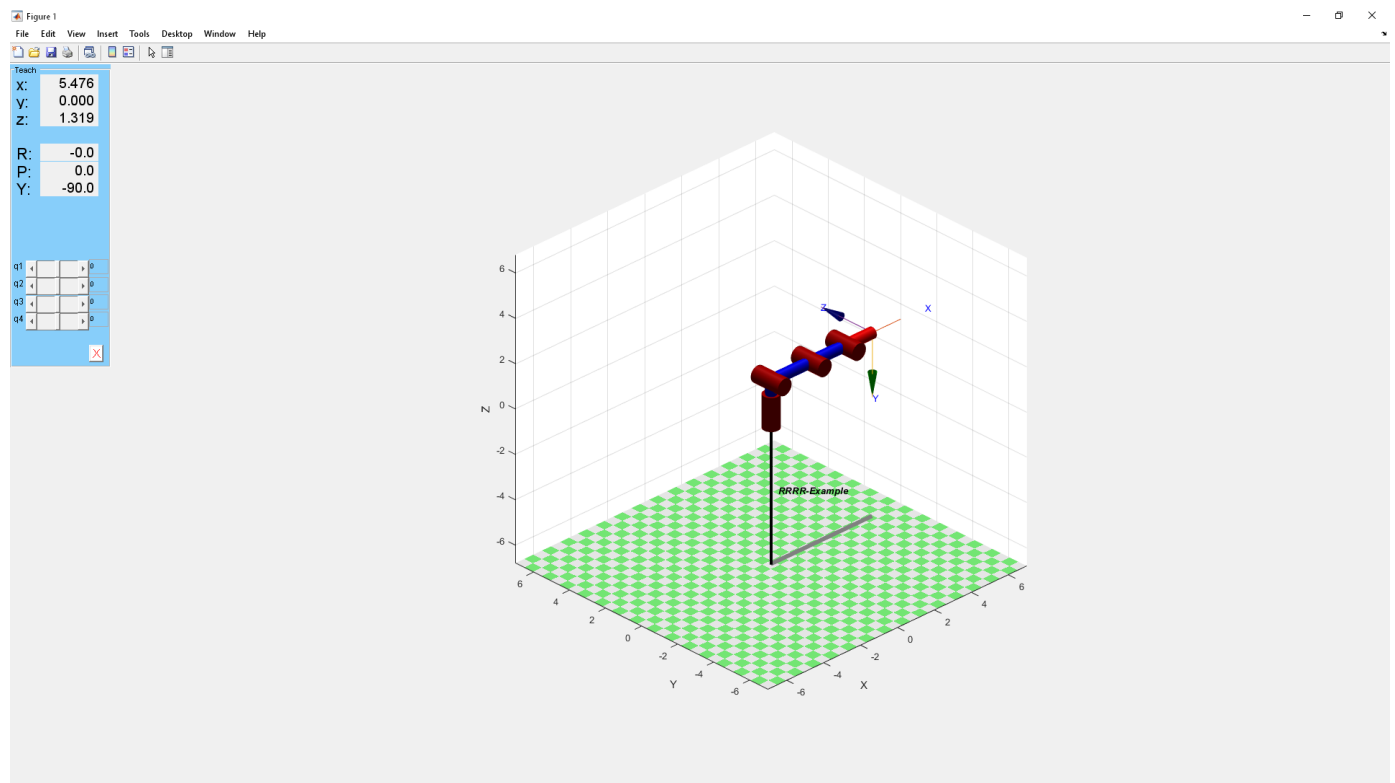
**Robot Right View**

## **B. CAD to URDF**

1. The robot was assembled in Fusion 360
2. The robot was assembled in Fusion 360. Then, we used [this](#) plugin.
3. Steps to install the plugin
  - For Windows (in PowerShell):
    - `cd <path to fusion2urdf>`
    - `Copy-Item ".\URDF_Exporter\" -Destination "${env:APPDATA}\Autodesk\Autodesk Fusion 360\API\Scripts\" -Recurse`
  - For macOS (in bash or zsh):
    - `cd <path to fusion2urdf>`
    - `cp -r ./URDF_Exporter "$HOME/Library/Application Support/Autodesk/Autodesk Fusion 360/API/Scripts/"`
4. Next, we used the plugin to directly convert to URDF and exported the robot to the RIVO\_description directory.
5. Now, we have successfully converted CAD to URDF.
6. A detailed description of CAD model
7. Detailed dimensions of links
8. Various views of the assembled CAD model
9. CAD to URDF

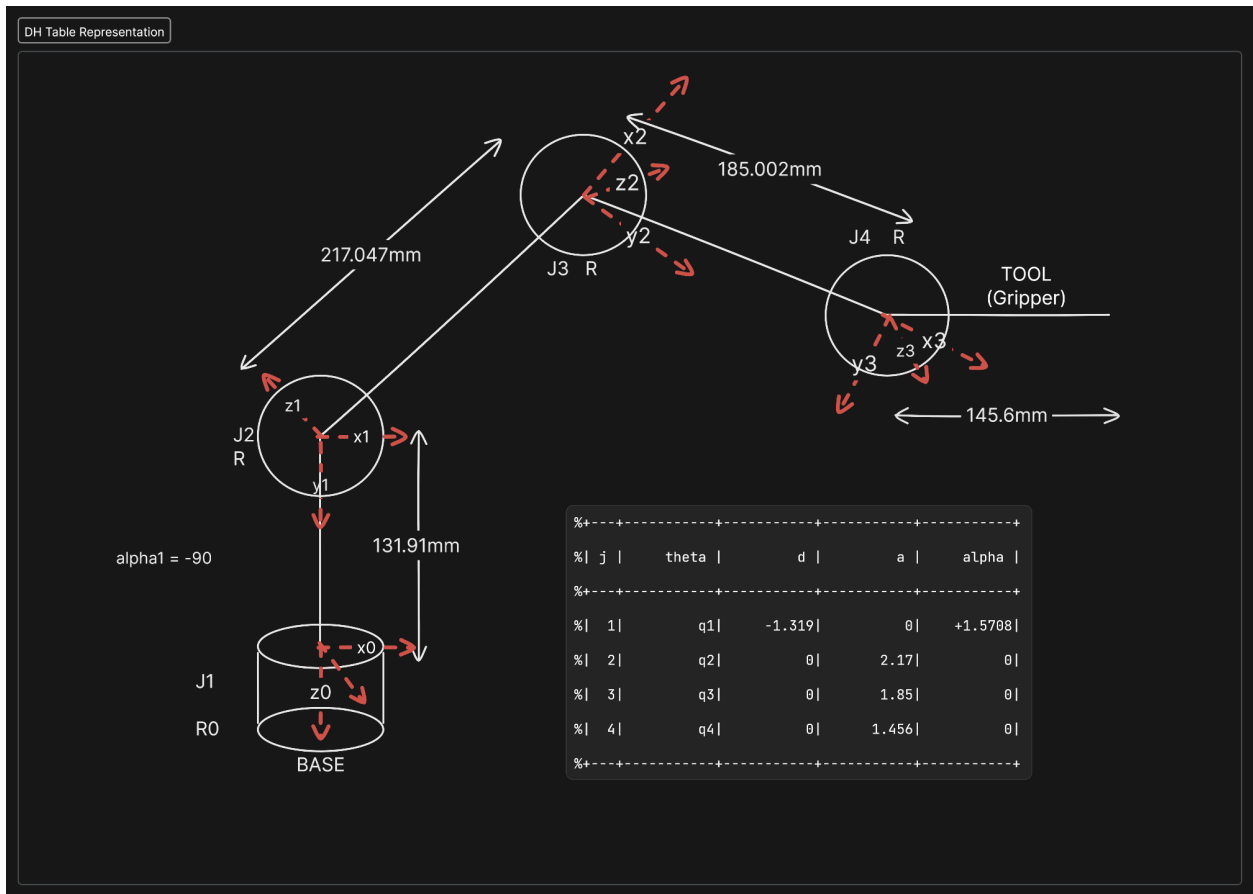
## **3. FORWARD KINEMATICS**

The following screenshot represents the robot at its home position where all theta values are 0. It has been done using Peter Corke's Toolbox by scaling the lengths down by a factor of x100. The end effector co-ordinates as mentioned would turn out to be [547.6, 0, 131.9] transpose after scaling it up to the real world.



### D-H Frames and Table

Link	$\alpha$	$a$	$\Theta$	$d$
1	$90^\circ$	0	$\Theta_0^*$	-131.9mm
2	0	217mm	$\Theta_1^*$	0
3	0	185mm	$\Theta_2^*$	0
4	0	145.6mm	$\Theta_3^*$	0



### a. Final Transformation Matrix (between base and end-effector)

For Finding the transformation matrix,  $\alpha_0$  is taken as -90 degrees and  $d_0 = + 1.319$  from the above mentioned DH table. It basically means that  $z_0$  axis is taken along the link in the upwards direction as per convention. This choice allows the alignment of the  $z_0$  axis parallel to the link in the upward direction, following convention. The inversion does not alter the parameters while still maintaining the consistency of the framework.

// Write intro and explain approach

$$\begin{pmatrix} m(eap - afq) + n(-afp - eaq) & m(-afp - eaq) - n(eap - afq) & b & mo(eap - afq) + no(-afp - eaq) - afqr + eag + eapr \\ m(ebp - bfq) + n(-bfp - ebq) & m(-bfp - ebq) - n(ebp - bfq) & -a & mo(ebp - bfq) + no(-bfp - ebq) - bfqr + ebg + e bpr \\ m(fp + eq) + n(ep - fq) & m(ep - fq) - n(fp + eq) & 0 & d + fg + mo(fp + eq) + no(ep - fq) + fpr + eqr \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The Final Transformation Matrix is:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

$$\begin{bmatrix} a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Where the following terms are:

$$\mathbf{a11:} (-\sin(\theta_2)\sin(\theta_3)\cos(\theta_1) + \cos(\theta_1)\cos(\theta_2)\cos(\theta_3))\cos(\theta_4) + (-\sin(\theta_2)\cos(\theta_1)\cos(\theta_3) - \sin(\theta_3)\cos(\theta_1)\cos(\theta_2))\sin(\theta_4)$$

$$\mathbf{a12:} -(-\sin(\theta_2)\sin(\theta_3)\cos(\theta_1) + \cos(\theta_1)\cos(\theta_2)\cos(\theta_3))\sin(\theta_4) + (-\sin(\theta_2)\cos(\theta_1)\cos(\theta_3) - \sin(\theta_3)\cos(\theta_1)\cos(\theta_2))\cos(\theta_4)$$

$$\mathbf{a13:} -\sin(\theta_1)$$

$$\mathbf{a14:} 145.6*(-\sin(\theta_2)\sin(\theta_3)\cos(\theta_1) + \cos(\theta_1)\cos(\theta_2)\cos(\theta_3))\cos(\theta_4) + 145.6*(-\sin(\theta_2)\cos(\theta_1)\cos(\theta_3) - \sin(\theta_3)\cos(\theta_1)\cos(\theta_2))\sin(\theta_4) - 185*\sin(\theta_2)\sin(\theta_3)\cos(\theta_1) + 185*\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) + 217*\cos(\theta_1)\cos(\theta_2)$$

$$\mathbf{a21:} (-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) + \sin(\theta_1)\cos(\theta_2)\cos(\theta_3))\cos(\theta_4) + (-\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_3)\cos(\theta_2))\sin(\theta_4)$$

$$\mathbf{a22:} -(-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) + \sin(\theta_1)\cos(\theta_2)\cos(\theta_3))\sin(\theta_4) + (-\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_3)\cos(\theta_2))\cos(\theta_4)$$

$$\mathbf{a23:} \cos(\theta_1)$$

$$\mathbf{a24:} 145.6*(-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) + \sin(\theta_1)\cos(\theta_2)\cos(\theta_3))\cos(\theta_4) + 145.6*(-\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_3)\cos(\theta_2))\sin(\theta_4) - 185*\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) + 185*\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) + 217*\sin(\theta_1)\cos(\theta_2)$$

$$\mathbf{a31:} (\sin(\theta_2)\sin(\theta_3) - \cos(\theta_2)\cos(\theta_3))\sin(\theta_4) + (-\sin(\theta_2)\cos(\theta_3) - \sin(\theta_3)\cos(\theta_2))\cos(\theta_4)$$

$$\mathbf{a32:} (\sin(\theta_2)\sin(\theta_3) - \cos(\theta_2)\cos(\theta_3))\cos(\theta_4) - (-\sin(\theta_2)\cos(\theta_3) - \sin(\theta_3)\cos(\theta_2))\sin(\theta_4)$$

$$\mathbf{a33:} 0$$

$$\mathbf{a34:} 145.6*(\sin(\theta_2)\sin(\theta_3) - \cos(\theta_2)\cos(\theta_3))\sin(\theta_4) + 145.6*(-\sin(\theta_2)\cos(\theta_3) - \sin(\theta_3)\cos(\theta_2))\cos(\theta_4) - 185*\sin(\theta_2)\cos(\theta_3) - 217*\sin(\theta_2) - 185*\sin(\theta_3)\cos(\theta_2) + 1319$$

a41: 0

a42: 0

a43: 0

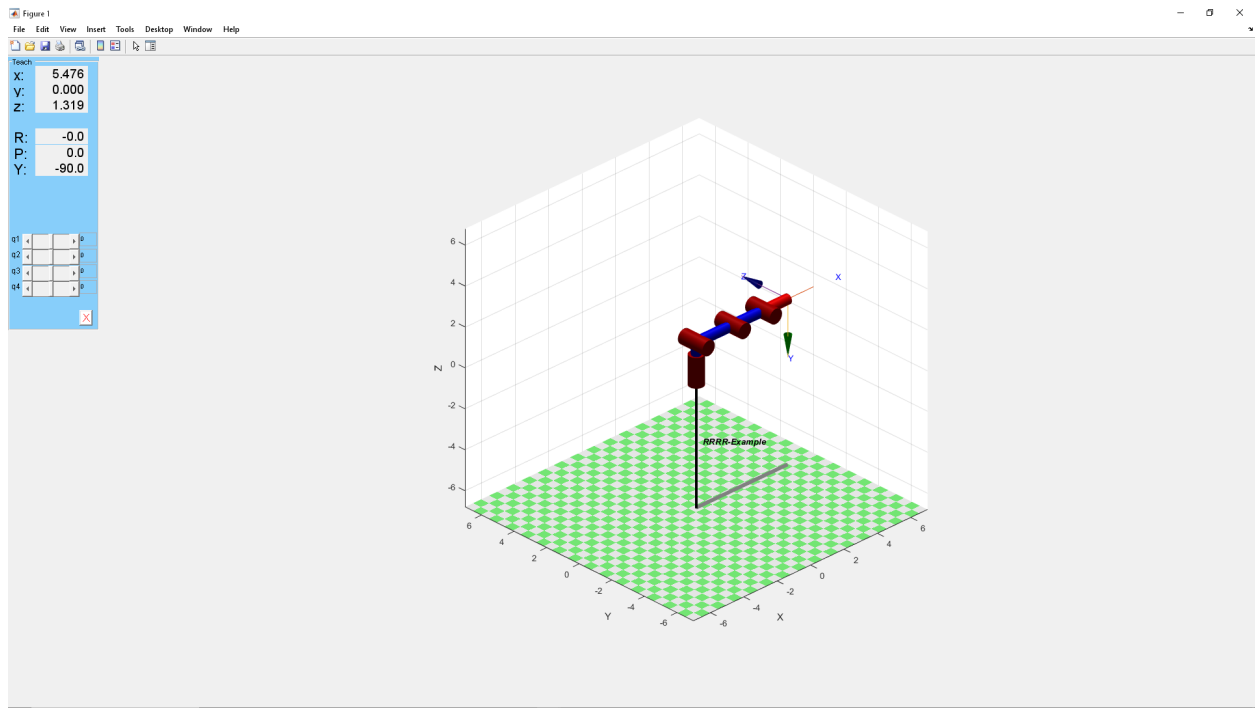
a44: 1

## 4. FORWARD KINEMATICS VALIDATION

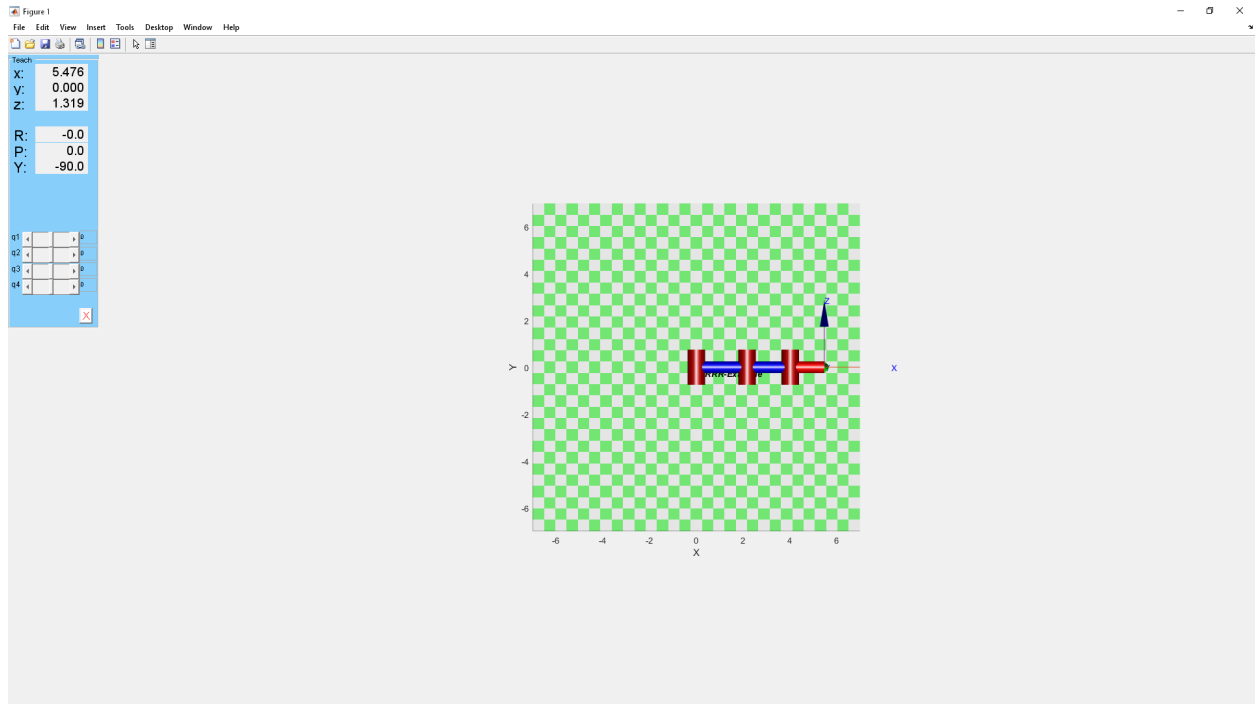
Validating Forward Kinematics is a key step in robotics, making sure that where the robot's end effector matches the math we expect from its joint movements. By using the Peter Corke Toolbox in MATLAB, this process checks that the math we use to understand the robot's motion matches what happens. To do this, we pick a few known spots where the robot should end up and see if the math agrees with where it goes. Finding any differences helps us trust that the math we're using will guide the robot accurately in real-world tasks.

To represent the position and infer forward kinematics correctly,  $\alpha_0$  is taken as  $-90$  degrees and  $d_0 = +1.319$ . These transformations flip the first frame i.e. the world frame to make it easier to find and validate points.

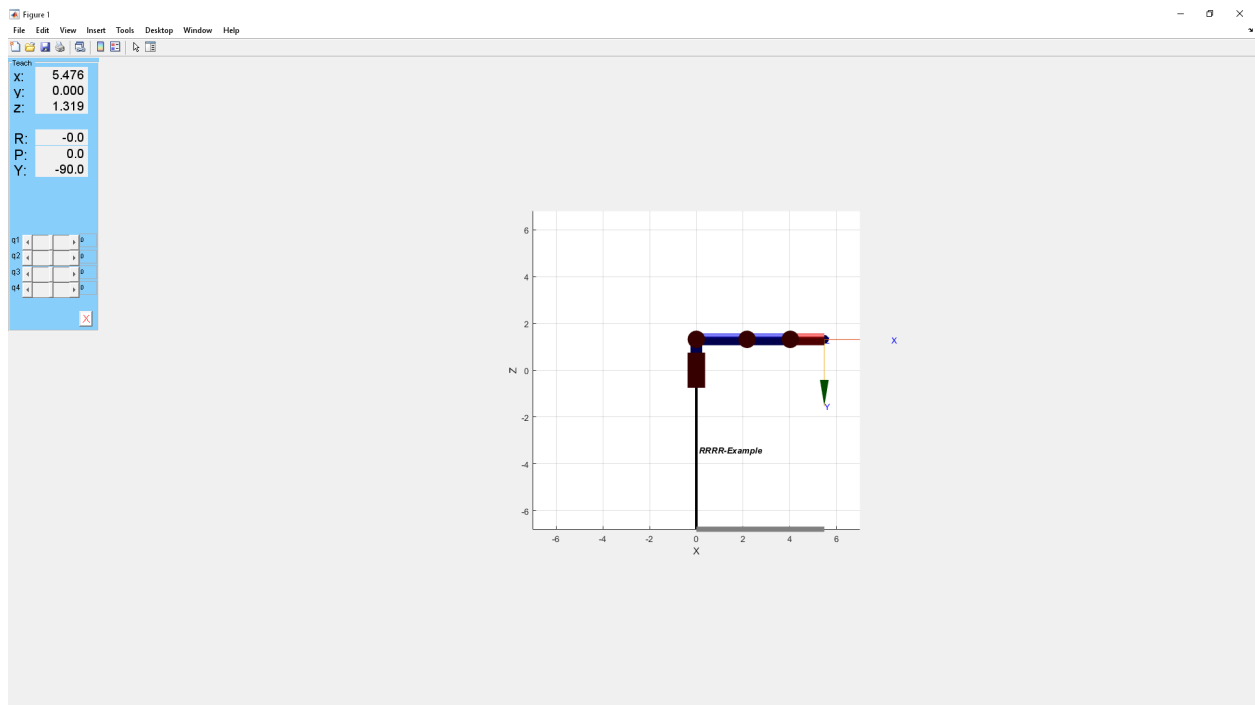
- A. Respawning at Home Position Using Peter Corke's Toolbox with  $\theta_1 = \theta_2 = \theta_3 = \theta_4 = 0$ .



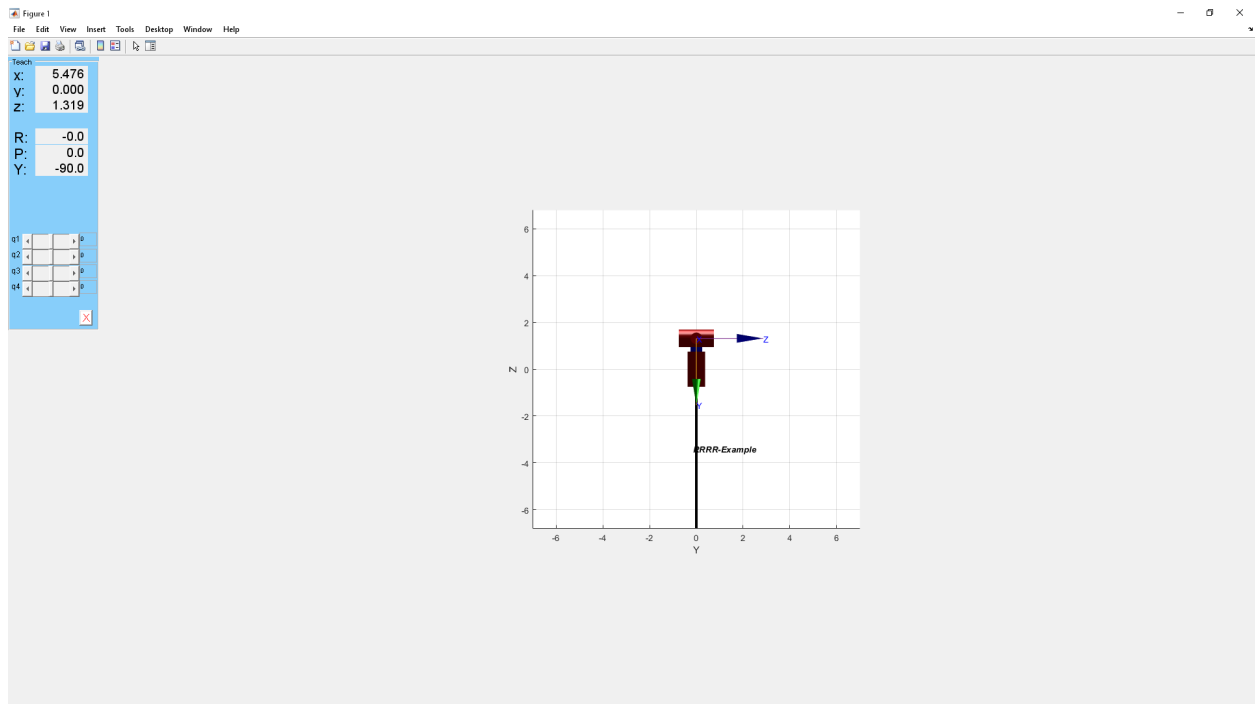
## 1. XY View



## 2. XZ View



### 3. YZ View



B. Validation - 2 is done for the given input theta values:

$\theta_1 = 0$

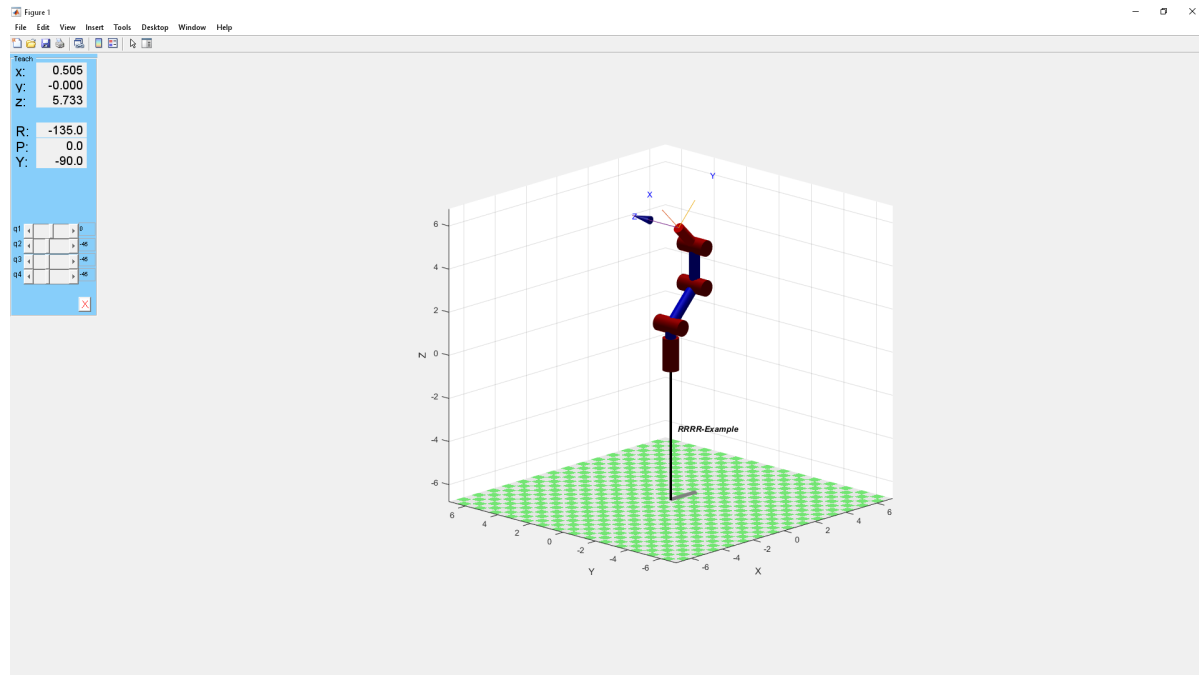
$\theta_2 = -45$

$\theta_3 = -45$

$\theta_4 = -45$



Output Validation: The co-ordinates of the end effector are given in the left corner.



C. Validation - 3 is done for the given input theta values:

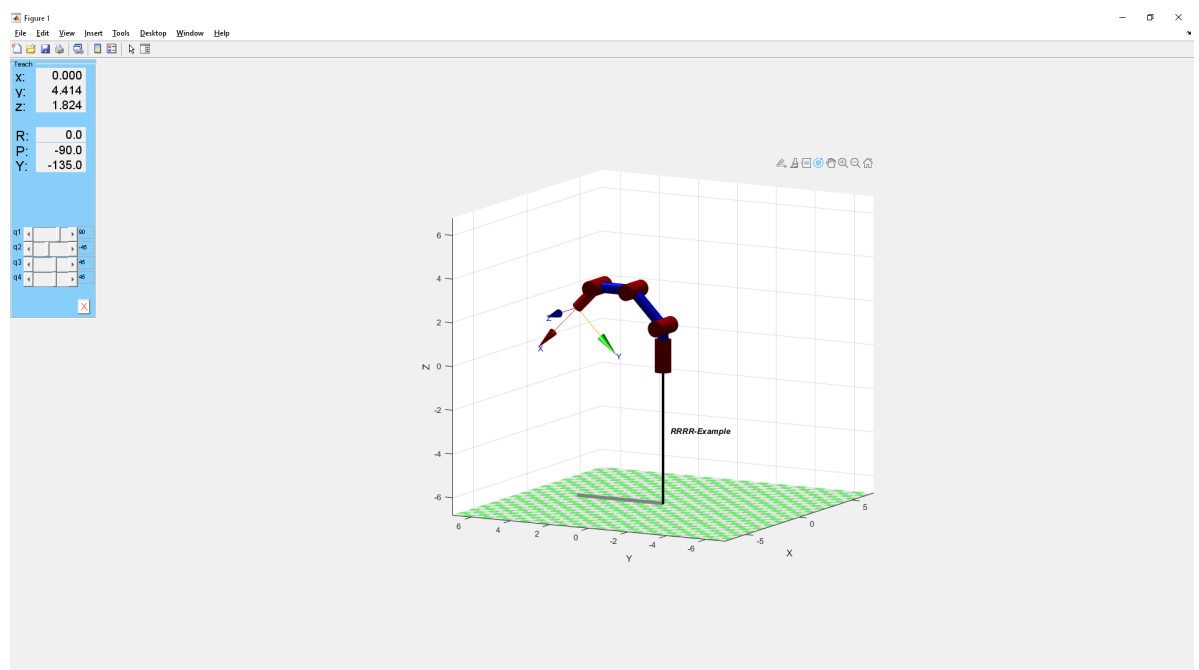
$\theta_1 = 90$

$\theta_2 = -45$

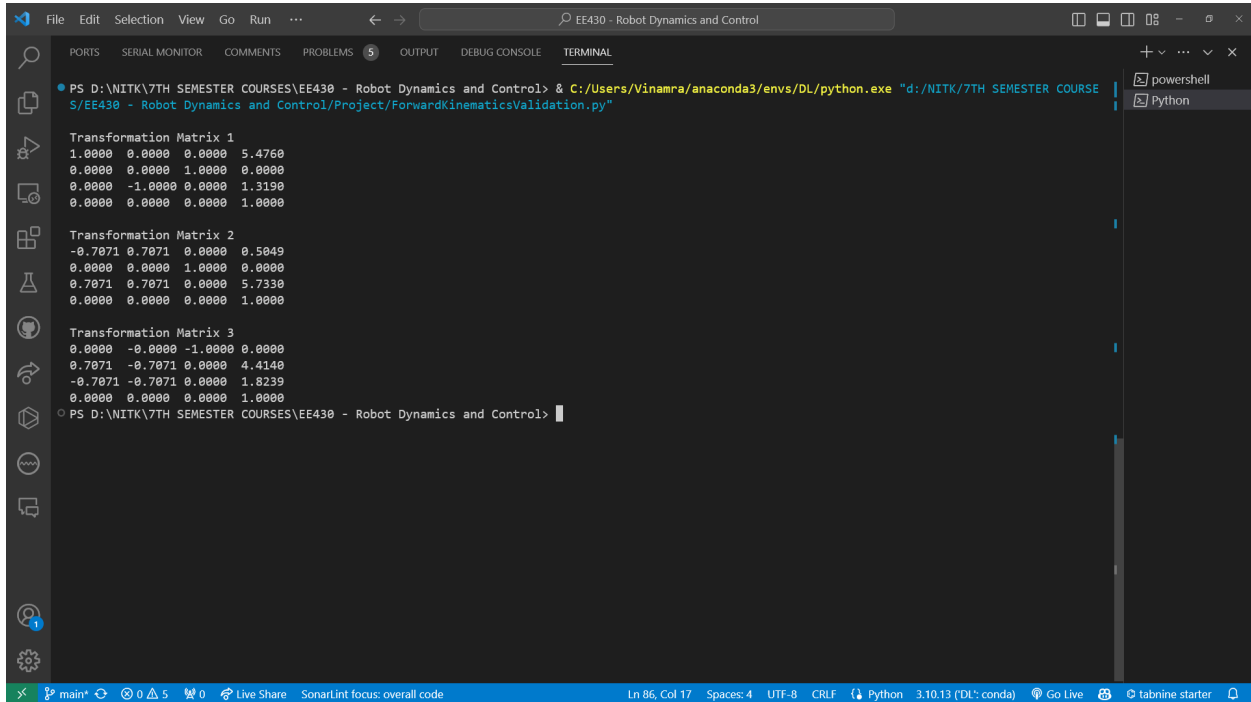
$\theta_3 = +45$

$\theta_4 = +45$

Output Validation: The co-ordinates of the end effector are given in the left corner.



## Transformation Matrix Output for the three Positions validated



```
PS D:\NITK\7TH SEMESTER COURSES\EE430 - Robot Dynamics and Control> C:/Users/Vinamra/anaconda3/envs/DL/python.exe "d:/NITK/7TH SEMESTER COURSE S/EE430 - Robot Dynamics and Control/Project/ForwardKinematicsValidation.py"

Transformation Matrix 1
1.0000 0.0000 0.0000 5.4760
0.0000 0.0000 1.0000 0.0000
0.0000 -1.0000 0.0000 1.3190
0.0000 0.0000 0.0000 1.0000

Transformation Matrix 2
-0.7071 0.7071 0.0000 0.5049
0.0000 0.0000 1.0000 0.0000
0.7071 0.7071 0.0000 5.7330
0.0000 0.0000 0.0000 1.0000

Transformation Matrix 3
0.0000 -0.0000 -1.0000 0.0000
0.7071 -0.7071 0.0000 4.4140
-0.7071 -0.7071 0.0000 1.8239
0.0000 0.0000 0.0000 1.0000

PS D:\NITK\7TH SEMESTER COURSES\EE430 - Robot Dynamics and Control>
```

The Forward Kinematics coordinates were checked using the Peter Corke Toolbox, and they matched up well with where the robot was supposed to be. This agreement at different points assures that the robot's math for moving around is accurate and dependable.

## 5. INVERSE KINEMATICS

- Inverse kinematics in robotics involves computing the joint parameters that achieve a desired position and orientation of the robot's end-effector. This process is particularly crucial in the RRRR (revolute-revolute-revolute-revolute) joint configuration of the project's robot, where the orientation of the first joint is around the z-axis, and the subsequent joint movements are determined accordingly
- These equations represent the inverse kinematics of the RRRR robot configuration. Given specific values for x,y,z, and the end effector angle ( $\theta$ ), you can use these equations to determine the corresponding joint angles
- Joint Configuration: RRRR
- Joint Orientation: First joint rotates about the z-axis, and all other joints are perpendicular to it and planar to each other.
- Angles Corresponding to Joints:
  - $\theta_1$  Angle for Joint 1 (Rotates about the z-axis).

- $\theta_2$  Angle for Joint 2 (Parallel to the xy plane).
- $\theta_3$  Angle for Joint 3 (Parallel to the xy plane).
- $\theta_4$  Angle for Joint 4 (Parallel to the xy plane).
- $\theta$ : End effector angle from the xy plane.

$$\theta_0 = \tan^{-1} \left( \frac{y}{x} \right)$$

Represents the angle of the first joint (rotation about the z-axis) calculated based on the Cartesian coordinates (x,

$$y = y - 145.6 \cdot \cos(\theta) \cdot \sin(\theta_0)$$

$$x = x - 145.6 \cdot \cos(\theta) \cdot \cos(\theta_0)$$

Calculate adjustments to the Cartesian coordinates (x, y) due to the robot's arm length and end effector

$$\theta_2 = -\cos^{-1} \left( \max \left( \min \left( \frac{x^2 + y^2 + (z - 131.9)^2 - a_1^2 - a_2^2}{2 \cdot a_1 \cdot a_2}, 1 \right), -1 \right) \right)$$

Determines the angle for Joint 3 using the inverse cosine function, based on the robot's geometric configuration and constants.

$$\theta_1 = \tan^{-1} \left( \frac{z - 131.9}{\sqrt{x^2 + y^2}} \right) - \tan^{-1} \left( \frac{a_2 \cdot \sin(\theta_2)}{a_1 + a_2 \cdot \cos(\theta_2)} \right)$$

Calculates the angle for Joint 2 considering the robot's geometric configuration and the angles of Joint 1 and 3.

$$\theta_3 = -\theta - \theta_1 - \theta_2$$

Determines the angle for Joint 4, taking into account the end effector angle and the angles of Joints 1, 2, and 3.

**Joint Angle Calculation:** IK provides a mathematical framework to determine the joint angles required for the RRRR robot to position its end-effector accurately within its workspace. By utilizing geometric and trigonometric calculations, IK computes the precise joint configurations needed to achieve specific positions and orientations.

**Solving Kinematic Equations:** With a RRRR robot having multiple revolute joints, IK involves solving a system of nonlinear equations derived from the robot's kinematic structure. These equations relate the joint angles to the position and orientation of the end-effector. Various mathematical methods such as numerical techniques, closed-form solutions, or iterative approaches are employed to solve these equations.

**Geometric Constraints and Solutions:** IK accounts for geometric constraints, including joint limits and workspace boundaries, while determining feasible solutions for the joint angles. Mathematical algorithms ensure that the computed joint configurations adhere to these constraints, avoiding singularities or physical limitations of the robot's motion.

## 6. WORKSPACE STUDY

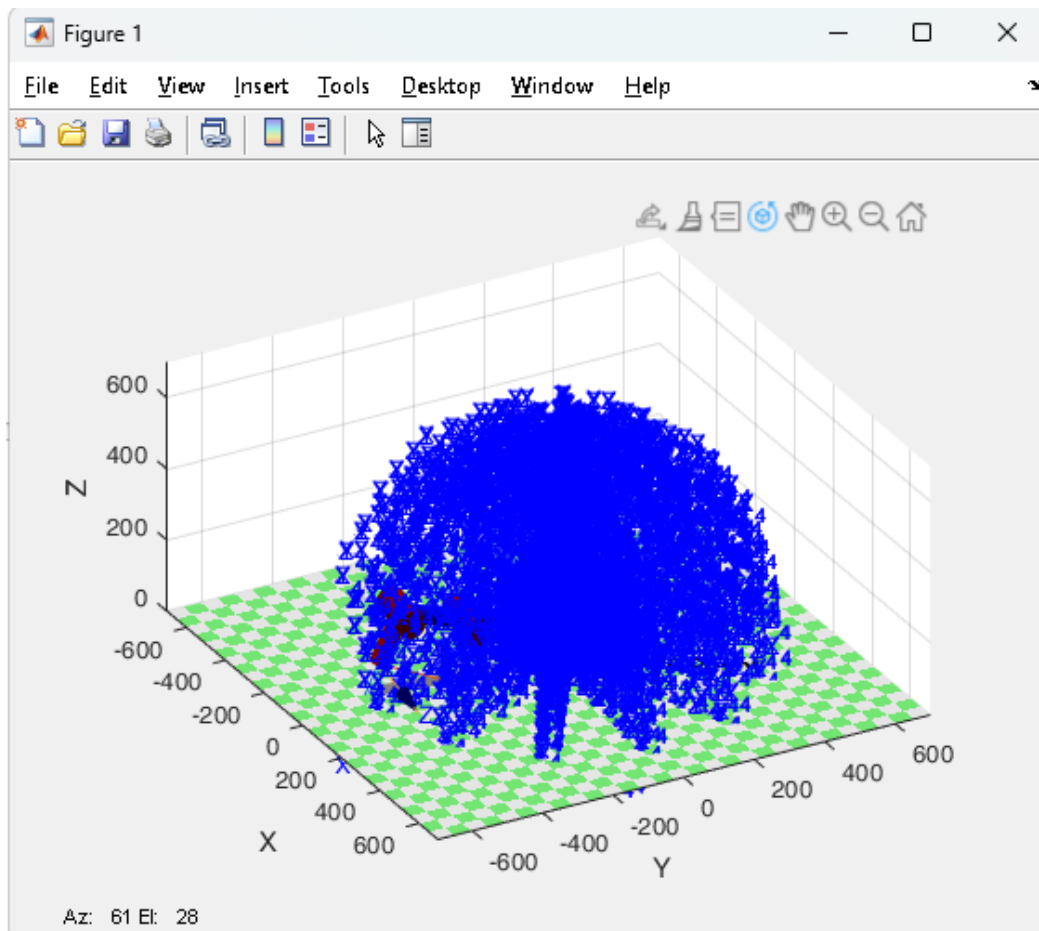
```
L(1) = Link ( [ 0, 131.9, 0, pi/2, 0 ] );  
  
L(2) = Link ( [ 0, 0, 217, 0, 0 ] );  
  
L(3) = Link ( [ 0, 0, 185, 0, 0 ] );  
  
L(4) = Link ( [ 0, 0, 145.6, 0, 0 ] );  
  
robot = SerialLink(L);  
  
robot.name = 'RIVO 4DOF Robot';  
  
for q1 = -pi/2:0.5:pi/2  
    for q2 = 0:0.5:pi  
        for q3 = -pi:0.5:0  
            for q4 = -pi/2:0.5:pi/2  
                robot.plot ([q1,q2,q3,q4], 'workspace', [-700 700 -700 700 0 700])  
  
                hold on  
  
                [T,A] = robot.fkine([q1,q2,q3,q4]);  
  
                trplot(A(4), 'frame', num2str(4))  
  
                drawnow;  
  
            end  
        end  
    end  
end  
end
```

The provided MATLAB code was used to visualize the workspace of our robot.

The limits for joint variables  $q_1$ ,  $q_2$ ,  $q_3$ , and  $q_4$  were set in accordance with the defined constraints in the URDF file.

By iterating over all possible combinations of  $q_1$ ,  $q_2$ ,  $q_3$ , and  $q_4$  within the specified limits, the workspace was systematically plotted.

It was observed that the workspace forms an approximately hemispherical region with a radius of 547.6 mm, corresponding to the sum of the lengths of the robot's links (217 mm + 185 mm + 145.6 mm).



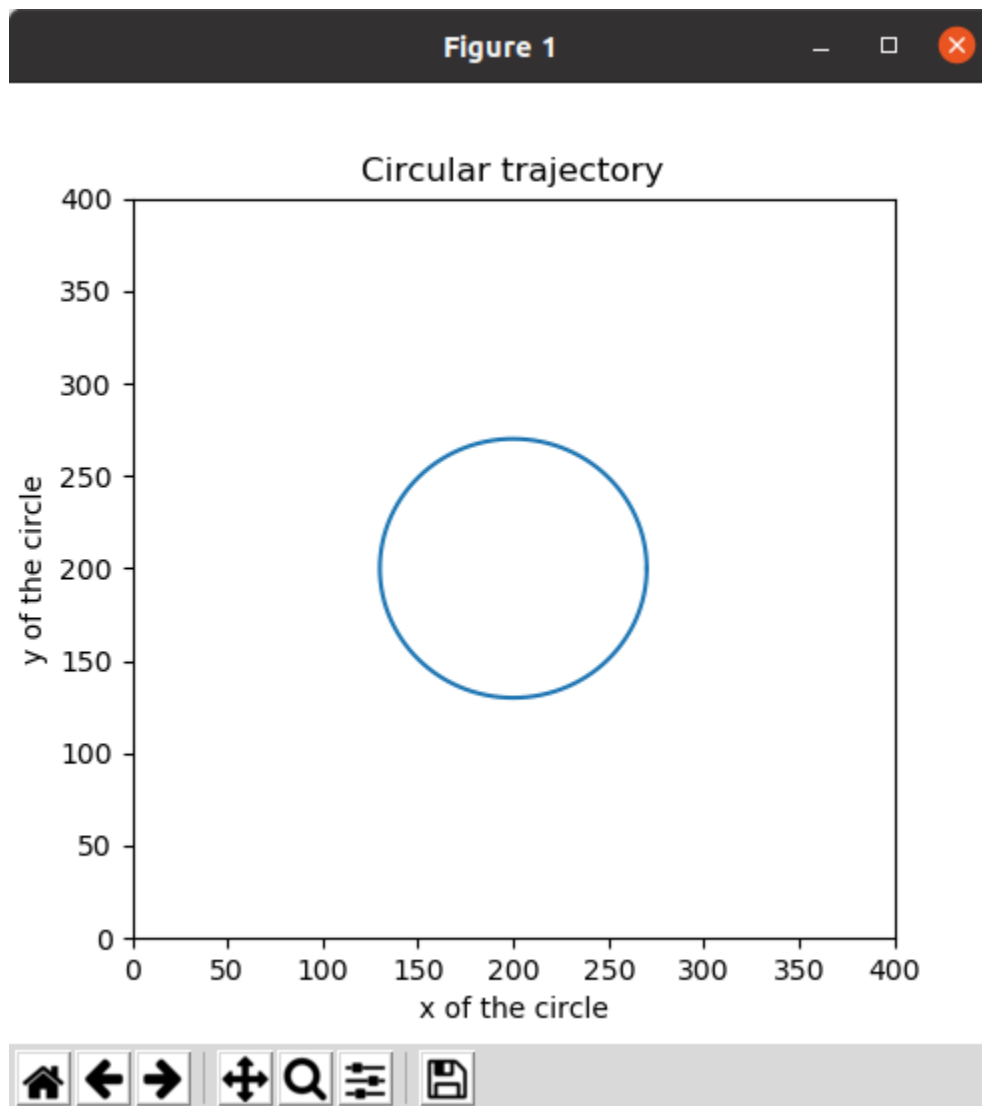
To plot the entire workspace, higher computation power is required. However, due to our relatively slower CPU, we used a step size of 0.5 radians for all the joint angles in the code.

## 7. SIMULATION

- a) We have implemented both a circle trajectory and a rectangle trajectory in our robot motion planning. ([Video Links](#))

### Circle Trajectory:

- For the circle trajectory, we plot the circle in the xy-plane ( $z=0$ ). The user provides inputs for the circle's center coordinates ( $x, y$ ) and radius ( $r$ ).
- Using the circle equation  $x_1=x+r\cos(\theta)$  and  $y_1=y+r\sin(\theta)$ , where ( $x, y$ ) are the coordinates of the circle center and  $\theta$  varies from 0 to 360 degrees, we generate points along the circle.
- For each point ( $x_1, y_1$ ), we use our inverse kinematics function to calculate the corresponding joint angles.
- These joint angles are then published to a topic, enabling the robot to execute the circle trajectory.



### Rectangle Trajectory:

- In the case of the rectangle trajectory, we define a rectangle in the xy-plane.
- The user provides inputs for the rectangle, specifying the center of the rectangle and the lengths of its sides (side 1 and side 2).

Points along the perimeter of the rectangle are obtained by iterating over its edges.

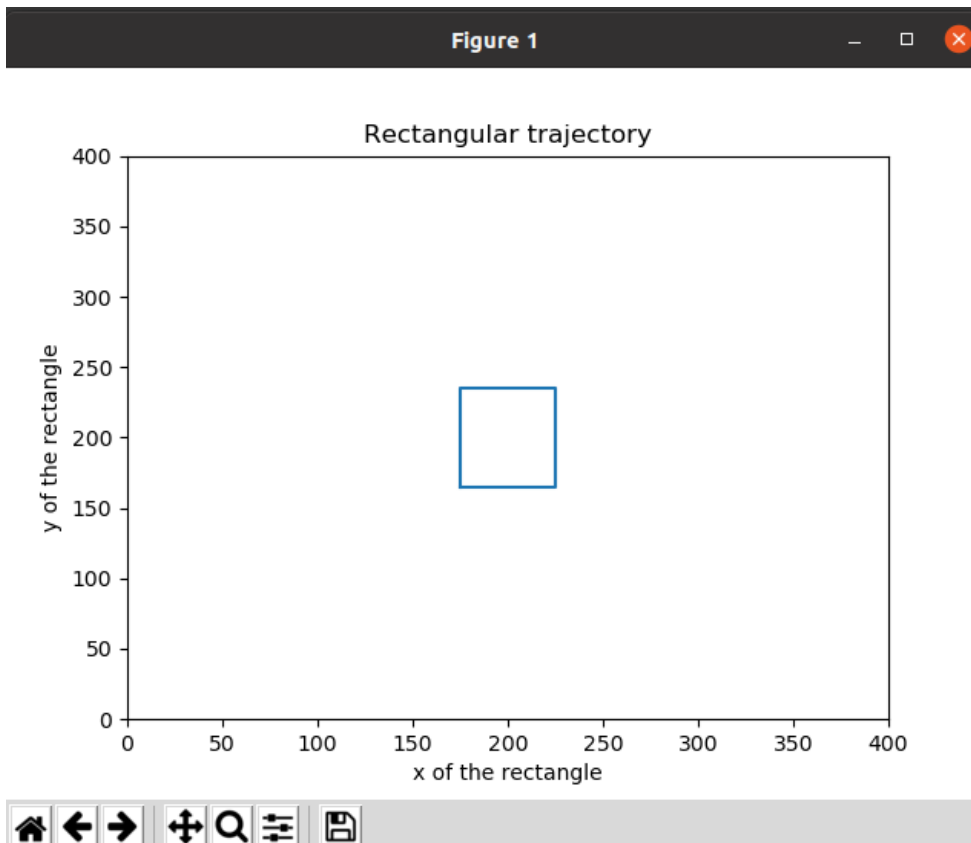
```
for x1 in np.arange(x - s2/2, x + s2/2 + 1, 5):
    calculate_coordinates(x1, y - s1/2)

for y1 in np.arange(y - s1/2, y + s1/2 + 1, 5):
    calculate_coordinates(x + s2/2, y1)

for x1 in np.arange(x + s2/2, x - s2/2 - 1, -5):
    calculate_coordinates(x1, y + s1/2)

for y1 in np.arange(y + s1/2, y - s1/2 - 1, -5):
    calculate_coordinates(x - s2/2, y1)
```

- Similar to the circle trajectory, we utilize the inverse kinematics function to calculate joint angles for each point.
- These joint angles are then published to the topic, enabling the robot to follow the rectangle trajectory.



b)

- After converting the CAD model to URDF using a dedicated plugin, we moved the resulting directory within the Catkin workspace.
- Subsequently, we established a 'config' folder within this workspace and created a 'controllers.yaml' file, detailing the proportional (P), integral (I), and derivative (D) values for all the joints.
- We used an analytical approach to find P, I, D values of each joint.
- To facilitate the visualization and simulation processes, we designed two launch files – 'display.launch' and 'gazebo.launch.'
- The former is responsible for launching RViz, while the latter is employed to initiate Gazebo.
- To address an axis misalignment issue that surfaced during the transition from CAD to URDF, adjustments were made to the axis direction within the 'RIVO.xacro' file under the URDF folder.
- Once these preparations were complete, we executed simulations in Gazebo.
- Subsequently, a Python script, 'thetaPublisher.py,' was developed to directly publish joint angles to the controller topics.
- As we had given names of the controllers as dof\_1\_control, dof\_2\_control, dof\_3\_control, dof\_4\_control in controller.yaml.
- Topics named dof\_1\_control/command, dof\_2\_control/command, dof\_3\_control/command, dof\_4\_control/command were created.
- So we need to publish joint angles to these topics in order to move our robot.
- Following successful implementation, we proceeded to create 'InverseKpublisher.py.'
- This script prompts the user for inputs (x, y, z) and utilizes an inverse kinematics function, encapsulated within 'InverseKinematics.py,' to calculate joint angles.
- The resulting angles are then published to the respective controller topics.
- Upon validating the functionality of 'InverseKpublisher.py,' additional scripts, namely 'circle.py' and 'rectangle.py,' were created to define trajectories using circular and rectangular paths, respectively.

## 8. CHALLENGES FACED

- **Fusion to URDF:** Initially, we faced challenges in converting our CAD model in Fusion to URDF. Attempts to export the model to Onshape proved time-consuming due to the large size of the STEP file. Subsequently, we discovered a [plugin](#) that simplifies the conversion process from Fusion to URDF.
- **Toppling of Robot:** Upon starting the robot, we encountered an issue where it would topple. After investigating we found out that the mass of the base was not enough to hold all the connected components. Due to gravitational forces acting on other components, the robot experienced toppling. To address this, we increased the mass of the robot.
- **Sliding of Robot:** Upon providing input for robot movement, an unexpected sliding behavior was observed. It was identified that insufficient friction between the base link and the ground caused this issue. To mitigate sliding, we increased the friction in the vicinity of the base.
- **Movement of Base:** Another challenge emerged as the base joint moved unintentionally when other joints were in motion. Upon revisiting the inertia matrix of the base link, we realized that its values were not adjusted after increasing the mass. After aligning the inertia matrix values with the increased mass, the undesired movement ceased.



- **“Controller Spawner Couldn't Find the Expected Controller\_Manager ROS Interface” Error:** While employing the thetaPublisher script to move the robot, we encountered a perplexing issue where the robot remained stationary despite correct configurations. After meticulous debugging, we discovered a warning in the terminal: "Controller spawner couldn't find the expected controller\_manager ROS interface." After exploring solutions on multiple websites (website1, website2), we identified a mismatch in the specified namespace. Adjusting the namespace, although attempted initially, did not resolve the issue. Ultimately, removing the namespace and keeping it as default resolved the problem
- **Inverse Kinematics:** As we initiated the calculation for finding the inverse kinematics, we divided our robot into two parts. First, the DOF\_1 joint, whose angle was relatively easy to calculate; it was simply the arctangent of  $y/x$ . The second part involved three revolute joints in a planar configuration. Despite numerous attempts to solve it mathematically and geometrically, we were unable to find a solution. Upon researching the inverse kinematics of a 3-revolute planar robot on the internet, we discovered that it is indeed challenging to determine using simple trigonometry and a geometric approach. In many cases, we encountered difficulties leading to infinite solutions. To address this issue and make the inverse kinematics viable, we decided to incorporate user input for the gripper's angle, i.e., the end effector's angle with respect to the ground. With this constraint in place, we successfully determined the values for  $\theta_0$ ,  $\theta_1$ ,  $\theta_2$ , and  $\theta_3$ .
- **PID Control :** Tuning the PID control parameters ( $K_d$ ,  $K_p$ ,  $K_i$ ) for the RRRR Robot involves intricate calculations due to the system's non-linear, high-order transfer functions. To navigate this complexity, an analytical approach was employed to derive approximate values for these control constants. By leveraging general  $K_d$  and  $K_p$  values commonly observed in real-world robotic systems, our tuning process centered around refining and adjusting these values to optimize the robot's performance in our simulations

## 9. INDIVIDUAL CONTRIBUTIONS

### Sumukh C Prakash

- Motion Planning Strategy Implementation: Implementing various motion planning strategies (e.g., circle, rectangle trajectories) for simulating the robot's movements.
- Workspace Analysis: Conducting a comprehensive study of the robot's operational space using simulation tools.
- Testing and Validation: Testing the robot's capabilities and limitations within the simulated environment to ensure practical real-world applicability.
- Performance Optimization: Identifying areas for performance enhancement and refining the design and models based on simulation results.
- CAD Assembly Design: Responsible for creating, assembling, and verifying the CAD models of the robotic system.
- Component Detailing: Detailing individual components, links, and mechanisms necessary for the robot's assembly and functionality.
- Setting up report

## Vinamra Parakh

- Model Validation: Validating kinematic models using tools like the Peter Corke Toolbox for accuracy verification.
- Integration and Validation: Integrating PID control into the simulation environment and validating its effectiveness in controlling the robot's behavior and movements
- Denavit-Hartenberg (DH) Table Calculation: Calculating the DH parameters and establishing transformation matrices for the robot's joints.
- Testing and Validation: Testing the robot's capabilities and limitations within the simulated environment to ensure practical real-world applicability.
- Performance Optimization: Identifying areas for performance enhancement and refining the design and models based on simulation results.
- Setting up report

## Harsh Nahata

- Forward and Inverse Kinematics Derivation: Deriving mathematical models for forward and inverse kinematics of the robot.
- Denavit-Hartenberg (DH) Table Calculation: Calculating the DH parameters and establishing transformation matrices for the robot's joints.
- Joint Movement and End-effector Positioning: Ensuring precise joint movements and accurate end-effector positioning in the derived models.
- Tuning PID Constants (Kp, Ki, Kd): Setting up and fine-tuning the PID control parameters for the robot's control system.
- Analytical Approximation: Employing analytical methods to derive approximate PID values based on real-world robot standards.
- Setting up report

## 10. REFERENCES

- 1) [https://grabcad.com/library/robot-arm-rivo-3d-printed-robot-arm-1/details?folder\\_id=13498434](https://grabcad.com/library/robot-arm-rivo-3d-printed-robot-arm-1/details?folder_id=13498434)
- 2) [GitHub - syuntoku14/fusion2urdf: A Fusion 360 Script to export URDF](#)
- 3) [Kinematic model of an RRR robotic arm. | Download Scientific Diagram](#)
- 4) [Problem with start controller with roslaunch: controller spawner couldn't find the expected controller\\_manager ROS interface - Gazebo: Q&A Forum](#)
- 5) [Fusion 360 to URDF | Adding Gazebo plugin | Keyboard Teleop | LiDAR | Part -2](#)
- 6) [gazebo crashes immediately using roslaunch after installing gazebo ros packages](#)
- 7) [\(PDF\) Simulation 3-DOF RRR Robotic Manipulator under PID Controller](#)