ENPM662 Project 2

Delbot - A delivery robot

Technical Report

Badrinarayanan Raghunathan Srikumar (119215418)
Aniruddh Balram (119206416)

MEngg. Robotics
University of Maryland
College Park
23rd November,2022

ACKNOWLEDGMENT

We would like to thank our supervisor Dr. Reza Monfaredi for his consistent support and guidance during the course of this project. Additionally, we would also extend our gratitude to Adarsh Malapaka, Pavan Mantripragada, and Aneesh Chodisetty for their relentless assistance in both doubt clearance and software assistance, without which successful completion of the project would not have been possible

I. INTRODUCTION:

Manipulators are being used extensively in industries for a variety of purposes. Starting from a simple planar 2 DOF robot, to highly complex manipulators. Manipulators are used in almost every industry for faster and precise automation of manufacturing products. Their applications range from inspecting tiny electronic parts to lifting heavy vehicle parts, thus making their use extremely versatile. Nowadays, manipulator *Cobots* exist which enable humans to work along with the manipulator. There is one disadvantage though: Manipulators have a constrained workspace. The reason for this is, generally all the available *Cobots* have a fixed base for better stability and dynamics. This disadvantage prevents the manipulator from catering to tasks which it is very much capable of such as delivery, nursing, sanitation and many more. One way to overcome this problem is to introduce a mobile base-frame upon which the manipulator *Cobot* is fixed. This way, the *Cobot* will have the same workspace at every coordinate point on the space of operation which the mobile base can achieve, therefore increasing the workspace of the robot as a whole.

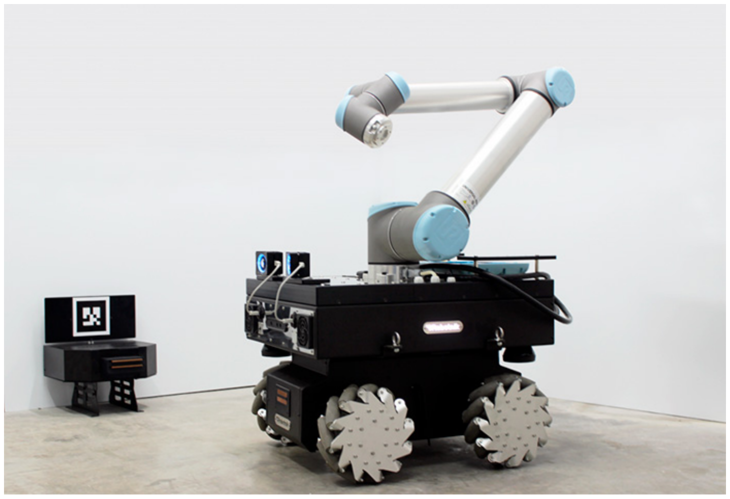


Figure 1: *An example of a mobile manipulator*

In this report, we explain about the design and development of a mobile manipulator *Cobot* which would not only solve the former disadvantage but also increase the workability and versatility of a *Cobot* manipulator.

II. MOTIVATION:

Covid-19 brought about many changes in the lifestyle of people. Shops were closed, accessibility to groceries were limited and people feared stepping out of the house. This caused a dearth in supply and increase in demand. So, certain delivery companies had to step-up to ensure that the demands were met risking the life of their delivery employees. Not just that, in certain cases, during surged demands, certain deliveries fail and such a situation is risky when the delivery is of critical medications. As a concept, our intent is to develop a Robot which could assist these delivery companies with respect to delivery of food and groceries from a pick-up point to a drop-point.

III. APPLICATION:

The objective of this project is to design and develop a mobile manipulator Robot which can perform pick and place operations given known end-points. The motivation for this project comes from the shortage of home supplies during Covid-19. The working of this Robot is quite-straightforward. The delivery companies receive delivery requests from customers through their mobile application. With that, we have two sets of information: 1) The products to be delivered 2) The end location where it needs to be delivered. The robot then moves to a point in their warehouse where the products are, picks them up, reaches the end location and delivers them to the customers.

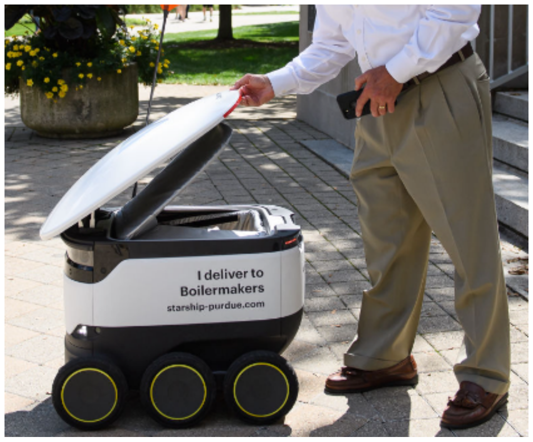


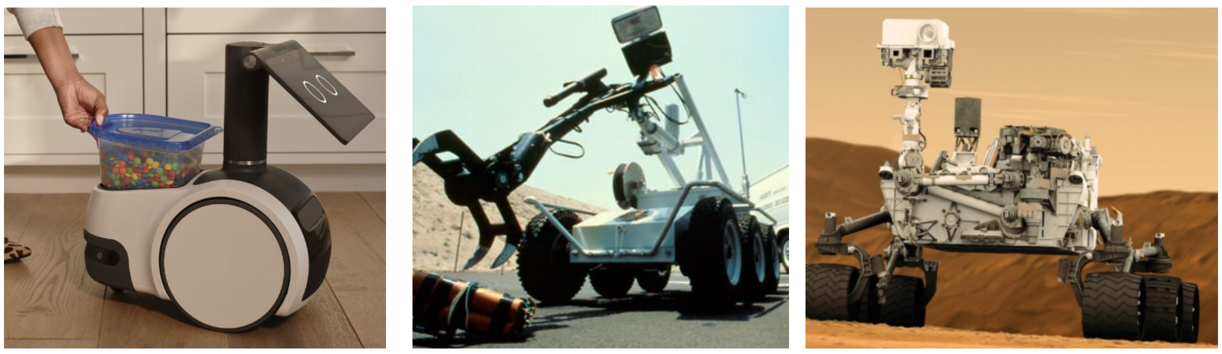
Figure 3: *An example of a delivery robot*

As you can see, the current delivery robots don't have a manipulator attached to them. Therefore, they are dependent on humans to keep the products of delivery in their carriers. With the improvements in AI, the process of pick and place can also be automated along with delivery, which our robot is designed to do.

Although our motivation is to build a mobile manipulator to assist pick and place of groceries and food items, the use case of such robots can be easily scaled to other industries with a few tweaks in design and size. Some of the applications that our Robot can perform or extended to are as follows:

1. **Warehouse managemen**t: To move carton packages from one point to another within the warehouse.
2. **Home Assistant**: As a home assistant which can help with cleaning, fetching coffee etc,.
3. **War-zones**: In war-ridden regions or areas of distress, mobile manipulators can help in recognition of mines and defusal of bombs.
4. **Mining industry**: Mines can be extremely risky for humans with the risk of collapse at any moment in time. To solve this problem, mobile manipulators can be utilized to help with mining activity.

5. **Space Applications**: Mobile robots have been extremely useful in space exploration especially for Mars missions.



a. **b.** **c.**

Figure 2: *a) Amazon's Astro being utilized as a home assistant, b) a robot carrying out bomb inspection, c) a robot used in space exploration*

IV. ROBOT DESCRIPTION:

IV. A. ROBOT TYPE

The robot we have designed is a mobile manipulator robot specifically for the purpose of deliveries.

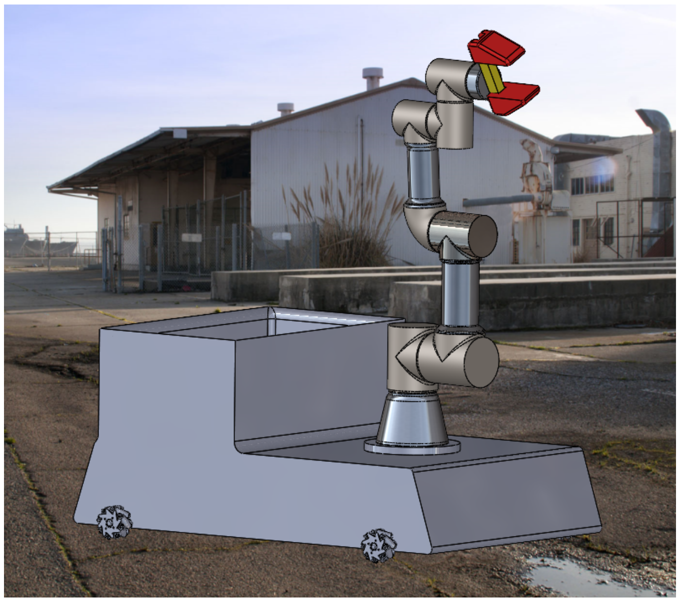


Figure 3: *A Glimpse of the Robot we have designed*

IV. B. MODEL DESCRIPTION:

IV. B. i. Degrees of Freedom of the Robot:

The Robot we have designed has multiple aspects which offer it a degree of freedom. Holonomicity is defined as the coverage of all degrees of freedom of a Robot in its plane/space of operation. The utilization of Mechanum wheels makes our Robot Holonomic in its plane of movement, i.e it covers all 3 degrees of planar motion. The arm attached to the mobile (ignoring end-effector) is a 6 - DOF Holonomic Robot. Therefore, the Robot as a whole covers 9DOF thus making it a redundant robot in space. The end - effector is a prismatic joint offering 1 DOF for picking an object. The DOF description is given below:

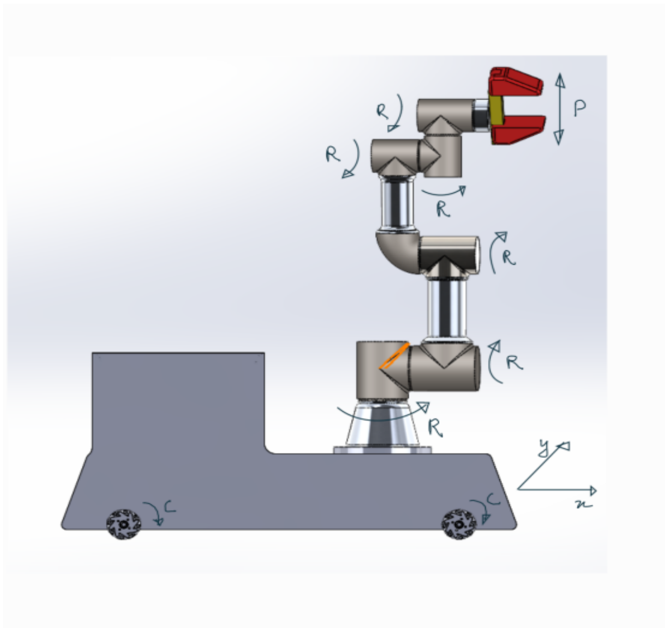
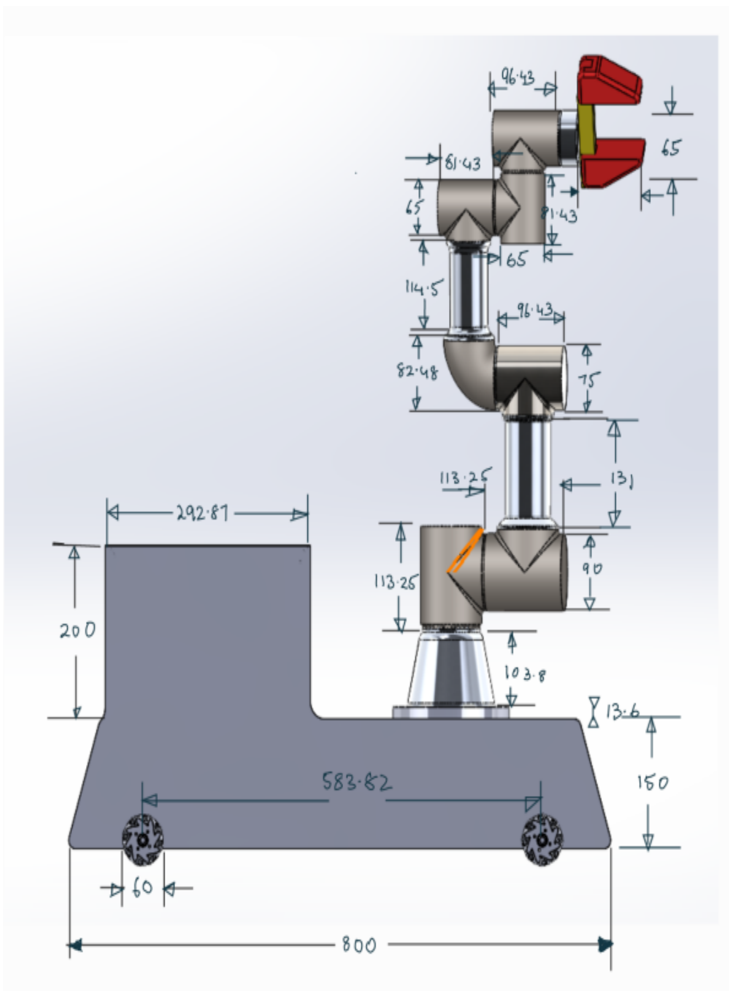


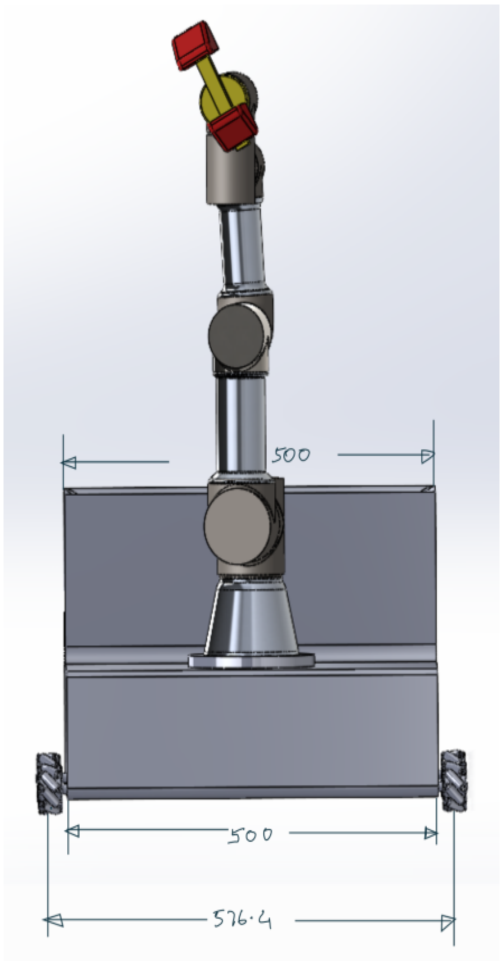
Figure 4: *DOF description of the Robot*

PART	JOINT	DOF
Mobile Base	Consists of 4 continuous joints to rotate a revolute joint and each mechanum wheel has 8 inclined roller revolute joints (at 45° each).	3 DOF in its plane of operation. The combination of 4 mechanum wheels have the capability to turn in place.
Arm	The arm is inspired from the UR3 Cobot. It consists of 6 revolute joints.	The 6 Revolute joints offer 6 DOF in it's space of operation
End - Effector	The end-effector consists of two prismatic fingers operating on the same axis of prismatic motion.	Since the prismatic axis is the same, it could be assumed to be a single prismatic joint. Therefore it offers 1 DOF.
Total DOF (Including the end-effector) = 10 R - Revolute Joint C - Continuous Joint P - Prismatic Joint		

IV. B. ii. Dimensions of the Robot

The robot contains a mobile base propelled by four mechanum wheels. An arm inspired from the UR3 (Universal Robots) is attached to the mobile base. The mobile base also contains a carrier for dropping the product to be delivered. *Figure 5* shows the front-view and side-view of the Robot with dimensions. All dimensions are in mm.

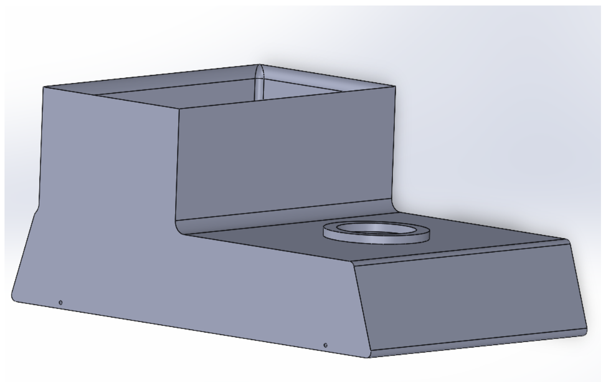
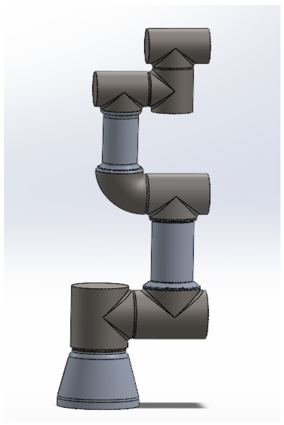
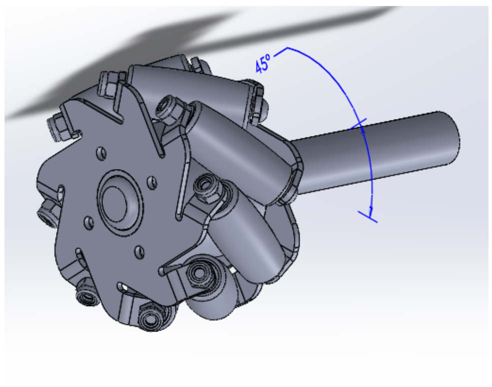




a. **b.**

Figure 5: *a) Side-View of the Robot, b) Front-View of the Robot*

The following table has the dimension description of individual components with images.

PART	**DESCRIPTION**
Base 	Body: Height : 350mm Length: 800mm Width: 500mm Carrier Box Dimensions Height: 150mm Length: 500mm Width: 292.87mm
Arm 	The dimensions of the arm are mentioned in *Figure 5: (b)*. The dimensions of the robotic arm are not very essential. The DH Parameters are, which are given in the next section.
Mechanum Wheels 	Wheel Diameter: 60mm Each roller is a continuous joint connected at 45°. Roller Diameter: 10.56mm Roller Length: 28mm

IV. C. DENAVIT-HARTENBERG PARAMETERS:

The Denavit-Hartenberg (DH) parameters are the 4 parameters associated with a particular convention for attaching reference frames to the links of a spatial kinematic chain, or robot manipulator. These 4 parameters are defined as follows:

PARAMETER	DESCRIPTION
d	offset along previous link's z-axis between the x-axes of the previous and current links' frames
θ	angle about previous link's z-axis , from x -axis of previous link's frame to current link's x-axis.
a	offset along previous link's x-axis between the z-axes of the previous and current links' frames.
α	angle about previous link's x-axis , from z -axis of previous link's frame to current link's z-axis.

The procedure followed to designate the DH Coordinate Frames:

1) Designate the z-axis at each joint about the axis of rotation in case of a revolute joint and along the axis of prismatic motion in case of a revolute joint.
2) For the base joint assume x-direction in any way but perpendicular to that z-direction.
3) The subsequent x-axes must be perpendicular to both current z and previous z-axes.
4) The x-axis drawn must intersect the previous z-axis.
5) The y-axis can be obtained by applying right-hand-thumb rule on z-axis and x-axis on each joint

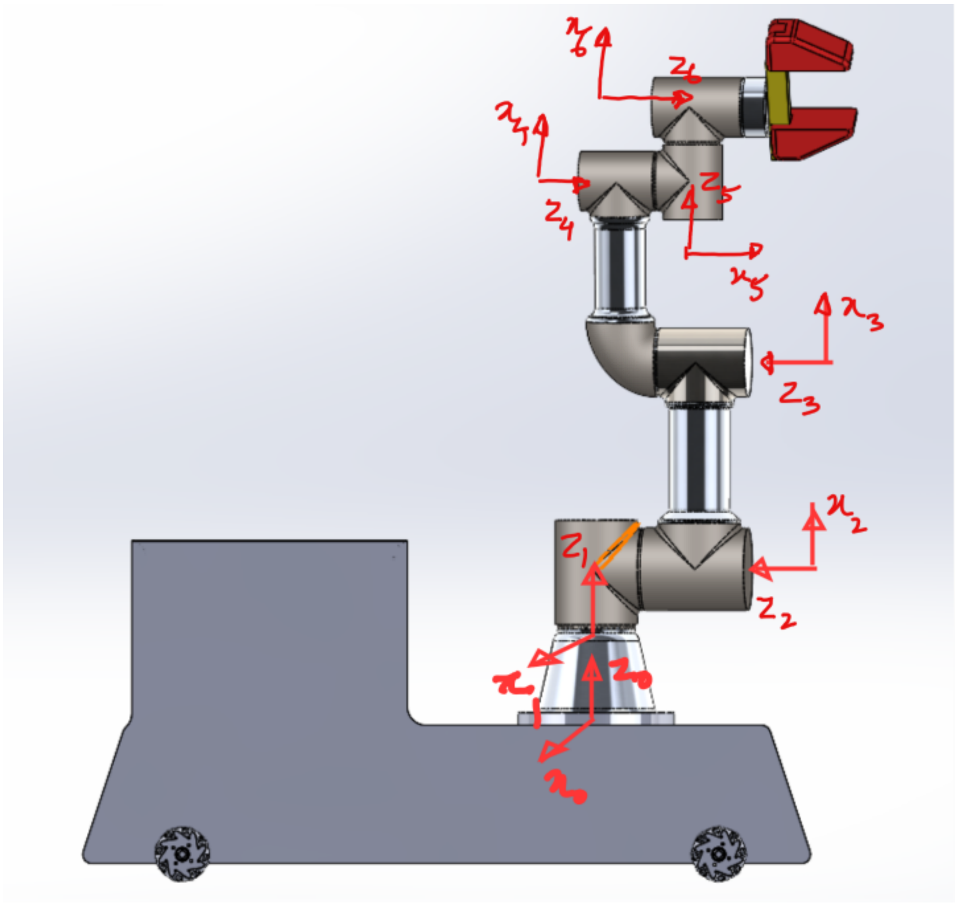


Figure 6: *DH Coordinate Frames*

The DH table for the manipulator used in our robot is given below:

Joints/Parameters	θ	a[m]	d[m]	α[radians]
0	0	0	0.1519	$\pi/2$
1	0	-0.24365	0	0
2	0	-0.21325	0	0
3	0	0	0.11235	$-\pi/2$
4	0	0	0.08535	$\pi/2$
5	0	0	0.0819	0

V. KINEMATICS

V. A. FORWARD KINEMATICS OF MANIPULATOR:

Once we have the DH Parameter table setup, we can obtain the final transformation matrix as follows:

$$T^0_{eff} = T^0_1.T^1_2.T^2_3.T^3_4.T^4_5$$

Where

$$T^{i-1}_i = Rot(z_i).Trans(z_i).Trans(x_i).Rot(x_i)$$

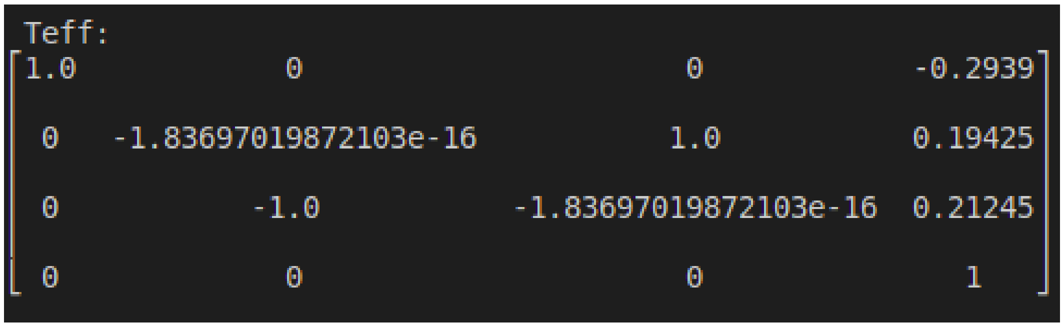
Here T^{i-1}_i is the transformation of the next link with respect to the previous link. Since, there are 6 joints; there can only be 5 relative transformations. The 0th joint mentioned is the first revolute joint relative to which end-effector position is determined. Now, as an additional step it is always important to obtain the end-effector position with respect to the base of the robot. Therefore, the end-effector position w.r.t joint 0, T^0_{eff} is multiplied by T^b_0 which results in the final transformation of the end-effector w.r.t the base of the robot

T^0_{eff} is calculated as:

$$\begin{matrix} 1 & 0 & 0 & 0.163 \\ 0 & \cos(\pi) & -\sin(\pi) & 0 \\ 0 & \sin(\pi) & \cos(\pi) & 0.279 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Thus the final transformation matrix can be given as:

$$T^b_{eff} = T^b_0.T^0_{eff}$$



```
Teff:
⎡1.0          0                      0                 -0.2939⎤
⎢                                                             ⎥
⎢ 0   -1.83697019872103e-16         1.0               0.19425⎥
⎢                                                             ⎥
⎢ 0          -1.0          -1.83697019872103e-16  0.21245⎥
⎢                                                             ⎥
⎣ 0           0                      0                    1   ⎦
```

Figure 7: *Transformation Matrix obtained for our manipulator*

V. B. WORKSPACE STUDY:

The workspace of a manipulator is measured as its volume of reach or in other words, the workspace it can function in. Additionally, for our Robot we have a mobile base. So, our Robot will have a fixed workspace but the workspace itself moves wherever the Robot moves, thus increasing the overall workspace of the Robot. Forward Kinematics' most important application is to establish the final workspace of the Robot. Once we have obtained the transformation matrix, the immediate next step is to have an understanding of the Robot's workspace. For ease, we have only done study of the Arm's workspace, since with a mobile base, it'll be the same with respect to the position of the mobile base in space. There are two common ways to obtain the workspace of the Robot:

1) One procedure is to vary all the joint angles in small steps in a loop and apply the obtained transformation matrix to get the end points which basically becomes all the points that the robot can reach.

2) Another procedure is to extend the Robot's arm in such a way that it becomes the maximum length and then change the first two joint angles which would give us the maximum reachable points of the Robot. But that procedure will not tell us about all the reachable points within the workspace.

The procedure implemented for obtaining the approximate workspace of the Robot.

1) Have defined a random array of size 5 between $-\pi$ to π.

2) Looping each joint 5 times consecutively with a complexity of O(n^6).

3) The final homogenous matrix is multiplied with the base point i.e [0, 0, 0, 1] to obtain different end-effector points. The different end-effector points are the workspace values of the manipulator.

The diagram gives the workspace of the manipulator for 3^6 points.

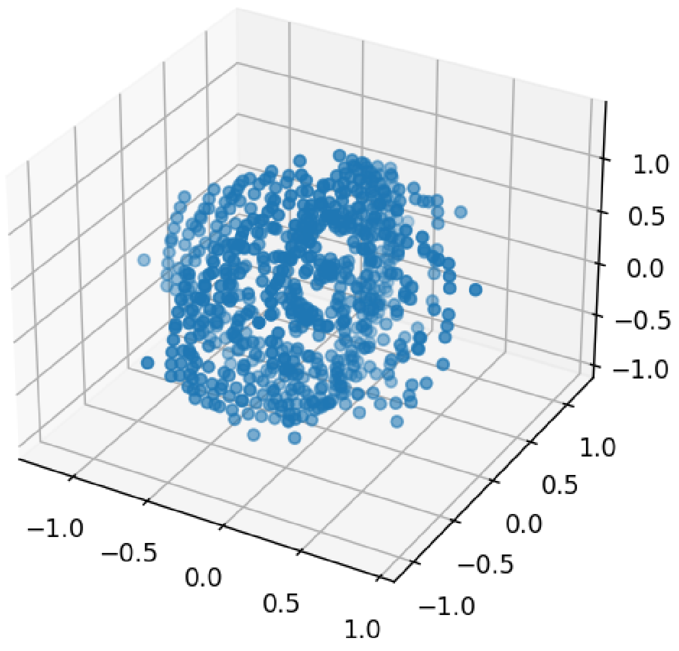


Figure 8: *Workspace of the manipulator*

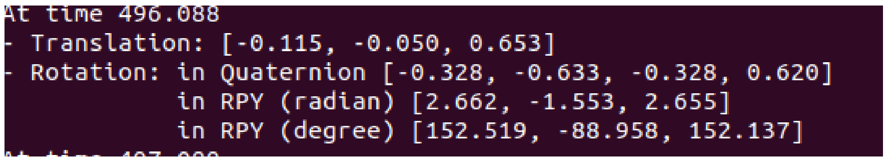
The workspace produces all the points that are reachable by the Robot. But, in fact all points are not reachable because programmatically speaking, the link thickness is not considered. In reality, links have some thickness because of which self-collision can occur. Therefore, we can say that *Figure 7* describes an approximate workspace of the Robot without the consideration of the possibility of self-collision. Another assumption is that, there are no constricting boundaries around the Robot. The Robot can move freely wherever it can move.

V. C. FORWARD KINEMATICS VALIDATION:

For forward kinematics, we have written a fk_validation.py script. The script takes a mock initial and final angle and computes the transformation matrix. We spawn the Robot on Gazebo and measure it's initial position using the command:

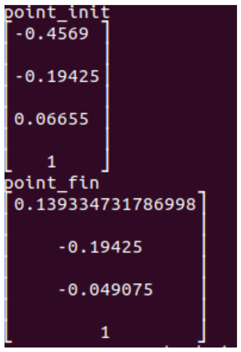
rosrun tf tf_echo /u3_base_link /ee_link

This is the initial position of the Robot at angles $[0, 0, 0, 0, 0, 0]$. The value comes out to be - Translation: [-0.115, -0.051, 0.653]. In fk_validation.py script, we print out the transformation of the point [0, 0, 0] for initial angles and final angles. This is where we face the challenge. The final transformation matrix will result in end points that are different from end-points on gazebo. This is because the initial configuration of the Robot in Gazebo is unknown. It always gives angles as [0, 0, 0, 0, 0, 0] however the configuration maybe. But in the program, for angles [0, 0, 0, 0, 0, 0], there is a defined end position. So to tackle this challenge,
we validate forward kinematics by taking euler distances between the points obtained through code and points obtained through tf_echo after the Robot moves through the final angles which is published in fk_validation.py script.



```
At time 496.088
- Translation: [-0.115, -0.050, 0.653]
- Rotation: in Quaternion [-0.328, -0.633, -0.328, 0.620]
            in RPY (radian) [2.662, -1.553, 2.655]
            in RPY (degree) [152.519, -88.958, 152.137]
```

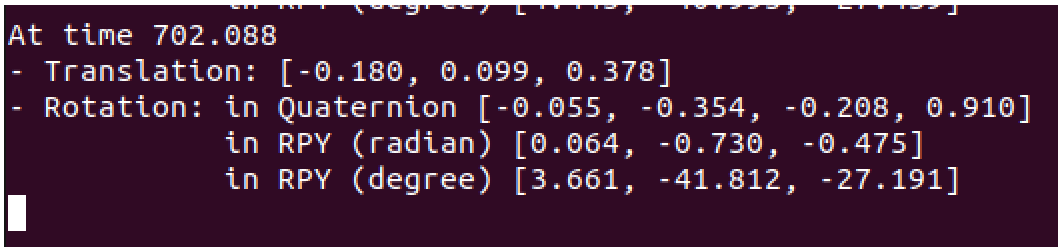
a.



```
point_init
[-0.4569  ]

[-0.19425 ]

[0.06655  ]

[   1     ]
point_fin
[0.139334731786998]

[     -0.19425    ]

[     -0.049075   ]

[        1        ]
```

b.

As a final step, we publish the final angle values to Gazebo and change the Robot's position and measure the Robot's end-effector final position using the above command.



```
At time 702.088
- Translation: [-0.180, 0.099, 0.378]
- Rotation: in Quaternion [-0.055, -0.354, -0.208, 0.910]
            in RPY (radian) [0.064, -0.730, -0.475]
            in RPY (degree) [3.661, -41.812, -27.191]
```

c.

Figure 9: *a) The initial position of end-effector on Gazebo, b) The initial and final position obtained through fk_validation.py, c) The final position of end-effector on Gazebo*

We use the Euclidean distance formula:

$$D = \sqrt{((x_2 - x1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2)}$$

We apply the Euclidean distance formula and find out the distance between the initial and final positions obtained through the program and initial and final distance obtained through Gazebo.

We see that, the distance between two points through Gazebo = 0.32
And, the distance between two points through code = 0.6069

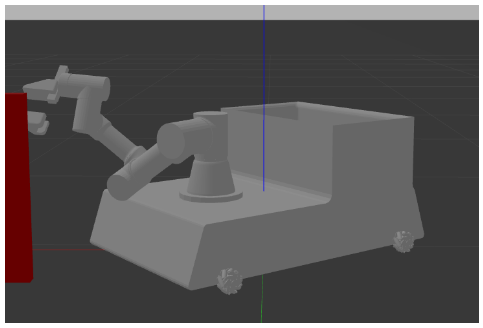


Figure 11: *The orientation of the Robot changes when the Arm moves*

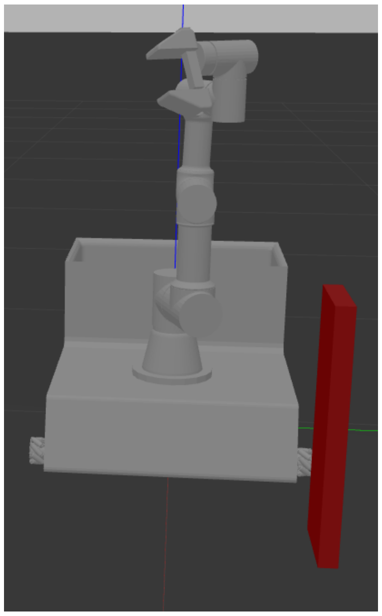


Figure 12: *The initial Orientation of the Robot*

The error in measurement of the end-effector position is due to the orientation shift of the mobile base when the arm moves. This causes an error in tf measurement on Gazebo. The transformation matrix used is correct because we use the same transformation matrix for inverse kinematics to generate a circular trajectory which it does on Gazebo.

V. D. INVERSE KINEMATICS:

Inverse kinematics is the problem of finding the joint angles of the manipulator, given its end effector coordinates.
In order to find the joint angles, the first step is to find the Jacobian matrix.
The Jacobian matrix can be calculated as follows :

Revolute Joints

$$J = \begin{bmatrix} J_v \\ J_w \end{bmatrix} = \begin{bmatrix} R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (d^0_n - d^0_{i-1}) \\ R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix}$$

Prismatic Joints

$$J = \begin{bmatrix} J_v \\ J_w \end{bmatrix} = \begin{bmatrix} R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{bmatrix}$$

From the DH table that was shown earlier, Jacobian matrix can be calculated for a given configuration. The Jacobian matrix calculation method for a revolute joint and a prismatic joint is shown above. The upper half of the Jacobian matrix constitutes the linear components of velocity and the lower half constitutes the angular components of the velocity. Thus, using the Rotation matrix which is shown as R^0_{i-1} , it can be substituted to the Jacobian and the final Jacobian matrix can be calculated.

After the Jacobian matrix is calculated, it can be used to calculate $\dot{q}$ which is the rate of change of joint angle, using the formula:

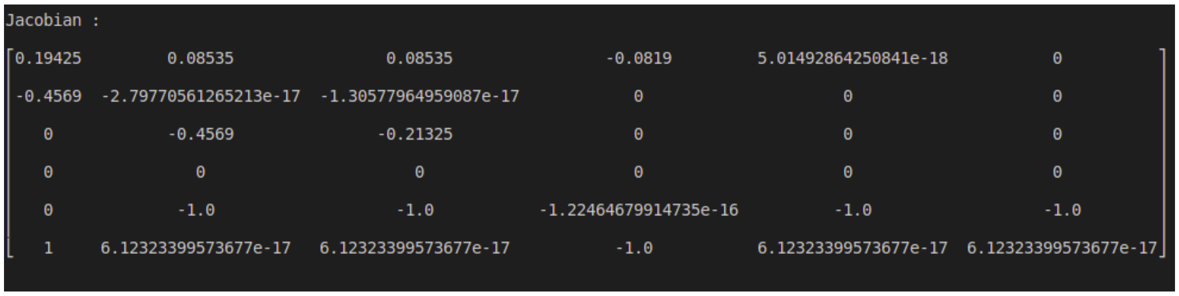
$$\dot{q} = J^{-1}.v$$

Where J^{-1} is the inverse of the Jacobian matrix, and v is the velocity of the end effector.

Now, the joint value is updated as follows:

$$q_{new} = q + \dot{q} \,.\, \Delta t$$

The joint angles are updated as stated above. This can be done in a loop to find the final set of joint angles for a given trajectory from a point $(x1, y1)$ to $(x2, y2)$.

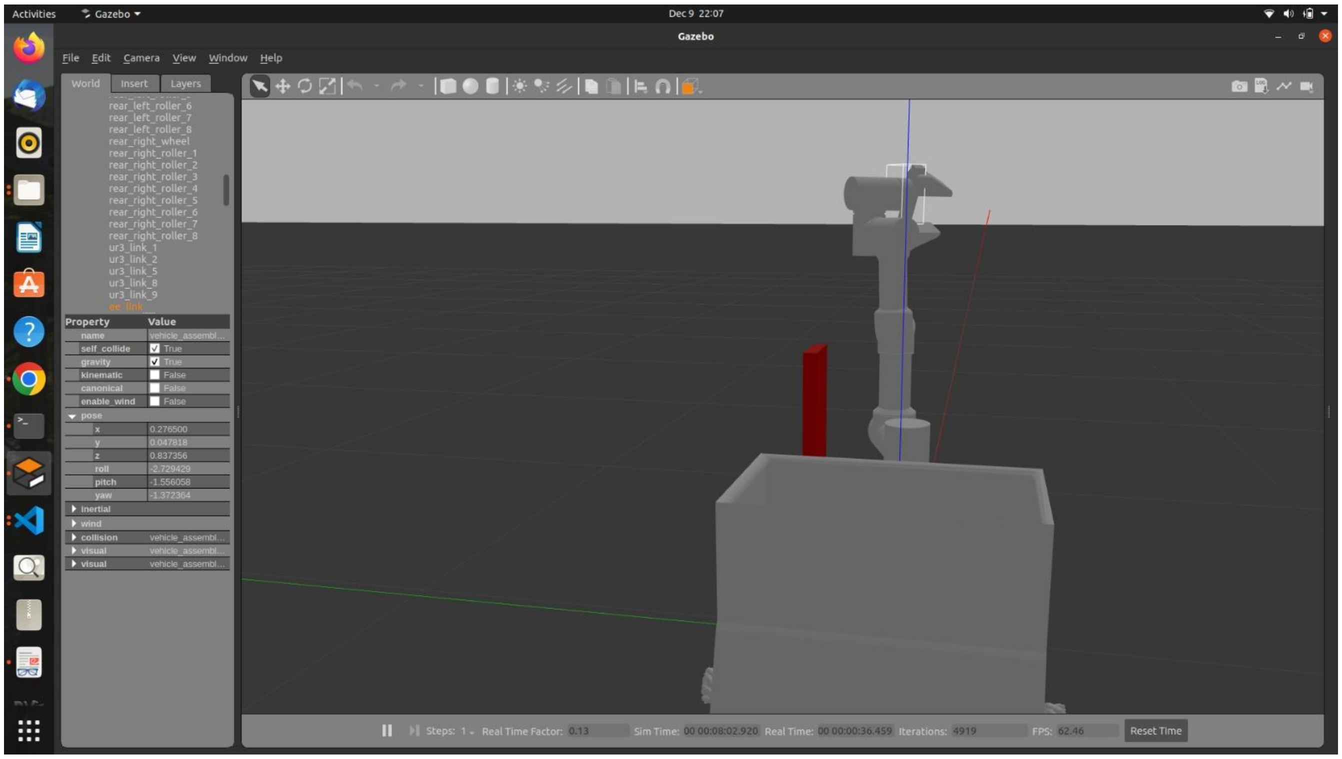


```
Jacobian :
⎡0.19425         0.08535               0.08535               -0.0819         5.01492864250841e-18           0          ⎤
 -0.4569  -2.79770561265213e-17  -1.30577964959087e-17          0                   0                     0
    0            -0.4569               -0.21325                 0                   0                     0
    0               0                     0                     0                   0                     0
    0              -1.0                  -1.0          -1.22464679914735e-16      -1.0                  -1.0
⎣   1     6.12323399573677e-17   6.12323399573677e-17         -1.0         6.12323399573677e-17  6.12323399573677e-17⎦
```

Figure 13: *Jacobian obtained for the initial Robot configuration at* $\theta = [0, 0, 0, 0, 0, 0]$

V. E. INVERSE KINEMATICS VALIDATION:

For the inverse kinematics validation, we have considered velocity inverse kinematics wherein we generate a circular trajectory. The joint-angles obtained using inverse kinematics are published into the Gazebo joints to obtain a circular end-effector motion. The motion is extremely slow and not clearly visible, therefore to further validate, we took the initial position note on Gazebo for the end-effector, then let the motion run and took the final position value. The two values match with a minor error as they should after completion of a circular trajectory. The error comes because of errors in numerical integration methods. Numerical methods to integrate are an approximation of the actual integration process and therefore there is bound to be some error. If the steps aren't small enough, the curves can't be approximated to straight lines because of which errors occur in integration

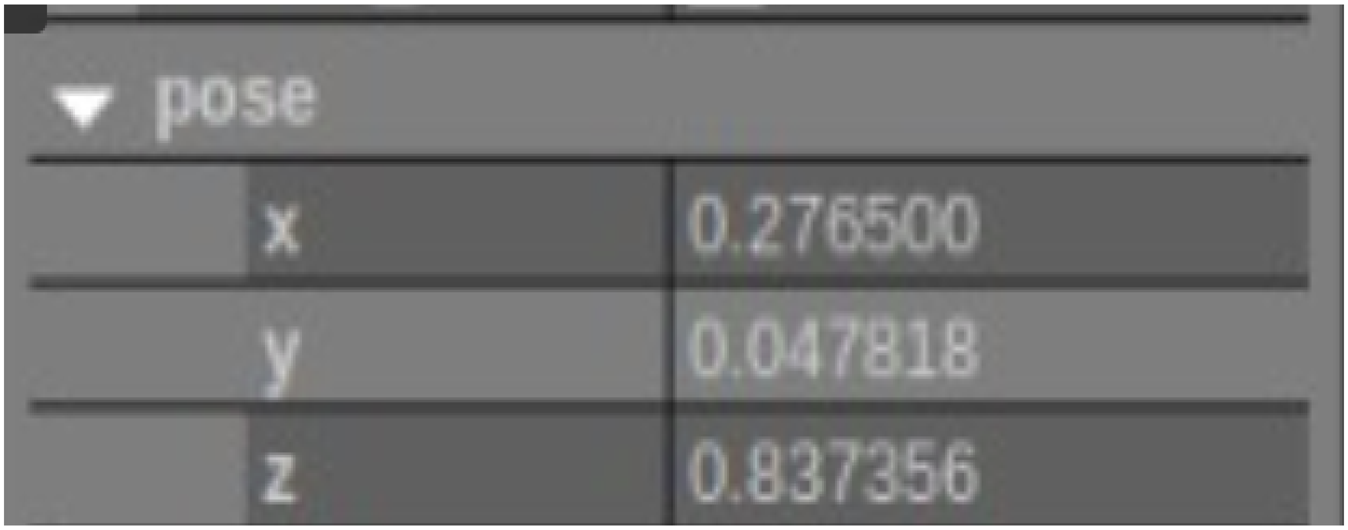


```
rear_left_roller_6
rear_left_roller_7
rear_left_roller_8
rear_right_wheel
rear_right_roller_1
rear_right_roller_2
rear_right_roller_3
rear_right_roller_4
rear_right_roller_5
rear_right_roller_6
rear_right_roller_7
rear_right_roller_8
ur3_link_1
ur3_link_2
ur3_link_5
ur3_link_8
ur3_link_9
ee_link
```

Property	Value
name	vehicle_assembl...
self_collide	✓ True
gravity	✓ True
kinematic	False
canonical	False
enable_wind	False
pose	
x	0.276500
y	0.047818
z	0.837356
roll	-2.729429
pitch	-1.556058
yaw	-1.372364
inertial	
wind	
collision	vehicle_assembl...
visual	vehicle_assembl...
visual	vehicle_assembl...

```
Steps: 1  Real Time Factor: 0.13  Sim Time: 00 00:08:02.920  Real Time: 00 00:00:36.459  Iterations: 4919  FPS: 62.46  Reset Time
```

Figure 14: *Initial position of the Robot in Gazebo*



pose	
x	0.276500
y	0.047818
z	0.837356

Figure 15: *The position values before the start of trajectory*

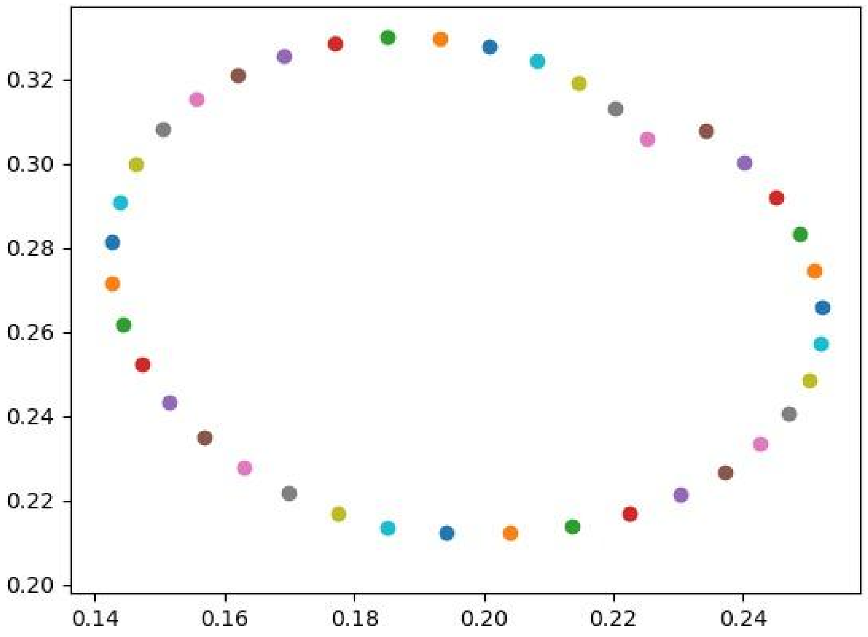
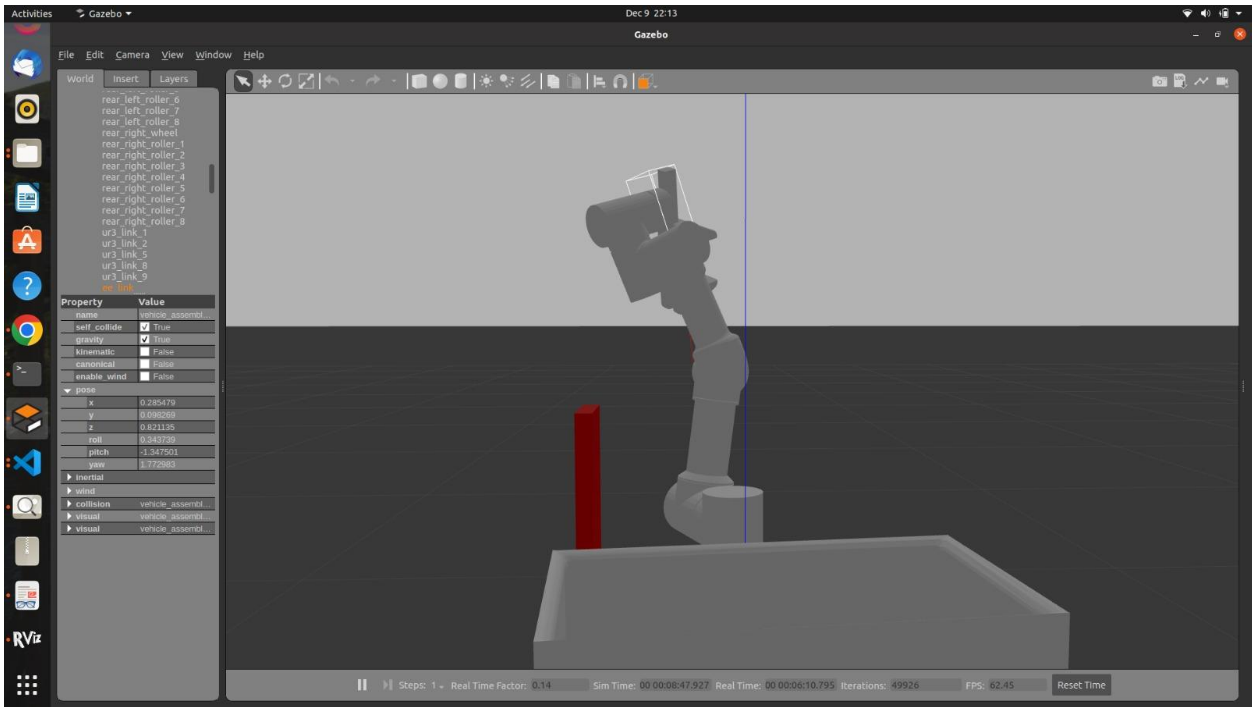


Figure 16: *The approximate circular trajectory generated using inverse kinematics*

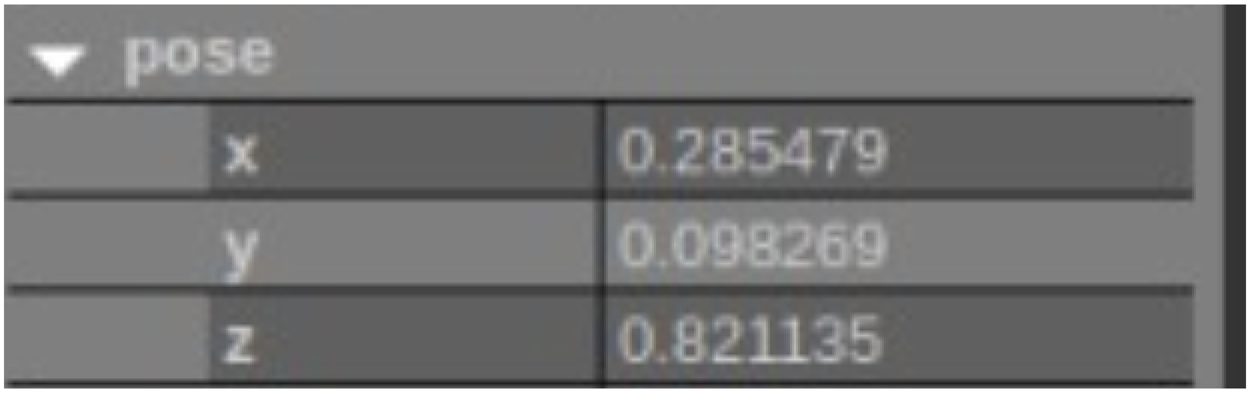


rear_left_roller_6
rear_left_roller_7
rear_left_roller_8
rear_right_wheel
rear_right_roller_1
rear_right_roller_2
rear_right_roller_3
rear_right_roller_4
rear_right_roller_5
rear_right_roller_6
rear_right_roller_7
rear_right_roller_8
ur3_link_1
ur3_link_2
ur3_link_5
ur3_link_8
ur3_link_9
ee_link

Property	Value
name	vehicle_assembl...
self_collide	✓ True
gravity	✓ True
kinematic	False
canonical	False
enable_wind	False
pose	
x	0.285479
y	0.098269
z	0.821135
roll	0.343739
pitch	-1.347501
yaw	1.772983
inertial	
wind	
collision	vehicle_assembl...
visual	vehicle_assembl...
visual	vehicle_assembl...

Steps: 1 Real Time Factor: 0.14 Sim Time: 00 00:08:47.927 Real Time: 00 00:06:10.795 Iterations: 49926 FPS: 62.45 Reset Time

Figure 17: *The Robot while making the trajectory*



```
▾ pose
      x     0.285479
      y     0.098269
      z     0.821135
```

Figure 18: *Final Position of the Robot after making the trajectory*

VI. CONTROL METHODS:

For the Gazebo simulations, we have utilized position controllers on each arm joint and for each wheel we have utilized PID velocity controllers. The pick and place operation is performed using moveit which creates a joint group that utilizes PID position_controllers/JointTrajectoryController. For MoveIt simulations, we have created two groups, one for the arm joints and the other for hand joints. For the movement of the mobile base, PID velocity_controllers/ VelocityJointController is used which receives velocity input to propel the mobile base

VII. MODEL ASSUMPTIONS

Certain assumptions were made during the design and development of the Robot:

- The links, chassis and joints are rigid i.e., they undergo no deformations
- Environmental resistances like drag, friction are not considered.
- There are no thermal effects taken into consideration.
- Self-collision matrix is not considered.
- The environment in which the robot has been spawned is static, i.e there are no moving obstacles in the environment.
- The actuators are assumed to have any possible torque, as much as is required to move the robot.

VIII. GAZEBO AND RVIZ SIMULATIONS:

The Gazebo and RVIZ simulations are recorded as a video in the following link:
https://drive.google.com/file/d/1qItt6UyC16aCXbElswLgnpFnVTCxZOQI/view?usp=share_link

IX. CAD FILES

The CAD Files are present in the drive link:
https://drive.google.com/drive/folders/18lf77_RY5AEO_JbB1f75tzr0ey_o-69q?usp=sharing
The assembly 'vehicle_assembly' is the final assembly of the Robot. For the arm assembly go to '/Arm/ur3_assembly'

X. PROBLEMS FACED :

1. The CAD model generated using SOLIDWORKS for the manipulator, was inspired from UR3. But, the design was slightly modified according to the requirements of our objective and simulations. Hence, setting up DH parameters for the modified version was a bit difficult. An approximate DH table has been set up and hence, there are some variations between the expected and actual end effector positions.
2. Dealing with namespaces in ROS was another huge challenge. The controllers relied on namespaces and thus, it caused a lot of problems. The controllers didn't load properly many times in the course of the project.
3. Setting up moveit controllers to work along with Gazebo controllers was another tedious task. The Gazebo controllers are single joint controllers while Moveit controllers work on a joint group. Pairing the two caused a lot of issues

4. The object pickup was very difficult since it kept slipping. For this, we had to use an external plugin which increased the friction between the object and the gripper to lift it.
5. Movement of mechanum wheels was difficult initially. We used a plugin for the robot to have a planar motion with Mechanum wheels.
6. Understanding the home position of the Robot i.e, we were not able to figure out what the initial spawn angles of the Robot were because whenever the Robot spawns moveit always gives 0 angle values.

XI. LESSONS LEARNED

1. Having extensively used Solidworks for this project, we learned how to efficiently design certain basic structures in a fast manner.
2. Namespaces in ROS was a huge problem in our project. We learned how to deal with them. Now, we understand, where to check and which file to access in order to rectify namespace errors.
3. We learned how to configure controllers properly. We had to make two sets of controllers to run at once.
4. Setting up and using moveit for future projects, would now be easier since we extensively worked on it as well.

XII. FUTURE WORKS

While the Robot is minimalistically designed to cater to basic pick, place and move tasks, it is still very primitive in its functionality. Currently, it just does command-based pick, place and move motions. Use for such Robots drastically increases with the introduction of Autonomy. As future works, introduction of vision techniques to make its movements autonomous, to recognize the object of delivery based on the input image given through the application, reach its end point automatically after receiving the end location data through the application and improve its capability to work collaboratively with humans are some of the features which we would like to work on.
Additionally, development of a software application which could take orders from the customers and supply the data to the Robot would be very interesting to do.

XIII. REFERENCES

1) https://www.starship.xyz/
2) Richard M. Murray, S. Shankar Sastry, and Li Zexiang. A Mathematical Introduction to Robotic Manipulation. CRC Press, Inc., Boca Raton, FL, USA
3) Robot Dynamics and Control by Mark W Spong
4) https://wiki.ros.org
5) Youtube tutorials on solidworks

XIV. INDIVIDUAL CONTRIBUTIONS

Aniruddh Balram:

1. Designing the model
2. Setting up URDF
3. Workspace study and validation
4. Forward Kinematics and validation

Badrinarayanan Raghunathan Srikumar:

1. Setting up controllers and plugins
2. Python code for simulation.
3. Inverse Kinematics and validation

Moveit was an entirely new package. Hence we set it up together.