ENPM 662 – Introduction to Robot Modelling

PROJECT 2 REPORT

Tele operable Rover with Multipurpose Manipulator

Submitted by Team 29

- Piyush Goenka – 120189500
- Gautam Sreenarayanan Nair - 119543092

Table of Contents

Introduction

This project primarily focusses on implementing forward and inverse kinematics on a tele-operable mobile rover with manipulator arm attached to it. The forward and inverse kinematics are validated using sympy and by Gazebo simulation.

Application

The robot is a tele-operable rover with a multi-purpose arm. The rover has a toolbox of end-effectors that the manipulator could pick up from depending on the application. For the purpose of this project, we have utilized a vacuum gripper as end effector to present the implementation of forward and inverse kinematics. Other end-effector tools could be implemented in future depending on the applicational requirement of the robot accordingly.

Robot type

It is a mobile rover with a manipulator arm.

DOFs and dimensions

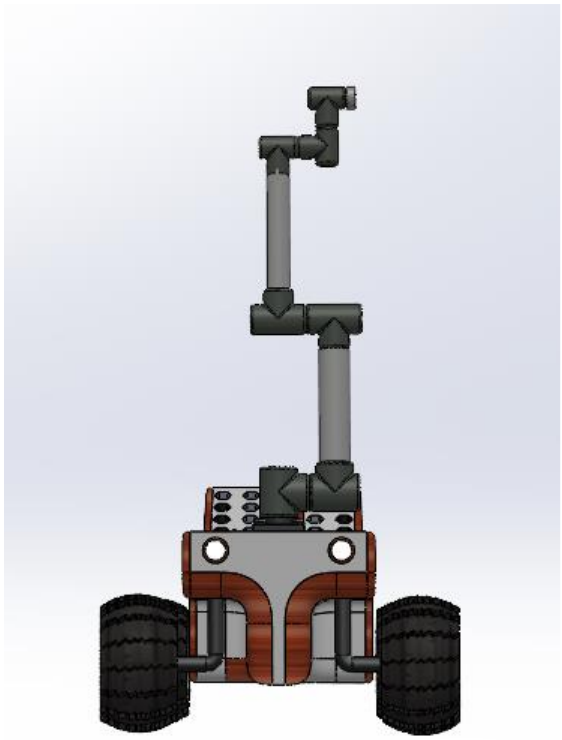
The rover has three degrees of freedom, and the manipulator arm has six degrees of freedom.

The rover has a height of around 850mm and a length and breadth of 1694 mm and 750mm respectively. The manipulator is inspired from UR10 of Universal Robotics. The UR10 is a medium-duty industrial collaborative robot that combines both long reach and high payload. It can seamlessly integrate into a wide range of applications and delivers endless automation possibilities [1]. The height of the manipulator is around 1.73 meters, and all the joints are of revolute type. Please refer the joint rotations section of this document for more details regarding the same.

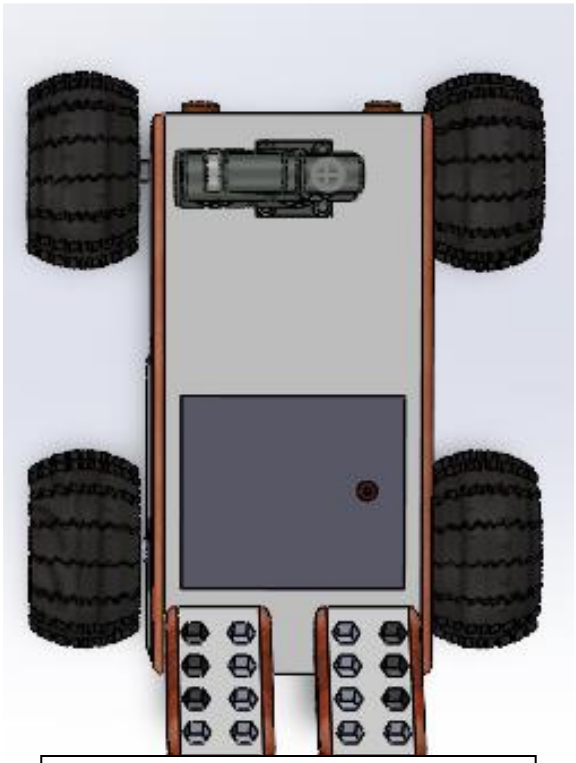
CAD Model

The model contains a rover and a manipulator attached to it. The rover is rear wheel driven and both the front wheels are individually steerable. There is a tool set holder at the rear of the rover that can hold multiple end-effectors. These end-effectors can be changed and used by the manipulator itself depending on application. The demonstration of the same is out of scope of this project and can be implemented as a future work.

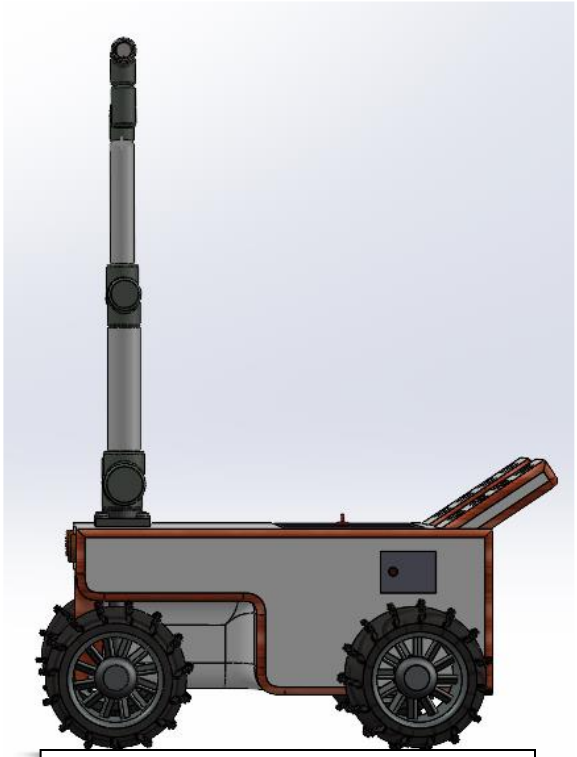
Rendered Model



Front View



Top View



Side View

Denavit-Hartenberg convention

The Denavit-Hartenberg or DH convention is a commonly used method for selecting frames of reference in robotic applications. In this method the homogenous transformation is a product of four basic transformations resulting in the below given transformation matrix [2].

$$T_{i-1}^{i} = \begin{bmatrix} cos\theta_i & -cos\alpha_i sin\theta_i & sin\alpha_i sin\theta_i & a_i cos\theta_i \\ sin\theta_i & cos\alpha_i cos\theta_i & -sin\alpha_i cos\theta_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

D-H Parameter definitions :

a_i: Distance from z_{i-1} to z_i axes along the x_i axis.

d_i: Distance from origin O_{i-1} to the intersection of the z_{i-1} and x_i axes along the z_{i-1} axis.

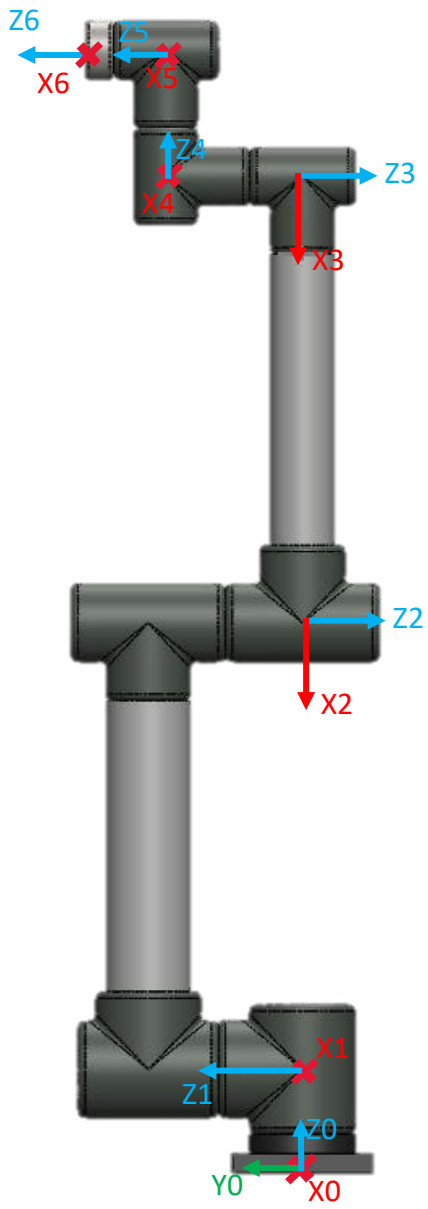
α_i: Angle between z_{i-1} and z_i axes about the x_i axis.

θ_i: Angle between x_{i-1} and x_i axes about the z_{i-1} axis.

These parameters are found out after frame assignment and the parameters for this project is given below in the DH parameter table section. The transformation matrix given above is used to find the transformation between links ‘i’ and ‘i-1’.

Manipulator Frames

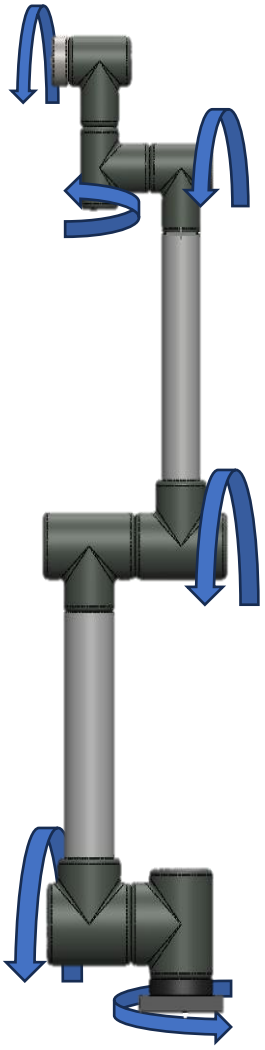
The frame assignment for DH parameter calculation is given below. For ease of representation y-axis is neglected from the diagram as it is not very relevant for DH parameter calculations. If required, the same can be easily calculated using right hand rule.



Frame Assignment

Joint Rotations

The manipulator has only revolute joints. In total there are 6 revolute joints available in the manipulator and a diagrammatic representation of the joint rotations are given below.



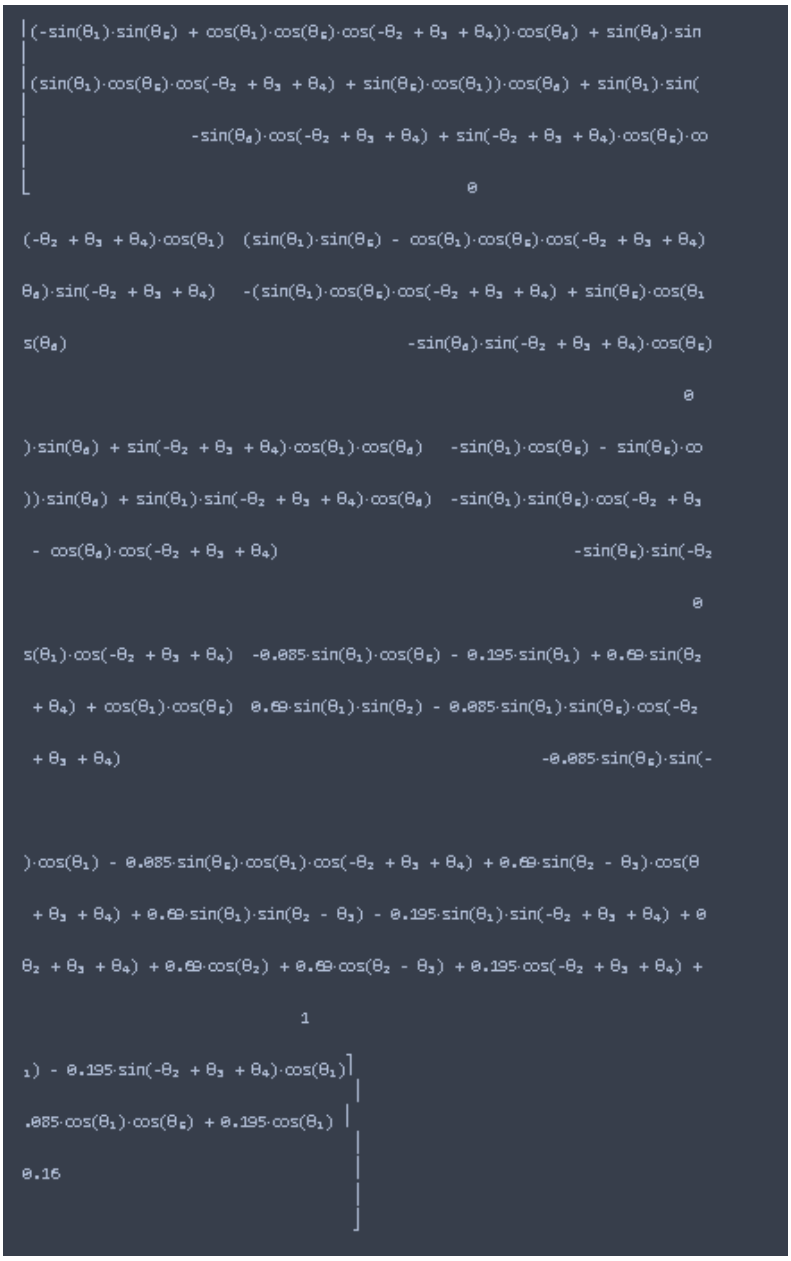
Joint Rotations

DH Parameter Table

Link	a_i	alpha_i (degrees)	d_i	Theta (degrees)
1	0	-90	160 mm	theta_1
2	-690 mm	180	0 mm	theta_2 + 90
3	-690 mm	0	0 mm	theta_3
4	0	-90	-195 mm	theta_4+ 90
5	0	-90	195 mm	theta_5
6	0	0	85 mm	theta_6

Forward kinematics

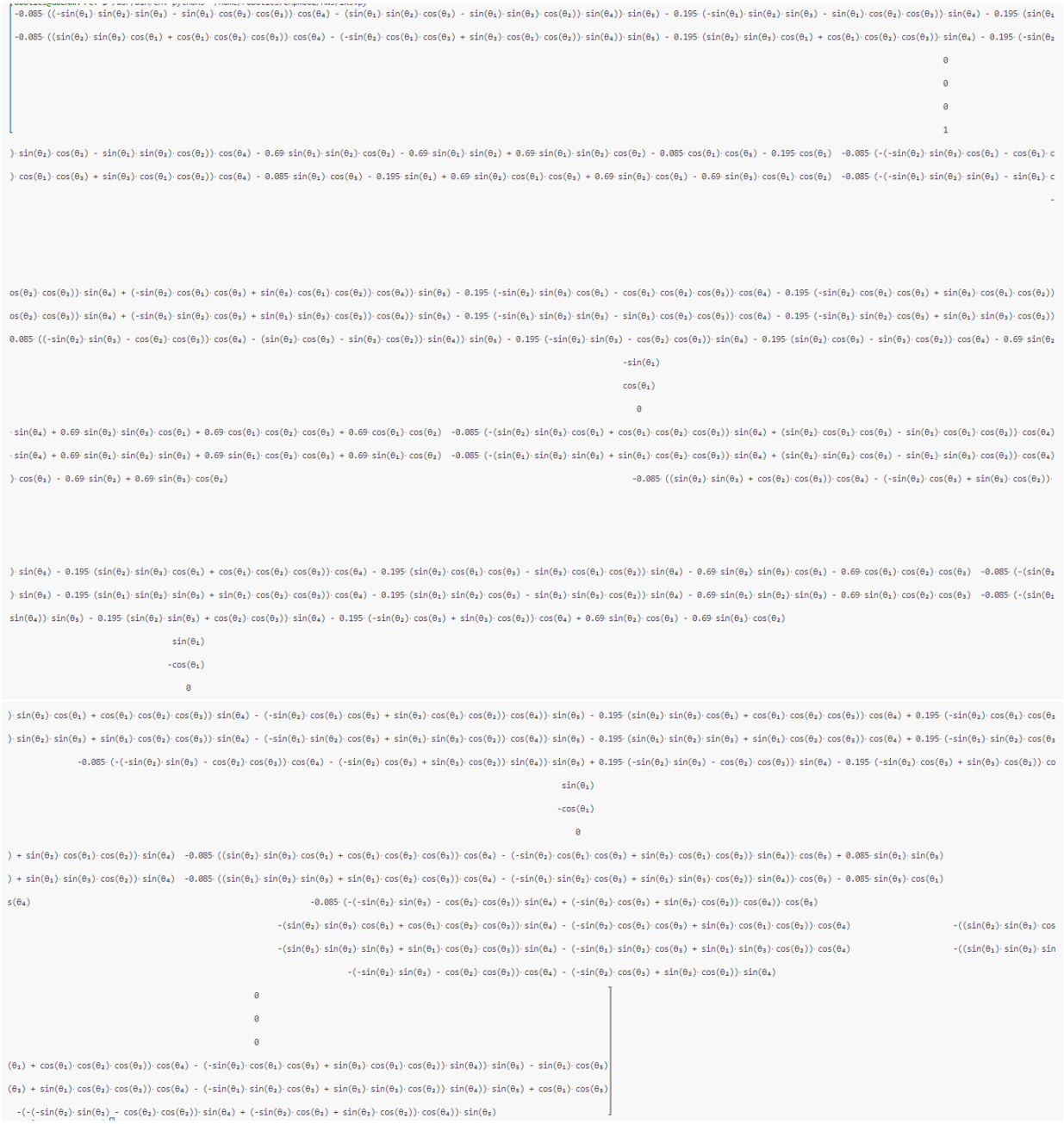
Forward kinematics involves using the joint variables of a robot to calculate the position and orientation of each link, including the end-effector. By assigning a coordinate frame to each link and utilizing rigid motion methods, we establish the configuration of each link in relation to its neighboring frames. The process of determining the position and orientation of all robot links relative to one another is known as forward kinematics [3]. By substituting the joint variables (joint angles in our case), in the final transformation matrix between the final and base link, the end effector pose can be found out. The final transformation matrix between the final link and the base link is given below.

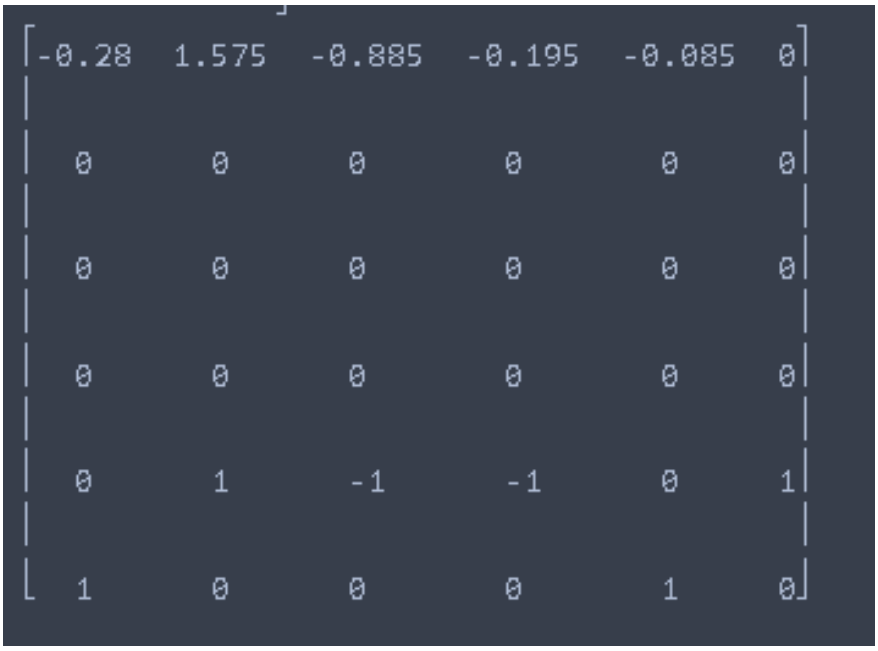


Final Transformation Matrix

Inverse kinematics

Inverse Kinematics (IK) involves the challenge of finding a suitable set of joint configurations that facilitate the smooth, rapid, and accurate movement of end effectors to predefined positions [4]. The joint configurations in this application are joint angles. The Jacobian matrix was found for our application. The Jacobian matrix is given below.





-0.28	1.575	-0.885	-0.195	-0.085	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

Jacobian matrix at zero joint angles.

Forward kinematics validation

We validated the forward kinematics in ROS by first setting the joint angles to some known set of values. Then we obtained the transformation matrix of the end effector link w.r.t the base link of the arm manipulator. Using the x, y and z position values of this transformation matrix, we validated our forward kinematics.

The validation of Forward Kinematics was performed for 3 configurations:

Configuration 1:

Θ1	Θ2	Θ3	Θ4	Θ5	Θ6
4.73	0.09	1.62	-1.51	-0.26	0.11

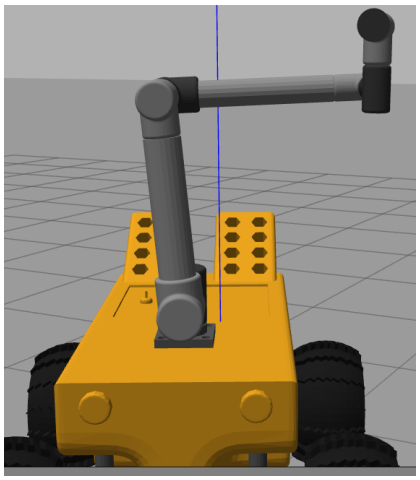
Sympy Output: (refer the x,y,z values from output)

```
#  Forward Kinematics
robot = RobotArm()
end_effector_pose = robot.forward_kinematics(4.73,0.09,1.62,-1.51,-0.26,0.11)
rounded_end_effector_pose = end_effector_pose.evalf().applyfunc(lambda x: roun
sp.pprint(rounded_end_effector_pose, use_unicode=True, num_columns=120, wrap_l
```

```
⎡-0.239  0.027   0.971  0.266⎤
⎢                            ⎥
⎢-0.967  0.087   -0.24  0.614⎥
⎢                            ⎥
⎢-0.091  -0.996  0.005  1.071⎥
⎢                            ⎥
⎣  0       0       0     1.0 ⎦
```

Gazebo output

```
EE position
x: 0.26
y: 0.61
z: 1.07
```



Configuration 2:

Θ1	Θ2	Θ3	Θ4	Θ5	Θ6
0	0	0	0	0	0

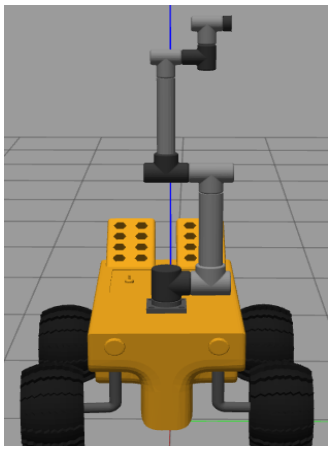
Sympy Output: (refer the x,y,z values from output)

```
#  Forward Kinematics
robot = RobotArm()
end_effector_pose = robot.forward_kinematics(0, 0, 0, 0, 0, 0)
sp.pprint(end_effector_pose)
```

```
⎡1  0   0    0  ⎤
⎢                ⎥
⎢0  0   1  0.28 ⎥
⎢                ⎥
⎢0  -1  0  1.735⎥
⎢                ⎥
⎣0  0   0    1  ⎦
```

Gazebo Output

```
EE position
x: 0.0
y: 0.27
z: 1.73
==================================
^Crobotics@ubuntu:~/colcon_ws$
```



Configuration 3:

Θ1	Θ2	Θ3	Θ4	Θ5	Θ6
1.56	0	0	0	0	0

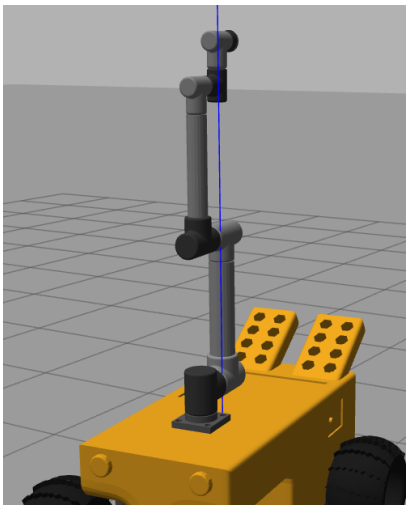
Sympy Output: (refer the x,y,z values from output)

```
#  Forward Kinematics
robot = RobotArm()
end_effector_pose = robot.forward_kinematics(1.56,0,0,0,0,0)
rounded_end_effector_pose = end_effector_pose.evalf().applyfunc
sp.pprint(rounded_end_effector_pose, use_unicode=True, num_colu
```

```
⎡0.011   0    -1.0   -0.28⎤
⎢                        ⎥
⎢ 1.0    0    0.011  0.003⎥
⎢                        ⎥
⎢  0    -1.0   0     1.735⎥
⎢                        ⎥
⎣  0     0     0      1.0 ⎦
```

Gazebo Output:

```
EE position
x: -0.27
y: 0.0
z: 1.73
```



Inverse kinematics validation

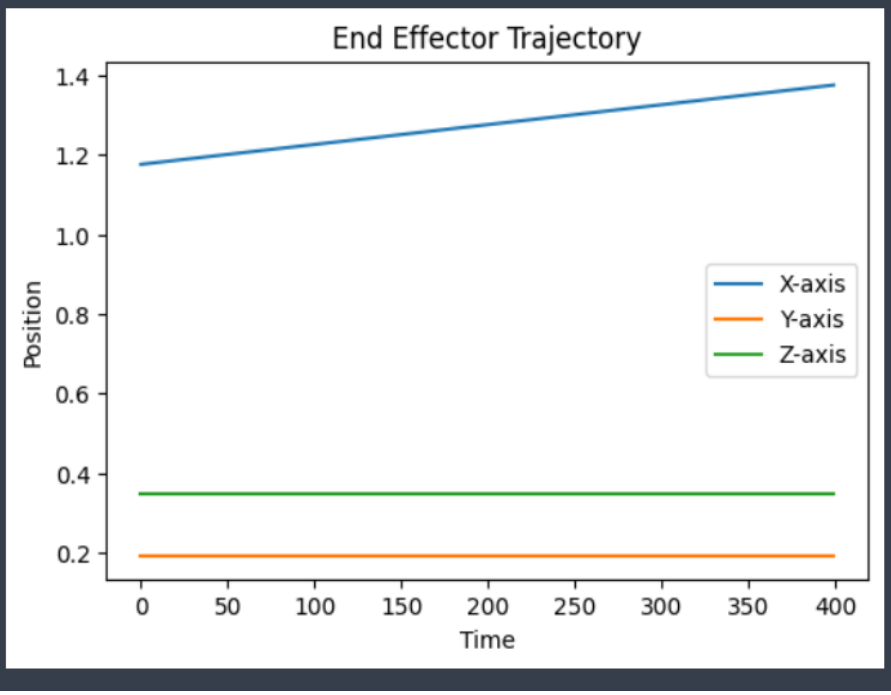
We validated the inverse kinematics in ROS. First, we obtain the end effector pose with respect to the base link by looking up the tf2 transform. Then our inverse kinematics solver computes a straight-line trajectory for our end effector in the positive x-direction for 0.2m. The explanation for the velocity component is given in the section – inverse kinematics.

Our inverse kinematics solver extracts the initial joint values of the arm by subscribing to the Joint States topic and reading the latest joint states. Then it computes the required joint angles for a straight-line trajectory and simultaneously publishes these updated joint angles in the ROS network. Our solver keeps computing the trajectory till the desired distance is reached.

Once the trajectory finishes, we then obtain the end effector pose with respect to the base link by looking up the tf2 transform. We validated the inverse kinematics by looking at the difference in the initial position of the end effector and final position of the end effector.

A straight-line motion of the end-effector (in positive x-axis) was used as trajectory for the purpose of inverse kinematics validation.

```
#Inverse Kinematics
robot = RobotManipulator()
x_traj, y_traj, z_traj = robot.move_robot(0.2, 'x_axis', sp.Matrix([[0],
                                [0.56], [-1.49], [0.5], [4.68], [0.03]]))
```



The robot_move method that is used as a script for implementing inverse kinematics takes three arguments. The distance, axis of motion and current joint angles. It can be seen in the figure given above that we wanted to move the end-effector in positive x-axis by 0.2m. The plot shows that the end-effector has moved 0.2m and there is no motion in y and z axis. This validated the inverse kinematics implemented in this model.

The logic used for implementing inverse kinematics is given below in pseudo code format. The velocity is taken as a constant (0.01m/s). All end effector movements would be with a velocity of 0.01m/s here. The distance is input to this script. In our package, we have kept the distance to be moved for a particular key constant. The time taken for movement can be found easily by dividing distance by velocity. This time is used to run a loop and the using numerical integration the joint configurations are updated for a small value of time (delta_t).

Pseudo code

```
linear_velocity = 0.01
end_time = distance/ linear_velocity
time=0
  while time < end_time:
        J_var = self.J.subs(q_current)
        J_inv = J_var.inv()
        q_dot = (J_inv) * end_effector_velocity.subs(t, time).evalf()
        q_current = q_current + q_dot.evalf() * delta_t
        time = time + delta_t
        position = self.T_final.subs(q_current)  # for the purpose of evaluation.
```

Gazebo Validation

Start Position

```
EE position
x: 1.17
y: 0.19
z: 0.35
```

End Position

```
EE position
x: 1.37
y: 0.19
z: 0.35
```

Video link of validation in Gazebo -> https://youtu.be/k8ei-SBzB-c

Assumptions

The following are the general assumptions made.

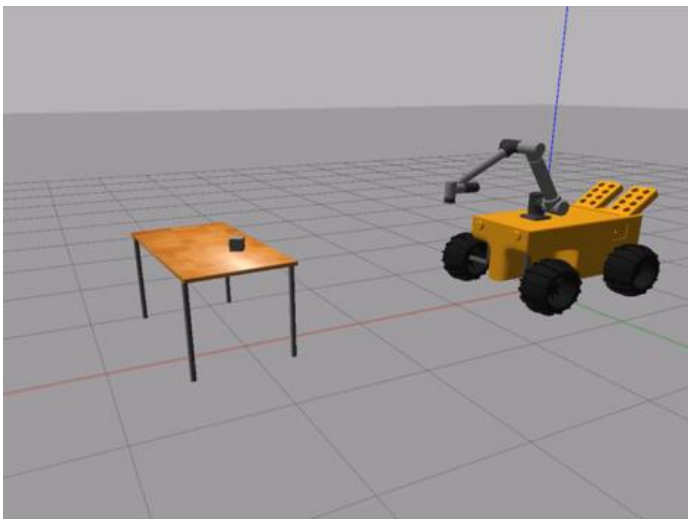
1. All links are rigid with no deformation.
2. The frictional loss during joint rotations is negligible.

Control method

The control method we used was open-loop control. We chose this method since our application was designed such that the robot was to be tele-operated by a human. Hence, this control method was perfect for our application since this control method gives complete control of the robot's movements to the user.

Gazebo and Rviz visualization

In the below scene, the robot is tele-operated using the keys described in the table. The scene consists of our robot, a table, and a cube box on top of it. Our robot is attached with a vacuum gripper at the end-effector. We can move the robot and the end-effector using the keys. The goal of the application is for the user to pick-up the cube by tele-operating the robot and place it wherever they desire. Steps to run the code can be found in the README file submitted with this document.



Key	Command
W	Move Rover Forward
A	Steer Rover Left
S	Move Rover Reverse
D	Steer Rover Right
Q	Stop Robot
I	Move Arm Forward
J	Move Arm Left
K	Move Arm Backward
L	Move Arm Right
M	Move Arm Upwards
N	Move Arm Downward
H	Set Arm in Home pose
R	Set Arm in READY pose
C	Vacuum Gripper ON
V	Vacuum Gripper OFF
ESC	Quit Teleop

Video link https://www.youtube.com/watch?v=8nf7IrT_3io

Note: In our tele-op node, we are controlling the end effector position of the arm using the keys. **In the background inverse kinematics is used to compute the required joint angles to move the arm in the required trajectory**. We are NOT manipulating the arm's joints directly. Inverse Kinematics is being implemented here.

Problems faced

One problem faced was that a couple of the joints were moving in different directions in our URDF as compared to what we defined in our SolidWorks model. We rectified it by changing the axis of rotation of the joints in our URDF to match with the one we desired.

Another problem we faced was that in Gazebo our joints would fly away/break with time since we had only defined position controllers in our URDF for those joints. We rectified it by adding velocity controllers in the URDF for our joints and then the joints did not break.

Lesson learned

We learnt the practical implementation of forward and inverse kinematics in this project. We learnt the end-to-end development and usage of a robot, specifically an arm manipulator. Starting with the 3D modeling in SolidWorks, setting up the DH table and constructing the inverse kinematics solver. Then setting up controllers in ROS and simulating the entire system. We are confident now that we can model and use our custom robots with ease.

Conclusion

This project attains implementation of forward and inverse kinematics for a tele-operable mobile rover, seamlessly integrating a versatile manipulator arm, and validating its capabilities through Sympy and Gazebo simulation. The successful validation of both forward and inverse kinematics within ROS, utilizing open-loop control, endows users with precise control over the end-effector for nuanced object manipulation. The project's resilience is evident in overcoming challenges like joint misalignment and Gazebo instability.

Future work

1. Adding path planning/motion planning and perception capabilities.
2. Designing and adding other end effectors and demonstrating the end-effector changing capacity of the manipulator.

References

1. https://www.universal-robots.com/products/ur10-robot/

2. Spong, M. W., Hutchinson, S., & Vidyasagar, M. (n.d.). *Robot Modeling and Control*. John Wiley & Sons.

3. Jazar, R.N. (2022). Forward Kinematics. In: Theory of Applied Robotics. Springer, Cham. https://doi.org/10.1007/978-3-030-93220-6_5

4. Aristidou, Andreas & Lasenby, Joan. (2009). Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver.

5. https://docs.ros.org/en/galactic/Tutorials.html

6. https://enpm-662introduction-to-robot-modelling.readthedocs.io/en/latest/