ENPM 662 - Fall 2022
Introduction to Robot Modeling
Project-02: Report

Robot Name: CR-7

(Cleaning Robot -7)

DIFFERENTIAL DRIVE MOBILE ROBOT WITH A ROBOTIC ARM

Made by ~ Rishikesh Avinash Jadhav (rjadhav1)
UID-119256534
Nishant Awdeshkumar Pandey (npandey2)
UID-119247556

Table of Contents

S.N.	Topics	S.N.	Topics
1.	Introduction	9.	Workspace Study
2.	Motivation	10.	Assumptions
3.	Process	11.	Control Method
4.	Robot Description	12.	Gazebo and RViz Visualization
5.	Forward KInematics	13.	Problems Faced
5.1	D-H Parameters	14.	Lessons Learned
6.	Inverse Kinematics	15.	Conclusions
7.	Forward KInematics Validation	16.	Future Work
8.	Inverse Kinematics Validation	17.	References

1. Introduction

Modeling a mobile robot (differential drive) with a robotic arm mounted on top of it. The simple purpose for making an assembly like this is to clean the entire space without the worry of larger objects in the path of the robot.
The idea of making this bot is inspired by the Roomba which is a series of autonomous robotic vacuum cleaners sold by the company iRobot.

The entire project went through a series of analyses and processes, namely, analysing the robot's kinematics (Inverse and forward) and validating the same, spawning the robot on Gazebo(in the desired environment) and RViz.
The idea was to use the derived equations to use the robot for the desired application (moving to the goal location, picking and placing the object from one place to another).

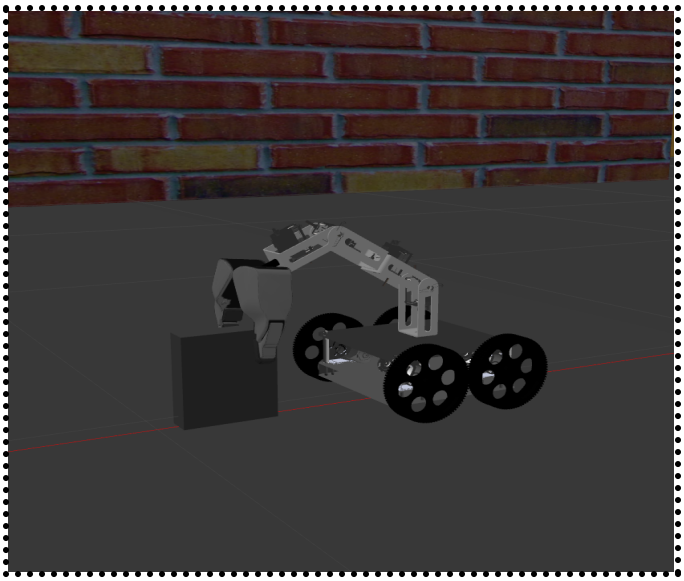


Fig1: Picture of CR-7 taken in Gazebo.

2. Motivation

Fig2: Robot used as design reference (UR5 + Ridgeback).

The idea of a room-cleaning robot has been encouraged by multiple companies and this technology has seen many advancements. We wanted to take it further by providing the moving robot with a robotic arm. Our version of the arm will help the robot to navigate across a lot of clutter as it will clear the bigger objects in the room

No matter how many links are attached to the robotic arm the workspace of any arm still is not enough for wide area applications. As the D.O.F of the arm increases the complexity of calculation also increases. To satisfy a variety of demands, making a more well-rounded robot becomes the need of the hour.

Applications like cleaning a room with scattered objects can be easily executed because of less weight and size of the objects to pick and place. The small size of the bot facilitates operations in crowded spaces.

3. Process explanation with a flow chart

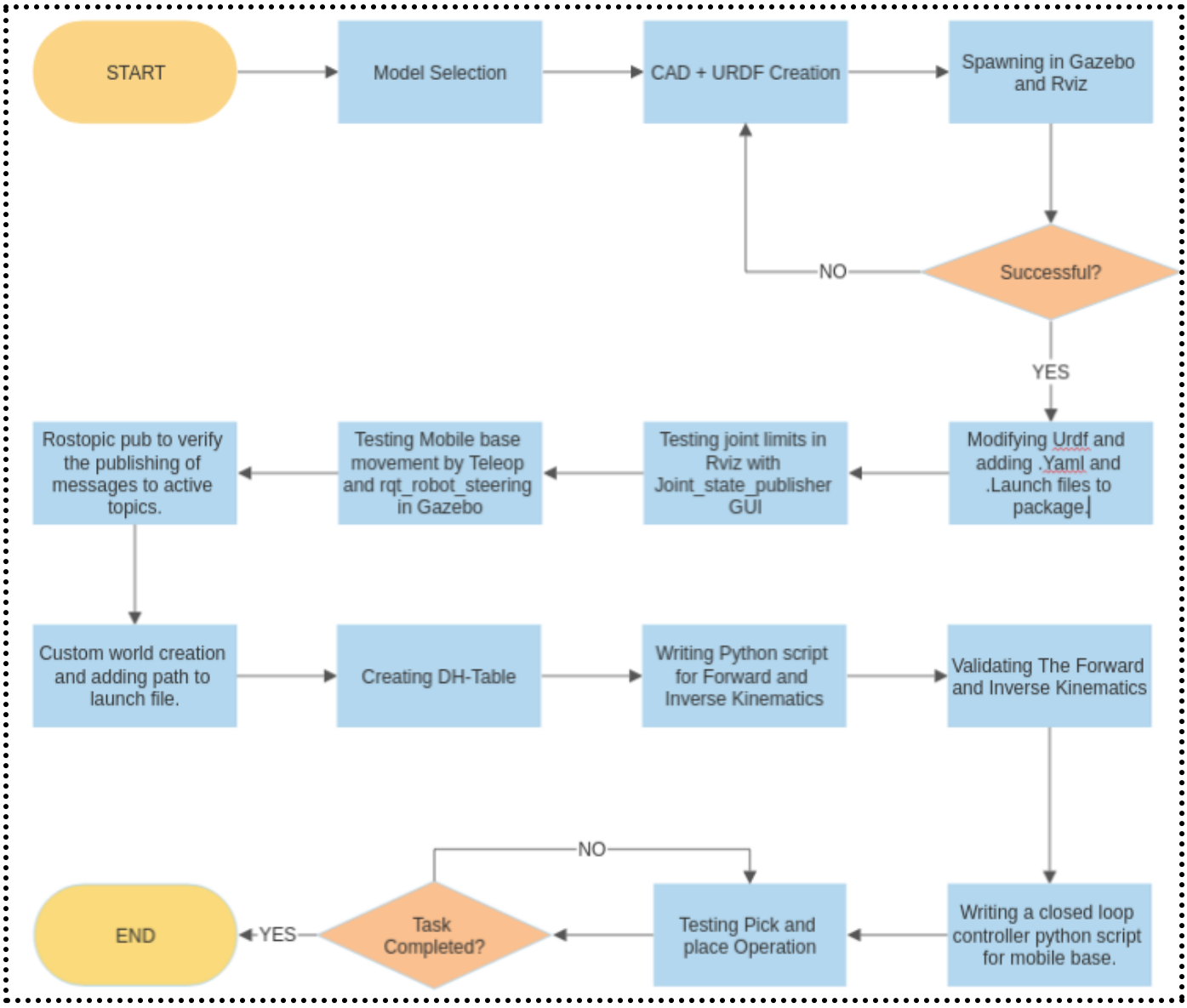


Fig 3: Flow chart of all the steps involved.

4. Robot description

Fig 4: Robot as spawned in gazebo

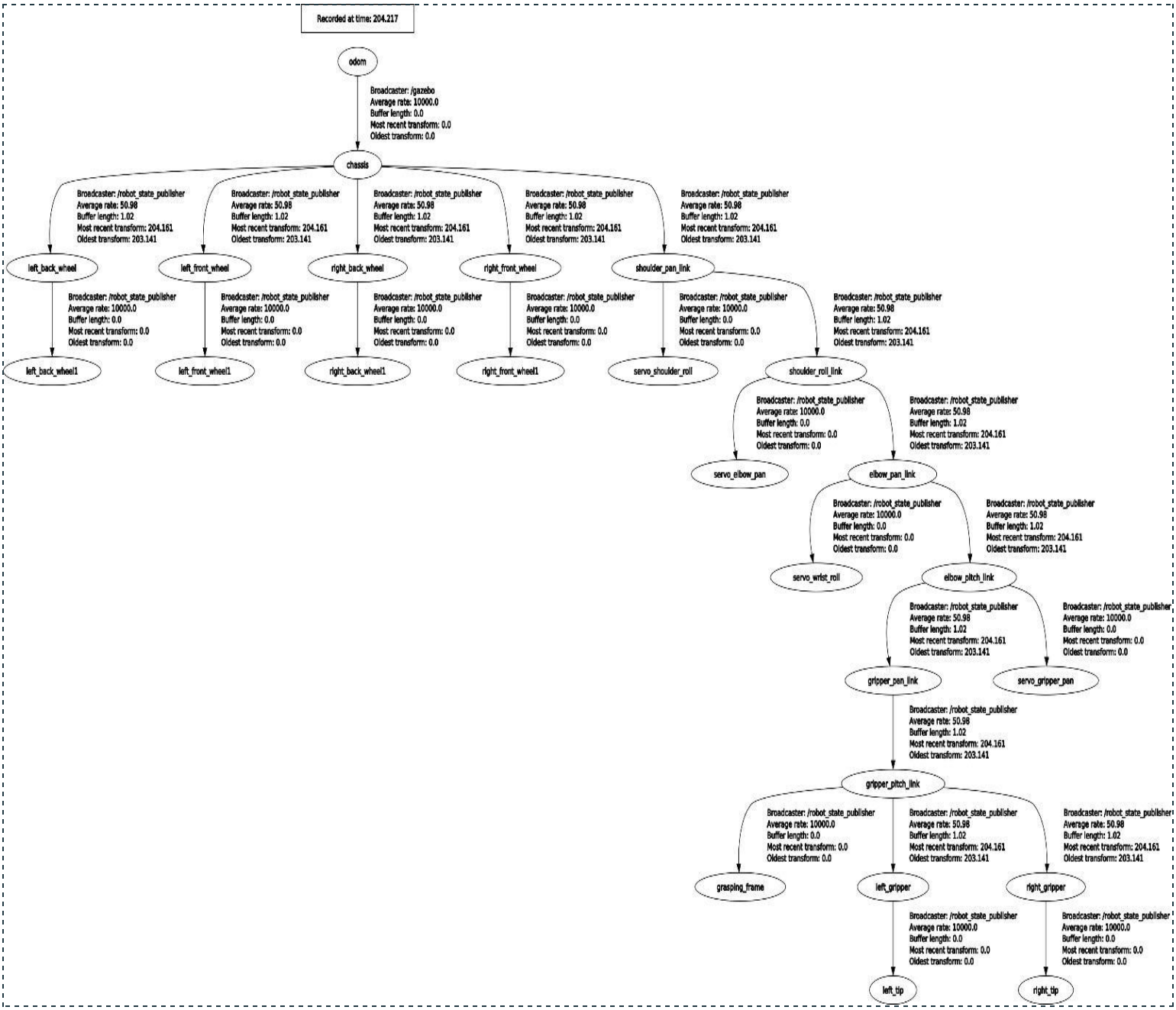
Robot Type: MOBILE MANIPULATOR

Total DOF (9) = Robotic arm DoF (6)+ Mobile robot DoF (3)

DIMENSIONS:

- Chassis property(units): Length: 0.19, Width: 0.19, Height: 0.070
- Wheel property: Radius(units):0.055, width: 0.022, Mass=0.2
- Shoulder roll link(units): Length: 0.09, Width: 0.047, Height: 0.035
- Shoulder pan link(units): Length: 0.09, Width: 0.047, Height: 0.035
- Elbow pan Link(units): Length: 0.09, Width: 0.047, Height: 0.035
- Elbow pitch Link(units): Length: 0.09, Width: 0.047, Height: 0.035
- Gripper(units): Length: 0.05, Width: 0.010
- Gripper type: 2 Finger gripper

4.1 TF TREE



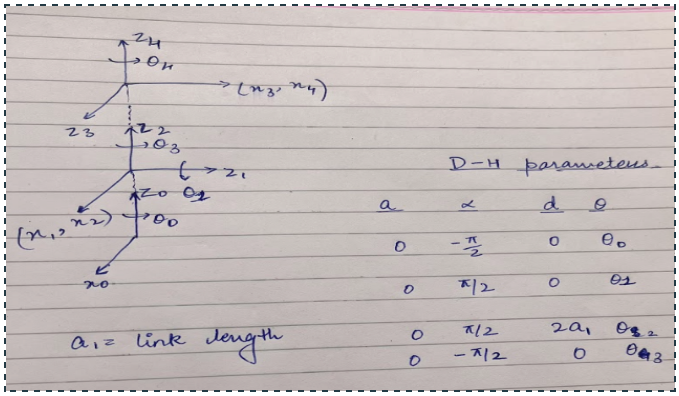
The above Tf tree shows how all the parts and their parent and child link. This gives the basic idea of the structure of the bot.

5. Forward kinematics

Steps for the execution of forward kinematics:

- DH- parameters are obtained after analyzing the various frames.
- Calculation of Transformation matrices for all joints w.r.t. base frame using the DH- parameters
- The angles are all assumed to be zero and then the verification of the final frame (elbow_pitch joint) positions is done in RViz.
- Final Transformation matrix is obtained.

5.1 D-H Parameters



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.

Fig 5: DH table and its parameter's definitions (Spong convention).

The above table was obtained by using origin shifting method a1 here is 0.08 units. All the angles are described in the above pictorial representation. This analysis facilitates in understanding the manner in which the joints rotate and will be used for further calculations and validations.

Following this analysis of the DH- parameters the transformation matrices can be obtained and multiplied to obtain the final homogeneous transformation matrix. This matrix gives the position and orientation of the end-effector with respect to the base link (shoulder pan joint).

The obtained transformation matrices are as follows:

$$H={}^{1}_{0}T*{}^{2}_{1}T*{}^{3}_{2}T*{}^{4}_{3}T$$

Homogeneous Transformation Matrix between Links 'i' and 'i-1' :

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

Fig 6: General formula for transformation matrix

5.2 Code

```
import rospy
from trajectory_msgs.msg import JointTrajectory
from std_msgs.msg import Header
from trajectory_msgs.msg import JointTrajectoryPoint
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64
from std_msgs.msg import Float64MultiArray
import sys, select, termios, tty
from sympy import *
from sympy.matrices import Matrix
from sympy import pprint, Array
import numpy as np
from math import pi
from sympy import diff,cos,sin,Symbol
import math
#Function to create a Transformation matrix using DH parameters - α, θ, d, a
def Transformation(alpha, theta, d, a):
    d = float(d)
    a = float(a)
    T = Matrix([[cos(theta), -sin(theta)*cos(alpha), sin(theta)*sin(alpha), a*cos(theta)],
    [sin(theta), cos(theta)*cos(alpha), -cos(theta)*sin(alpha), a*sin(theta)],
    [0, sin(alpha), cos(alpha), d],
    [0, 0, 0, 1]])
    return T
x = []
y = []
z = []
#link lengths

a1=0.08
#joint variables
q1_1 = Symbol('theta1')
q2_1 = Symbol('theta2')
q3_1 = Symbol('theta4')
q4_1 = Symbol('theta5')



T1 = Transformation(-pi/2,q1_1,a1,0)          #Creating homogeneous Transformation matrix for 0-1
T2 = Transformation(pi/2,q2_1,0,0)            #Creating homogeneous Transformation matrix for 1-2
T3 = Transformation(pi/2,q3_1,2*a1,0)         #Creating homogeneous Transformation matrix for 2-3
T4 = Transformation(-pi/2,q4_1,0,0)         #Creating homogeneous Transformation matrix for 3-4


H1 = T1
H2 = H1*T2
H3 = H2*T3
H4=  H3*T4
```

Fig 7: Code for obtaining the transformation matrices for the bot.

The above code illustrates the manner in which the desired matrices were obtained.

6. Inverse Kinematics

Steps for the execution of inverse kinematics:

- After obtaining the homogeneous transformation matrices the Jacobian (J) was calculated.
- For calculating J the second method is applied. In this method, the fourth column of the transformation matrix is differentiated with respect to joint angles.
- The obtained matrices then become the upper half of every column in the jacobian matrix and the lower half is the third column of each transformation matrix. The obtained Jacobian in this case is a 6 X 4 matrix.
- Where the upper half gives the linear velocity component for each link and the lower half gives the angular velocity component for each link.
- Both Jv and Jw vary based on the type of joint, i.e they are different for revolute and prismatic joints.
- The jacobian is then used to get the joint angles from the joint velocities. By using the below formula:

$$q = J^{-1} * q\_dot$$

Where q is the joint angle and q_dot is the joint velocity. J^{-1} in this case is the pseudo-inverse of the Jacobian. J^{-1} is a 4 x 6 matrix.

- Numerical integration is used to then obtain the joint angles and further the end effector trajectory is obtained.

6.1 Code

```
H1 = T1
H2 = H1*T2
H3 = H2*T3
H4 = H3*T4
Z0 = Matrix([[0], [0], [1]])                # Z component of base frame
Z1 = Matrix([[H1[2]], [H1[6]], [H1[10]]])   #Taking Z component of H1
Z2 = Matrix([[H2[2]], [H2[6]], [H2[10]]])   #Taking Z component of H2
Z3 = Matrix([[H3[2]], [H3[6]], [H3[10]]])   #Taking Z component of H3
Z4 = Matrix([[H4[2]], [H4[6]], [H4[10]]])   #Taking Z component of H4
Xp = Matrix([[H4[3]], [H4[7]], [H4[11]]])   #Translation component of final transformation matrix - H4
C1 = diff(Xp, q1_1) #Differentitaing Xp w.r.t theta 2
C2 = diff(Xp, q2_1) #Differentitaing Xp w.r.t theta 3
C3 = diff(Xp, q3_1) #Differentitaing Xp w.r.t theta 4
C4=  diff(Xp, q4_1)
J1 = Matrix([[C1], [Z1]]) #Computing Jacobian 1st column
J2 = Matrix([[C2], [Z2]]) #Computing Jacobian 2nd column
J3 = Matrix([[C3], [Z3]])
J4 = Matrix([[C4], [Z4]])
J = Matrix([[J1, J2, J3, J4]])
```

Fig 8: Code for obtaining the Jacobian matrix.

The above code illustrates the manner in which the Jacobian was obtained during the trajectory plotting and robot arm manipulation.

7. Forward Kinematics Validation

Forward kinematics is validated in the following manner:

- Joint angles are fixed.
- The final transformation matrix is printed using the code in fig 7.
- The robot is spawned in RViz and then the robot arm's joint angles are set to the value same as that entered in the code.
- Position of the end effector in RViz and the homogenous matrix are compared and if they are the same then the validation is considered successful.

These steps are executed in the video and the forward kinematics of the arm are then verified.

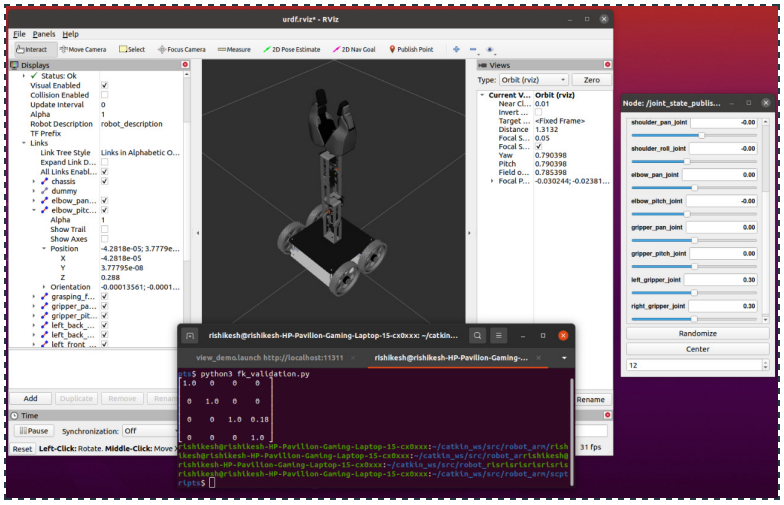


Fig 9: Screenshot of validation

8. Inverse Kinematics Validation

Inverse kinematics is validated in the following manner:

- The trajectory of the bot is decided. In this case, it is selected to go back and forth about the shoulder pitch joint.
- The trajectory of the end effector is plotted.

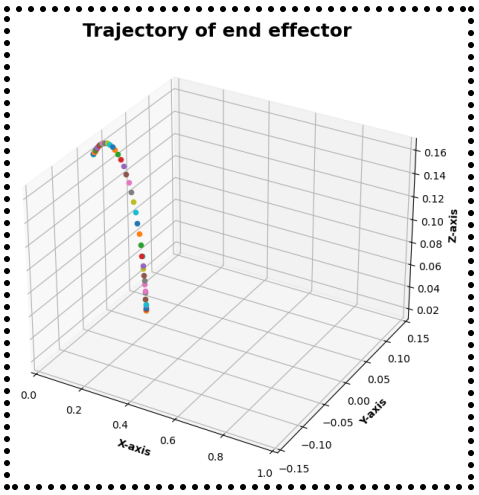
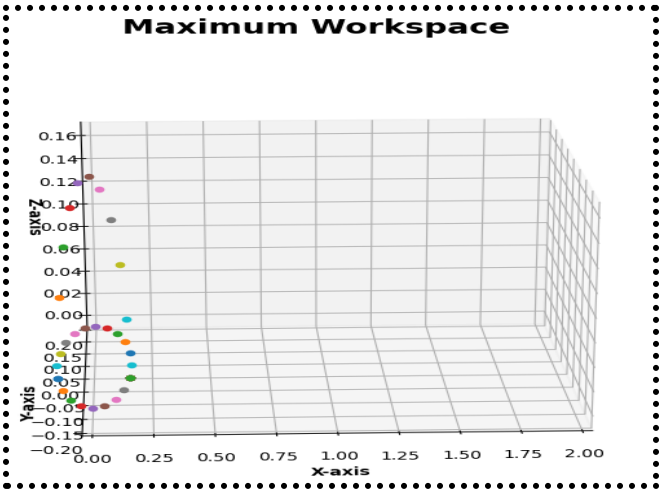


Fig 10: Trajectory obtained after validation

- The robot is again spawned in RViz to verify if the robot works in the same manner as it is designed for.
- The working of the code is similar to that illustrated in section 6.
- This video shows the validation of inverse kinematics.

9. Workspace Study

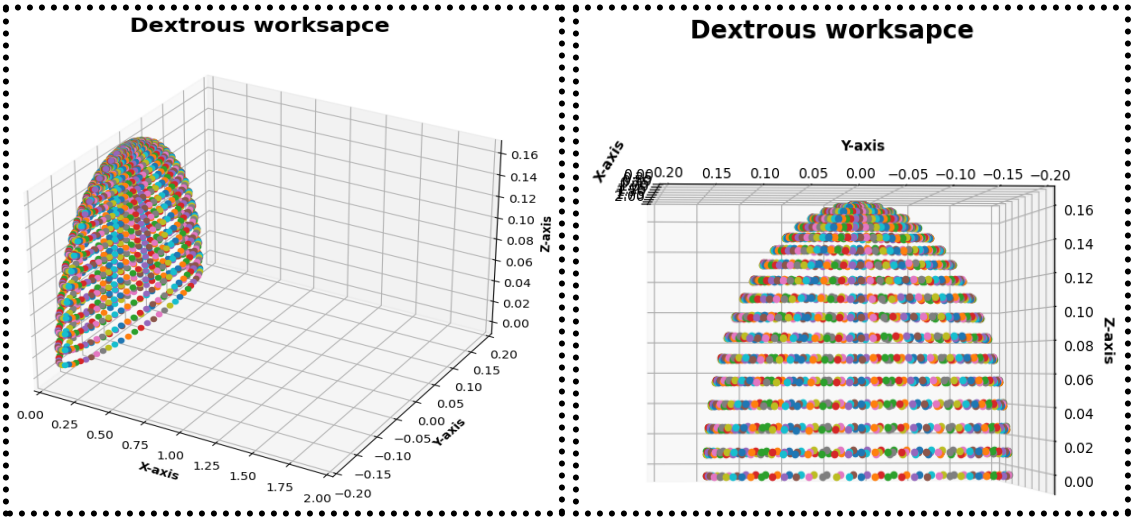
Maximum workspace of the arm is as follows:



The above figure is obtained by selecting the angle limits as follows:

- Theta 1= 0:2*pi
- Theta 2= -pi/2:pi/2
- Theta 3=0:pi/2
- Theta 4= -pi/2:pi/2

Whereas the dexterous workspace of the arm will be that of an ellipsoid as shown in the figure below:



10. Assumptions

Robot Model design assumptions:

- All the joints and objects are considered to be rigid.
- The friction and other external disturbances are not taken into account.
- Robot self-collision is considered along with the external obstacles and is taken into account for this scope of the project.
- The path of the arm or the robot is just one solution among all the other solutions it can have, this may or may not be the optimal solution.

11. Control method

The moving bot contains a differential drive and is controlled using a closed-loop mechanism the following method is used to illustrate its working:

- The goal is known beforehand and is input in the code.
- The starting position of the bot is also fixed.
- Then the control inputs are given to the wheels and the current status of the bot is known using the Gazebo odometer.
- The current distance of the bot is obtained and subtracted from the goal and then it is controlled accordingly by giving linear and angular velocity accordingly.
- The bot stops 0.25 unit distance before the box to pick up.
- The bot can also go back to its original or home position.

12. Gazebo and RViz visualization

- The video of the Pick and Place operation showcases the simulation in Gazebo and RViz.

- The validation of the robot functionalities is given in this video folder. This includes videos which show Teleop, rqt_steering, arms control and mobile base control.

13. Problems Faced

- Correctly spawning the robot in the Gazebo and Rviz which was spawning with dislocated parts.
- Finding the DH parameters as the robot used by us is not universal.
- Multiple errors were solved which we faced while creating the launch and YAML files.
- Writing the python script for forward, inverse kinematics and a closed loop controller for the differential drive robot.
- Correctly simulate the application.
- Sensors could not be added instead we had to use a Gazebo odometer to write the closed-loop controller for the mobile robot.
- In some cases specifically while writing the code we entered the wrong DH parameter values and after code completion this minuscule error caused us to spend more time than required on finding the root of this problem. (which we found and corrected).

14. Lessons Learned

➔ Time management is key.
➔ Solidwork assembly to URDF file conversion.
➔ Learned adding plugins and parts by creating a Xacro file.
➔ Understood the structure and wrote the controllers at appropriate joints of the robot(.YAML).
➔ Gained an understanding of how the launch file was supposed to be created (reference taken from last project as well as online

resources) and made a single launch file along with an inclusion of a custom world in it.

➔ Wobbling of a few parts while moving in the simulation was fixed in the end by adjusting PID values.

➔ Learned writing python scripts for our project with reference to previous homework and published the topics as per the task requirements in the code.

➔ Kinematic Analysis has helped us validate the desired equations and numerical values which can be further implemented for practical use.

➔ Understood the Complexity of the Pick and place functionality of manipulators.

➔ Biggest takeaway was to test output at every minor milestone of the project.

15. Conclusion

- The mobile manipulator URDF file was properly spawned in Rviz and the gazebo.
- Wrote a joint state publisher GUI and checked the working of joints successfully.
- Validation for the movement of the robot was done through simulation using ROS in a Gazebo environment.
- Created a rqt_robot_steering, Teleop to make sure the differential drive robot was able to traverse the world properly.
- Wrote a closed-loop controller script and ensured that the mobile robot stopped at a certain distance before the goal position to avoid a collision.
- Created a custom test world for the robot environment and spawned the robot in it successfully.
- Wrote the Python scripts for finding the Homogeneous

transformation matrix via the DH-table and then found the Jacobian. (HW-4 code was referred to for this purpose)
- Verification of forward and inverse kinematics was conducted and verified.
- After addressing and debugging multiple issues in the final stages, the application simulation was executed successfully.

16. Future Work

- As we know our robot was built to navigate cluttered environments for cleaning purposes. It has room for improvements to be made in the future.
- There can be multiple advancements which can further improve the efficiency and make the robot multi-functional, which can address multiple monotonous cleaning tasks without much need of supervision.
- By adding a vacuum feature similar to the Roomba for dust collection, integrating contact and vision sensors as well as a few more changes to the design and/or feature additions will help the robot navigate tight spaces with ease.
- This gives an opportunity to move towards a fully functional automated cleaning mobile manipulator.
- The application for this robot can be used for small spaces such as households, and hospital wards as well as scaled-up versions of the mobile manipulator can be used in airports, warehouses and huge facilities which require a lot of effort, and supervision to clean.

17. References

1. Robot Modeling and Control, Mark W. Spong, Seth Hutchinson, and M. Vidyasagar
2. HW3 or HW4 of ENPM-662
3. Gazebo.sim
4. Ros.org
5. https://moveit.ros.org/
6. https://www.solidworks.com/
7. youtube.com
8. grabcad.com