DESIGN AND DEVELOPMENT OF FOOD PREPARING ROBOT

Final-Project Report
for
Introduction to Robot Modeling (ENPM662)

Authors
Pratik Bhujbal UID: 117555295
Aditi Ramadwar UID: 117093575

Date
12/10/2021

Abstract

The traditional pizza-making process involves Making the Pizza Dough, Preparing the Pizza dough for applying sauce and cheese, Applying the Toppings over, and Baking it.

Our Robot system will streamline the tedious and repetitive process of pouring different kinds of toppings and cheese on different pizzas on a conveyor oven which will then be ready to serve. The robot is a 12 dof with two arms of 6 dof each and a gripper. Appropriate robot model and specifications are expounded below with proper consideration of the robot's duty. Also explained the plans and methods of implementation and tools required to develop the whole.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

PID - Proportional-Integral-Derivative
DoF - Degree of Freedom
mm - Millimeter
m - Meter
kg - Kilogram

1. INTRODUCTION

Currently, the food industry, especially in the United States, is rapidly changing. As the population increases, the demand for the quantity of food generated increases exponentially as well. On top of that, the consumers demand high-quality food which is more sustainable. By keeping all the demands of the consumers as well as the rise in the number of consumers in mind, robots can be used effectively to resolve this problem of demand over supply and also produce better quality products than those produced by humans currently. [1]

The robot's ability to be absolutely precise with the task is useful when it comes to delivering high-quality food to consumers. Robots can be made adaptable to the type of ingredients to be used to various different food products. They provide higher precision results than those by human employees. Robots can also be made consistent and made to work for long hours without the need to pay them the minimum wage. This makes robots perfect to work in the food industry where repetitive tasks need to be carried out consistently.

Robotics and automation are a key part of the solution. Compared to other industries, the food manufacturing sector has been relatively slow to adopt robotics. But, over the last couple of years robotics has started to make its way into almost every link in the food supply chain, from the field to the kitchen[1]. There is another sector in the food industry where robots can take over next and that is the fast-food industry. The fast-food industry needs consistent results with the utmost precision to keep up with the brand's reputation. Hence industrial robots are the perfect fit for the fast-food industry and we have developed a food preparing robot being used to cook a pizza. A huge reason to choose this application is to show that the robot can work in extreme conditions like high temperatures near a pizza oven.

2. APPLICATION

In our project, we have made our robot to cook a pizza by adding different types of toppings on the pizza which will be ready for baking. While we are using our robot for making a pizza, the robot can also be used for other applications in the food industry such as delivering the pizza, as well as developing processed food in the food manufacturing sector.

The robot in this project is made to work in a restaurant setting where it prepares pizzas for the customers by adding toppings onto the pizza base. We have given two different kinds of toppings to put on the pizza base for the robot to add.

3. ROBOT DoF and DIMENSIONS

The designed robot is a non-mobile industrial robot manipulator. The robot is a 12 dof with two arms of 6 dof each and a gripper. An isometric view of the designed robot is shown in Figure 3.1.

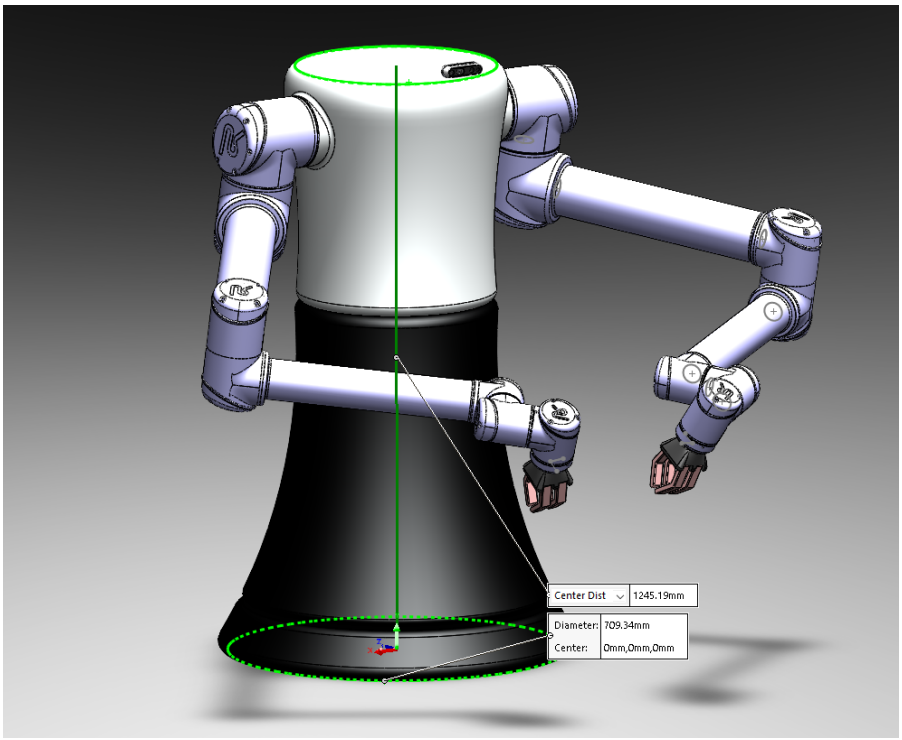


Fig 3.1 Body Dimension

Figure 3.1 also shows the dimension of the body where the height of the robot is 1245.19 mm from the base of the robot with a diameter of 709.34 mm.

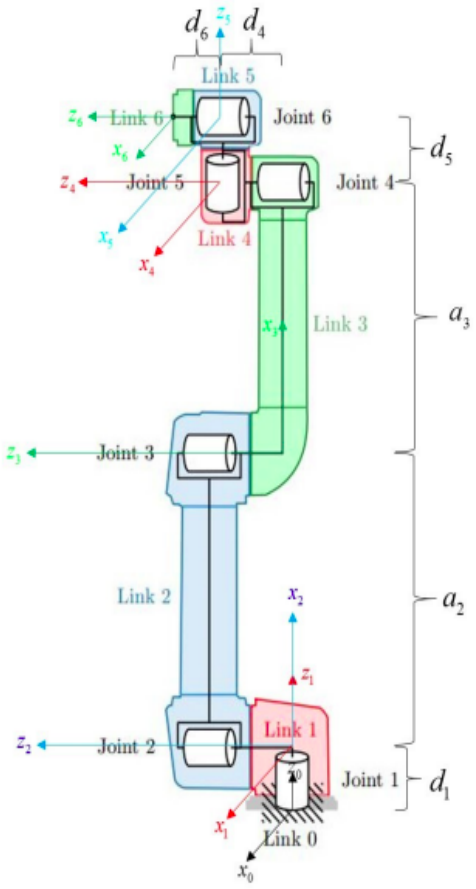


Fig 3.2 UR10 Dimensions

According to the annotations in Figure 3.2, Table 3.1 shows the dimensions of each link and the mass of each link.

di	Length (m)
d1	0.1273
d4	0.163941
d5	0.1157
d6	0.0922
a2	-0.612
a3	-0.5723

Link	Mass(kg)
Link 1	7.778
Link 2	12.93
Link 3	3.87
Link 4	1.96
Link 5	1.96
Link 6	0.202

Table 3.1 Link Dimension and masses

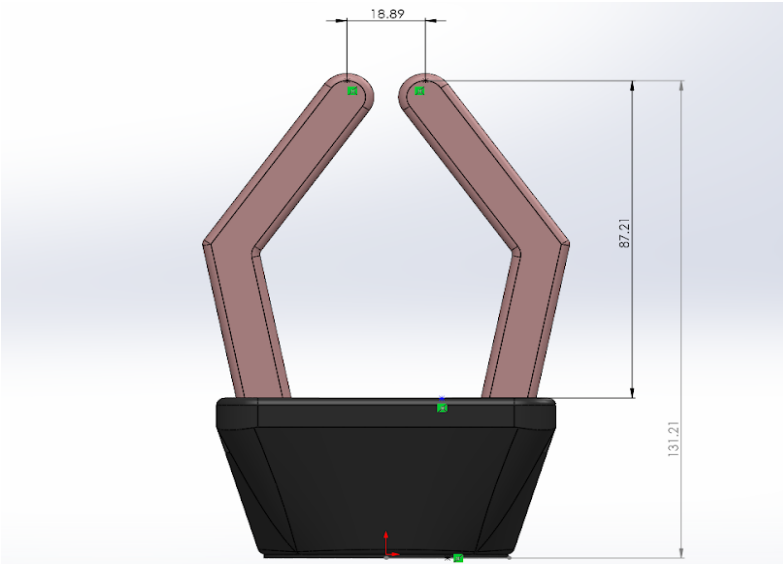


Fig 3.3 Gripper Dimensions

4. CAD MODELS

The CAD Model for UR10 was taken from GrabCad - https://grabcad.com/library/ur10-with-movement-1 and exported to URDF using SolidWorks by assigning frames and joint axis as shown in Figure 4.2 (a)
For the rest robot assembly parts modeling were done in Solidworks and exported to URDF. Figure 4.1 shows the robot assembly.

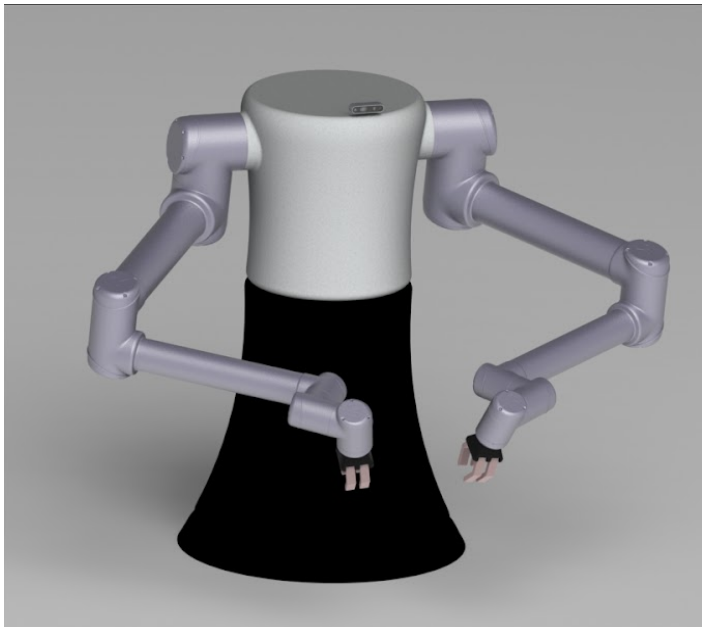
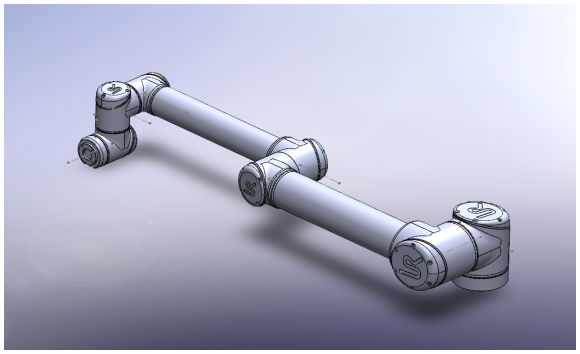
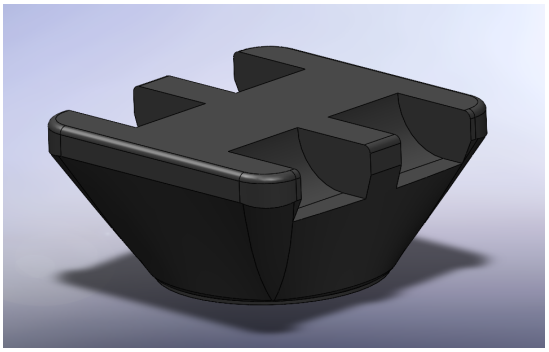


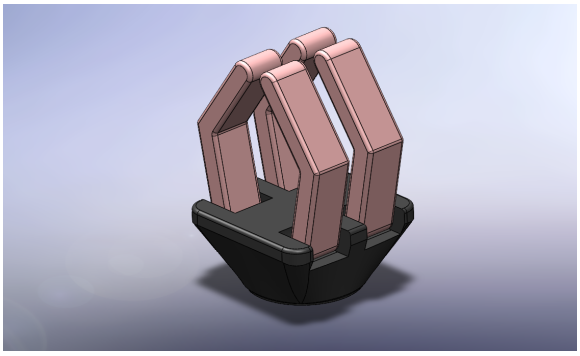
Fig. 4.1 Robot Assembly



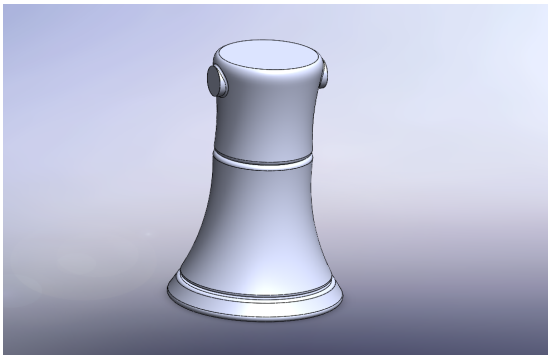
a) Ur10 Arm



b) Gripper Base



c) Gripper Assembly



d) Robot's Body

Fig. 4.2 Individual parts

5. Denavit-Hartenberg (DH) PARAMETERs

For accurate calculation and derivation of forward and inverse kinematics, we need to describe the motion of all joints between all the links present on the arm. At first, we analyzed the robot arm in its home position and assumed that all the joint angles are at zero radians. We drew the axis of each joint by referring to [4].

Since all our joints are revolute joints, the z-axis is made to lie on the axis of rotation for each joint. The x-axis is made to lie along the "common normal", which is the shortest orthogonal line between the previous z-axis and the current z-axis. The y-axis is then naturally selected to satisfy the right-hand rule.[4]

Figure(1) is a representation of the coordinate frames we have selected for each joint in our arm. Since the arm has six degrees of freedom, we need to define 6 axes and derive DH parameters for all six joints. To derive the four parameters

of the DH parameters, we have taken each joint parameter with reference to the previous joint. They are calculated in reference to the "common normal" drawn while figuring out the axis of each joint.
Table 1 shows the DH parameters derived for our arm. Since both the arms in the robot are identical, the same DH parameters are used for the other arm as well. [2]

Joints	**Theta**	**d**	**a**	**alpha**
Shoulder Yaw	Theta1	0.1273	0	90
Shoulder Pitch	Theta2	0	-0.612	0
Elbow	Theta3	0	-0.5723	0
Wrist 1	Theta4	0.163941	0	90
Wrist 2	Theta5	0.1157	0	-90
Wrist 3	Theta6	0.0922	0	0

Table 5.1 D-H table

The forward kinematics of each arm is being calculated separately to decrease complications in integration.

6. FORWARD KINEMATICS

For the calculated DH parameters of each arm, we calculated the forward kinematics of each joint by keeping the robot at it's base configuration. [3]
The base configuration of the arm is assumed to be when all the joint theta angles are such that the jacobian would run into singularity issues. So theta1 and theta 3 are kept as -pi while all others are zero.

T06 =
r11 = ct6*(ct5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2)) + st1*st5) + st6*(ct4*(-ct1*ct2*st3 - ct1*ct3*st2) - st4*(ct1*ct2*ct3 - ct1*st2*st3))

r12 = ct6*(ct4*(-ct1*ct2*st3 - ct1*ct3*st2) - st4*(ct1*ct2*ct3 - ct1*st2*st3)) - st6*(ct5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2)) + st1*st5)

r13 = ct5*st1 - st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))

r21 = ct6*(-ct1*st5 + ct5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2))) + st6*(ct4*(-ct2*st1*st3 - ct3*st1*st2) - st4*(ct2*ct3*st1 - st1*st2*st3))

r22= ct6*(ct4*(-ct2*st1*st3 - ct3*st1*st2) - st4*(ct2*ct3*st1 - st1*st2*st3)) - st6*(-ct1*st5 + ct5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2)))

r23= -ct1*ct5 - st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2))

r31= ct5*ct6*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)) + st6*(ct4*(ct2*ct3 - st2*st3) - st4*(ct2*st3 + ct3*st2))

r32= -ct5*st6*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)) + ct6*(ct4*(ct2*ct3 - st2*st3) - st4*(ct2*st3 + ct3*st2))

r33= -st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3))

px = -0.5723*ct1*ct2*ct3 - 0.612*ct1*ct2 + 0.5723*ct1*st2*st3 - 0.1157*ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + 0.0922*ct5*st1 + 0.163941*st1 + 0.1157*st4*(ct1*ct2*ct3 - ct1*st2*st3) - 0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))

py= -0.0922*ct1*ct5 - 0.163941*ct1 - 0.5723*ct2*ct3*st1 - 0.612*ct2*st1 - 0.1157*ct4*(-ct2*st1*st3 - ct3*st1*st2) + 0.5723*st1*st2*st3 + 0.1157*st4*(ct2*ct3*st1 - st1*st2*st3) - 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2))

pz= -0.5723*ct2*st3 - 0.5723*ct3*st2 - 0.1157*ct4*(ct2*ct3 - st2*st3) - 0.612*st2 + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)) + 0.1273

7. VALIDATION OF FK

Following are the transformation matrices of each joint configuration with respect to the base link for the initial configuration.

T01

```
-1.0 , 0.0 , -0.0 , 0.0
-0.0 , -0.0 , 1.0 , 0.0
0.0 , 1.0 , 0.0 , 0.13
0.0 , 0.0 , 0.0 , 1.0
```

T02

```
-1.0 , 0.0 , -0.0 , 0.61
-0.0 , -0.0 , 1.0 , 0.0
0.0 , 1.0 , 0.0 , 0.13
0.0 , 0.0 , 0.0 , 1.0
```

T03

```
1.0 , -0.0 , -0.0 , 0.04
0.0 , 0.0 , 1.0 , 0.0
-0.0 , -1.0 , 0.0 , 0.13
0.0 , 0.0 , 0.0 , 1.0
```

T04

```
1.0 , -0.0 , 0.0 , 0.04
0.0 , 1.0 , 0.0 , 0.16
-0.0 , 0.0 , 1.0 , 0.13
0.0 , 0.0 , 0.0 , 1.0
```

T05

```
1.0 , -0.0 , 0.0 , 0.04
0.0 , 1.0 , 0.0 , 0.16
-0.0 , 0.0 , 1.0 , 0.13
0.0 , 0.0 , 0.0 , 1.0
```

T06

```
1.0 , -0.0 , -0.0 , 0.04
0.0 , 0.0 , 1.0 , 0.26
-0.0 , -1.0 , 0.0 , 0.24
0.0 , 0.0 , 0.0 , 1.0
```

The end effector position with respect to the base link is [0.04, 0.26, 0.24]
For the angle configuration: [-pi, 0, -pi, 0, 0, 0]

8. INVERSE KINEMATICS

The steps to calculate the Jacobian matrix for this arm were:

1. Calculate T0i for each joint, so total 6 transformation matrices.
2. Obtain Oi
3. Obtain Zi
4. Calculate Ji using the following formula:

$$J_i = \begin{bmatrix} Z_{i-1} \times (O_n - O_{i-1}) \\ Z_{i-1} \end{bmatrix}$$

5. Integrate the 6 jacobian arrays for individual joints into one matrix.

Our Jacobian matrix is a 6x6 matrix at the home position

```
-0.26 , 0.12 , 0.12 , 0.12 , -0.09 , -0.0
0.04 , 0.0 , -0.0 , 0.0 , -0.0 , -0.0
0.0 , -0.04 , 0.57 , -0.0 , 0.0 , 0.0
0.0 , -0.0 , -0.0 , -0.0 , 0.0 , -0.0
0.0 , 1.0 , 1.0 , 1.0 , 0.0 , 1.0
1.0 , 0.0 , 0.0 , 0.0 , 1.0 , 0.0
```

This is the general jacobian matrix we have derived through SymPy:

*J1 = Matrix([[0.0922*ct1*ct5 + 0.163941*ct1 + 0.5723*ct2*ct3*st1 + 0.612*ct2*st1 + 0.1157*ct4*(-ct2*st1*st3 - ct3*st1*st2) - 0.5723*st1*st2*st3 - 0.1157*st4*(ct2*ct3*st1 - st1*st2*st3) + 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2))], [-0.5723*ct1*ct2*ct3 - 0.612*ct1*ct2 + 0.5723*ct1*st2*st3 - 0.1157*ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + 0.0922*ct5*st1 + 0.163941*st1 + 0.1157*st4*(ct1*ct2*ct3 - ct1*st2*st3) - 0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))], [0]])*

J2 = Matrix([[-ct1(-0.5723*ct2*st3 - 0.5723*ct3*st2 - 0.1157*ct4*(ct2*ct3 - st2*st3) - 0.612*st2 + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)))], [-st1*(-0.5723*ct2*st3 - 0.5723*ct3*st2 - 0.1157*ct4*(ct2*ct3 - st2*st3) - 0.612*st2 + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)))], [ct1*(-0.5723*ct1*ct2*ct3 - 0.612*ct1*ct2 + 0.5723*ct1*st2*st3 - 0.1157*ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + 0.0922*ct5*st1 + 0.163941*st1 + 0.1157*st4*(ct1*ct2*ct3 - ct1*st2*st3) - 0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))) + st1*(-0.0922*ct1*ct5 - 0.163941*ct1 - 0.5723*ct2*ct3*st1 - 0.612*ct2*st1 - 0.1157*ct4*(-ct2*st1*st3 - ct3*st1*st2) + 0.5723*st1*st2*st3 + 0.1157*st4*(ct2*ct3*st1 - st1*st2*st3) - 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2)))]])*

J3 = Matrix([[-ct1(-0.5723*ct2*st3 - 0.5723*ct3*st2 - 0.1157*ct4*(ct2*ct3 - st2*st3) + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)))], [-st1*(-0.5723*ct2*st3 - 0.5723*ct3*st2 - 0.1157*ct4*(ct2*ct3 - st2*st3) + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)))], [ct1*(-0.5723*ct1*ct2*ct3 + 0.5723*ct1*st2*st3 - 0.1157*ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + 0.0922*ct5*st1 + 0.163941*st1 + 0.1157*st4*(ct1*ct2*ct3 - ct1*st2*st3) - 0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))) + st1*(-0.0922*ct1*ct5 - 0.163941*ct1 - 0.5723*ct2*ct3*st1 - 0.1157*ct4*(-ct2*st1*st3 - ct3*st1*st2) + 0.5723*st1*st2*st3 + 0.1157*st4*(ct2*ct3*st1 - st1*st2*st3) - 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2)))]])*

J4 = Matrix([[-ct1(-0.1157*ct4*(ct2*ct3 - st2*st3) + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)))], [-st1*(-0.1157*ct4*(ct2*ct3 - st2*st3) + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)))], [ct1*(-0.1157*ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + 0.0922*ct5*st1 + 0.163941*st1 + 0.1157*st4*(ct1*ct2*ct3 - ct1*st2*st3) - 0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))) + st1*(-0.0922*ct1*ct5 - 0.163941*ct1 - 0.1157*ct4*(-ct2*st1*st3 - ct3*st1*st2) + 0.1157*st4*(ct2*ct3*st1 - st1*st2*st3) - 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2)))]])*

J5 = Matrix([[-(-ct4(ct2*ct3 - st2*st3) + st4*(ct2*st3 + ct3*st2))*(-0.0922*ct1*ct5 - 0.1157*ct4*(-ct2*st1*st3 - ct3*st1*st2) + 0.1157*st4*(ct2*ct3*st1 - st1*st2*st3) - 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2))) + (-ct4*(-ct2*st1*st3 - ct3*st1*st2) + st4*(ct2*ct3*st1 - st1*st2*st3))*(-0.1157*ct4*(ct2*ct3 - st2*st3) + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)))], [(-ct4*(ct2*ct3 - st2*st3) + st4*(ct2*st3 + ct3*st2))*(-0.1157*ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + 0.0922*ct5*st1 + 0.1157*st4*(ct1*ct2*ct3 - ct1*st2*st3) -*

*0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))) - (-ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + st4*(ct1*ct2*ct3 - ct1*st2*st3))*(-0.1157*ct4*(ct2*ct3 - st2*st3) + 0.1157*st4*(ct2*st3 + ct3*st2) - 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)))], [(-ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + st4*(ct1*ct2*ct3 - ct1*st2*st3))*(-0.0922*ct1*ct5 - 0.1157*ct4*(-ct2*st1*st3 - ct3*st1*st2) + 0.1157*st4*(ct2*ct3*st1 - st1*st2*st3) - 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2))) - (-ct4*(-ct2*st1*st3 - ct3*st1*st2) + st4*(ct2*ct3*st1 - st1*st2*st3))*(-0.1157*ct4*(-ct1*ct2*st3 - ct1*ct3*st2) + 0.0922*ct5*st1 + 0.1157*st4*(ct1*ct2*ct3 - ct1*st2*st3) - 0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2)))]])*

*J6 = Matrix([[-0.0922*st5*(-ct1*ct5 - st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2)))*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3)) + st5*(-0.0922*ct1*ct5 - 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2)))*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3))], [-st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3))*(0.0922*ct5*st1 - 0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))) + 0.0922*st5*(ct4*(ct2*st3 + ct3*st2) + st4*(ct2*ct3 - st2*st3))*(ct5*st1 - st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2)))], [-(-ct1*ct5 - st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2)))*(0.0922*ct5*st1 - 0.0922*st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2))) + (-0.0922*ct1*ct5 - 0.0922*st5*(ct4*(ct2*ct3*st1 - st1*st2*st3) + st4*(-ct2*st1*st3 - ct3*st1*st2)))*(ct5*st1 - st5*(ct4*(ct1*ct2*ct3 - ct1*st2*st3) + st4*(-ct1*ct2*st3 - ct1*ct3*st2)))]])*

9. VALIDATION OF IK

For the validation of our inverse kinematics implementation, we took help of the code developed for the previous assignments and modified it as per the DH parameters of our robot [3].

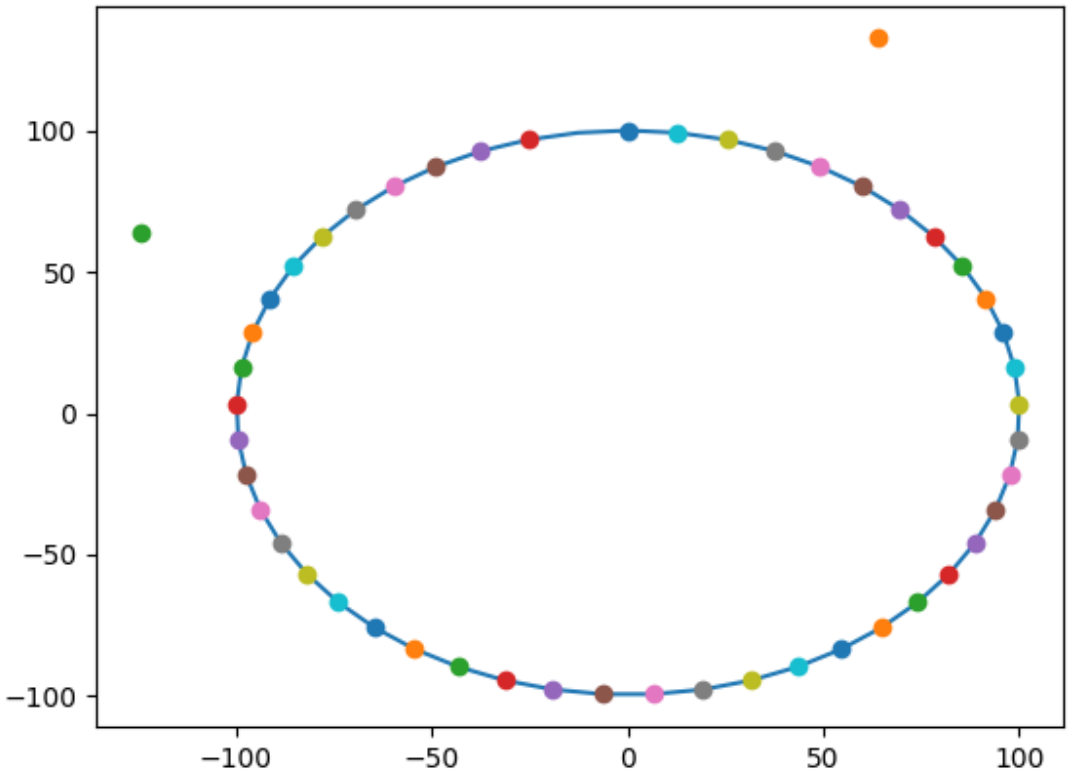


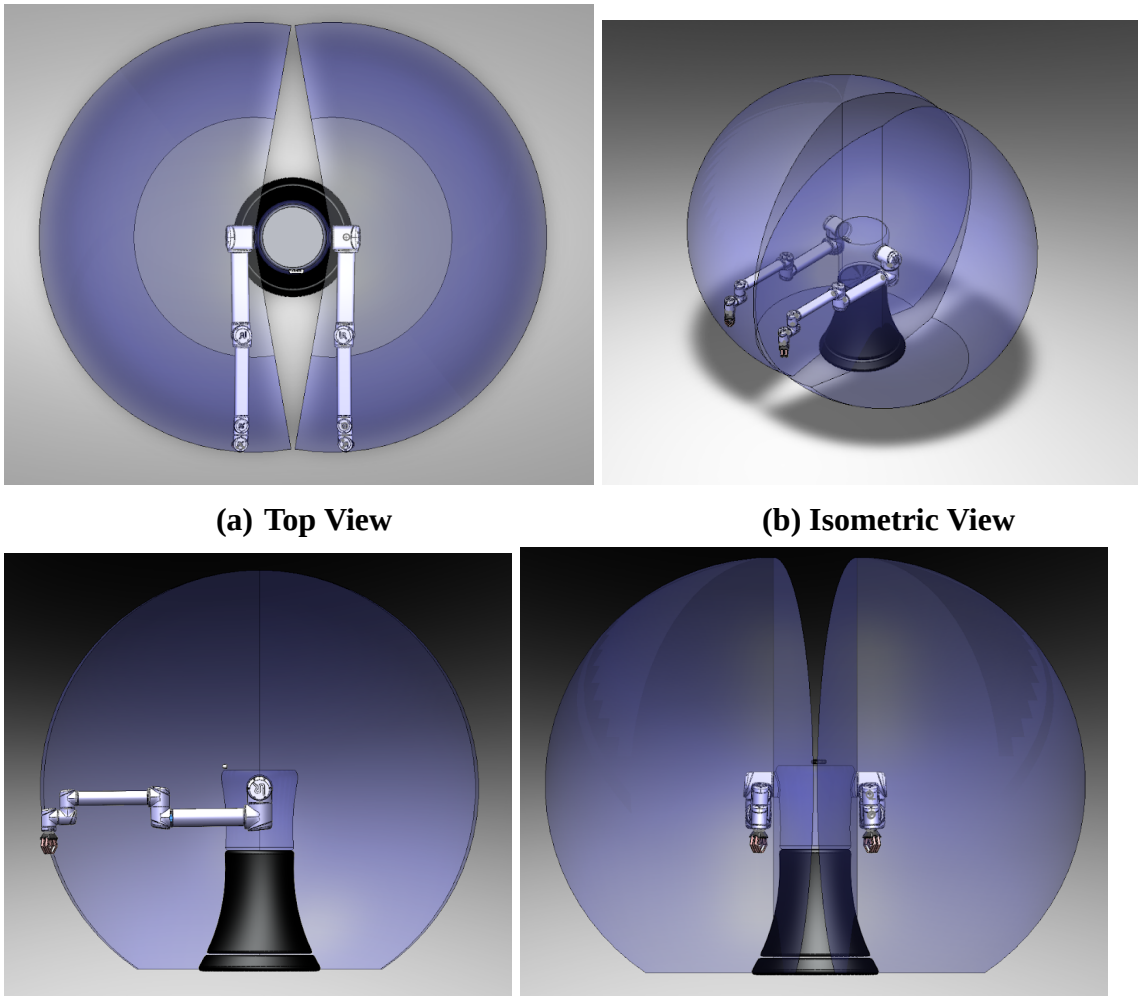
Figure 9.1 IK validation

Fig 9.1 shows the validation of our IK solver. The blue circle is the desired trajectory that we needed the arm to follow and the color dots are the actual points of the trajectory the arm followed.
The following parameters were added for the verification.

thetas = [-pi, 0, -pi, 0, 0, 0]
alpha = [pi/2, 0, 0, pi/2, -pi/2, 0]
d = [0.1273, 0, 0, 0.163941, 0.1157, 0.0922]
a = [0, -0.612, -0.5723, 0, 0, 0]

For this validation, we chose the (y, z) plane for the robot to draw since that is the plane where our use case lies.

10. WORKSPACE STUDY



(a) Top View **(b) Isometric View**

(c) Side View **(d) Front View**

Figure 10.1 Maximum Envelope for Robot

Figure 10.1 shows the workspace of the robot with maximum envelope.

11. ASSUMPTIONS

1. The trajectory of each movement to the robot has been predetermined using inverse kinematics which was derived from the designed model.
2. The location of the pizza is known.
3. The position of each topping with respect to the robot is known.
4. All the joints are rigid.
5. Robot's paths are unobstructed and humans don't enter the workspace

12. CONTROL METHOD

To control all the revolute joints on our robot, we implemented the JointPositionController[5]. The reason for choosing this controller is that through our kinematics, we derive the desired theta values for each joint angle and the Joint Position Controller commands a desired position to the joint by taking in the joint angle in radians. Hence, this controller is perfect for our use case. Another reason for using this controller is that the commands from the position controller will be constrained by the safety limits of the robot, so position commands near the joint limits may not be achieved[6].

The desired angle of the joint is achieved by the controller using the PID control to specify the effort to the joint. The rqt_gui and dynamic reconfigure (Fig 12.1) were used to set the PID gains for all joints of the arm.

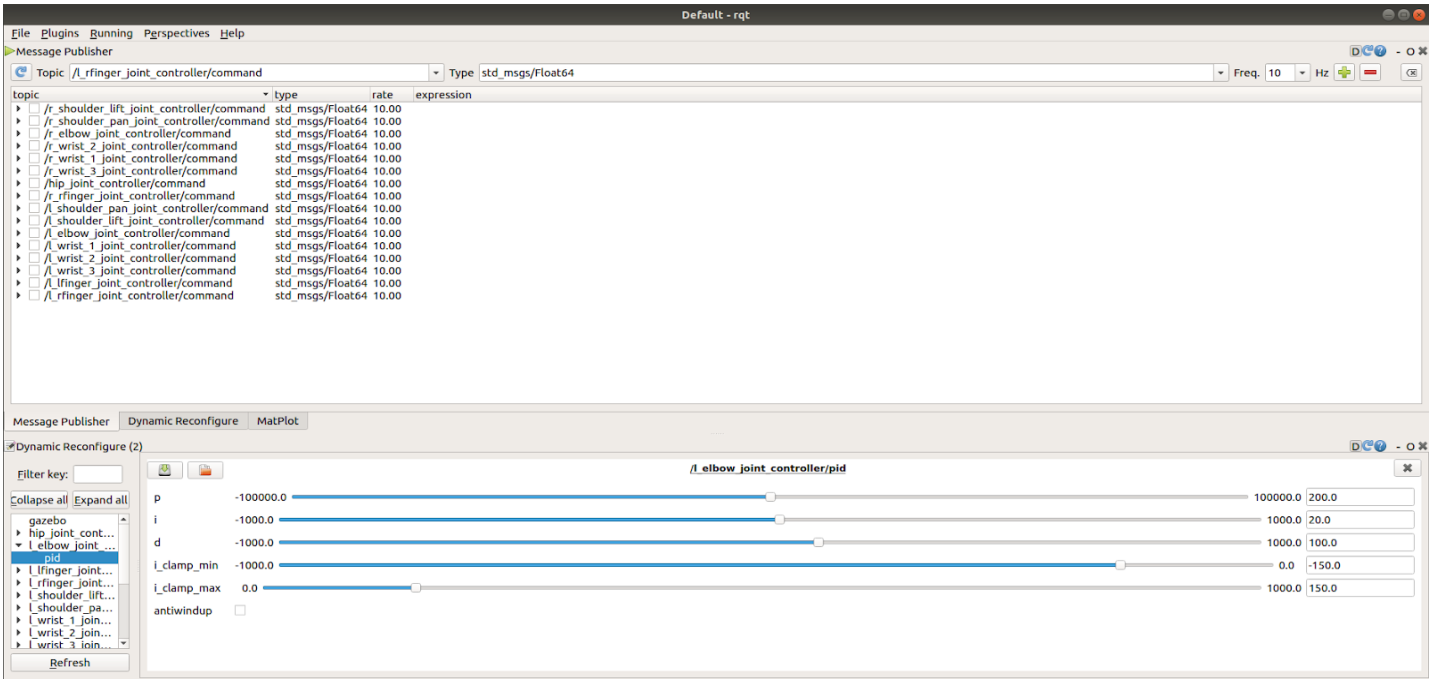


Figure 12.1 rqt_gui

Figure 12.2 PID tuning with dynamic reconfigure & matplot

For example, this is the gains we set to get desired joint angle values :-
l_elbow_joint_controller:

type: effort_controllers/JointPositionController
joint: l_elbow_joint
pid: {p: 200, i: 20, d: 100, i_clamp: 150}

13. GAZEBO AND RVIZ VISUALIZATION

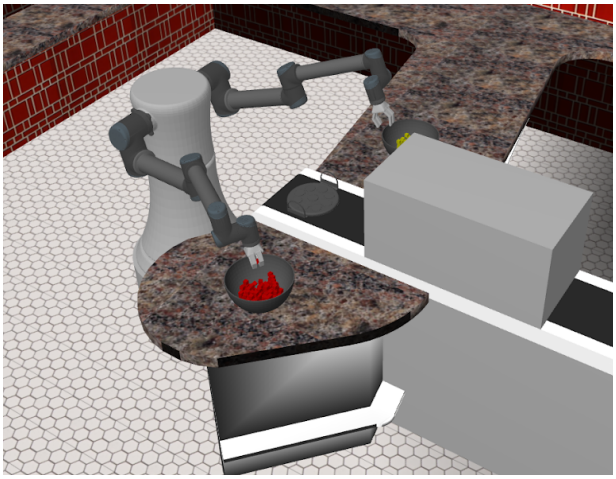


Figure 13.1 Robot in Gazebo Simulator

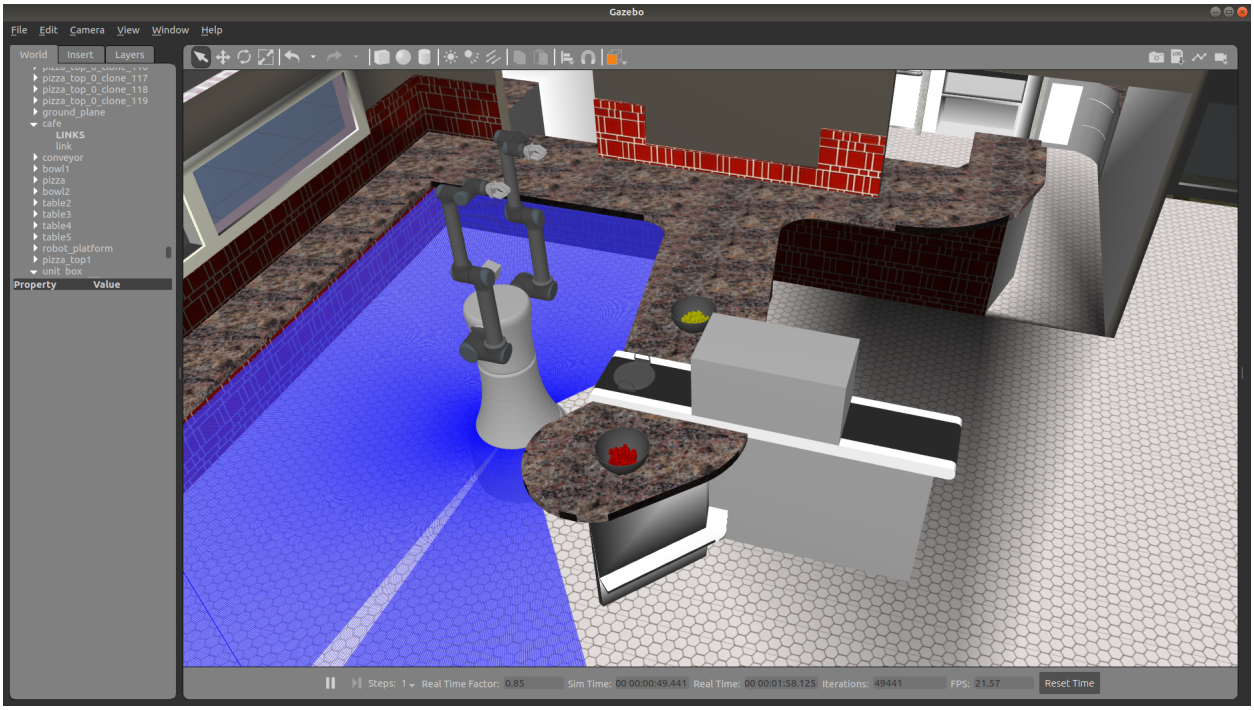


Figure 13.2 Robot in Gazebo World

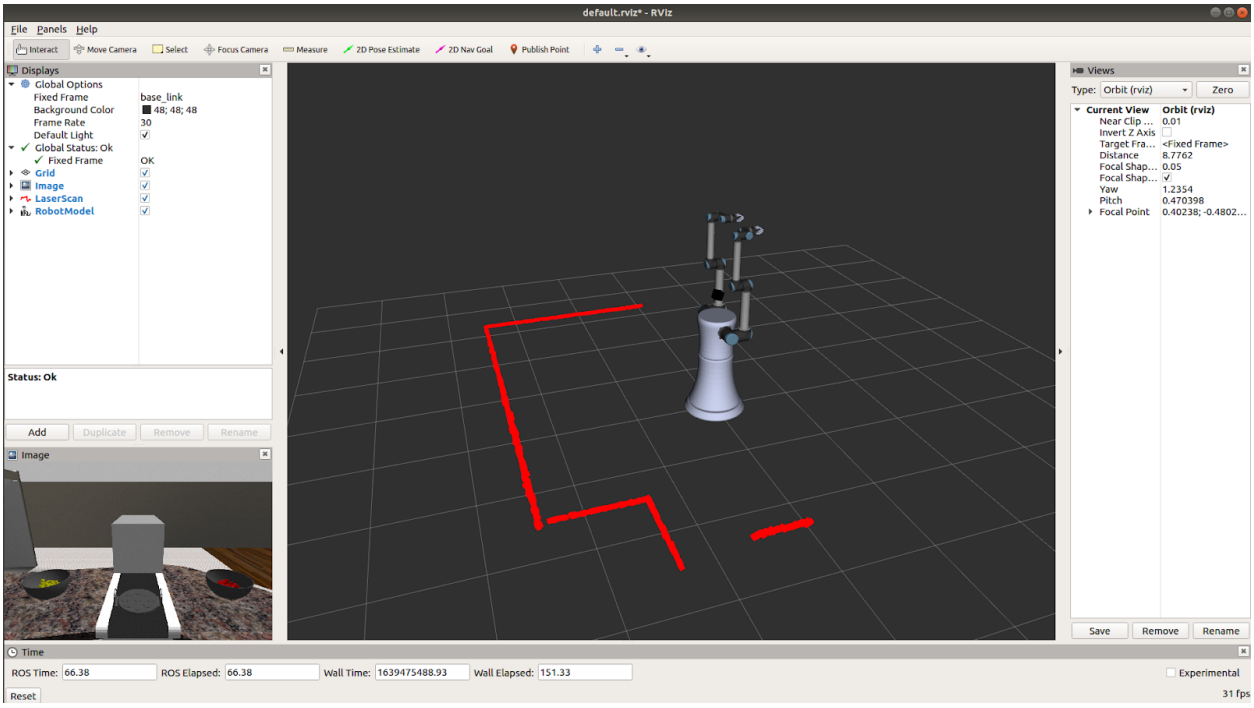


Figure 13.3 Rviz LidarLidarLidarLidarVisualization

14. PROBLEMS FACED

1. The body of the robot would topple when the arms were being moved. This was resolved by increasing the mass of the body to keep it in place.
2. Extra joint limitations were needed to be added since arms were colliding with the body of the robot.

15. LESSON LEARNT

Throughout the development of this project we learned quite a few things that are listed below.

We learned how to,

1. Design and integrate a robot with particular dimensions that will be compatible with the environment we want to use it in.
2. Carry out arm joint controller calibration using a PID controller.
3. Actuate the robot in a gazebo world by publishing messages on various topics.
4. Apply limitations of each joint so that the arms won't collide with each other as well as the body of the robot.
5. Create a robust model that can be used for various different tasks in the fast-food industry.
6. Figure out the workspace of a given robot.
7. Implement FK and IK on real-time robots

16. CONCLUSION

After modeling this food-making robot, we can verify that by implementing kinematics, we can make the robot follow a desired trajectory accurately. This accuracy and repeatability of the robot increase the scope of usage in the food industry.

17. SCOPE OF IMPROVEMENT

1. For further development of the robot and a wider workspace, an actuator on the hip can be applied for the robot to turn 360 degrees.
2. Lidar data can be used further for dynamic obstacles.

18. INDIVIDUAL CONTRIBUTIONS

Aditi- Robot and Gripper design, DH Parameters, FK & IK, Trajectory points mapping.
Pratik- URDF export, World Design, Joint Position Controller, PID Configurations, Launch files.

19. IMPORTANT LINKS

Project repository (Github) -
https://github.com/Prat33k-dev/food_maker
Simulation Video -
https://drive.google.com/file/d/1ToFM6Ej-sCzrsyW8UmaYlT9Ouy-rchKu/view?usp=sharing
Cad Models -
https://drive.google.com/drive/folders/1eIMCfKIN6-10OEwQIi32q56CKl-o1Jfy?usp=sharing

20. REFERENCES

1) Farah Bader, Shahin Rahimifard, A methodology for the selection of industrial robots in food handling, Innovative Food Science & Emerging Technologies, Volume 64, 2020.
2) Wilson, Mike. (2010). Developments in robot applications for food manufacturing. Industrial Robot: An International Journal. 37. 498-502. 10.1108/01439911011081632.
3) https://blog.robotiq.com/top-5-ways-robotics-is-changing-the-food-industry
4) https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/
5) ENPM 662 Fall 2021 Homework 3 and 4. University of Maryland, 2021.
6) https://www.youtube.com/watch?v=rA9tm0gTln8
7) http://wiki.ros.org/robot_mechanism_controllers/JointPositionController
8) https://www.theconstructsim.com/ros-qa-112-how-to-tune-a-pid-with-ros-control/