UNIVERSITY OF
MARYLAND

# PICK-PACK:
# UR5 KITCHEN ROBOT

This Report is for the ENPM662 Introduction to Modelling Course Final Project.


AUTHORS:

SUMEDH REDDY KOPPULA
UID:117386066

RAHUL KARANAM
UID:118172507

SUPERVISOR
Reza Monfaredi, PhD



**University of Maryland- College Park, 2021**
Maryland Robotics Center
INSTITUTE FOR SYSTEMS RESEARCH

# Contents

# Chapter 1

# Introduction and Organization

The use of robots in the food industry has increased drastically over the last decade. Major research has been developing in automating the cooking, packaging, and delivering processes using robots which operate autonomously or assisting in the overall process.

The Market we are focusing is on fast service restaurants and fast-food industry as these two combine a $280 Billion industry globally and has been increasing rapidly since the past few years since COVID-19.Rising labor costs coupled with global labor shortage,soaring rents have made food industries to shift over automating their repetitive tasks replacing the human force.

There is a dire need to automate the food packaging and delivering services where the process should be efficient and processed with no human intervention.

Pick-Pack comes into the help of these restaurant owners and managers by providing a solution to the labor shortage by implementing these robots in their kitchen to automate the food packaging process and can be later integrated as a delivery service provider using mobile robots. This project shows the initial development of a kitchen robot used primarily in food industries where delivery, labor force, and infrastructure is a major concern.

**Organization**

| | |
|---|---|
| 1.Introduction and organisation | This chapter will provide the introduction to the project and explains how the project is organized and structured in the report. |
| 2.Motivation | It contains the motivation behind this project and the problem statement defining the purpose of the project. |
| 3.Robot Description | This chapter will describe the robot specification and overall structure of the robot |
| 4.Robot Degree of Freedom and Dimensions | It describes the functionalities and capabilities of the robot. |
| 5.Robot Design | This will explain the design of the robot and various other parts used. |
| 6.DH Parameters | This section explain the DH calculations for the UR5 robot. |
| 7.Forward Kinematics | This section explains the forward kinematics calculation. |
| 8.Forward Kinematics Validation | This explains about the validation of FK solution. |
| 9.Inverse Kinematics<br>This explains about the validation of IK solution. | IK Validation |
| 10.Workspace Study | It showcases our workspace study. |
| 11.Model Assumptions | It explains about the assumptions considered for the above robot description and the environment. |
| 12.Control | It explains our control structure. |
| 13.Simulation Results | It shows our simulated results in gazebo and rviz. |
| 14.Problems faced | This showcases our issues faced during implementation. |
| 15.Lessons Learned | It explains the lessons learned after completing the project. |
| 16.Milestones and Timelines | This chapter explains about various stages of our project implementation. |
| 17.Conclusion | Discussion about our project. |
| 18.Future Work | It explains the future implementation or posibilities of this project. |
| 19.References | This section refers to all the resources used in developing the project. |

# Chapter 2

# Motivation

From burger flipping to cooking robots, many food industries are beginning to discover the benefits of automating their tedious, repetitive tasks with robots.

The major challenge of any restaurant industry is finding, training, and retaining staff due to an acute shortage in the labor force.

Leveraging these robots to automate repetitive tasks with utmost precision which can work 24/7 with minimum errors will reap profits for the restaurant owners. We can use these robots for packaging, cooking, integrating with the delivery robots and other processes, making fewer mistakes, reducing food wastage, and working all day long without a pay rise.

We propose Pick Pack which is a 6 axis UR5 robot which can assist restaurants by packaging their food products or helping out in the cooking process.

For example, Pick Pack can make a salad bowl mixing up to 8 ingredients. A customer can build his bowl by providing the ingredients as inputs to the robot.

**Problem Statement**

The goal of this project is **to design and simulate a six axis robot to assist food packaging and processing used in food industries such as restaurants.** The problem of labour shortage, delayed order processing, contact less food processing are solved.

1. In regards to our problem statement,we have proposed to design and develop a 6 axis floor mounted robot used for packaging food items in a restaurant.As per the scope we are using the robot to work in a environment where it will automate tasks such as picking,packaging and delivering of finished food products in a restaurant.It takes cares of all the obstacles in its work space by using path planning algorithms.

2. We are designing this robot in a way to reduce the overall space and increase overall efficiency of the output.Implementing this robot will help us to find out the rigid body kinematics such as Forward and Inverse Kinematics of the robot and designing the work space environment will help us in the future to test for different scenarios.

3. In this project we are interested in finding out "**How can we assist the restaurants to compete with the fast pacing world using robots?**"

# Chapter 3

# Robot Description

**UR5 Robot Specifications:** We have chosen UR5 ( Universal Robots ) robotic arm to perform pick-and-place tasks. The base of the 6-axis UR5 robot is clamped on to the above mentioned robot base.

**End Effectors and its design:**
We plan to use modular end-effectors, The UR5 robot will change its end effector according to the task assigned.

Food order processing is a time critical task, hence, instead of using multiple robots at different locations for order processing. we intend to design a floor mounted robotic system, where a single robotic arm traverse from two fixed point in XYZ directions in an assigned workspace to perform the pick and place task / application.

The end effector consists of 2 outward extruded plug like structures. Where, various types of spoons are attached to this plug like end effectors depending upon the task/ planning of the robot. For example,The robot end effector uses a blow shaped spoon to pick vegetables and place them into another bowl.

But as for this project we are going with a adaptive robotiq 85 gripper which is modular and interchangeable.

**Applications:**

1. Packaging and material handling
2. Quality control and inspection
3. Metal fabrication,machining, welding, cutting, cladding etc.
4. Food and beverage processing Planning and decision making

## UR5 Technical specifications

6-axis robot arm with a working radius of 850 mm / 33.5 in

| | |
|---|---|
| **Weight** | 18.4 kg / 40.6 lbs |
| **Payload** | 5 kg / 11 lbs |
| **Reach** | 850 mm / 33.5 in |
| **Joint ranges** | +/- 360° |
| **Speed** | All joints: 180°/s.<br>Tool: Typical 1 m/s. / 39.4 in/s. |
| **Repeatability** | +/- 0.1 mm / +/- 0.0039 in (4 mils) |
| **Footprint** | Ø149 mm / 5.9 in |
| **Degrees of freedom** | 6 rotating joint |
| **Control box size (WxHxD)** | 475 mm x 423 mm x 268 mm / 18.7 x 16.7 x 10.6 in |
| **I/O ports** | Digital in: Controlbox: 2, Tool conn.: 2<br>Digital out: Controlbox: 16, Tool conn.: 2<br>Analog in: Controlbox: 2, Tool conn.: 2<br>Analog out: Controlbox: 2,Tool conn.: - |
| **I/O power supply** | 24 V 2A in control box and 12 V/24 V 600 mA in tool |
| **Communication** | TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP |
| **Programming** | 1. C++, python with ROS<br>2. Polyscope graphical user interface on 12 inch touchscreen with mounting |
| **Noise** | Comparatively noiseless |
| **IP classification** | IP54 |
| **Power consumption** | Approx. 200 watts using a typical program |
| **Collaboration operation** | 15 Advanced Safety Functions<br>Tested in accordance with: EN ISO 13849:2008 PL d<br>EN ISO 10218-1:2011, Clause 5.4.3 |
| **Materials** | Aluminum, PP plastic |
| **Temperature** | The robot can work in a temperature range of 0-50°C |
| **Power supply** | 100-240 VAC, 50-60 Hz |
| **Cabling** | Cable between robot and control box (6 m /236 in)<br>Cable between touchscreen & control box(4.5m/177 in) |
| **Motors** | KBM motors from Kollmorgen |
| **Sensors** | USB camera for perception and active vision |

# Chapter 4

# DOF and dimension

We intend to propose an inverted 6-axis UR5 floor mounted robot which plays a vital role in pick and place of objects from one location to another. This robot consists of two parts:

1. **1 DOF** robotiq 85 adaptive gripper attached to the end-effector of the manipulator.
2. **6 DOF** UR5 is a lightweight, adaptable collaborative industrial robot that tackles food processing applications with ultimate flexibility with modular end-effectors.

Overall, 7 DOF kitchen robot systems provide the advantage of large work areas and better positioning accuracy,providing the ability to place an object correctly.This weight of the robot is 19.4 Kg (gross wt) and capacity of payload is approximately 5 kg. Please refer to Figure 3.1 for the robot design shown in solid works.

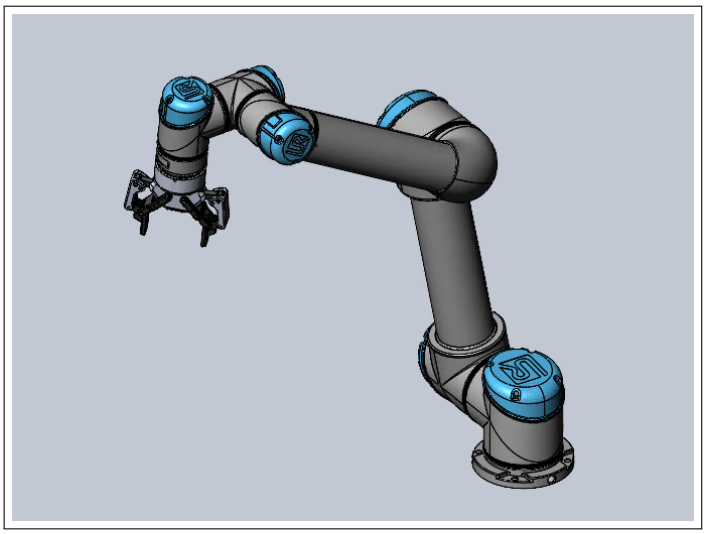

Figure 4.1: SolidWorks design of UR5 Robotic arm with robotiq85 gripper.

# Chapter 5

# CAD

1. **Gantry Design**
   Initially we have designed a CAD model consisting of single axis gantry system matted with UR5 robot. However, while the assembly is exported in the form of URDF, We have noticed design constraints in simulation world in regards with x-axis of gantry system.

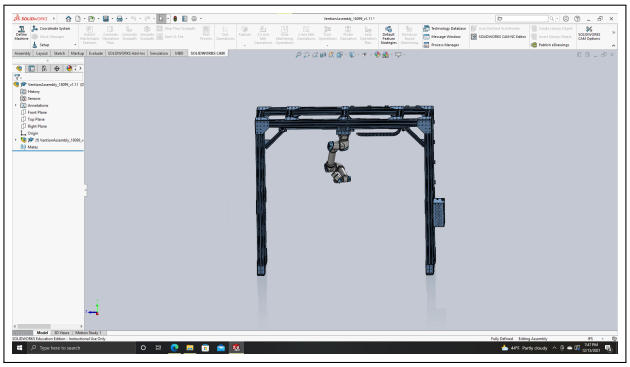

Figure 5.1: Solid Works design of Gantry system with UR5 Robotic arm

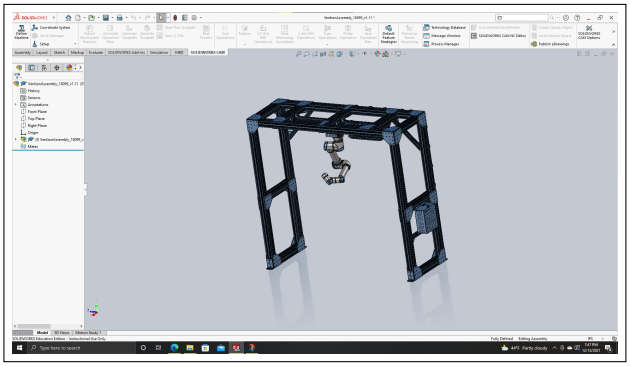

Figure 5.2: Solid Works design of Gantry system with UR5 Robotic arm.

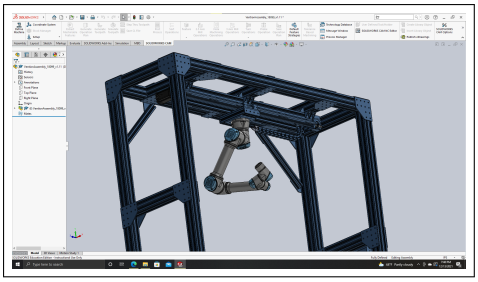

Figure 5.3: Solid Works design of Gantry system with UR5 Robotic arm

2. **UR5 with Robotiq 85 Gripper**

As a fall back plan, We have re-designed our ground mounted UR5 robot with Robotiq 2 finger gripper of pick and place/ packing of the food objects. We have exported this model design in the form of URDF files for gazebo simulation.

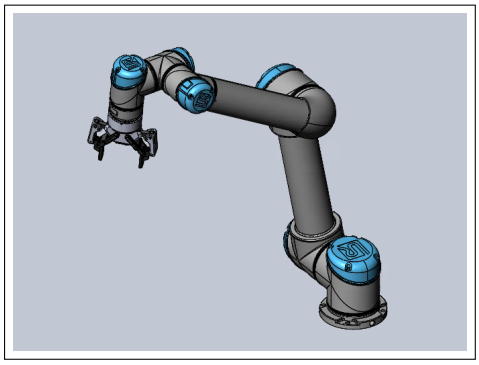

Figure 5.4: SolidWorks design of UR5 Robotic arm with robotiq85 gripper.

# Chapter 6

# DH Parameters

1. A commonly used convention for selecting frames of reference in robotic applications is the Denavit-Hartenberg, or D-H convention. In this convention, each homogeneous transformation Ai is represented as a product of four basic transformations

   $A_i = R_z, \theta_i Tran_z, d_i Trans_x, a_i R_x, \alpha_i$

2. where the four quantities $\theta_i$, ai, di, $\alpha_i$ are parameters associated with link i and joint i. The four parameters ai, $\alpha_i$ , di, and $\theta_i$ are generally given the names link length, link twist, link offset, and joint angle, respectively. These names derive from specific aspects of the geometric relationship between two coordinate frames, as will become apparent below. Since the matrix Ai is a function of a single variable, it turns out that three of the above four quantities are constant for a given link, while the fourth parameter, i for a revolute joint and di for a prismatic joint, is the joint variable.

| UR5 | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Kinematics** | theta [rad] | a [m] | d [m] | alpha [rad] | **Dynamics** | Mass [kg] | Center of Mass [m] |
| Joint 1 | 0 | 0 | 0.089159 | π/2 | Link 1 | 3.7 | [0, -0.02561, 0.00193] |
| Joint 2 | 0 | -0.425 | 0 | 0 | Link 2 | 8.393 | [0.2125, 0, 0.11336] |
| Joint 3 | 0 | -0.39225 | 0 | 0 | Link 3 | 2.33 | [0.15, 0.0, 0.0265] |
| Joint 4 | 0 | 0 | 0.10915 | π/2 | Link 4 | 1.219 | [0, -0.0018, 0.01634] |
| Joint 5 | 0 | 0 | 0.09465 | -π/2 | Link 5 | 1.219 | [0, 0.0018,0.01634] |
| Joint 6 | 0 | 0 | 0.0823 | 0 | Link 6 | 0.1879 | [0, 0, -0.001159] |

Figure 6.1: Frame analysis of UR5 Robot

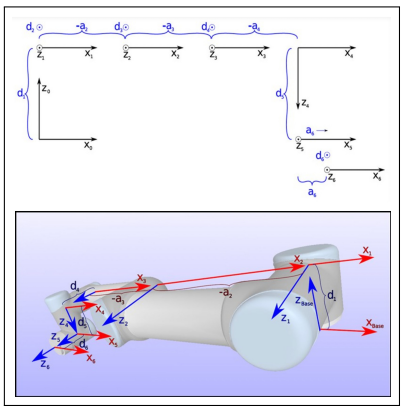

Figure 6.2: DH Analysis of UR5 Robot

3.

# Chapter 7

# Forward Kinematics of the UR5 robot manipulator

Forward kinematics is the basically mapping between the joint space and with regards to our s position of the end-effector.
We first begin by giving the forward kinematics, describing the position of the end effector as a function of joint angles:
$T_6^0$ is the Transformation matrix with respect to the end effector position.
We calculate the $T_6^0$ using the DH paramaters of the UR5 robot which we have calculated in the previous section.

$$T_6^0 = \begin{bmatrix} R11 & R12 & R13 & x \\ R21 & R22 & R23 & y \\ R31 & R32 & R33 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.1}$$

$$R_{11} = c_6(s_1 s_5 + c_{234} c_1 c_5) s_{234} c_1 s_6$$

$$\begin{gathered} \mathrm{R}_{21} = c_6(c_1 s_5 c_{234} c_5 s_1) s_{234} s_1 s_6 \\ \mathrm{R}_{31} = c_{234} s_6 + s_{234} c_5 c_6 \\ \mathrm{R}_{12} = s_6(s_1 s_5 + c_{234} c_1 c_5) s_{234} c_1 c_6 \\ \mathrm{R}_{22} = s_6(c_1 s_5 c_{234} c_5 s_1) s_{234} c_6 s_1 \\ \mathrm{R}_{32} = c_{234} c_6 + s_{234} c_5 s_6 \\ \mathrm{R}_{13} = c_5 s_1 c_{234} c_1 s_5 \\ \mathrm{R}_{23} = c_1 c_5 c_{234} s_1 s_5 \\ \mathrm{R}_{33} = s_{234} s_5 \\ \mathrm{x} = \mathrm{d}_6(c_5 s_1 c_{234} c_1 s_5) + d_4 s_1 + c_1(a_3 c_{23} + a_2 c_2) + d_5 s_{234} c_1 \\ \mathrm{y} = \mathrm{s}_1(a_3 c_{23} + a_2 c_2) d_4 c_1 d_6(c_1 c_5 + c_{234} s_1 s_5) + d_5 s_{234} s_1 \\ \mathrm{z} = \mathrm{d}_1 + a_3 s_{23} + a_2 s_2 d_5 c_{234} d_6 s_{234} s_5 \end{gathered} \tag{7.2}$$

**x** is the end-effector X position.
**y** is the end-effector Y position.
**z** is the end-effector Z position.

$$s_1 = sin(theta_1)$$

$$\begin{gathered} \mathrm{c}_1 = cos(theta_1) s_{234} = sin(theta_2 + theta_3 + theta_4) \\ \mathrm{c}_{234} = cos(theta_2 + theta_3 + theta_4) \end{gathered} \tag{7.3}$$

# Chapter 8

# FK Validation

The forward kinematics calculates the pose of the robot's end-effector from joint states. This means the state of each joint in the articulated body of a robot needs to be defined.

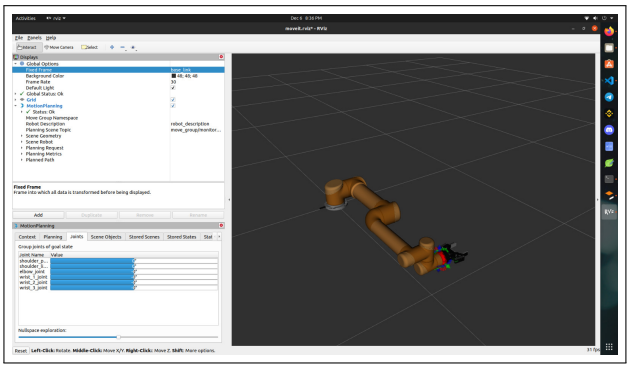

Figure 8.1: Forward Knematics validation using RViz by setting theta values.

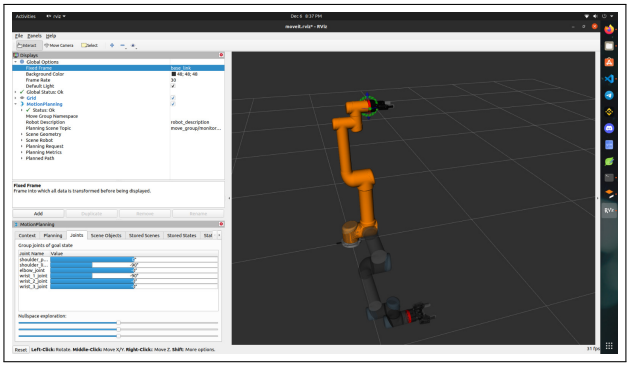

Figure 8.2: Forward Knematics validation using RViz by setting theta values.

# Chapter 9

# IK and IK Validation

**Computation of joint positions from a given end effector position in Cartesian space:** IK Validation Video link

1. We have used numerical approach of KDL library to implement the Inverse kinematics Validation of UR5 robot. By moving the joint configuration (angles) step by step to approach the desired goal position for our end effector position. Therefore, we try to reduce an error function. In our case, we want to minimize the error between our current end effector pose and the desired end effector pose.

2. The End effector pose (lets call it **X**) consists of the position $x_{x/y/z}$ and the orientation $x_{Rx/Ry/Rz}$ which sums up to six degrees of freedom. When we have a robot with n joints, every degree of freedom is dependent on the joint angles $q_{1/.../n}$ i.e. is a function of the joint angles (X = F(Q) )

3. Lets assume a 6*1 matrix X which has desired end effector pose, Let Q [n*1] be the iterative guess for the joint positions and let $\Delta X$ be the distance between the end effector pose that results from the guess and our desired pose, we can write the error function as

$$\Delta X = X - F(Q) \tag{9.1}$$

   where (F(Q)) is the end effector pose for our current guess of joint angles. We want to minimize the error so if we set $\Delta X$ to 0, we end up with the following equation:

$$X = F(Q) \tag{9.2}$$

   The function F() represents our forward kinematics which is calculated in the above chapte. Now, we want to know Q based on X (the inverse) which is not quite as easy so we need to do some tricks.When we differentiate both sides, we get an expression that is very fundamental in robotics:

$$v = J(q) * \dot{q} \tag{9.3}$$

   This formula expresses the relation between the end effector velocity v in cartesian space and the joint velocities $\dot{q}$. The Jacobian matrix J is dependent on the robot kinematics and on the joint angles q in the current configuration and maps the joint velocities to the end effector velocity. As we are interested in the joint angles q that lead to a certain end effector pose so we divide by the Jacobian and bring it to the other side of the equation:

$$v = J(q)^{-1} * \dot{q} \tag{9.4}$$

4. The inverse $J(q)^{-1}$of a Jacobian matrix can only be directly computed if it is a square matrix i.e. if the number of degrees of freedom matches the number of joints n. Unfortunately, the UR5 has 7 joints so we need to approximate the inverse with a so called pseudo-inverse Jacobian ($J(q)^{i}$)

$$v = J(q)^i * \dot{q} \tag{9.5}$$

let's have a look at how this done in KDL. We know that constructing the IK position solver ik-solver, KDL takes the computed chain ur5-chain, a forward kinematics solver fk-solver, an IK velocity solver vel-ik-solver and an integer value for the maximum amount of iterations as arguments.

```
KDL::ChainIkSolverVel_pinv vel_ik_solver(ur5_chain, 0.0001, 1000);
KDL::ChainIkSolverPos_NR ik_solver(ur5_chain, fk_solver, ...
   vel_ik_solver, 1000);
```

5. lets check ik-solver function inside the KDL library that converts the Cartesian end effector position to the joint angles.

```
int ChainIkSolverPos_NR::CartToJnt(const JntArray& q_init, const ...
    Frame& p_in, JntArray& q_out)
    {
        if (nj != chain.getNrOfJoints())
            return (error = E_NOT_UP_TO_DATE);

        if(q_init.rows() != nj || q_out.rows() != nj)
            return (error = E_SIZE_MISMATCH);

        q_out = q_init;

        unsigned int i;
        for(i=0;i<maxiter;i++){
            if (E_NOERROR > fksolver.JntToCart(q_out,f) )
                return (error = E_FKSOLVERPOS_FAILED);
            delta_twist = diff(f,p_in);
            const int rc = iksolver.CartToJnt(q_out,delta_twist,delta_q);
            if (E_NOERROR > rc)
                return (error = E_IKSOLVER_FAILED);
            // we chose to continue if the child solver returned a ...
               positive
            // "error", which may simply indicate a degraded solution
            Add(q_out,delta_q,q_out);
            if(Equal(delta_twist,Twist::Zero(),eps))
                // converged, but possibly with a degraded solution
                return (rc > E_NOERROR ? E_DEGRADED : E_NOERROR);
        }
        return (error = E_MAX_ITERATIONS_EXCEEDED);        // failed ...
           to converge
    }
```

6. After initializing the array of joint positions q-out with the initial joint positions q-init, the starting end effector position is computed with the forward kinematics solver, which gives us our initial F(Q). The distance between the initial end effector pose f and the goal end effector pose p-in is differentiated over a time frame of 1 second so we get our end effector velocity v (delta-twist) which at the same time is our $\Delta X$. Then we call the CartToJnt function of our IK velocity solver. This function takes the current joint position estimate q-out (the starting joint positions q-init in case of the first iteration) and the velocity v as an argument and computes the joint velocity as shown before with the equation. The function stores its result for in qdot-out.

7. After the Jacobian J(q) dependent on the joint positions is computed, the pseudo-inverse J(q)i is approximated with a method called singular value decomposition which is called with an argument for the maximum amount of iterations. The decomposition looks as follows:

$$J(q)^i = V * S_{pinv} * U_t \tag{9.6}$$

8. When the computation is done, the resulting  is stored in the qdot-out array. After returning to the IK position function, the joint velocity is integrated because in the end we want to know the joint position. The integration is done in the CartToJnt function of the position ik-solver by simply adding  to the current estimate of the joint positions q-out. Afterwards, the next iteration starts with this new value. This continues until the end effector velocity resulting from the differentiation of the distance between current end effector pose for the estimated joint configuration and desired end effector pose lies under a certain threshold. Then, the current pose with the estimated joint positions is close enough to the desired pose for our purposes.

**Calculation of inverse kinematics UR5 robot for circular trajectory**

```
 from sympy import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math

# DH parameters of UR5 Robots
# a = [0.00000, -0.42500, -0.39225,  0.00000,  0.00000,  0.0000]
# d = [0.089159,  0.00000,  0.00000,  0.10915,  0.09465,  0.0823]
# alpha = [ 1.570796327, 0, 0, 1.570796327, -1.570796327, 0 ]
# theta_home = [0, -1.570796327, 0, -1.570796327, 0, 0]

# Declaring the variable for joint angles
theta1, theta2, theta4, theta5, theta6, theta7 = symbols("theta1 theta2 ...
    theta4 theta5 theta6 theta7")
d1, d3, d5, d7 = symbols("d1 d3 d5 d7")
d1 = 89.2
d3 = 425
d5 = 392.5
d7 = 205.5



T_01 = Matrix([[cos(theta1), 0, -sin(theta1), 0], [sin(theta1), 0, ...
    cos(theta1), 0],
                                  [0, -1, 0, d1], [0, 0, 0, 1]])
T_12 = Matrix([[cos(theta2),    0, sin(theta2), 0], [sin(theta2), 0, ...
    -cos(theta2), 0],
                                  [0, 1, 0, 0], [0, 0, 0, 1]])
T_23 = Matrix([[cos(0), 0, sin(0), 0], [sin(0), 0, -cos(0), 0],
                                  [0, 1, 0, d3], [0, 0, 0, 1]])
T_34 = Matrix([[cos(theta4), 0, -sin(theta4), 0], [sin(theta4), 0, ...
    cos(theta4), 0],
                                  [0, -1, 0, 0], [0, 0, 0, 1]])
T_45 = Matrix([[cos(theta5), 0, -sin(theta5), 0], [sin(theta5), 0, ...
    cos(theta5), 0],
                                  [0, -1, 0, d5], [0, 0, 0, 1]])
T_56 = Matrix([[cos(theta6), 0, sin(theta6), 0], [sin(theta6), 0, ...
    -cos(theta6), 0],
                                  [0, 1, 0, 0], [0, 0, 0, 1]])
T_67 = Matrix([[cos(theta7), -sin(theta7), 0, 0], [sin(theta7), ...
    cos(theta7), 0, 0],
                                  [0, 0, 1, d7], [0, 0, 0, 1]])

# Calculating transformation matrix from the base frame to end effector frame
T_02 = T_01 * T_12
T_04 = T_01 * T_12 * T_23 * \
                          T_34
T_05 = T_01 * T_12 * T_23 * \
                          T_34 * T_45
T_06 = T_01 * T_12 * T_23 * \
                          T_34 * T_45 * T_56
```

```
T_07 = T_01 * T_12 * T_23 * \
                          T_34 * T_45 * T_56 * \
                          T_67


theta_mat = Matrix([theta1, theta2, theta4, theta5, theta6, theta7])

z_1 = T_01[:3, 2]
z_2 = T_02[:3, 2]
z_3 = T_04[:3, 2]
z_4 = T_05[:3, 2]
z_5 = T_06[:3, 2]
z_6 = T_07[:3, 2]

xp = T_07[:3, 3]

dh_q = xp.jacobian(theta_mat)

z_mat = ...
    z_1.row_join(z_2).row_join(z_3).row_join(z_4).row_join(z_5).row_join(z_6)

J_0 = dh_q.col_join(z_mat)

q_new = Matrix([[0], [-pi/2], [-pi/4], [0], [112*pi/180], [0]])
subs_m = N(J_0.subs([(theta1, q_new[0]), (theta2, q_new[1]), (theta4, ...
    q_new[2]), (theta5, q_new[3]),
                     (theta6, q_new[4]), (theta7, q_new[5])]), 5)
J_0_inv = N(subs_m.inv(), 5)


x_dot = []
z_dot = []
q_dot = []
th = pi/2
xes = []
yes = []
zes = []

fig = plt.figure()
ax = Axes3D(fig)
ax.set_xlim3d(-550, -450)
ax.set_ylim3d(-100, 100)
ax.set_zlim3d(400, 600)

for i in range(40):
    x_dot.append(N(((-1)*1.25*100*sin(th)), 5))
    z_dot.append(N((100*1.25*cos(th)), 5))
    th += pi/20

for i in range(40):
    j_vel_mat = Matrix([[0], [x_dot[i]], [z_dot[i]], [0], [0], [0]])
    q_dot.append(J_0_inv*j_vel_mat)
    q_new = (q_new + q_dot[i]*0.05).evalf()
    transform = N(T_07.subs([(theta1, q_new[0]), (theta2, q_new[1]), ...
        (theta4, q_new[2]),
                                                 (theta5, q_new[3]), ...
                                                    (theta6, q_new[4]), ...
                                                    (theta7, q_new[5])]), 5)

    xes.append(transform[0, 3])
    yes.append(transform[1, 3])
```

```
    zes.append(transform[2, 3])

    subs_m = N(J_0.subs([(theta1, q_new[0]), (theta2, q_new[1]), (theta4, ...
        q_new[2]), (theta5, q_new[3]),
                         (theta6, q_new[4]), (theta7, q_new[5])]), 5)
    J_0_inv = N(subs_m.inv(), 5)
    ax.scatter3D(transform[0, 3], transform[1, 3], transform[2, 3])
    plt.pause(0.1)
    print(i, subs_m.det())


plt.show()
```

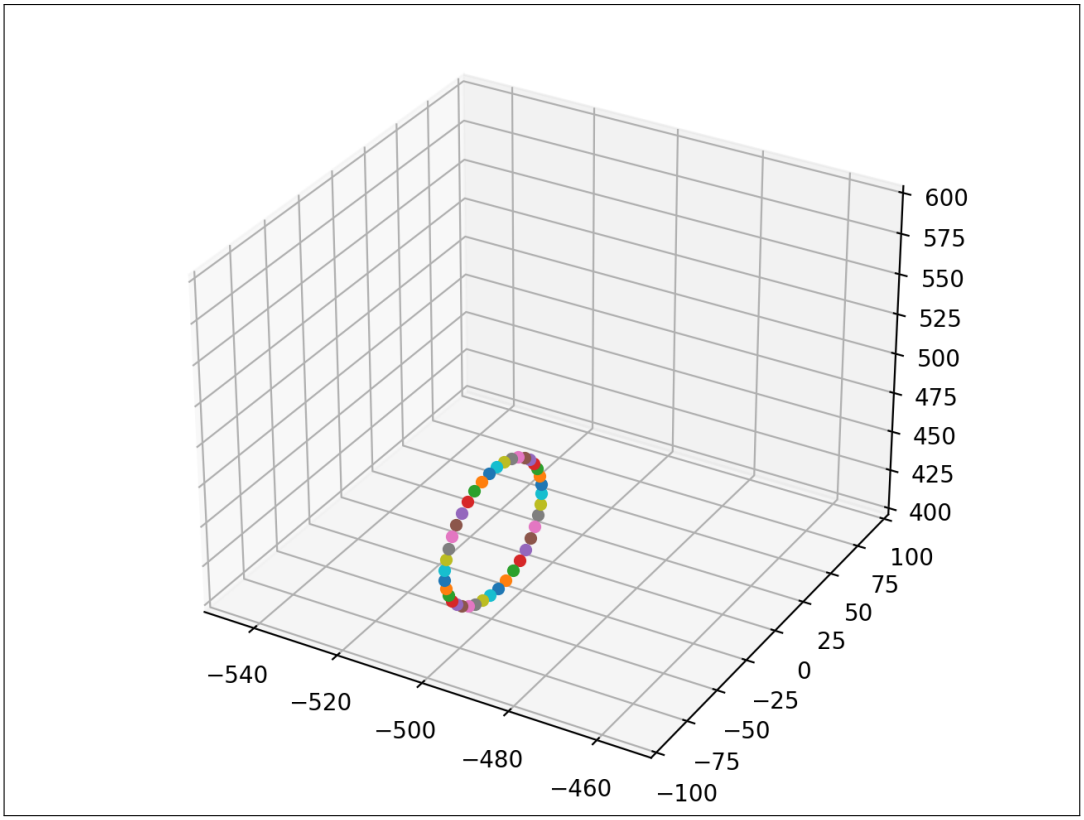


Figure 9.1: Making circular trajectory using Jacobian.

# Chapter 10

# Workspace study

The workspace of the UR5 robot extends 850 mm from the base joint. It is important to consider the cylindrical volume directly above and directly below the robot base when a mounting place for the robot is chosen. Moving the tool close to the cylindrical volume should be avoided if possible, because it causes the joints to move fast even though the tool is moving slowly, causing the robot to work inefficiently and the conduction of the risk assessment to be difficult.

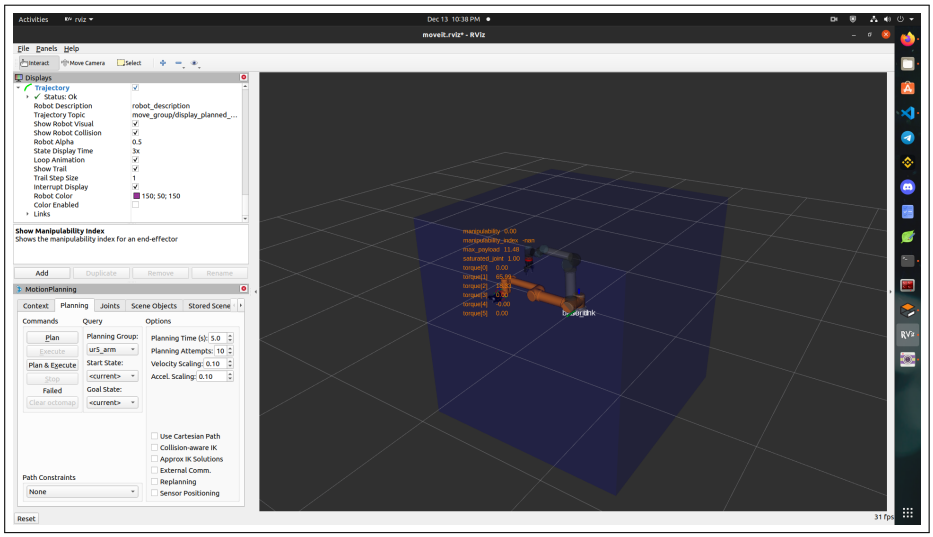


Figure 10.1: Workspace study of UR5 Robot in Rviz

**Simulation video of Workspace :** https://youtu.be/Q8qiyR$_ZVNs$

# Chapter 11

# Model Assumptions

Our Robotic system has 3 stages of design and each stage has few assumption

**Stage 1 Design of a UR5 Robotic arm with robotiq85 gripper**

1. Assuming Repeatability as +/- 1 mm
2. UR5 robotic arm have a payload capacity of 5 kg.
3. The friction and the other external disturbances are not taken into account.

**Stage 2 UR5 Robotic Arm**

1. All the joints are considered to rigid
2. working radius of 850 mm / 33.5 in
3. The base of the robot is attached to the table.
4. Created the custom world with solid objects to perform the pick and place operations.
5. We have considered a set of PID gains which have performed well with the simulation.

**Stage 3 Pick and Place gazebo simulation**

1. Robot is mounted on a table which is of 1.21 height from the ground.
2. Assuming 4-6 different types of objects for pick and place.
3. Planning of the path is predefined and given by the C++ client to the robot using moveit.
4. All the picked items are placed into a bowl. This bowl is assumed to be fixed at a single position

# Chapter 12

# Control of the UR5 robot using PID ROS-Control

1. After, Exporting URDF from Solid works, we have modified URDF files to add transmissions to every joint of UR5 and ros-control to tune controller gains

```
    1. Adding gazebo ROS control
        <plugin name="gazebo_ros_control"            ...
           filename="libgazebo_ros_control.so">
        </plugin>
    2. Adding transmissions to joints
        <transmission name="shoulder_lift_trans">
            <type>transmission_interface/SimpleTransmission</type>
            <joint name="shoulder_lift_joint">
            <hardwareInterface>EffortJointInterface</hardwareInterface>
            </joint>
            <actuator name="shoulder_lift_motor">
            <mechanicalReduction>1</mechanicalReduction>
            </actuator>
        </transmission>
```

   We choose SimpleTransmission as transmission interface which represents a simple reducer transmission. In order to use our controllers in Gazebo we define an EffortJointInterface, because in the end, our position controller output and the value that is commanded on the joint motors is the effort (or force). We pretend the transmission between our actuator and the joint to be 1:1 so we set the <mechanicalReduction> property to 1.

2. Adding the yaml configuration file

```
# Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers ---------------------------------------
  shoulder_pan_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: shoulder_pan_joint
    pid: {p: 500.0, i: 0.01, d: 50.0, i_clamp_min: -100.0, ...
       i_clamp_max: 100.0}
  shoulder_lift_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: shoulder_lift_joint
    pid: {p: 500.0, i: 100.0, d: 30.0, i_clamp_min: -400.0, ...
       i_clamp_max: 400.0}
  elbow_joint_position_controller:
    type: effort_controllers/JointPositionController
```

```
    joint: elbow_joint
    pid: {p: 10000.0, i: 0.01, d: 50.0, i_clamp_min: -100.0, ...
        i_clamp_max: 100.0}
  wrist_1_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: wrist_1_joint
    pid: {p: 200.0, i: 10.0, d: 20.0, i_clamp_min: -400.0, ...
        i_clamp_max: 400.0}
  wrist_2_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: wrist_2_joint
    pid: {p: 100.0, i: 0.1, d: 10.0, i_clamp_min: -100.0, i_clamp_max: ...
        100.0}
  wrist_3_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: wrist_3_joint
    pid: {p: 100.0, i: 0.1, d: 10.0, i_clamp_min: -100.0, i_clamp_max: ...
        100.0}
```

3. Loading controllers to launch file

```
<!-- load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" ...
     type="spawner" respawn="false"
    output="screen" args="shoulder_pan_joint_position_controller ...
        shoulder_lift_joint_position_controller ...
        elbow_joint_position_controller ...
        wrist_1_joint_position_controller ...
        wrist_2_joint_position_controller ...
        wrist_3_joint_position_controller joint_state_controller"/>
```

4. start the robot-state-publisher that publishes the joint state topic of the joint-state-controller transformer to tf

**Analysing UR5 PID control of each joint using RQT**

1. Initially we are launching UR5 in empty gazebo world by running following command in the terminal **roslaunch kitche-bot iksolver.launch**

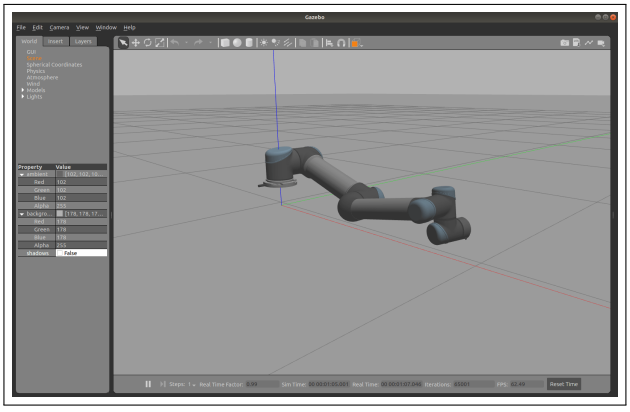

Figure 12.1: UR5 spanning in empty world

2. In new terminal, we should execute the below command to analyze PID control of each joint of UR5 **rosrun rqt-gui rqt-ui**

3. Now we can choose topics from the list and publish on them. In the upper selection bar next to Topic you can simply type in the topic name and then press the green plus button to add them. We will go through the tuning process for the wrist-1 joint so please add the /wrist-1-joint-position-controller/command topic. We can set a static value by extending the topic we want to publish on and

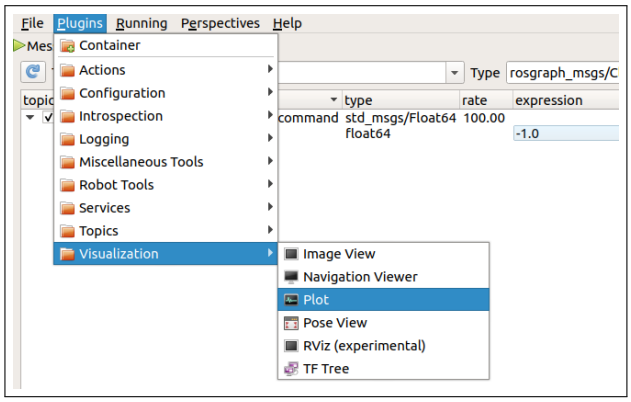


Figure 12.2: Navigate to Plot in Visualizations of RQT panel

put a value into the expression column under the data property of the topic. We will set it to 100 for now. As soon as you put a tick into the box to the left of the topic, you should see the joint reacting to your command.

4. Below we can see the difference between one badly and one well tuned controller that both react to the same position command. You can see that we need tuned control parameters In order to move the robot nice and smoothly. Here, once more, rqt-gui comes in handy.

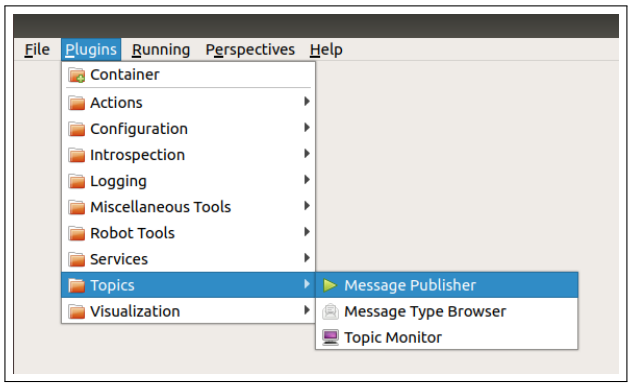


Figure 12.3: Navigate to Message Publisher Topic

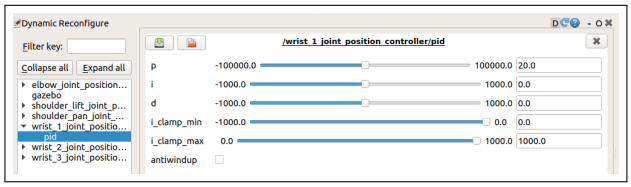


Figure 12.4: Dynamic Reconfigure from the Plugins menu

# Chapter 13

# Simulation Results

1. Please refer our repository for running our project https://github.com/sumedhreddy90/KitchenRobot
2. Simulation Images:

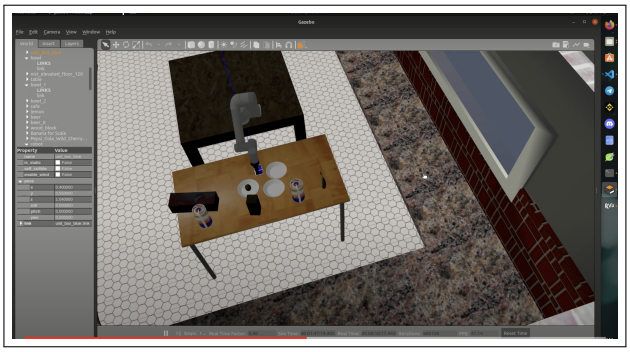

Figure 13.1: Dynamic Reconfigure from the Plugins menu

3. Simulation Videos: click me: https://youtu.be/OZsbqnFeqTQ
   **Installation Instructions:**
   (a) Clone this repository into your workspace:

```
git clone https://github.com/sumedhreddy90/KitchenRobot
catkin_make
source devel/setup.bash
```

   (b) Inverse Kinematics Validation we have implement a C++ program that allows us to move the tcp in Cartesian space. We have created a ROS node that uses KDL for our kinematic calculations and that interfaces with our simulated robot in Gazebo. Launch UR5 onto empty gazebo world

```
roslaunch kitchen_bot iksolver.launch
```

   Launch UR5 Robot onto kitchen gazebo world

```
rosrun kitchen_bot IKSolver
```

   (c) Pick and Place Food objects Usage:

```
roslaunch kitchen_planning demo_gazebo.launch
```

   Run the below script for planning the robot to pick food objects and place them in a plate

```
rosrun kitchen_bot KitchenPlan
```

   Plotting UR5 joint angles with RQT

```
roslaunch kitchen_bot jointRQT.launch
```

# Chapter 14

# Problems faced

1. Integrating end-effector to the gripper has caused issues while planning the robot which has been resolved by correcting the URDF files and creating a virtual joint between them.
2. Due to design complexities and time constraints we had to cut down our project to a floor mounted robot rather than ceiling mounted gantry robot which was proposed earlier.
3. Configuration of the controller in move it has caused to function the robotic arm erratic and crashing rviz several times. We had to delete the config.cache files inorder to reset the previous config cache files.
4. End-effector was not able to grip food items which resulted in dropping of items, in order to show our functionality we have showed using solid objects like a cube.

# Chapter 15

# Lessons Learned

In this section, we present our work timelines and the learning outcomes from this project:

**Stage 1:**

1. In this stage, we plan to design CAD models for our UR5 robot, end effectors and any other possible parts to our robotic system.
2. Furthermore, We would calculate the forward, inverse kinematics using the DH parameters and perform jacobian analysis. After validating the kinematics of the robot we export the models accordingly.

**Stage 2:**

1. In this stage, we have imported the URDF (exported from solidworks) to the gazebo for simulation.
2. Furthermore, We have created world for our simulation and added custom objects.On top of that, we have added controllers and configured through moveit interface.

**Stage 3:**

1. In this stage, we have used the moveit configuration to check the controllers positions in rviz by planning our desired home positions.
2. Furthermore, We have written C++ node to send commands to the moveit group interface in order to move the joints to the desired trajectory.
3. We performed pick-and-place operation using the C++ client node and simulated the operation by placing food ingredients into various bowls.

**Learning Outcomes**

1. Design and calculate the Forward and Inverse Kinematics of Universal Robot UR5.
2. Validating the forward and inverse kinematics of a robotic manipulator.
3. Study about creating world in Gazebo and Rviz replicating different scenario of the project.
4. Picking and Placing of various ingredients into a bowl by planning path which avoids obstacles.
5. Understanding of integration of Sensors,interchangeable end-effectors, controllers in Gazebo or Rviz.
6. Study about various path planning algorithms for planning the trajectory of PickPack.
7. Studied and learned different software tools such as ROS, Solidworks , Moveit , Gazebo and Rviz.

# Chapter 16

# Milestones and Timelines

| **Task** | **Days** | **Total Days** |
|---|---|---|
| Stage 1.a Research and Design of x- axis Gantry, UR5 Robot | 11-07-2021 to 11-07-2021 | 1 |
| Stage 1.b CAD Design and model for Gantry system | 11-08-2021 to 11-08-2021 | 1 |
| Stage 1.c CAD Design and model for UR5 Robot | 11-09-2021 to 11-09-2021 | 1 |
| Stage 1.d CAD Design and model for end effectors | 11-10-2021 to 11-10-2021 | 1 |
| Stage 1.e Any other possible parts design to our robotic system | 11-11-2021 to 11-11-2021 | 1 |
| Stage 1.f Forward, inverse kinematics calculations and validations using the DH convention method and Jacobian analysis | 11-12-2021 to 11-13-2021 | 2 |
| **""Found issues with the gantry mounted system, had to switch to a floor mounted robot""** | | |
| Stage 2 Part 1 To build a custom simulation world with objects clamped with UR5 Robot in gazebo | 11-13-2021 to 11-17-2021 | 5 |
| Stage 2 Part 1.a Import the UR5 robotic URDF (generated from the solidworks exportURDF tool) arm with the gripper to gazebo for simulation. | 18-07-2021 to 22-07-2021 | 3 |
| Stage 2 Part 1.b Import and place table consisting of different food objects and a bowl placed at fixed position | 18-07-2021 to 22-07-2021 | 3 |
| Stage 2 Part 1.c Include other objects into our world environment and integrate sensors, controllers, transmission and generate final URDF | 18-07-2021 to 22-07-2021 | 3 |
| Stage 2 Part 2 Control the simulated robot in Gazebo using Moveit/ Moveit 2 | 18-07-2021 to 22-07-2021 | 3 |
| **Stage 2 Part 2.a** Install Moveit and launch Moveit setup Assistant with RViz and Plan a cartesian path directly by specifying a list of waypoints for the end-effector to go through. I.e create moveit config and configure in Rviz | 11-18-2021 to 11-19-2021 | 1 |
| **Stage 2 Part 2.b** creating launch files with custom models and custom world | 11-20-2021 to 11-21-2021 | 2 |
| **Stage 2 Part 2.c** write a C++ client or node files to automatically send control commands | 11-22-2021 to 11-23-2021 | 2 |
| **Stage 2 Part 2.d** Carry out a complete autonomous motion planning by using visual feedback | 11-24-2021 to 11-25-2021 | 2 |
| **Stage 3** Script and ROS nodes, topics, publisher-subscriber, services and actions | 11-29-2021 to 12-01-2021 | 3 |
| **Stage 3.a** communication and control between UR5 Robot | 12-02-2021 to 12-04-2021 | 2 |
| **Stage 3.b** Add or remove features accordingly | 12-05-2021 to 12-05-2021 | 1 |

# Chapter 17

# Conclusion

In this project, we have designed and simulated our kitchen robot to perform pick-and-place operations on food ingredients.We have successfully performed forward and inverse kinematics of our robot.These kinematics have been validated using different methods which are mentioned in the above sections.

We planned our trajectory using move it group interface for simulation of our robot in gazebo.This robot can be further developed into a cooking robot by integrating with high-end sensors and using deep learning to analyse the environment and performing the assigned tasks.

Overall, we have controlled our robot and simulated in various scenarios. However, alongside with the difficulties and problems faced while doing this project, we have tried to fulfill all the objectives which we have mentioned in the above sections.

1. For validating the manual kinematics equations, python  C++ programming was used, to derive the matrices, and known joint angles and configurations will be substituted in the equations to check for correctness.
2. Rviz has been used to visualize the sensor data
3. Gazebo has been used to validate the robot model and it's proper functioning under various control inputs
4. For UR5 robot kinematics validation, we would check for a visual pose robot in ROS for given inputs using "MoveIt" or using a python package visual kinematics.

# Chapter 18

# Future Work

In this section, we look at the potential areas of improvement which can be implemented in the future.

1. Integrating with camera sensor data to detect different food objects.
2. Adding custom made end-effectors which can be used for any object.
3. Inverting the UR5 robot and mounting the robot to a single axis gantry for covering more work space.
4. Adding two UR5 robotic arms to the gantry axis to cover larger work space.
5. Designing multiple end effector for different application such as spoon shaped end effector for picking up vegetables,tong shaped end effector for picking leafy salads or or any other ingredient.
6. Collaborating with mobile robots for delivering of the food items across the restaurants or outdoor.

Figure 18.1: Kitchen Robot with two robotics arms on a gantry system

# Chapter 19

# References


1. ARCAM-Dolly Upside Down:The 6-Axis Robotic Arm on track which support the arm upside down
2. Machine Vision System for 3D Plant Phenotyping
3. Focus Group Evaluation Of An Overhead Kitchen Robot Appliance
4. Designing a perception pipeline to compute grasp poses and manipulate using move-it for the UR5 robot.
5. VISUAL PERCEPTION WITH UR5 ARM TOWARDS SEMANTIC MANIPULATION
6. Move-it Tutorials ROS
7. Universal Robot
8. Control of Robot using Moveit
9. Universal Robot UR5 Technical Datasheet
10. UR5 Forward and Inverse Kinematics