ENPM 662: Introduction to Robot Modeling

Modeling a Universal Robot Arm (UR 5) with Parallel Gripper to Assist Disabled people

Project Report

Kulbir Singh Ahluwalia (UID: 116836050)
Patan Sanaulla Khan (UID: 116950985)

13 December 2019

Contents

List of Figures

Abstract

As we move towards robots becoming sentient, it is clear that we must start to rethink what robots mean to society and what their role is to be. This project presents the ability of UR5 arm as a model with robotiq 85 as a gripper to assist a disabled person.

First, the Motivation of the project is discussed. The following sections of the report focuses more on the description of the Robot and the gripper, and the steps to replicate the simulating process for the robot. The UR5 arm is simulated using Moveit and Rviz in Ubuntu 16.04 and ROS kinetic. Then the validation work is showed based on the DH parameters of the robot and computation of the forward and inverse kinematics along with validation and conclusion.

Forward Kinematics of robot manipulators is derived using a systematic approach based on the Denavit-Hartenberg convention and the use of homogeneous transformations. We validate the results using the Robotics toolbox in MATLAB.

Acknowledgement

I would like to thank Professor Chad Kessens for helping and motivating me to continue working on this project despite the tough times. Though we were not able to simulate it successfully in Gazebo, we learnt so much that it is all worth it in the end. We learnt about using ROS, git, Overleaf for online LaTeX editing of our report, Moveit for generating packages and Rviz for visualising in Gazebo. All these are in demand skills which will help me land an internship and provide a strong base for the semesters to come.

A big thanks to the TAs Sanchit and Rachith sir for their patience while clearing my doubts. I am also grateful to the self driven, passionate and helping colleagues who continue to inspire me with their amazing ideas, zeal to get stuff done and crazy ideas which they implement no matter what. I would also like to thank my seniors who helped me learn git and Gazebo and helped ease the learning curve. (or as I like to say, Learning "cliff")

Chapter 1

Introduction

1.1 Motivation

One billion people, or 15 percent of the world's population, experience some form of disability. The incidence of disability is higher for developing countries like India, China, Brazil and Mexico.[1] Disabled people are more likely to encounter negative social effects such as:

- Lesser schooling
- Poorer health outcomes
- Lower employment levels
- Higher rates of poverty

The results are relevant and applicable to a wide range of applications, including helping disabled people to interact independently with their environment, for assistance in labs, kitchens and workshops to handle tools.[1] Barriers to total social and economic integration for people with disabilities include:

- Inaccessible physical environments
- Unavailability of affordable technology
- Lack of ease of use of assistive devices

1.2 Problem Statement

Majority of disabled people in wheelchairs need physical assistance from other people and are thus dependent on them. In this project, the UR5 arm would be attached to a wheelchair of the disabled person to help him interact with his surroundings and become independent.

We have a person who would use the UR5 arm to grab a glass of milk from the shelf top and move it to his mouth all by himself by entering the coordinates of the glass. Hence the robot would mimic the normal human behavior of pick and place generally performed by the human hand.

1.3 Proposed Model

Our aim was to attach the robotic arm to the disabled person's wheelchair to help him interact with his surroundings and become independent. In order to mimic such a motion of the human hand, we require a 6-DOF robot arm where the robot uses 3 DOF for positioning and 3 DOF for orientation. The end effector which is a parallel gripper is needed to grip the object. The robot is mounted on one of the either sides of the chair and is arranged such that it provides maximum workspace to the arm.

When the wheel chair is at the desired location where the object is within the arm's range, The user selects an item by providing the end-effector co-ordinates and then activates the robot arm which picks it up. Once the item is within reach then the robot activates the grasp where the end effector moves the parallel gripper fingers to grab the item.

Once the item is picked up, the robot arm moves with the selected item to the pre-defined location or a well known location which is in front of the user's mouth using the 6-DOF and performs the tilt action for the user to sip the liquid (milk).

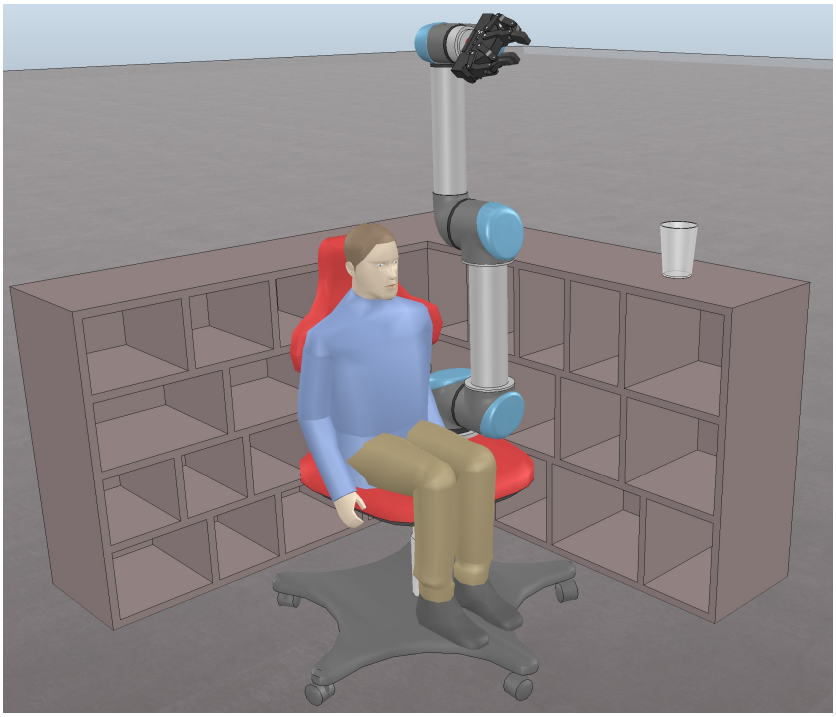


Figure 1.1: Proposed model of UR5 with robotiq 85 gripper on a wheelchair. The person uses the UR5 arm to grab a glass of milk from the top of the shelf and moves it all by himself to his mouth by entering the coordinates of the glass.

Chapter 2

Robot Description

2.1 Universal Robot-5 Arm

The Universal Robot (UR5) arm is composed of extruded aluminum tubes and joints. The end effector is attached at the end of wrist 3 joint. By coordinating the motion of each of the joints, the robot can move its tool around freely, apart from a cylindrical column directly above and directly below the base. The reach of the robot is 850 mm from the center of the base [2].(Refer Fig:2.1). Each of the link length and the center is given in the table 3.1.

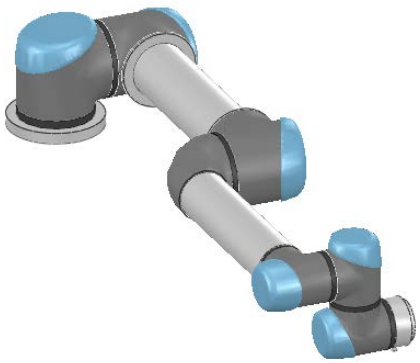


Figure 2.1: Universal Robot UR5 Arm [13]

Dynamics	Mass[Kg]	Center of Mass[m]
Link 1	3.7	[0, -0.02561, 0.00193]
Link 2	8.393	[0.2125, 0, 0.11336]
Link 3	2.33	[0.15, 0.0, 0.0265]
Link 4	1.219	[0, -0.0018, 0.01634]
Link 5	1.219	[0, 0.0018,0.01634]
Link 6	0.1879	[0, 0, -0.001159]

Table showing the Mass of each link along with the Center of masses of links

2.2 Workspace of the Robot

UR 5 has a range of up to 850 mm from the base joint. The robot has a cylindrical volume directly above and below it (refer Fig 2.2) where a mounting place is chosen. UR5 should not come close to the cylinder because of high chances of collisions and unstable velocities.

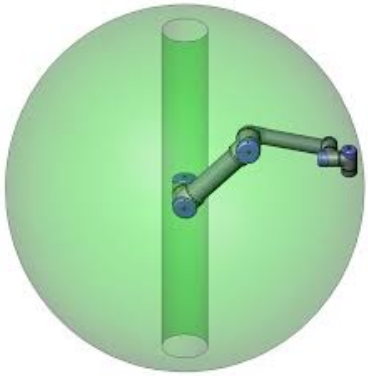


Figure 2.2: Workspace of the Arm

2.3 Kinematic Diagram

Now we see the frame diagram and co-ordinate diagram for the UR5 arm. These values are generated and validated using the process and steps mentioned in the article [13] on the UR5 robot.

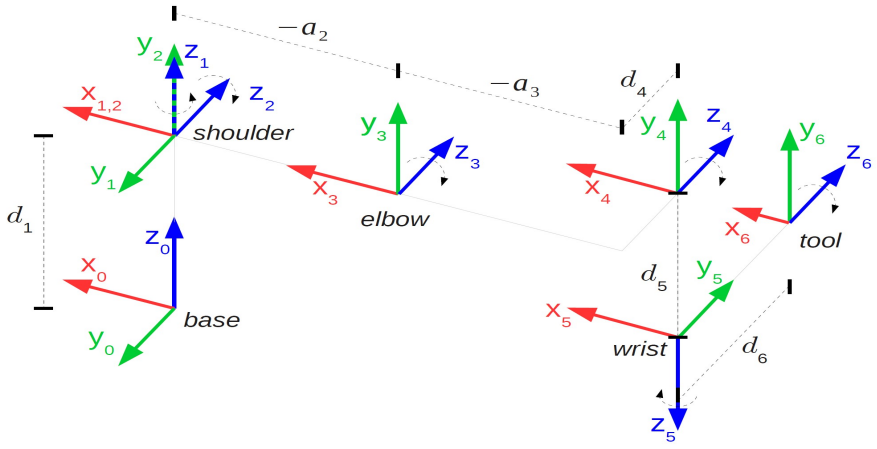


Figure 2.3: Denavit–Hartenber Co-ordinate Diagram [4]

i	θ_i	d_i	a_i	α_i
0	-	-	0	0
1	θ_1	0.08916	0	0
2	θ_2	0	0	$\frac{\pi}{2}$
3	θ_3	0	-0.425	0
4	θ_4	0.10915	-0.39225	0
5	θ_5	0.09456	0	$\frac{\pi}{2}$
6	θ_6	0.0823	-	$-\frac{\pi}{2}$

Figure 2.4: Denavit–Hartenberg Table for UR5 [4]

2.4 Robotiq-85 Gripper

The 2-Finger parallel gripper is designed for pick and place applications. The gripper has two articulated fingers that each have two joints (two phalanxes per finger). The grasp-type gripper can engage up to five points of contact with an object (two on each of the phalanges plus the palm).
The fingers of the gripper are under-actuated, meaning they have fewer motors than the total number of joints. This configuration allows the fingers to automatically adapt to the shape of the object they grasp, and it also simplifies the control of the grasp-type gripper[5].

Figure 2.5: Robotiq 85 attached to a UR5 arm[5]

Chapter 3

Methodology

3.1 Assumptions

For this project to perform as desired, the following set of assumptions are taken into account:-

1. All links are rigid and can handle the payload.
2. We assume that the point of contact is a soft touch grasp contact.
3. There is no slippage between the item and end-effector.
4. Assume the objects' geometry and the relative position of the object with respect to the Gripper is symmetric.
5. The robot satisfies all the environment conditions of the gripper and the arm.
6. The inertia of the chair is much greater than the inertia of the arm.
7. The chair does not topple in any scenario.
8. The workspace of the model is fixed with respect to the mount position in the wheelchair.
9. The object is within the vicinity of the arm and is modified to operate with the robot gripper.

10. The weight of the object to be picked is within the capacity range of the gripper and the arm.

11. The user is able to tell the coordinates of the object to be picked up with respect to the arm's base.

12. The arm moves slowly and stops if a collision force of more than 200 Newtons is detected for user safety.

13. There is a straw in the glass and the user can move his head to drink the liquid in the glass.

14. The UR5 arm can tilt the glass just as much as required and the liquid does not spill.

3.2 Tools Used

3.2.1 ROS

The Robot Operating System (ROS) is an open-source, meta-operating system. ROS is the de facto standard for writing interoperable and reusable robot software. It provides services like hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It uses the concepts of nodes, publishers, subscribers and messages to operate effectively between systems. It also provides tools and libraries for building, writing, and running code for robots. [11]

3.2.2 MoveIt

MoveIt provides functionality for kinematics, motion/path planning, collision checking, 3D perception and robot interaction. MoveIt is a primary source of a lot of the functionality for manipulation tasks in ROS.
MoveIt builds on the ROS messaging and build systems and utilizes some of the common tools in ROS like the ROS Visualizer (Rviz) and Unified Robot Description Format (URDF) files which follow XML format.
We have used the KDL Kinematic solver in Moveit for solving Inverse kinematics problems quickly.

3.2.3 RViz

Rviz which stands for ROS visualization is a 3D visualizer for displaying sensor data and state information from Moveit and other ROS systems. We can even see data from sensors like the kinect sensor which gives point clouds as output. It can also be used to validate poses and for visualizing path planning.

3.2.4 Robotics ToolBox by Peter Corke in MATLAB

The robotics toolbox provides pre defined models and convenient functions for calculating FK, Ik, Endf effector velocities and more. It uses a very general method of representing the kinematics and dynamics of serial-link manipulators as MATLAB® objects – robot objects can be created by the user for any serial-link manipulator. The toolbox also supports mobile robots with functions for robot motion models and path planning algorithms.

Chapter 4

Replicability

4.1 Pre-requisites

Clone the GitHub repository and follow the following steps to configure your machine with the initial requirements to replicate the process.

4.1.1 Setting up ROS

1. Make sure the system is running on the Ubuntu platform, Version - Xenial (Ubuntu 16.04.06)

2. Configure the system to Install ROS using the terminal and install ROS- Kinetic on the system

   ```
   $ sudo apt-get install ros-kinetic-desktop-full
   ```

3. Once the ROS is successfully installed in the machine we initiate ROS with the following commands.

   ```
   $ sudo rosdep init
   $ rosdep update
   ```

4. Use the following commands to add the setup.bash command to the .bashrc file, in order to run it everytime the computer starts.

   ```
   $ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
   $ source ~/.bashrc
   ```

5. Now that the ros is installed and ready to run, we have to create an ros workspace. Catkin make build the required subfolders and files automatically.

   ```
   $ mkdir -p ~/ws_662/src
   $ cd ~/ws_662/
   $ catkin_make
   ```

6. After running catkin_make, you should notice two new folders in the root of your catkin workspace: the build and devel folders.
   The build folder is where cmake and make are invoked, and the devel folder contains any generated files and targets, plus setup .*sh files so that you can use it like it is installed.

4.1.2 Setting up MoveIt

Install MoveIt package using the command:-

```
$ sudo apt install ros-kinetic-moveit
```

4.1.3 Setting up Git Files

Steps to clone the GIT repositories of the official Universal Robots and Robotiq, to get the URDF models of the robot arm and the gripper. Visit the official GIT repository of Universal Robots: https://github.com/ros-industrial/universal_robot.
Fork the repository and copy the clone path to clone the forked repository into the system. Create a new folder /GIT_REPOS and clone with the following commands

```
$ mkdir -p ~/GIT_REPOS/ur
$ cd ~/GIT_REPOS/ur
$ git clone  https://github.com/ros-industrial/universal_robot.git
$ ln -s ~/GIT_REPOS/ur ~/ws_662/src
```

Similarly repeat the above steps to clone the forked repository of the Robotiq official git repository. https://github.com/ros-industrial/robotiq

```
$ cd ~/GIT_REPOS/gripper
$ git clone  https://github.com/ros-industrial/robotiq.git
$ ln -s ~/GIT_REPOS/gripper ~/ws_662/src
```

Always do catkin make after you clone a repository to make sure all the subfolders and files are present.

```
$ catkin_make
```

4.2 Creating a URDF file

At this point, the individual repositories of the robot arm and the gripper are available in the source folder(aka src) of the created catkin workspace. The next step is to combine the gripper and the arm in to one URDF file and make it available in the catkin workspace.

- Create a new file in the /src directory of the workspace.
- Add the xml syntax and a basic robot tag with the robot name to be specified. Following the robot tag, create a link tag in the URDF file which will act like the parent link to the arm. Let it be called "world".

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro" name="ur5_robotiq">
<link name="world" />
```

- Now attach the arm to the world link at the particular xyz and rpy parameters using the following code.

```
<xacro:include filename="$(find ur_description)/urdf/ur5.urdf.xacro" />
<joint name="world_joint" type="fixed">
<parent link="world" />
<child link = "base_link" />
<origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0" />
</joint>
```

- Follow the same syntax to attach the end effector to the arm, where the path would be the path of the robotiq85 urdf or xacro file. And link the end-effector to the "ee_link" at the x, y and z values. Set all to 0.0 and the rpy as "0.0 0.0 1.5708".
- At the end include the following to include the ros_control plugin and end file.

```
<plugin name="ros\_control" filename="libgazebo_ros_control.so" />
</robot>
```

- Save the file as .urdf. And us the following command in terminal to check if the generated URDF file is error free and has no link issues.

```
$ check_urdf ur5_robotiq.urdf
```

4.3 Creating a MoveIt Config Package

1. Once the URDF file of the UR5 arm + gripper is ready, the file can be opened with MoveIt using the following command:-

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

2. Now select the option, "create a new MoveIt package" and browse the path to the newly created URDF file which has the robot (both arm and gripper). Once you have selected the desired path, launch the file in MoveIt.
   Upon successful import of the URDF file the setup tool allows the user make the necessary configurations for the moveIt. Click the tabs on the left side to continue the configuration process.

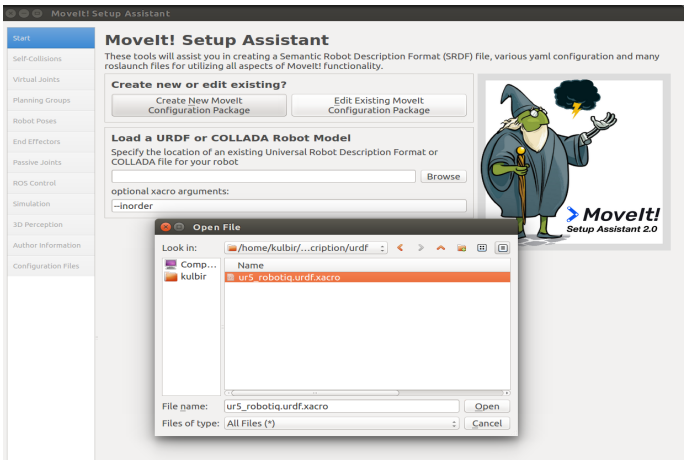


Figure 4.1: Creating a new MoveIt Package

3. First tab is self-collisions, which searches for pairs of robot links that can be safely be disabled from collision checking, in order to decrease the motion planning time. Give the needed sampling density and minimum collisions as 95% and generate the collision matrix. Based on the links of the robot, moveIt calculates the collision matrix and allows us to configure for changes.

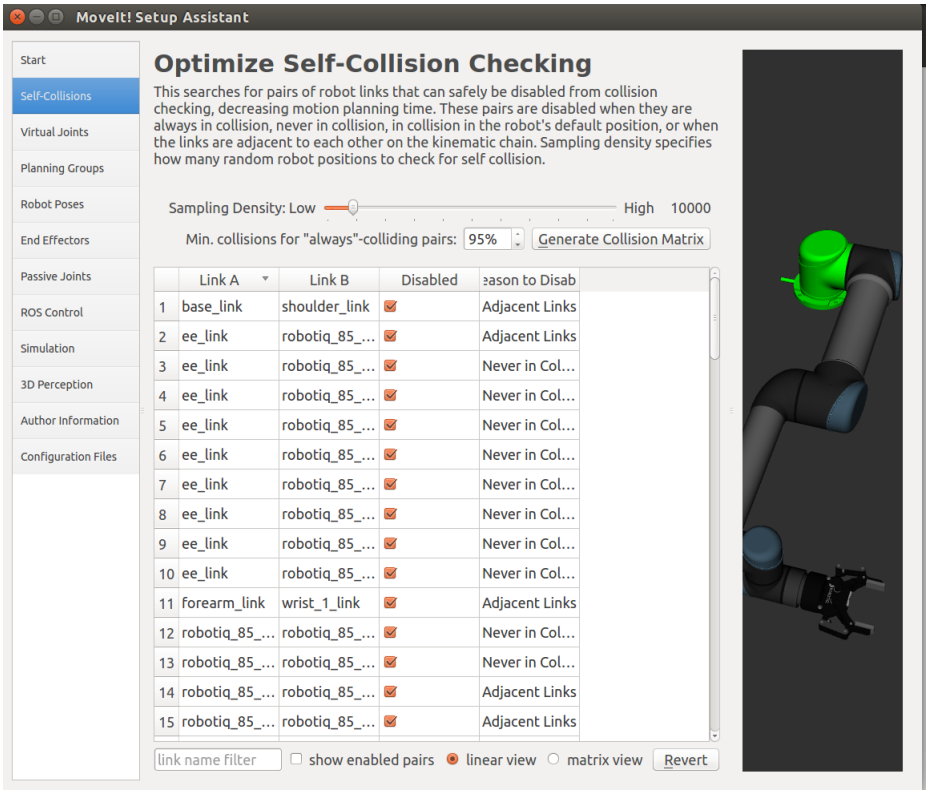


Figure 4.2: Self-Collision Checking

4. Next click “Virtual Joints” to create a virtual joint between the robot and the external frame of reference. Give a name for the virtual joint and give child link as the base link of the robot. Now give a parent frame name which is the base of the new frame of reference and define the joint type as “fixed”.

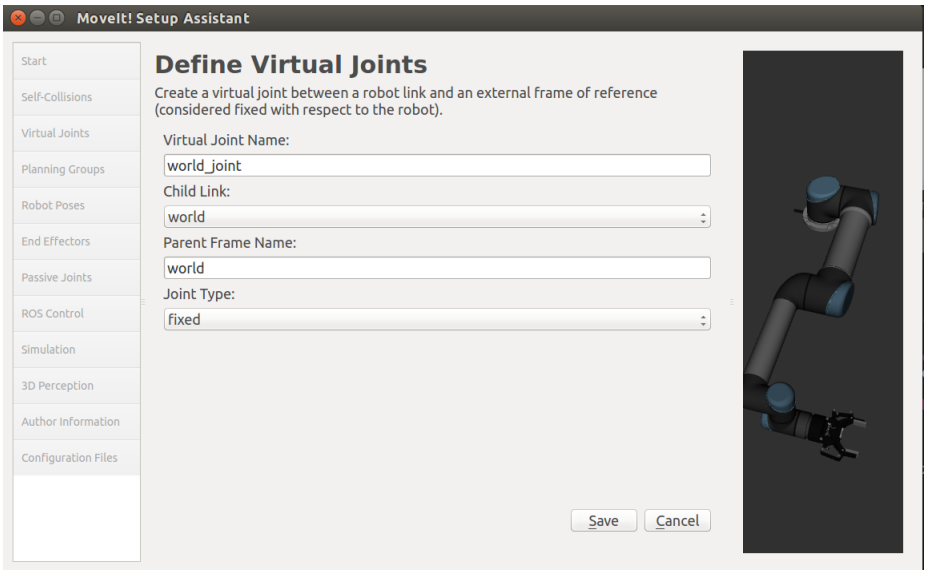


Figure 4.3: Defining Virtual Joints

5. Once the virtual joints are set, define the planning groups or a subset of joint-link pairs which are to be considered for planning and collision checking. Define the group for “arm”, select the kinematic solver as “KDLKinematicsPlugin”, with each kinematic resolution and timeout as 0.005 and solver attempts as 3. Choose a default planner for the group as “RRT”. Now add the joints which belong to the UR5 arm.

6. Similarly create the group to plan the motion of the Robotiq85 gripper. It is not necessary to define the kinematic solver and default planner for the gripper. Add the joints and save the groups.

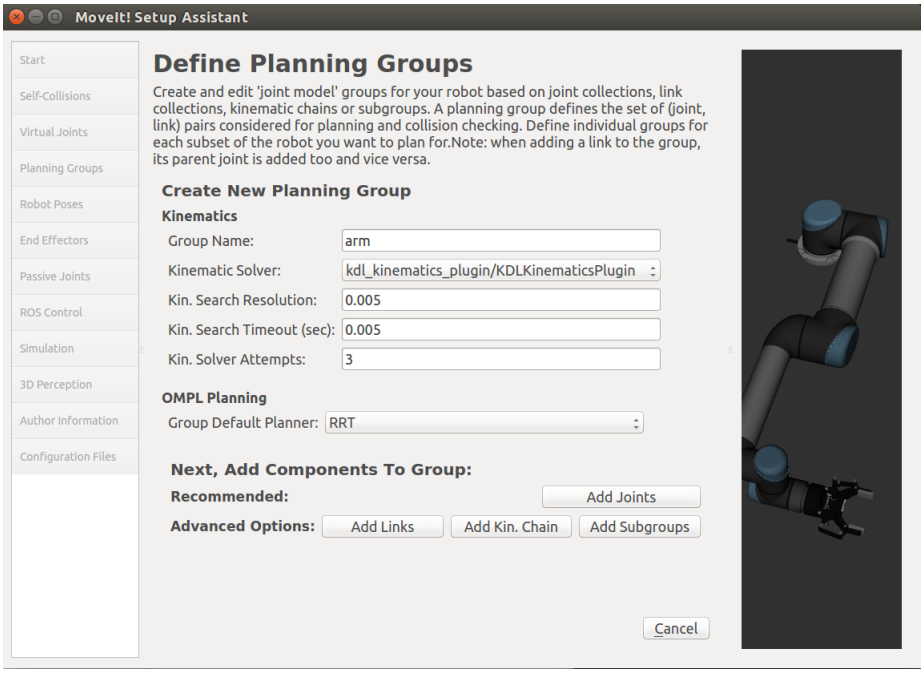


Figure 4.4: Defining Planning Groups

7. Next click "Robot Poses" to define the poses for the robot. These poses are sets of joint values for particular planning groups. Name the pose and set the values of the revolute joints to particular values (in radians) to set the desired pose. Save the pose.

```
Example: "rest_pose"
shoulder_pan_joint = 0.0000,
shoulder_lift_joint = -1.5708,
elbow_joint = 0.0000,
wrist_1_joint = -1.5708,
wrist_2_joint = 0.0000,
wrist_3_joint = -1.5708.
```

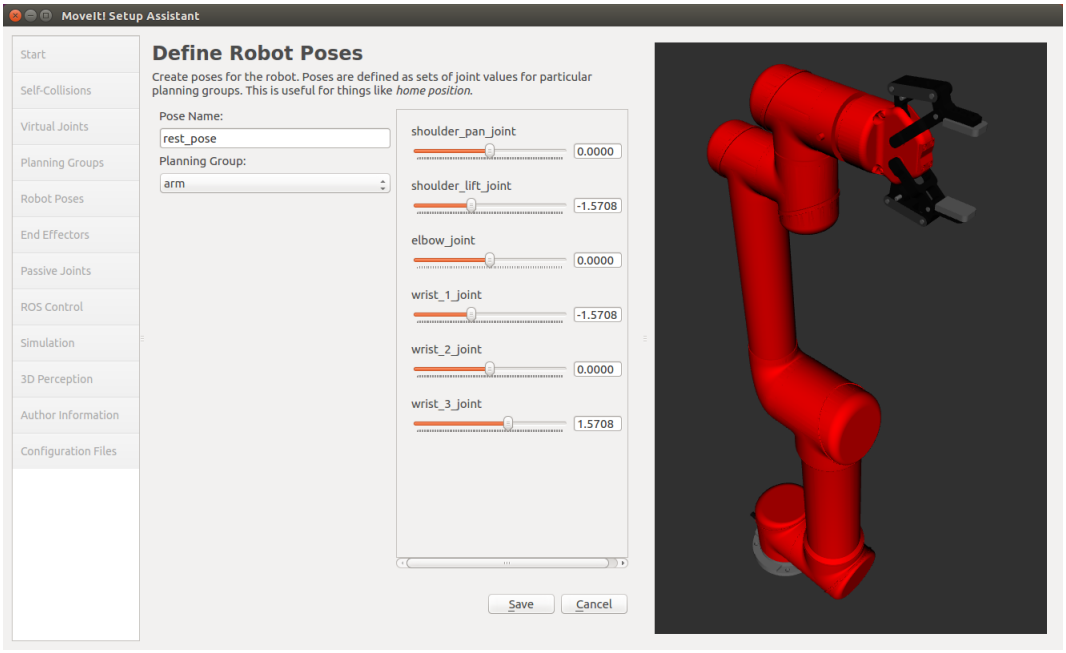


Figure 4.5: Defining Robot Poses

8. Next click "End Effectors" tab, create a name for the end effector which is the Roboitq85 gripper. Select the end-effector group related to the gripper, and select the parent link which is the base link which is attached to the gripper (i.e) "wrist_3_link" and the parent group as "arm", the group to which the gripper is attached.

9. Define Passive joints and ROS control and 3D perception if the robot involves camera etc.

10. Provide the author information as the Name of the editor and email address.

11. Generate the configuration files in the "src" of the newly created catkin workspace

    `~/ws_662/src/ur5_robotiq_arm`

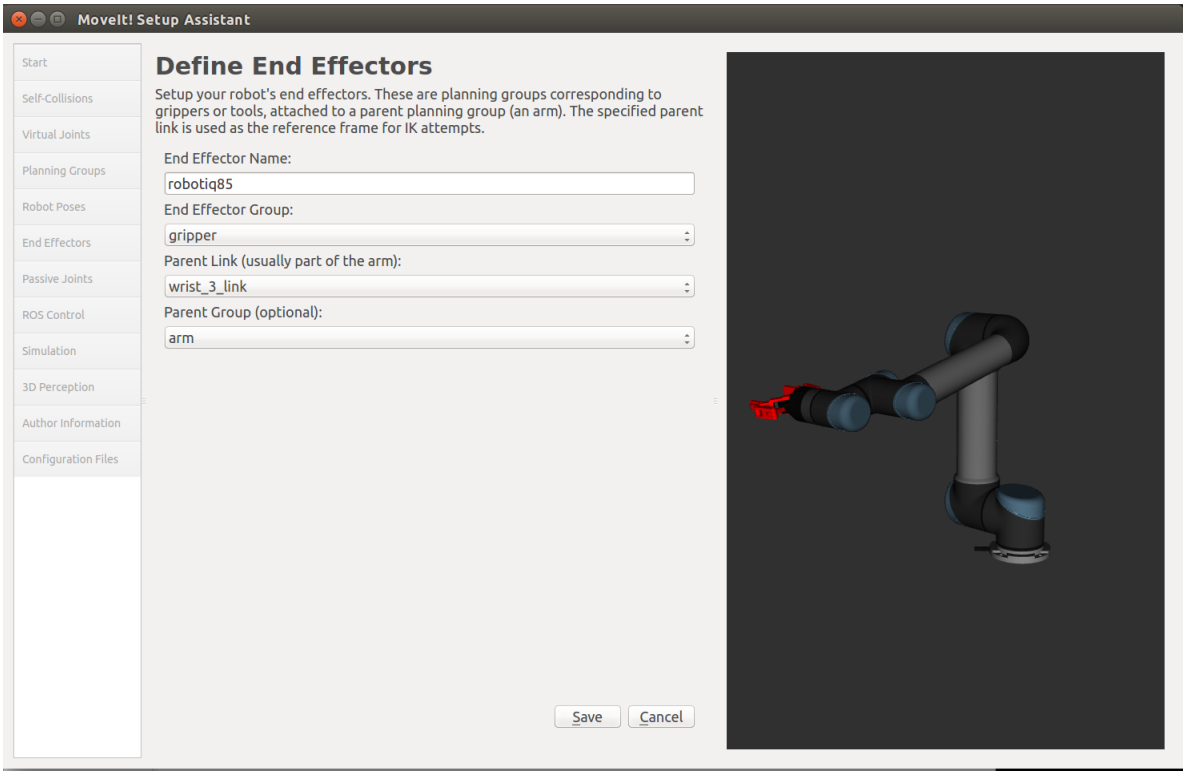


Figure 4.6: Defining end effector group and parent link

12. Click on "Generate Package" to generate the moveIt package followed by "Exit Setup Assistant" to exit the setupAssistant and launch the configuration. Note: The moveIt config package can be edited, by launching the moveIt setup assistant and selecting the "Edit Existing MoveIt Package". Make the required modifications and save the configuration files as the same package or a new package.

4.4 Creating a Controller File

Create a YAML configuration file called "controllers.yaml" file in the MoveIt config package under the ./config directory. This file will specify the controller configuration for your robot.

- *name*: The name of the controller.

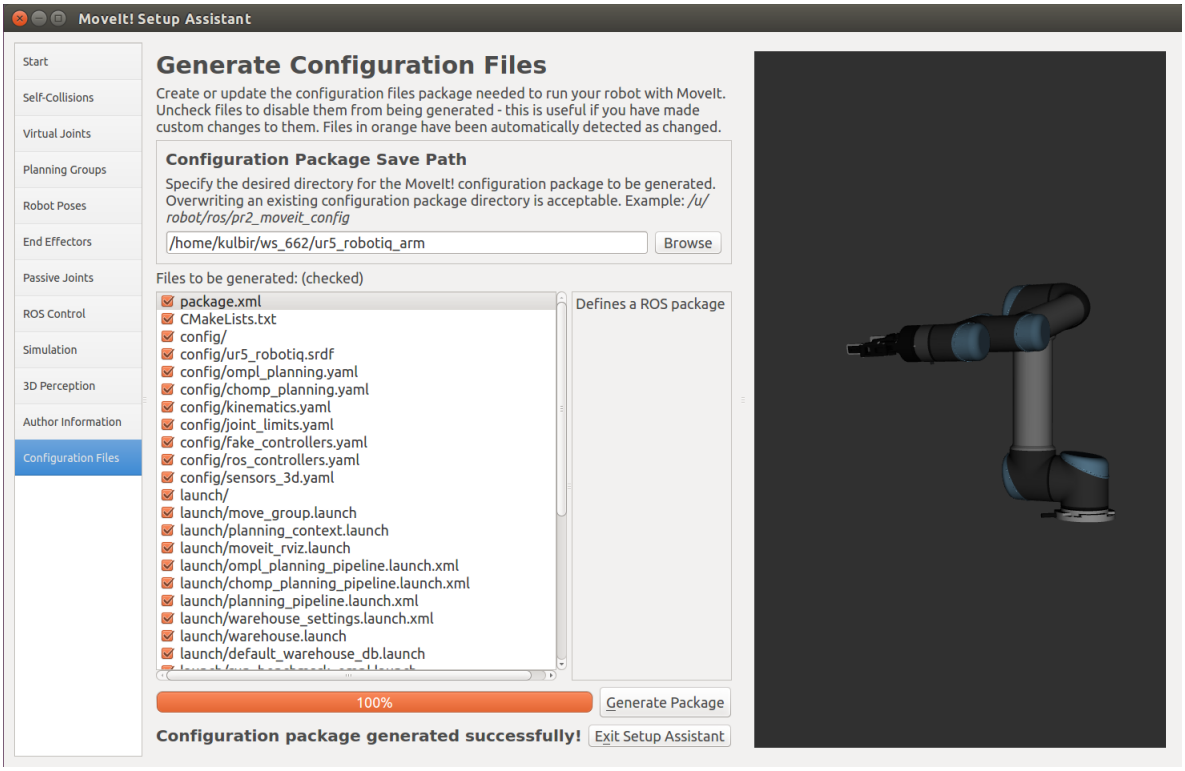


Figure 4.7: Generating Config files

- *action_ns*: The action namespace for the controller.
- *type*: The type of action being used
- *default*: The default controller is the primary controller chosen by MoveIt for communicating with a particular set of joints.
- *joints* : Names of all the joints that are being addressed by this interface.

4.5 Creating a launch File

1. Create the controller launch file (ur5_robotiq_arm_moveit_controller_manager.launch where the robot name needs to match the name specified when you created your MoveIt config directory). Add the following lines to this file

and save the file in the ./launch path of the config directory.

```
<launch>
<arg name="moveit_controller_manager"
default="moveit_simple_controller_manager/
MoveItSimpleControllerManager"/>
<param name="moveit_controller_manager"
value="$(arg moveit_controller_manager)"/>

<rosparam file="$(find ur5_robotiq_arm)/config/controllers.yaml"/>
</launch>
```

2. Now the system is ready to have MoveIt talk to your robot. To launch the files onto Rviz (robot visualiser) and Gazebo, create a new launch file to launch them simultaneously else they can also be launched in different terminals.

3. Run the new launch file and have the robot move, select the setting for configuration, motion planning selected. Also add the Robot to the Rviz and to interact move the robot in the predefined poses from motion planning tab.

4. Movement made in RViz replicates in Gazebo and hence the robot interact with the real world.

4.6 Planning in Rviz using a Moveit package

- We see that the initial pose is in grey while the desired robot pose is in solid orange colour as seen in figure 4.8. Note that we have checked "Collision aware IK" in Moveit so that the visualization in Rviz does not collide with itself or other objects.

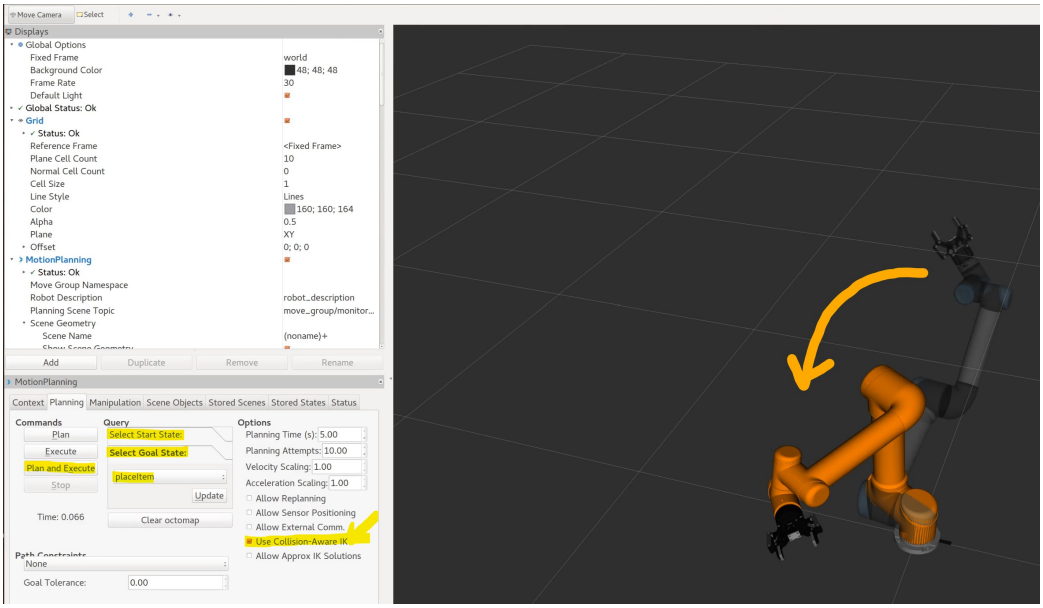


Figure 4.8: Moving the UR5 with Robotiq 85 gripper in Rviz using the KDL kinematics solver of Moveit

4.7 Github repository

- GitHub Repository - https://github.com/kulbir-ahluwalia/662-Final-Project-UR-5-arm
- The repository has all the files required to generate a Moveit package.
- The urdf.xacro file which is required by Moveit has the robotiq 85 gripper attached with the UR5 arm.
- There are three MATLAB code files with the extension ".m" used for validating the forward kinematics.
- The videos of the path planning in Rviz are attached.
- Zip folder of the workspace named "ws_662" is also uploaded with all the packages generated by Moveit.

4.8 Youtube videos

- Playlist of videos of the simulations in Rviz and validation on YouTube:- https://www.youtube.com/playlist?list=PLSaHroqSzdWR5xgQvIdhWTvgah-BKno
- The playlist has five videos as shown in the screenshot.

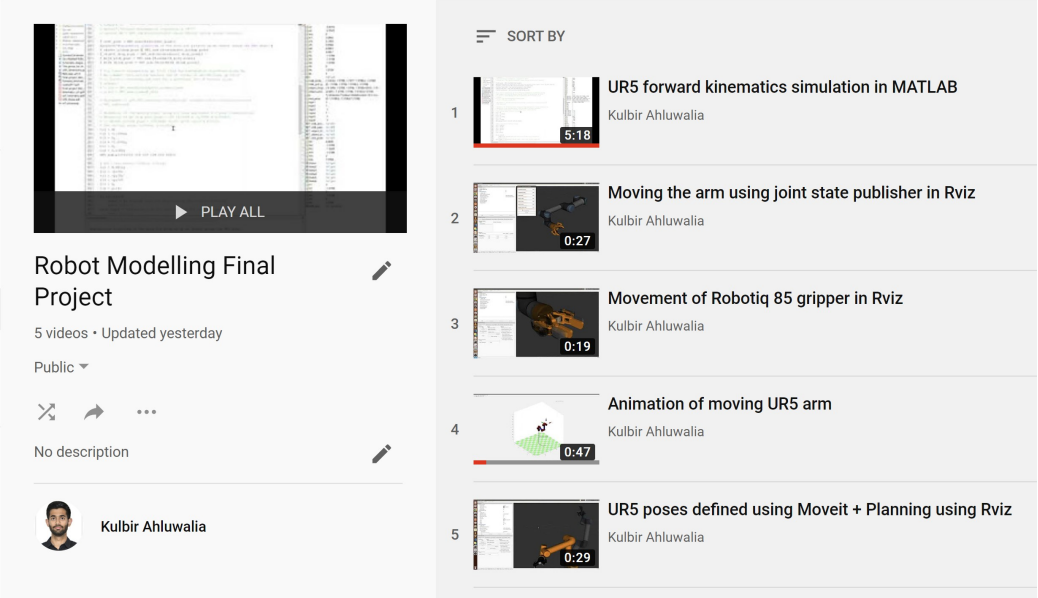


Figure 4.9: YouTube playlist with the explanations for MATLAB FK code and Rviz simulation videos with the joint state publisher, different poses and IK using the built in KDL kinematics solver in Moveit

Chapter 5

Validation

5.1 Using the Robotics toolbox in MATLAB

We have used the robotics toolbox developed by peter corke. It has various functions that aid in visualising and modelling robotic systems such as the plot command, function to generate the DH table and find the homogeneous transform for a robot pose using forward kinematics. [12]

Explanation of the MATLAB code for validation of Forward kinematics for the UR5 has been uploaded to YouTube on the channel "Kulbir Ahluwalia" and can be viewed at the link https://youtu.be/GjvW574I-Pc. The video of the moving UR5 arm can be seen at https://youtu.be/ISjVmBUFw6Y.

Some of the main functions used are:-

- $Object\_Name.plot(pose)$ - To plot a single pose of the robot. The robot pose is input as a row vector to the function.
- $Object\_Name.teach$ - To use GUI interface for physical insight and to configure poses.
- $Object\_Name.fkine(pose)$ - To obtain the homogeneous transform from the input pose using forward kinematics.

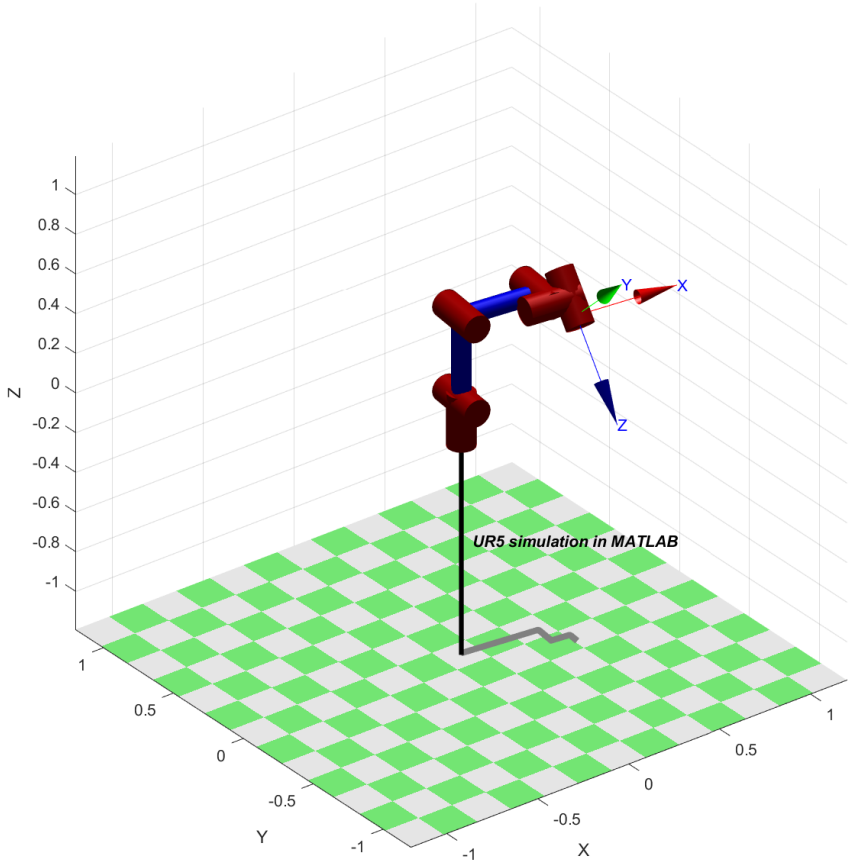


Figure 5.1: Pose of the UR5 as seen in MATLAB figure

5.2 Procedure for validating the Forward kinematics using MATLAB

- Find the homogeneous transform for a particular pose manually using forward kinematics.
- Use the Robotics toolbox to find the same homogeneous transform.
- The transforms obtained from the first and second approaches should be the same.

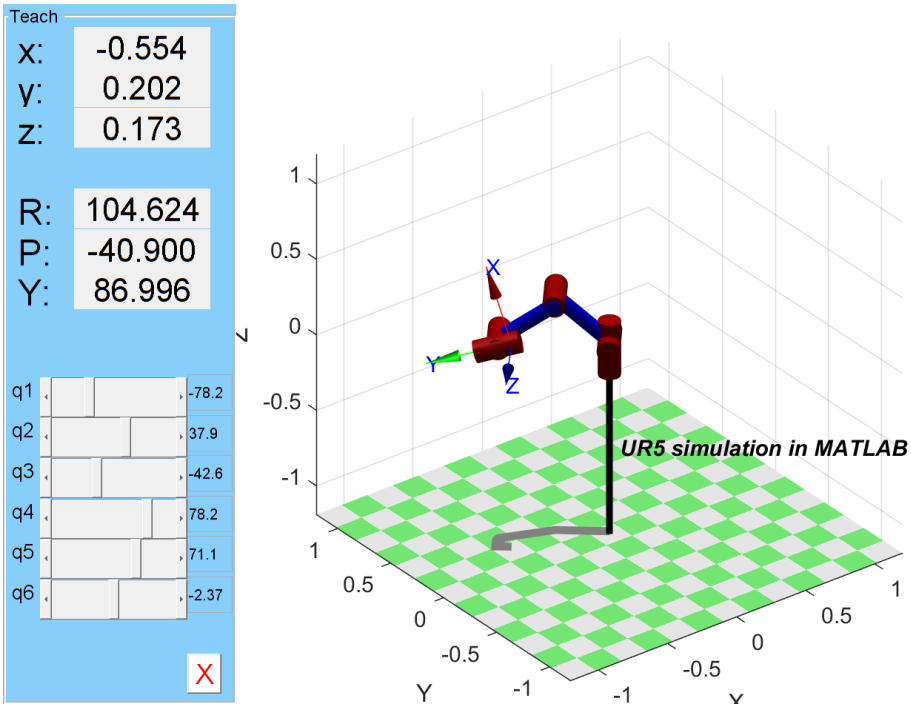


Figure 5.2: GUI interface using the teach function

5.3 Inverse kinematics of the UR5 arm in Matlab vs Moveit

For inverse kinematics, we first find the homogeneous transform using FK and then input the obtained pose to the ikine function:-

- $T\_inv = UR5\_arm.fkine(object\_pickup\_pose)$
- $q\_inv = UR5\_arm.ikine(T\_inv)$

We comment this section in the MATLAB code because the IK solver of matlab gives an error. It fails to converge and asks for a different set of initial joint values. This is because the 6 DOF arm is too complex for it and it exceeds the max number of iterations allowed for the solver. (A max of 100 iterations)

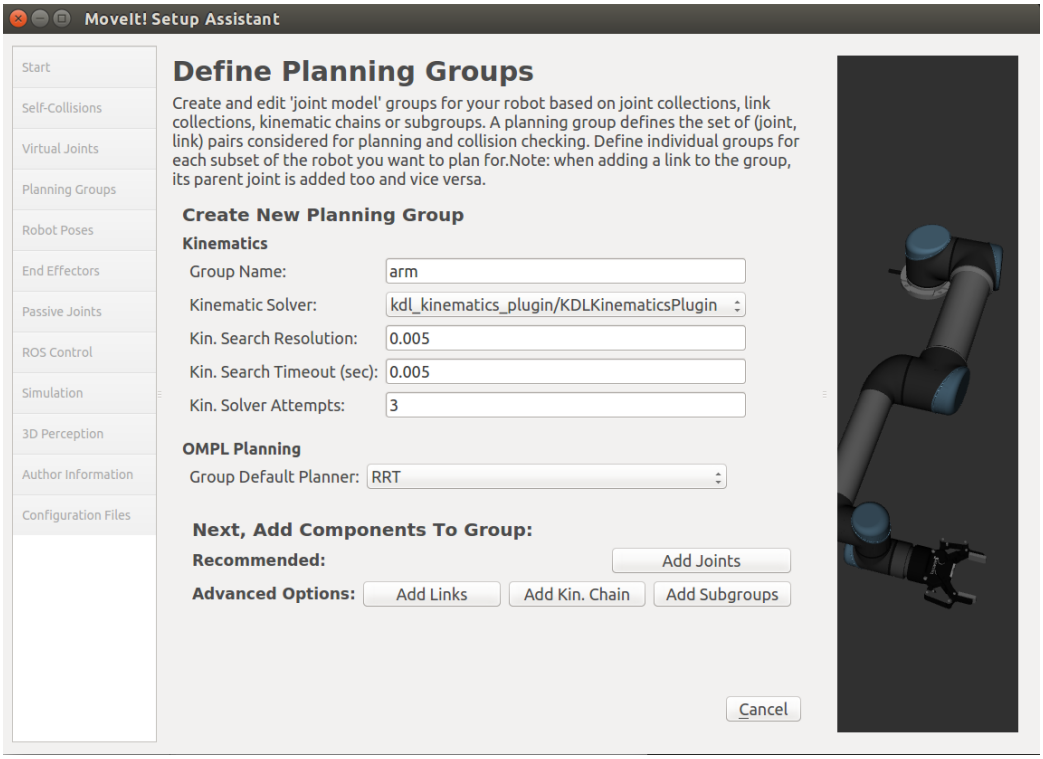


Figure 5.3: Using the KDL kinematics plugin in Moveit for moving the end effector using IK

The inverse kinematics for the UR5 can be found out manually using geometry as seen in the PDF "Kinematics of a UR5" [13]. We have not implemented the IK because it had errors while being simulated in MATLAB. However, the KDL kinematics plugin in Moveit is used in our case to move the robot using Inverse kinematics.

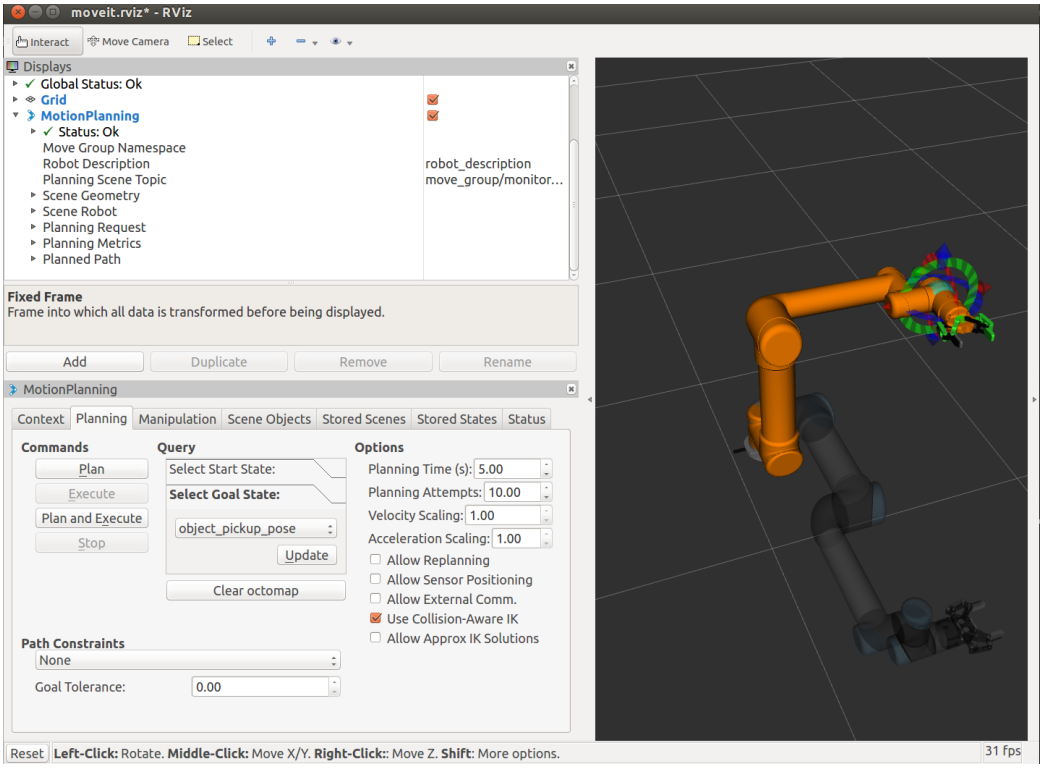


Figure 5.4: The sphere on the end effector can be used to give the desired position in Moveit for moving the end effector using IK

When we move the gripper in the simulation by clicking on the sphere shown in the figure 5.4, we can place the end effector anywhere in the reachable workspace and Moveit gets there using IK.

FORWARD KINEMATICS FOR UR5

⇒ PROCEDURE :—

① Make the frame diagram for UR5.

② Write the DH table

③ Find A1, A2, A3, A4, A5, A6 & multiply them to get T_6^0.

④ T_6^0 means the transformation matrix from frame 6 to frame 0 and is of the form :—

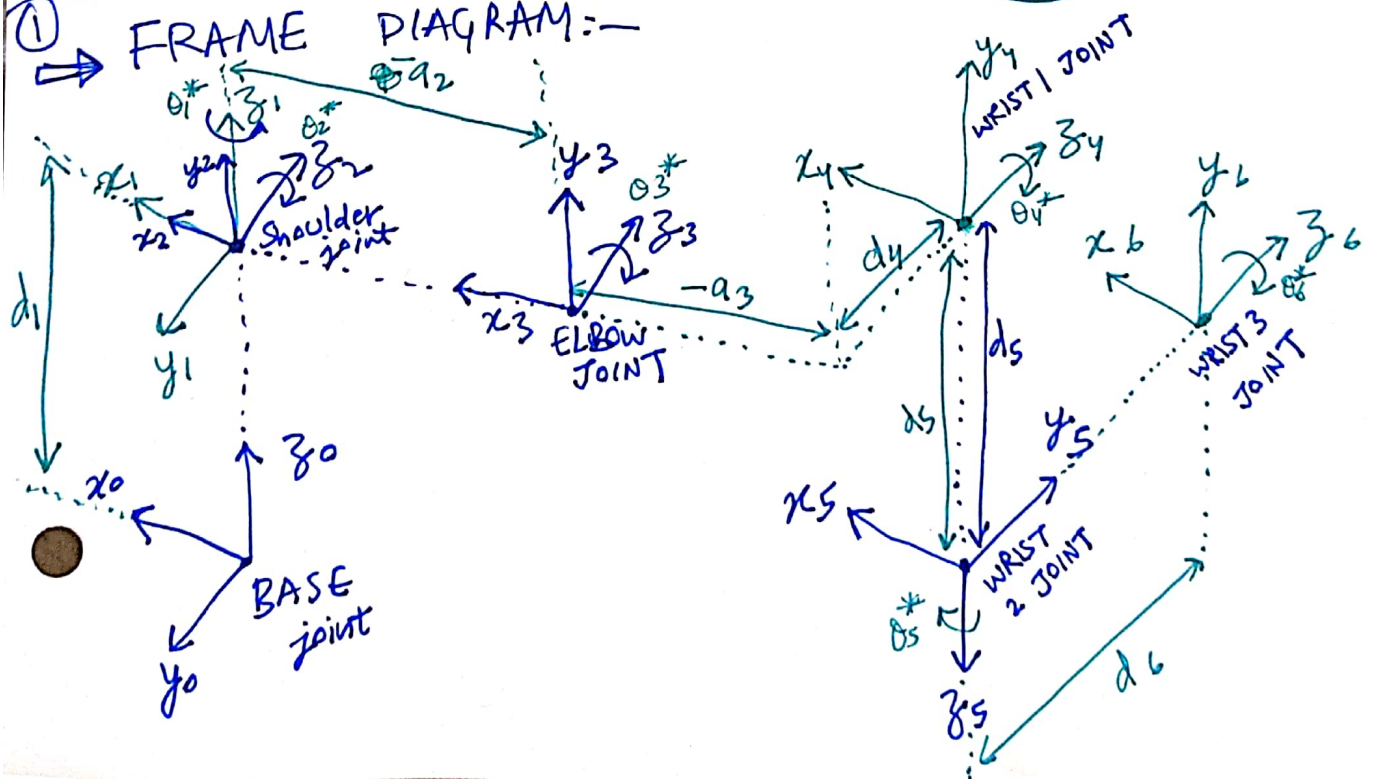
ORIENTATION, POSITION

$$T_6^0(\theta_1^*, \theta_2^*, \theta_3^*, \theta_4^*, \theta_5^*, \theta_6^*) = \begin{bmatrix} R_6^0 & \vdots & P_6^0 \\ \cdots & \vdots & \cdots \\ 0 & \vdots & 1 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

FUNCTION of the ~~control~~ joint variables

$$T_6^0 = A1^* A_2^* A_3^* A_4^* A_5^* A_6$$

① ⇒ FRAME DIAGRAM :—



(2) DH Table :-

TRANSFORM	$z_{\theta_{i-1}}$	$z_{d_{i-1}}$	x_{a_i}	x_{α_i}
$0 \rightarrow 1$	θ_1^*	d_1	0	0
$1 \rightarrow 2$	θ_2^*	0	0	$\pi/2$
$2 \rightarrow 3$	θ_3^*	0	$+a_2$	0
$3 \rightarrow 4$	θ_4^*	d_4	$+a_3$	0
$4 \rightarrow 5$	θ_5^*	d_5	0	$\pi/2$
$5 \rightarrow 6$	θ_6^*	d_6	0	$-\pi/2$

⇒ A1, A2, A3, A4, A5, A6 are found in MATLAB. (Code published & attatched)

⇒ We find $T_6^0 = A1 * A2 * A3 * A4 * A5 * A6$.

→ Then, for Forward Kinematics, we put in values of Theta1, Theta2 ..... Theta6 in T_6^0.

→ Shown in MATLAB. (Code Published is attatched).

(3)

→ For example, we do the FK for object pickup pose.

```
% MATLAB code to create a model of Universal Robotics UR5 6 DOF
 manipulator
% This file uses the function for UR5 from The Robotics Toolbox for
 MATLAB (RTB).
clc; clear; %clear the command window and the workspace
% Use run to load the Robotics Toolbox into Matlab's path
run('C:\Robotics Toolbox Matlab\robot-10.3.1\rvctools\startup_rvc.m')
% Use symbolic variables to define joint variables.
syms theta1 theta2 theta3 theta4 theta5 theta6

% Define variables for link lengths
%SI units of metres for length and radians for angle are used.
% a2 and a3 are taken to be negative because the poses were
% defined accordingly in moveit and then the DH table was made
d1 = 0.08916; a2 = -0.425; a3 = -0.39225;
d4 = 0.10915; d5 = 0.09456; d6 = 0.0823;
% Defining poses
rest_pose = ([0 -1.5708 0 -1.5708 0 1.5708]); % rest pose
object_pickup_pose = ([4.6851 -pi/2 -pi/2 -pi/2 0 pi/2]);
object_drop_pose = ([-0.1090 -pi/2 -pi/2 -pi/2 0.8353 -pi/2]);
milk_pick_pose = ([0 -pi/2 -pi/2 -pi/2 0 -pi/2]);
milk_drink_pose = ([-3.8182 -pi/2 -1.7977 -pi/2 0 -pi/2]);

% Enter the DH parameters as a vector, in case your joint is
 prismatic, use Link([....],'p')
% By default, the joint is revolute.
% L(1) means Link 1
L(1) = Link([0  d1   0   0]);
%L(1).qlim = [0 pi];  %In case you want to set limits on the link
 motion
L(2) = Link([0  0   0    pi/2]);
L(3) = Link([0  0   a2    0]);
L(4) = Link([0  d4   a3   0]);
L(5) = Link([0  d5   0  pi/2]);
L(6) = Link([0  d6   0    -pi/2]);

% Connect all links of the robot using the command serial link.
% Pass the vector L to serial link. We have the object "UR5_662" on
 the LHS.
UR5_arm = SerialLink(L);
% In order to name this robot, we use:-
UR5_arm.name = 'UR5 simulation in MATLAB';
% To see the DH table of the arm in the command window:-
fprintf('DH table of the Universal Robot 5 arm:- \n')
UR5_arm

% To plot a single pose of the robot, uncomment and use:-
% UR5_arm.plot(object_drop_pose)
% UR5_arm.plot([0 0 0 0 0 0])
% UR5_arm.plot(object_pickup_pose)

% To save images to a folder
```

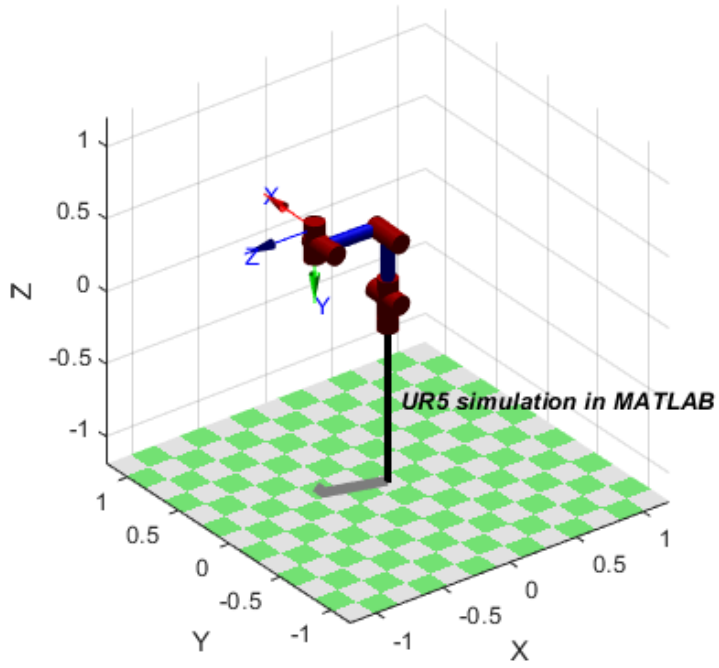
```
%saveas(gcf,'C:\Robotics Toolbox Matlab\ur5_imgs
\ur5_initial_pose.png')
%saveas(gcf,'C:\Robotics Toolbox Matlab\ur5_imgs
\ur5_object_pickup_pose.png')

% For the general relation: -
% Uncomment General_FK if you need it because it takes a long time to
% compute it
% fprintf('General Homogeneous transform of UR5')
% General_FK = UR5_arm.fkine([theta1 theta2 theta3 theta4 theta5
 theta6])

T_rest_pose = UR5_arm.fkine(rest_pose);
fprintf('Homogeneous transform of the pose for picking up an object
 using the UR5 arm:-')
T_object_pickup_pose = UR5_arm.fkine(object_pickup_pose)
T_object_drop_pose = UR5_arm.fkine(object_drop_pose);
T_milk_pick_pose = UR5_arm.fkine(milk_pick_pose);
T_milk_drink_pose = UR5_arm.fkine(milk_drink_pose);

% For inverse kinematics, we first find the homogeneous transform
 using fk,
% We comment this section because the IK solver of matlab gives an
 error
% It fails to converge and asks for a different set of initial joint
% values.
% T_inv = UR5_arm.fkine(object_pickup_pose)
% q_inv = UR5_arm.ikine(T_inv)

% Uncomment to use GUI interface for physical insight and to configure
 poses:-
% UR5_arm.teach

% Animation of the moving robot using for loop and pause for easy
 visualization
% Animation to go from rest_pose = ([0 -1.5708 0 -1.5708 0 1.5708])
% to object_pickup_pose = ([4.6851 -pi/2 -pi/2 -pi/2 0 pi/2])
% Set initial pose, t=theta, i=initial
ti1 = 0;
ti2 = -1.5708;
ti3 = 0;
ti4 = -1.5708;
ti5 = 0;
ti6 = 1.5708;
UR5_arm.plot([ti1 ti2 ti3 ti4 ti5 ti6])

% Set final pose, t=theta, f=final
fi1 = 4.6851;
fi2 = -pi/2;
fi3 = -pi/2;
fi4 = -pi/2;
fi5 = 0;
fi6 = pi/2;
```

```
if ti1-fi1<0
    sign1 = 1; %change sign for step size of for loop to positive
    %when fi1>ti1
else sign1 = -1;%change sign for step size of for loop to negative
    %when fi1<ti1
end

if ti2-fi2<0
    sign2 = 1; %change sign for step size of for loop to positive
    %when fi2>ti2
else sign2 = -1;%change sign for step size of for loop to negative
    %when fi2<ti2
end

if ti3-fi3<0
    sign3 = 1; %change sign for step size of for loop to positive
    %when fi3>ti3
else sign3 = -1;%change sign for step size of for loop to negative
    %when fi3<ti3
end

if ti4-fi4<0
    sign4 = 1; %change sign for step size of for loop to positive
    %when fi4>ti4
else sign4 = -1;%change sign for step size of for loop to negative
    %when fi4<ti4
end

if ti5-fi5<0
    sign5 = 1; %change sign for step size of for loop to positive
    %when fi5>ti5
else sign5 = -1;%change sign for step size of for loop to negative
    %when fi5<ti5
end

if ti6-fi6<0
    sign6 = 1; %change sign for step size of for loop to positive
    %when fi6>ti6
else sign6 = -1;%change sign for step size of for loop to negative
    %when fi6<ti6
end


% Using for loops to go from initial pose to final pose
% The animation can be made smoother by decreasing the step size in
 the for
% loop. The sign of the step size has to be changed according to
 situation
for th1 = ti1: sign1*0.06: fi1
    UR5_arm.plot([th1 ti2 ti3 ti4 ti5 ti6]);
    pause(0.25)
end
pause(0.25)
for th2 = ti2: sign2*0.06: fi2
```

```
    UR5_arm.plot([fi1 th2 ti3 ti4 ti5 ti6]);
    pause(0.1)
end
pause(0.25)
for th3 = ti3: sign3*0.06: fi3
    UR5_arm.plot([fi1 fi2 th3 ti4 ti5 ti6]);
    pause(0.1)
end
pause(0.25)
for th4 = ti4:  sign4*0.06: fi4
    UR5_arm.plot([fi1 fi2 fi3 th4 ti5 ti6]);
    pause(0.1)
end
pause(0.25)
for th5 = ti5: sign5*0.06: fi5
    UR5_arm.plot([fi1 fi2 fi3 fi4 th5 ti6]);
    pause(0.1)
end
pause(0.25)
for th6 = ti6: sign6*0.06: fi6
    UR5_arm.plot([fi1 fi2 fi3 fi4 fi5 th6]);
    pause(0.1)
end
```

```
Robotics, Vision & Control: (c) Peter Corke 1992-2017 http://
www.petercorke.com
- Robotics Toolbox for MATLAB (release 10.3.1)
 - ARTE contributed code: 3D models for robot manipulators (C:
\Robotics Toolbox Matlab\robot-10.3.1\rvctools\robot\data\ARTE)
 - pHRIWARE (release 1.2): pHRIWARE is Copyrighted by Bryan Moutrie
 (2013-2019) (c)
DH table of the Universal Robot 5 arm:-

UR5_arm =

UR5 simulation in MATLAB:: 6 axis, RRRRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|    0.08916|          0|          0|          0|
|  2|         q2|          0|          0|     1.5708|          0|
|  3|         q3|          0|     -0.425|          0|          0|
|  4|         q4|    0.10915|   -0.39225|          0|          0|
|  5|         q5|    0.09456|          0|     1.5708|          0|
|  6|         q6|     0.0823|          0|    -1.5708|          0|
+---+-----------+-----------+-----------+-----------+-----------+

Homogeneous transform of the pose for picking up an object using the
 UR5 arm:-

T_object_pickup_pose =
    0.0273         0   -0.9996   -0.3865
    0.9996         0    0.0273    0.2143
         0        -1         0    0.5965
```

```
    0         0         0         1
```



Published with MATLAB® R2019a

```
%the code is run for finding every Ai matrix after making the
 necessary changes to theta, a ,d and alpha
%A1, A2, A3... are stored in the workspace as we find each Ai
%then in the end, we just find the final T by multiplying all Ai

syms  alpha d a theta
syms theta1 theta2 theta3 theta4 theta5 theta6
syms d4 d5 d6 d1 a2 a3   %to create symbolic variables

rot_z = [cos(theta) -sin(theta) 0 0; sin(theta) cos(theta) 0 0; 0 0 1
 0; 0 0 0 1];      %Initialising homogeneous transforms
trans_z = [1 0 0 0; 0 1 0 0; 0 0 1 d; 0 0 0 1];
trans_x = [1 0 0 a; 0 1 0 0; 0 0 1 0; 0 0 0 1];
rot_x = [1 0 0 0; 0 cos(alpha) -sin(alpha) 0; 0 sin(alpha) cos(alpha)
 0; 0 0 0 1];
final = (rot_z*trans_z*trans_x*rot_x);

theta = theta6;
% theta5 = 0;
% theta6 = deg2rad(theta5);
% %theta =  theta6+theta3;
% theta =  theta6+0;

d = d6;
a = 0;

alpha = -90;
alpha = deg2rad(alpha);

final = subs(final); %updates values of variables
A6 = simplify(final);
%disp(A6);

% Following code can be uncommented when we have all Ais and want to
 see
% them and calculate T
% A1  %to see the final values of Ai matrices
% A2
% A3
% A4
% A5
% A6

% T = simplify(A1*A2*A3*A4*A5*A6)
```

Published with MATLAB® R2019a

```
% We have T from previous code, we use it here to find the homogeneous
% transform by substituting the values of theta1, theta2, theta3,
 theta4, theta5 and theta6
% We solve the Forward kinematics for object_pickup_pose = ([4.6851 -
pi/2 -pi/2 -pi/2 0 pi/2])

clc; clear; %clear the command window and the workspace

% Define variables for link lengths
% SI units of metres for length and radians for angle are used.
d1 = 0.08916; a2 = -0.425; a3 = -0.39225;
d4 = 0.10915; d5 = 0.09456; d6 = 0.0823;

% Substituting the vales of all the joint variables:-
theta1 = 4.6851;
theta2 = -pi/2;
theta3 = -pi/2;
theta4 = -pi/2;
theta5 = 0;
theta6 = pi/2;

% Values of A1 to A6 stored here from previous calculations
% so that they can be used when the workspace is cleared

 A1 =[ cos(theta1), -sin(theta1), 0,  0;
 sin(theta1),  cos(theta1), 0,  0;
           0,            0, 1, d1;
           0,            0, 0,  1]


A2 =[ cos(theta2), 0,  sin(theta2), 0;
 sin(theta2), 0, -cos(theta2), 0;
           0, 1,            0, 0;
           0, 0,            0, 1]


A3 =[ cos(theta3), -sin(theta3), 0, a2*cos(theta3);
 sin(theta3),  cos(theta3), 0, a2*sin(theta3);
           0,            0, 1,              0;
           0,            0, 0,              1]


A4 =[ cos(theta4), -sin(theta4), 0, a3*cos(theta4);
 sin(theta4),  cos(theta4), 0, a3*sin(theta4);
           0,            0, 1,             d4;
           0,            0, 0,              1]


A5 =[ cos(theta5), 0,  sin(theta5),  0;
 sin(theta5), 0, -cos(theta5),  0;
           0, 1,            0, d5;
           0, 0,            0,  1]
```

```
A6 =[ cos(theta6),  0, -sin(theta6),  0;
 sin(theta6),  0,  cos(theta6),  0;
          0, -1,            0, d6;
           0,  0,            0,  1]


% All the values are substituted in the general value of T from
 previous code
T = [ sin(theta1 + theta2)*sin(theta6) + cos(theta6)*(cos(theta1
 + theta2)*cos(theta3 + theta4)*cos(theta5) - cos(theta1 +
 theta2)*sin(theta3 + theta4)*sin(theta5)), -sin(theta3 + theta4 +
 theta5)*cos(theta1 + theta2),   sin(theta1 + theta2)*cos(theta6) -
 sin(theta6)*(cos(theta1 + theta2)*cos(theta3 + theta4)*cos(theta5)
 - cos(theta1 + theta2)*sin(theta3 + theta4)*sin(theta5)),
 d6*(cos(theta1 + theta2)*cos(theta3 + theta4)*sin(theta5)
 + cos(theta1 + theta2)*sin(theta3 + theta4)*cos(theta5)) +
 d4*sin(theta1 + theta2) + d5*sin(theta1 + theta2) + a3*cos(theta1 +
 theta2)*cos(theta3 + theta4) + a2*cos(theta1 + theta2)*cos(theta3);
 cos(theta6)*(cos(theta3 + theta4)*sin(theta1 + theta2)*cos(theta5) -
 sin(theta1 + theta2)*sin(theta3 + theta4)*sin(theta5)) - cos(theta1
 + theta2)*sin(theta6), -sin(theta3 + theta4 + theta5)*sin(theta1
 + theta2), - sin(theta6)*(cos(theta3 + theta4)*sin(theta1
 + theta2)*cos(theta5) - sin(theta1 + theta2)*sin(theta3 +
 theta4)*sin(theta5)) - cos(theta1 + theta2)*cos(theta6),
 d6*(cos(theta3 + theta4)*sin(theta1 + theta2)*sin(theta5)
 + sin(theta1 + theta2)*sin(theta3 + theta4)*cos(theta5)) -
 d5*cos(theta1 + theta2) - d4*cos(theta1 + theta2) + a3*cos(theta3 +
 theta4)*sin(theta1 + theta2) + a2*sin(theta1 + theta2)*cos(theta3);

                                                    sin(theta3 + theta4
 + theta5)*cos(theta6),                        cos(theta3 + theta4 +
 theta5),
                                                                 -sin(theta3
 + theta4 + theta5)*sin(theta6),

                                                                       d1
 + a3*sin(theta3 + theta4) + a2*sin(theta3) - d6*cos(theta3 + theta4 +
 theta5);


                  0,
 0,

                       0,


                                                                      1]


fprintf('We see that we get the same value of the homogeneous
 transform for the object pickup pose from both approaches \n')
```

```
fprintf('Hence, FK is validated for UR5 arm using MATLAB robotics
 toolbox')
```

```
A1 =

   -0.0273    0.9996         0         0
   -0.9996   -0.0273         0         0
         0         0    1.0000    0.0892
         0         0         0    1.0000


A2 =

    0.0000         0   -1.0000         0
   -1.0000         0   -0.0000         0
         0    1.0000         0         0
         0         0         0    1.0000


A3 =

    0.0000    1.0000         0   -0.0000
   -1.0000    0.0000         0    0.4250
         0         0    1.0000         0
         0         0         0    1.0000


A4 =

    0.0000    1.0000         0   -0.0000
   -1.0000    0.0000         0    0.3922
         0         0    1.0000    0.1091
         0         0         0    1.0000


A5 =

    1.0000         0         0         0
         0         0   -1.0000         0
         0    1.0000         0    0.0946
         0         0         0    1.0000


A6 =

    0.0000         0   -1.0000         0
    1.0000         0    0.0000         0
         0   -1.0000         0    0.0823
         0         0         0    1.0000


T =
```

```
    0.0273   -0.0000   -0.9996   -0.3865
    0.9996    0.0000    0.0273    0.2143
   -0.0000   -1.0000    0.0000    0.5965
         0         0         0    1.0000
```

```
We see that we get the same value of the homogeneous transform for the
 object pickup pose from both approaches
Hence, FK is validated for UR5 arm using MATLAB robotics toolbox
```

Published with MATLAB® R2019a

Chapter 6

Future work

This project is very interesting and has the potential to improve countless lives around the globe. We can work on the following aspects in the future:-

- We plan to simulate the arm and the gripper in **Gazebo** using **Rviz** and **Moveit**.
- The arm joints need PID tuning. We can set the values of the gains as it is important to prevent jerky movements.
- Set constraints on the workspace for user safety.
- Devise control algorithms for detecting collisions. For example, upon detecting an impact of 200 newtons, the arm should go into safe mode and stop exerting any force other than that required to prevent it from collapsing due to gravity.

Bibliography

[1] The World Bank - Disability inclusion
https://www.worldbank.org/en/topic/disability

[2] UR5 Manual Document
https://www.usna.edu/Users/weaprcon/kutzer/_files/documents/User%20Manual,%20UR5.pdf

[3] Consorcio
https://www.ci4-0.com/es/

[4] UR Robots Dynamics
https://www.universal-robots.com/how-tos-and-faqs/faq/ur-faq/parameters-for-calculations-of-kinematics-and-dynamics-45257/

[5] Robotiq 85 Gripper - Documentation
robotiq.com/website-assets/support_documents/document/2F-85_2F-140_Instruction_Manual_CB_PDF_20191018.pdf

[6] Install Ubuntu
http://wiki.ros.org/kinetic/Installation/Ubuntu

[7] ROS Workspace Create
http://wiki.ros.org/catkin/Tutorials/create_a_workspace

[8] ROS Documentation
https://moveit.ros.org/documentation/

[9] MoveIT Tutorials
http://docs.ros.org/indigo/api/pr2_moveit_tutorials/html/planning/src/doc/controller_configuration.html

[10] Robot Modeling and Control.
1st Edition by Mark W. Spong(Author), Seth Hutchinson(Author), M. Vidyasagar(Author)

[11] ROS Introduction
http://wiki.ros.org/ROS/Introduction

[12] Robotics, Vision and Control Fundamental Algorithms in MATLAB® by Peter Corke, DOI 10.1007/978-3-319-54413-7

[13] UR5 Kinematics Doc
http://rasmusan.blog.aau.dk/files/ur5_kinematics.pdf

Appendix

Code for the simulation is included here which includes: -

- Code for the URDF file that was written to combine the UR5 arm and the Robotiq 85 gripper. This urdf.xacro file was used for generating the package using Moveit.
- Code for the Controllers.yaml file.
- Code for the Launch file named after the robot package.
- Code for the Launch file to start the robot in Rviz or Gazebo.

CODE FOR THE SIMULATION

Code for the URDF file which is used for generating the robot model.

Path: Workspace/src/ur5_robotiq_arm.urdf.xacro

```
<?xml version="1.0"?>

<robot xmlns:xacro="http://wiki.ros.org/xacro"
name="ur5_robotiq" >



<xacro:arg name="transmission_hw_interface"
default="hardware_interface/PositionJointInterface"/>



<!-- ur5 -->

<xacro:include filename="$(find
ur_description)/urdf/ur5.urdf.xacro" />



<!-- arm -->

<xacro:ur5_robot prefix="" joint_limited="false"

transmission_hw_interface="$(arg transmission_hw_interface)"

/>

<link name="world" />

<gazebo>

<plugin name="gazebo_ros_control"
filename="libgazebo_ros_control.so">

<robotNamespace>ur5_robotiq</robotNamespace>

<robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimTy
pe>

<legacyModeNS>true</legacyModeNS>
```

```
</plugin>

</gazebo>


<joint name="world_joint" type="fixed">

<parent link="world" />

<child link = "base_link" />

<origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0" />

</joint>


<xacro:include filename="$(find
robotiq_description)/urdf/robotiq_85_gripper.urdf.xacro" />


<xacro:robotiq_85_gripper prefix="" parent="ee_link" >

<origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>

</xacro:robotiq_85_gripper>

</robot>
```

Code for the Controllers.yaml file.

Path: moveIt_package/config/Controllers.yaml

```
controller_list:

- name: grip_controller

action_ns: grripper_action

default: True

type: GripperCommand
```

```
joints:

- robotiq_85_left_inner_knuckle_joint

- name: ur5_controller

action_ns: follow_joint_trajectory

default: True

type: FollowJointTrajectory

joints:

- shoulder_pan_joint

- shoulder_lift_joint

- elbow_joint

- wrist_1_joint

- wrist_2_joint

- wrist_3_joint
```

Code for the Launch file named after the robot package.

Path:package/launch/ur5_robotiq_moveit_controller_manager.launch.xml

```
<launch>

<!-- loads moveit_controller_manager on the parameter server
which is taken as argument if no argument is passed,
moveit_simple_controller_manager will be set -->

<arg name="moveit_controller_manager"
default="moveit_simple_controller_manager/MoveItSimpleControll
erManager" />

<param name="moveit_controller_manager" value="$(arg
moveit_controller_manager)"/>
```

```
<!-- if the controller manager is running and we can talk to
it but need to know the name -->

<arg name="controller_manager_name"
default="simple_contoller_manager" />

<param name="controller_manager_name" value="$(arg
controller_manager_name" />



<!-- flag indicating whether teh controller manager should be
used or not -->

<arg name="use_controller_manager" default="true" />

<param name="use_controller_manager" value="$(arg
use_controller_manager" />



<!-- loads ros_controllers to the param server -->

<rosparam file="$(find
ur5_robotiq_arm)/config/controllers.yaml"/> <!--
ros_controllers -->

</launch>
```

Code for the Launch file to start the robot in Rviz or Gazebo.

<u>Path:</u> moveIt_package/launch/start_ur5_robotiq_arm.launch.xml

```
<launch>

     <arg name="config" default="true"/>

     <arg name="rviz_config" default="$(find
ur5_robotiq_arm)/launch/moveit.rviz"/>
```

```
     <include file="$(find
ur5_robotiq_arm)/launch/planning_context.launch">

          <arg name="load_robot_description" value="false"/>

     </include>



     <include file="$(find
ur5_robotiq_arm)/launch/move_group.launch">

          <arg name="allow_trajectory_execution"
value="true"/>

          <arg name="fake_execution" value="false"/>

     </include>



     <include file="$(find
ur5_robotiq_arm)/launch/moveit_rviz.launch">

          <arg name="config" value="$(arg config)"/>

     </include>



</launch>
```