ENPM 662: Introduction to Robot Modeling

Modeling a Universal Robot Arm (UR 5) with Parallel Gripper to Assist the Elderly (Pick and Place)

Project Report

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

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 simulate pick and place of items in a departmental store or grocery store to help the elderly.

First, the motivation of the project is discussed with a problem statement and thereby a proposed model is mentioned. Then we understand the basic assumptions which are to be considered for the robot and a brief introduction to the tools to be used in the proposed model simulation. The next section of the report focuses more on the description of the proposed model (i.e) the robot and the gripper including the forward kinematics. Then the steps to replicate the simulating process for the robot are mentioned in a detailed manner followed by which the validation tool is explained, but the code for validation and simulation is at the appendix of the report and therefore the report concludes with the results and future scope for the project.

Chapter 1

Introduction

1.1 Motivation

"Robots will play an important role in providing physical assistance and even companionship for the elderly." -Bill Gates. As per the census report there are 47.8 million Americans who are 65 and older and this number is expected to reach 98.2 million by 2060 [1]. In such a fast-paced world we often tend to forget the comforts of senior citizens. Among the various problems faced by the elderly in their day to day life the major concern is physical health as muscles and bones begin to weaken and the activities which require more strength become absurd [2].
An average person visits the grocery store 1.6 times a week and spends 43 minutes at the store [3]. There by the person spends a good amount of the time in reaching and placing items into the cart and carrying it to the counter. But this task becomes an issue for elderly people and providing a helping hand in such a scenario is the major source of motivation.

1.2 Problem Statement

Consider a scenario where a senior citizen uses a mobility scooter to drive around the departmental store, but the person is physically not able to add items into the cart due to inaccessibility or weight of the items. In such cases an assistive arm which would pick the item from the isle rack and add to the cart would be an ideal solution.

Hence the robot arm is meant to mimic the normal human behavior to pick an item and place in the cart, which in general is performed by human hand. Now to mimic such a motion of a human hand we require a robot of 6-DOF where the robot uses 3-DOF for positioning and 3-DOF for orientation. Also an end effector is needed to perform the selection of the item to pick the desired item.

1.3 Proposed Model

Consider an elderly person is seated on a locomotive chair and controls the robot arm to pick the required item. The robot is mounted on one of the either sides of the chair and is arranged such that it provides maximum workspace environment to the arm.

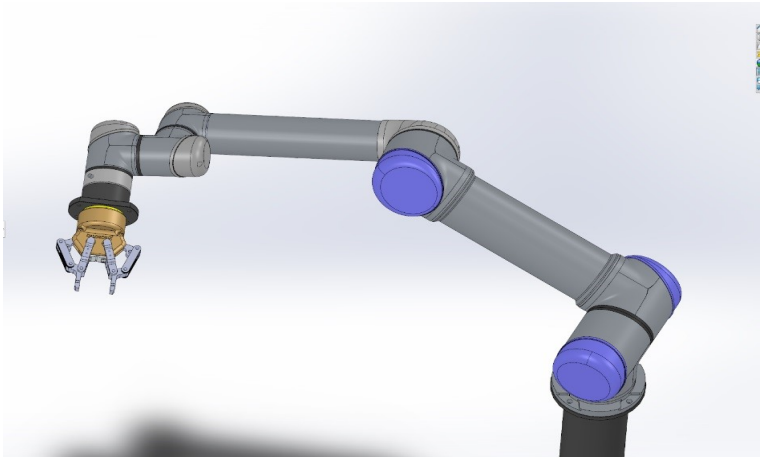


Figure 1.1: Proposed SolidWorks model

Now we assume the locomotive chair is at the desired location i.e. the isle where the user wants to select an item then the person activates the robot arm and selects the item on a provided screen or a provide end-effector co-ordinates i.e. the item position. The arm then moves it's position to reach the item (using the first 3-DOF) and then orients the wrist rotors (using the second 3-DOF) to get a best orientation to pick the item. Once the item is at the reach then the robot activates the grasp where the end effector moves

the parallel gripper fingers to select the item.

Once the item is picked the robot arm moves with the selected item to the pre-defined location or a well known location which is the cart using the 6-DOF and performs the release action for the item into the cart. The proposed robot is the UR5 arm with a 2 finger Robotiq 85 Gripper which will be used for pick and place of the items to help the user add things.

Chapter 2

Getting Started

2.1 Assumptions

To model the robot for the task, we consider the following set of assumptions under which the robot is expected to perform the desired task.

1. All links are rigid and will be able to handle the payload.
2. The workspace of the model is fixed with respect to the mount position in the wheelchair.
3. The object is within the vicinity of the arm and is modified to operate with the robot.
4. The weight of the object to be picked is within the capacity range of the gripper and the arm.
5. Assuming the point of contact will be a soft touch grasp contact and there is no slip between the surface of contact and gripper.
6. There is no slippage between the item and end-effector.
7. Assume the objects' geometry and the relative position of the object with respect to the Gripper is symmetric.
8. The robot satisfies all the environment conditions of the gripper and the arm.
9. The inertia of the chair is much greater than the inertia of the arm.

10. The chair does not topple in any scenario.

2.2 Tools Used

2.2.1 ROS

The Robot Operating System (ROS) is an open-source, meta-operating system. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for building, writing, and running code for robots.[13]

2.2.2 MoveIt

MoveIt provides functionality for kinematics, motion/path planning, collision checking, 3D perception, robot interaction and much, much more. MoveIt is a primary source of a lot of the functionality for manipulation (and mobile manipulation) 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 the ROS robot format (URDF).

2.2.3 RViz

(ROS visualization) is a 3D visualizer for displaying sensor data and state information from ROS. It can be used to validate poses, planning and motion planning. It is also used to simulate a non-real world to move the robot.

2.2.4 Robotics ToolBox

The Toolbox 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, path planning algorithms etc.

Chapter 3

Robot Description

3.1 Universal Robot-5 Arm

The Universal Robot (UR5) arm is composed of extruded aluminum tubes and joints. The Base is where the robot is mounted, and at the other end (Wrist 3) the tool of the robot is attached. By coordinating the motion of each of the joints, the robot can move its tool around freely, apart from the area directly above and directly below the base. The reach of the robot is 850 mm from the center of the base [4]. The robot has 6 links with the last link being connected to the end effector (Refer Fig:3.1). Each of the link length and the center is given in the table 3.1.

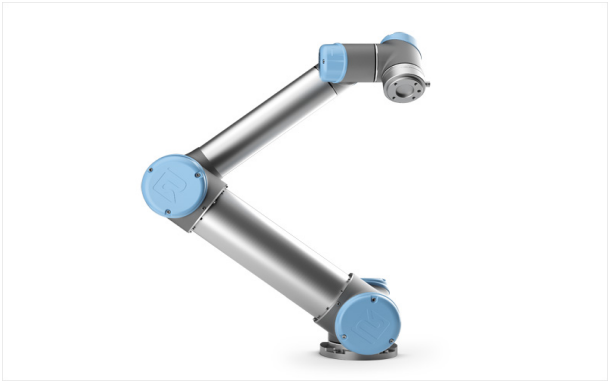


Figure 3.1: Universal Robotics UR5 Arm [5]

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 3.1 Link mass and Center of Mass

3.2 Workspace of the Robot

The robot UR 5 can extend up to 850 mm from the base joint. The robot has a cylindrical volume (refer Fig 3.2) directly above and below the base where a mounting place is chosen. Moving the robot joints close to this cylindrical volume is to be avoided.

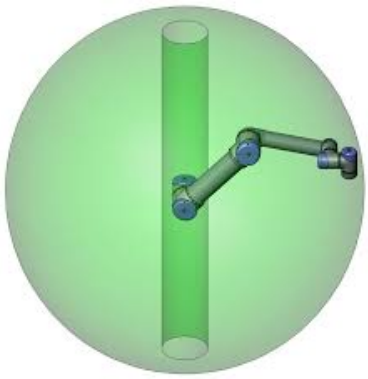


Figure 3.2: Workspace of the Arm

3.3 Kinematic Diagram

Following are 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 Robot Modeling and Control[12] book and applied on the UR5

robot.

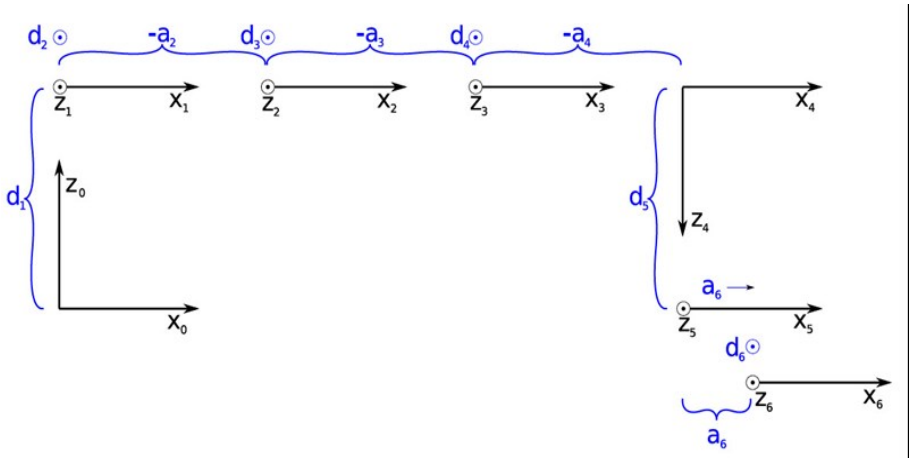


Figure 3.3: Denavit–Hartenber Co-ordinate Diagram [6]

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 3.4: Denavit–Hartenberg Table for UR5 [6]

3.4 Robotiq-85 Gripper

The 2-Finger Gripper is a robotic peripheral that is designed for industrial 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 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[7].

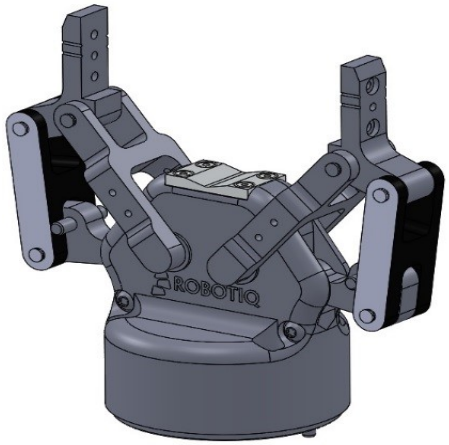


Figure 3.5: Parallel Gripper Robotiq 85 [7]

Chapter 4

Replicability

4.1 Pre-requisites

Configure the work machine with the initial requirements to work replicate the process.

4.1.1 Setting up ROS

1. Make sure the system is running on the Ubuntu platform, 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 might have to initiate the 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.

   ```
   $ 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

```
$ 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

Make sure the ROS has the MoveIt package installed, else use the commands in the terminal to install MoveIt.

```
$ 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
```

4.2 Creating a URDF file

Now that the individual repositories of the robot arm and the gripper are available in the source 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 xyz values all 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 full robot URDF file is ready, the file can be opened with MoveIt. Make sure the ROS has the MoveIt package installed, start the setup assistant to launch the MoveIt on the machine else use the terminal to install MoveIt and then launch.

   ```
   $ 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 selected the desired path launch the file on 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.

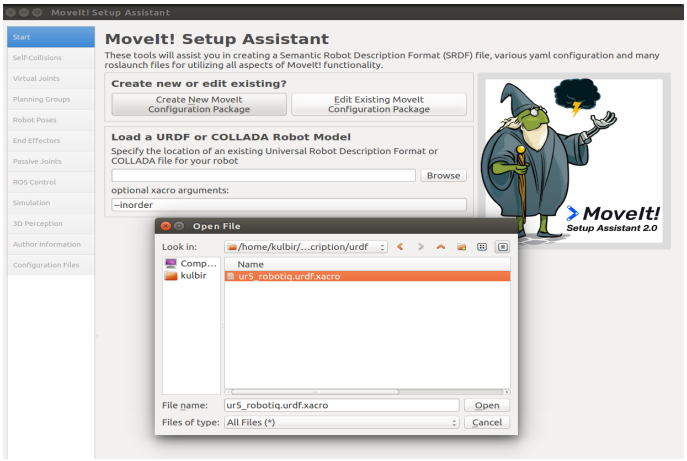


Figure 4.1: Creating a new MoveIt Package

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.

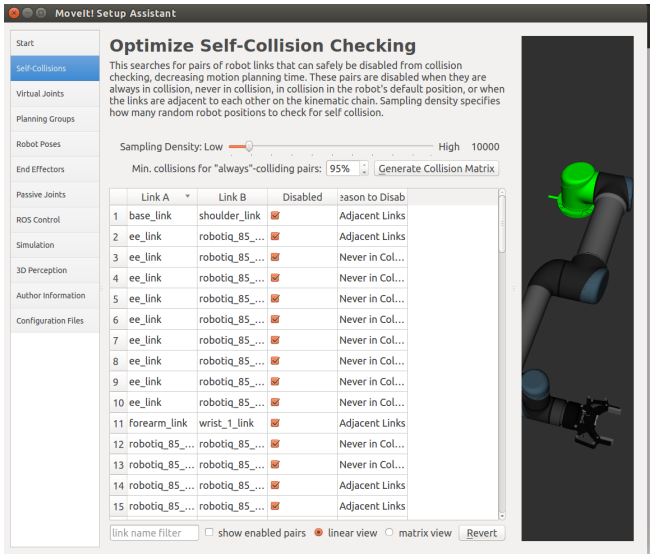


Figure 4.2: Self-Collision Checking

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”.

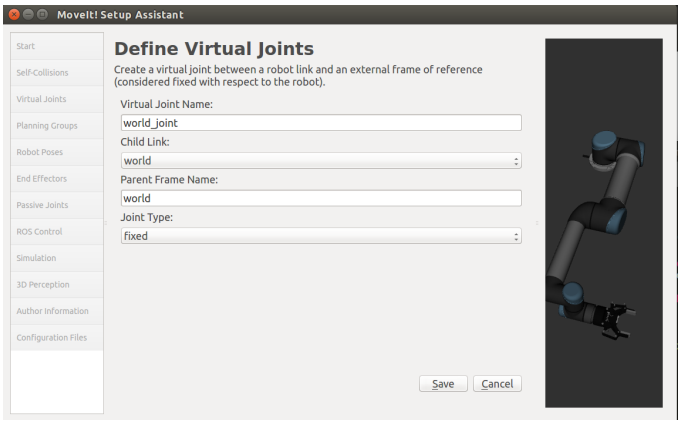


Figure 4.3: Defining Virtual Joints

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.

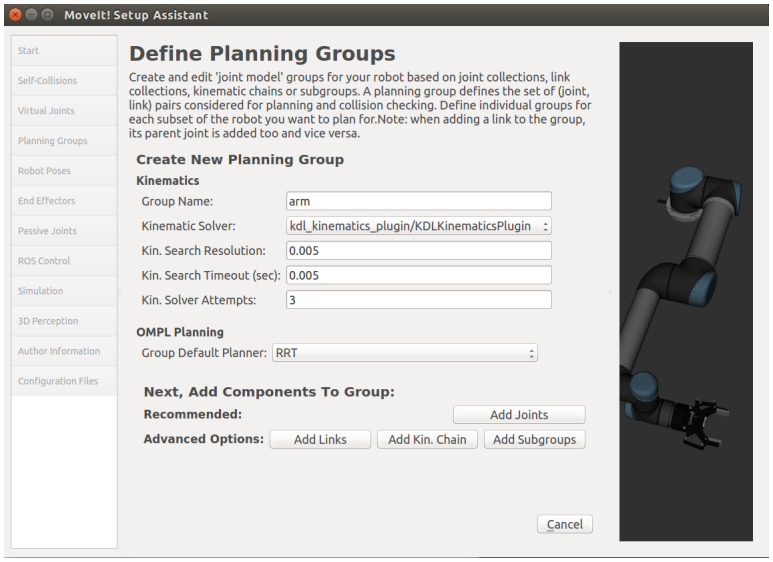


Figure 4.4: Defining Planning Groups

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.

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.
```

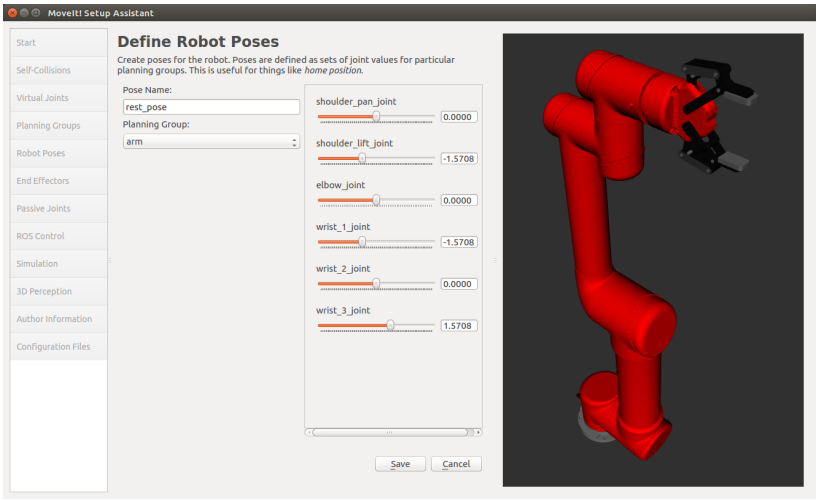


Figure 4.5: Defining Robot Poses

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
    ```

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.

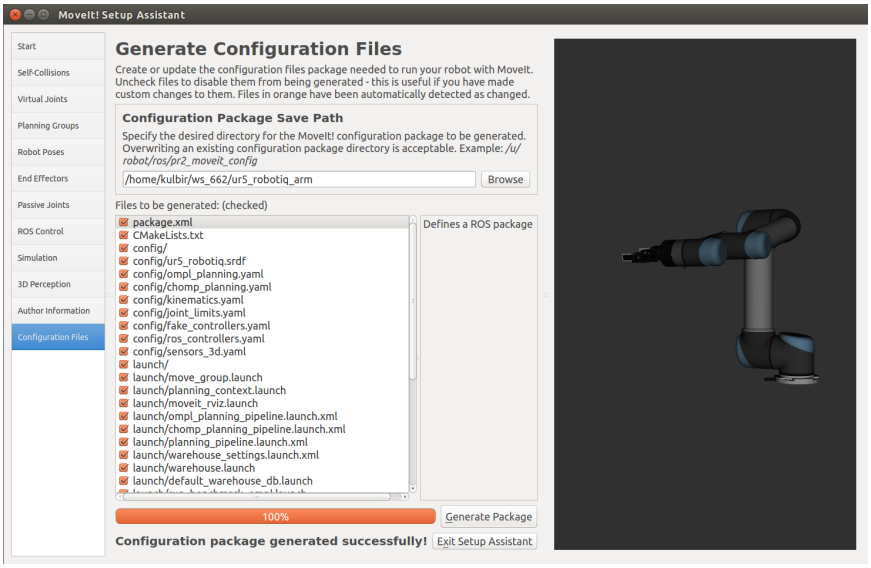


Figure 4.6: Generating Config files

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.
- *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.

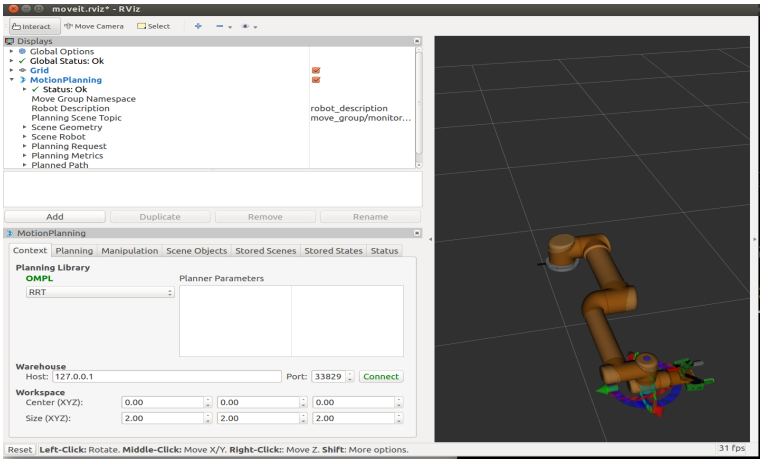


Figure 4.7: Moving the robot in Rviz

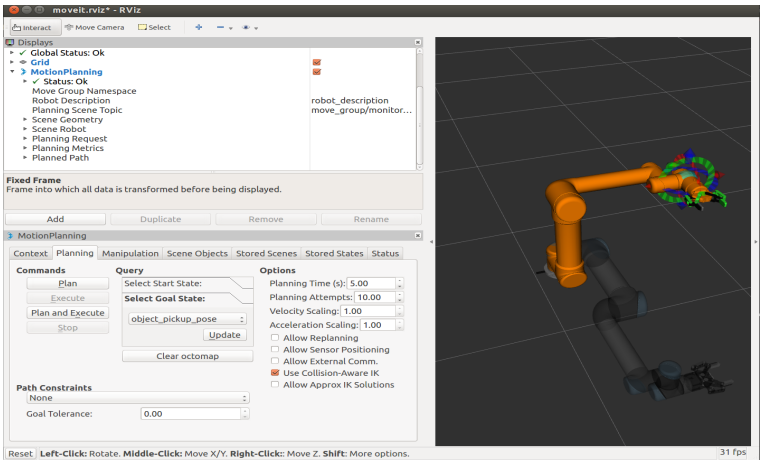


Figure 4.8: Changing the poses in Rviz

Chapter 5

Validation

For validating the results of the robot arm the robotics toolbox developed by Peter Corke has been used. It provides functions to visualize and model robotic systems. It also provides features for the ploting poses, generate the DH table and find the homogeneous transform for a robot pose using forward kinematics and inverse kinematics. [14]

The mainly used functions are:-

- $Object\_Name.plot(pose)$ - To plot a single pose of the robot.
- $Object\_Name.teach$ - 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.

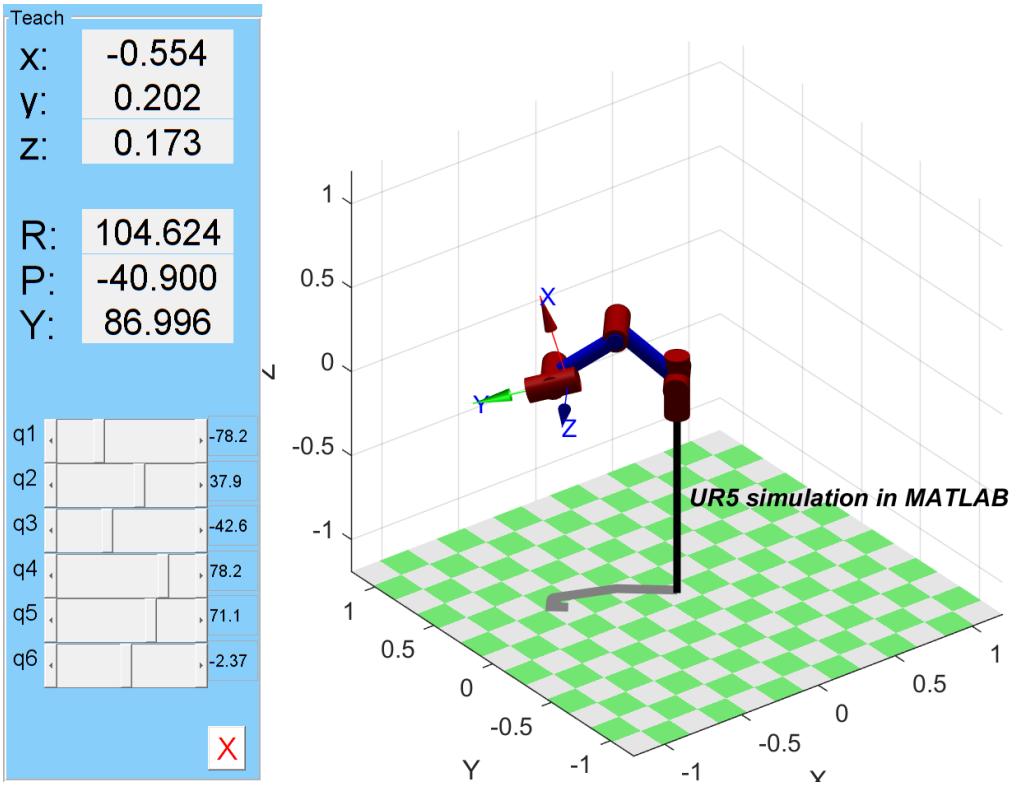


Figure 5.1: GUI interface using the teach function

5.1 Procedure for validating

1. Derive the homogeneous transform for a particular pose manually using forward kinematics.

2. Use the Robotics toolbox to find the same homogeneous transform.

3. The transforms obtained from the first and second approaches should be the same.

The code and the output generated by the Robotics Toolbox is at the Appendix of the report.

5.2 Source for Links

1. Videos for simulation and validation
   https://www.youtube.com/playlist?list=PLSaHroqSzd_WR5xg_QvIdhWTvgah-BKno

2. GIT Repository
   https://github.com/kulbir-ahluwalia/662-Final-Project-UR-5-arm

Chapter 6

Conclusion and Future Work

The aim of this project was to model a robot arm with a gripper to help elderly at the departmental or grocery store. Thus, kinematic modeling of the UR5 arm was carried out to simulate the motion of the arm. The motion of the arm was validated successfully, based on the values generated by the Robotics Toolbox and the manual calculations. The motion of the simulated robot in the predefined moveIt poses also proves the simulation results. But the grasp using the gripper was not successfully established and validated.

As I worked through this project, I have realized the efficiency of the robotic simulation tools and validation tools, as they calculate the dynamics and the kinematics with high accuracy. Also the idea of the robot arm was to be placed on a mobility scooter in a departmental store, hence this would effect the inertia of the robot arm. This realization opened many interesting challenges to the motion of the robot arm on a mobile platform, which could be the next step of the project.

In the end, I would like to thank Prof. Sean Gart for allowing me to work on this project and also the TAs for supporting and helping me throughout the course.

Bibliography

[1] Census Report 2017
https://www.census.gov/content/dam/Census/newsroom/facts-for-features/2017/cb17-ff08.pdf

[2] Common Problems for Elderly
https://www.familymattershc.com/common-problems-for-elderly/

[3] Grocery Shopping Facts
https://www.fivestarhomefoods.com/blog/grocery-shopping-facts

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

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

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

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

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

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

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

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

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

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

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

Appendix

1. Code for Simulation

2. MATLAB Robotics ToolBox Code

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>
```

```
% 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
```

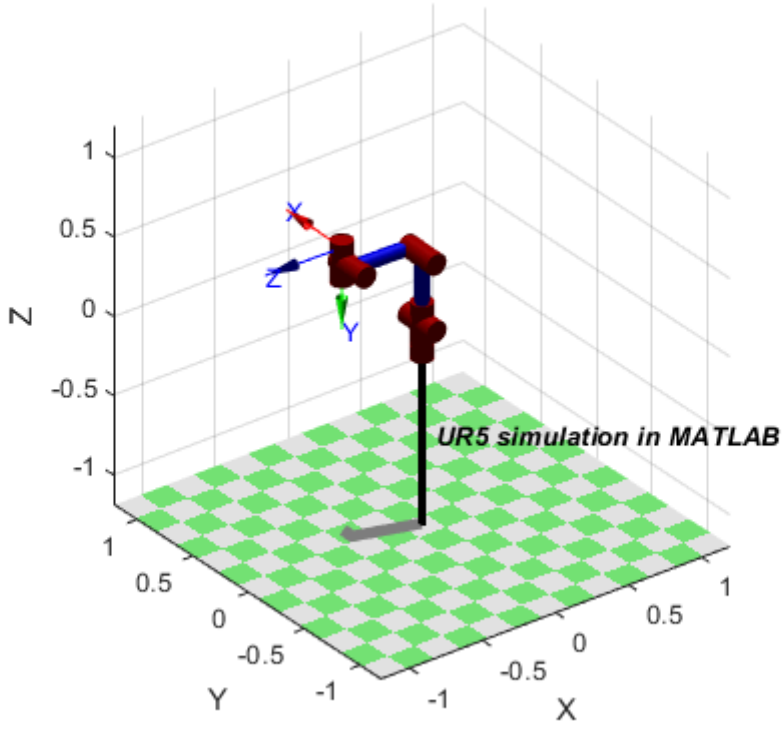
```
%saveas(gcf,'C:\Robotics Toolbox Matlab\ur5_imgs
\ur5_initial_pose.png')
%saveas(gcf,'C:\Robotics Toolbox Matlab\ur5_imgs
\ur5_object_pickup_pose.png')

% 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