UNIVERSITY OF MARYLAND

Introduction to Robot Modeling

On

Pick and Place Robot - UR5

By Akshitha Pothamshetty

UID: 116399326

Section 0101

Table of Content

Abstract

In this project, I utilized a 6 DOF industrial robotic arm, UR5 to pick and place small known objects, similar to bin picking process. The project starts with the theoretical calculation of Forward Kinematics and Inverse Kinematics of UR5. With the assumption of slow movement of the joints, Dynamics of the arm were neglected. I used MATLAB and Vrep to demonstrate the simulation of my project. Practical implementation begins with making Vrep and MATLAB communicate to each other. I used MATLAB as the scripting language to implement FK and IK. Once, Vrep and MATLAB were setup, I validated my implementation using theoretical FK and IK events and matching them with Vrep Output. In the end, I have demonstrated the pick & place operation with the robot tasked to segregate between different blocks and placing them on different tables.

UR5

Introduction

In this project, pick and place robot - UR5 is mounted on a stable stand, strategically positioned to reach the entire work envelope. UR5 is a type of mechanical arm, usually programmable, with similar functions to a human arm; the arm may be the sum total of the mechanism or may be part of a more complex robot. In this project, UR5 a 6 DOF, lightweight and adaptable robotic arm equipped with an RG2 gripper, picks small blocks (dimension: 5x5x5 cm) of various colors (RGB) and places them in a certain pattern.

Specifications

The UR5 is a robot developed by the company Universal Robots. Universal Robots is a Danish company and they sell Robotic Arms. Few of them include UR3, UR5 and UR10. The UR10 is somewhat bigger and able to handle a maximum of ten kilograms instead of five. According to the company the key benefits of their robots are that they are lightweight, safe and easy to use. The robotic arm is regarded as safe, because it will stop acting as soon as the robot hits an object sensed by a force sensor in one of the joints. Because of this, the robot does not need a safety cage and therefore it makes it ideal to work with as object for this project.

In the figure below the specifications given by Universal Robots are stated. One important statement from the specifications is the repeatability of 0.1 mm. Other comparable robots, in terms of size and payload, do much better. Both the IRB1200 of ABB and the T X60 of Stäubli have a repeatability of 0.02 mm. These robotic arms can acquire these values because they are much heavier and stiffer. This also means they are a lot more hazardous and more safety requirements are needed. For example a safety cage. This means these robots are a lot harder to work with.

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

Weight	18.4 kg / 40.6 lbs
Payload	5 kg / 11 lbs
Reach:	850 mm / 33.5 in
Joint ranges:	+/- 360° on all joints
Speed:	Joint: Max 180°/sec. Tool: Approx. 1 m/sec. / Approx. 39.4 in/sec.
Repeatability:	+/- 0.1 mm / +/- 0.0039 in (4 mil)
Footprint:	Ø149 mm / 5.9 in
Degrees of freedom:	6 rotating joints
Control box size (WxHxD):	475 mm x 423 mm x 268 mm / 18.7 x 16.7 x 10.6 in
I/O ports:	10 digital in, 10 digital out, 4 analogue in, 2 analogue out
I/O power supply:	24 V 1200 mA in control box and 12 V/24 V 600 mA in tool
Communication:	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP
Programming:	Polyscope graphical user interface on 12 inch touchscreen with mounting
Noise:	Comparatively noiseless
IP classification:	IP54
Power consumption:	Approx. 200 watts using a typical program
Collaboration operation:	Tested in accordance with sections 5.10.1 and 5.10.5 of EN ISO 10218-1:2006
Materials:	Aluminium, ABS plastic
Temperature:	The robot can work in a temperature range of 0-50°C
Power supply:	100-240 VAC, 50-60 Hz
Calculated Operating Life:	35,000 Hours

Figure 1. Specifications of UR5

In the figure below the UR5 is drawn with the dimensions of all the links. The robotic arm consists of six revolute joints. For the remainder of the report these joints will be referred to as Base, Shoulder, Elbow, Wrist 1, Wrist 2 and Wrist 3. The names of these joints are also stated in the figure. The UR5 has a rather standard layout for these types of robotic arms. The Shoulder and Elbow joint are rotating perpendicular to the Base joint. These three joints are connected with long links. In this way the robot has a long reach. The Wrist joints are mainly to manoeuvre the Tool Center Point (TCP) in the right orientation. Other robotic arms with similar layouts often use a spherical joint instead of three revolute joints. Those spherical joints are less prone to self-collision as the solution of the UR5.

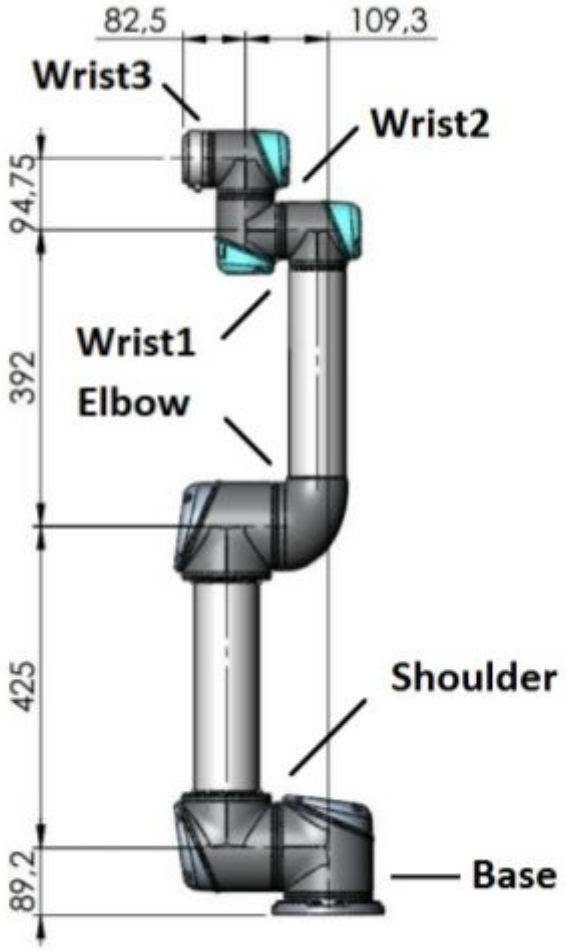


Figure 2. Dimensions of the UR5

Forward Kinematics

The Denavit–Hartenberg parameters (also called DH parameters) are the four parameters associated with a particular convention for attaching reference frames to the links of a spatial kinematic chain, or robot manipulator. They are used in order to standardize the coordinate frames for spatial linkages.In this convention, coordinate frames are attached to the joints between two links such that one transformation is associated with the joint, [Z], and the second is associated with the link [X]. The coordinate transformations along a serial robot consisting of n links form the kinematics equations of the robot,

$$[T] = [Z_1][X_1][Z_2][X_2]\dots[X_{n-1}][Z_n][X_n],$$

where [T] is the transformation locating the end-link.

Reference frames below illustrates a common assignment of Denavit-Hartenberg convention to the UR5 robot (shown with all joint angles at 0).

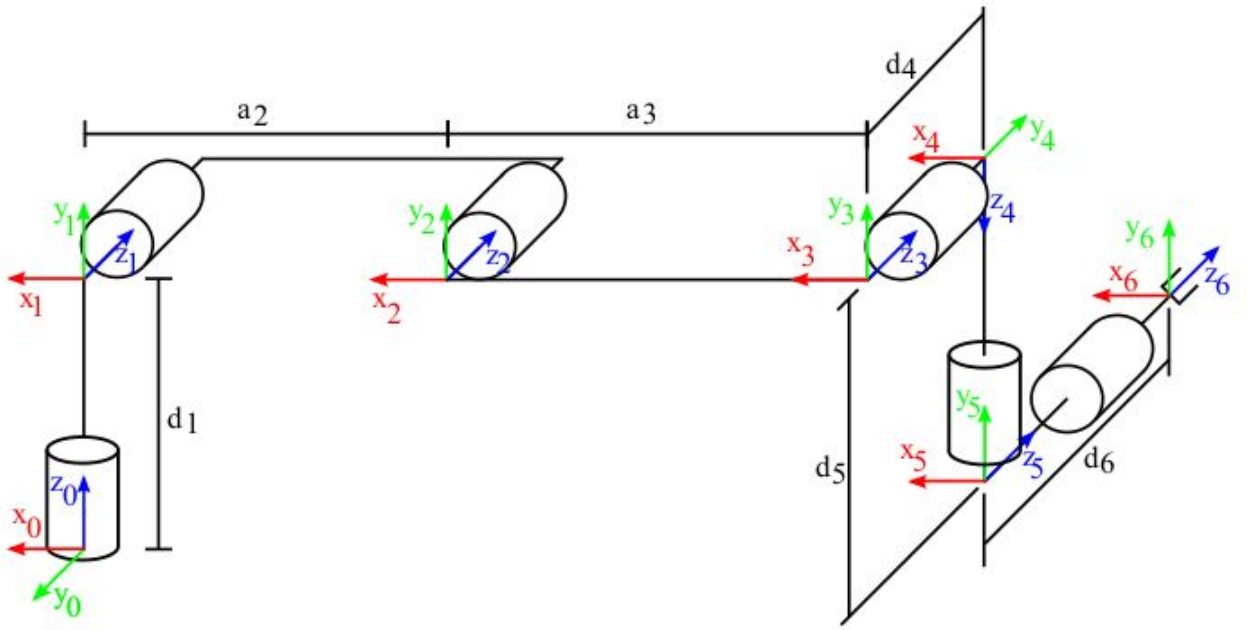


Figure 3. D-H Convention Frame Assignment

Joint	a	α	d	θ
1	0	$\pi/2$	0.089159	θ_1
2	-0.425	0	0	θ_2
3	-0.39225	0	0	θ_3
4	0	$\pi/2$	0.10915	θ_4
5	0	$-\pi/2$	0.09465	θ_5
6	0	0	0.0823	θ_6

Figure 4. D-H Parameters

As with any 6-DOF robot, the homogeneous transformation from the base frame to the gripper can be defined as follows:

$$T_6^0(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) = T_1^0(\theta_1)\,T_2^1(\theta_2)\,T_3^2(\theta_3)\,T_4^3(\theta_4)\,T_5^4(\theta_5)\,T_6^5(\theta_6) \tag{1}$$

Also remember that a homogeneous transformation T j i has the following form:

$$T_j^i = \begin{bmatrix} R_j^i & \vec{P}_j^i \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x_x & y_x & z_x & (P_j^i)_x \\ x_y & y_y & z_y & (P_j^i)_y \\ x_z & y_z & z_z & (P_j^i)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where $\vec{P}_j^i$ is the translation from frame i to frame j and each column of R_j^i is the projection of one of the axes of frame j onto the axes of frame i (i.e. $[x_x, x_y, x_z]^T \equiv \vec{x}_j^i$).

The equations below form the transformation matrix for the Forward Kinematics of the UR5. In these equations, I have used the following notations:

$$s_{i+j} = sin(\theta_i+\theta_j) \text{ and } c_{i+j} = cos(\theta_i+\theta_j)$$

The transformation matrix is calculated as:

$$H_6^0 = \begin{bmatrix} R_{11} & R_{12} & R_{13} & x \\ R_{21} & R_{22} & R_{23} & y \\ R_{31} & R_{32} & R_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

$$R_{11} = c_6(s_1s_5 + c_{234}c_1c_5) - s_{234}c_1s_6$$
$$R_{21} = -c_6(c_1s_5 - c_{234}c_5s_1) - s_{234}s_1s_6$$
$$R_{31} = c_{234}s_6 + s_{234}c_5c_6$$
$$R_{12} = -s_6(s_1s_5 + c_{234}c_1c_5) - s_{234}c_1c_6$$
$$R_{22} = s_6(c_1s_5 - c_{234}c_5s_1) - s_{234}c_6s_1$$
$$R_{32} = c_{234}c_6 + s_{234}c_5s_6$$
$$R_{13} = c_5s_1 - c_{234}c_1s_5$$
$$R_{23} = -c_1c_5 - c_{234}s_1s_5$$
$$R_{33} = -s_{234}s_5$$

$$x = d_6(c_5 s_1 - c_{234} c_1 s_5) + d_4 s_1 + c_1(a_3 c_{23} + a_2 c_2) + d_5 s_{234} c_1$$
$$y = s_1(a_3 c_{23} + a_2 c_2) - d_4 c_1 - d_6(c_1 c_5 + c_{234} s_1 s_5) + d_5 s_{234} s_1$$
$$z = d_1 + a_3 s_{23} + a_2 s_2 - d_5 c_{234} - d_6 s_{234} s_5$$

Inverse Kinematics

The first step to solving the inverse kinematics is to find θ_1 . To do so, we must first find the location of the 5th coordinate frame with respect to the base frame, P_5^0 . As illustrated in the figure below, we can do so by translating by d_6 in the negative z direction from the 6th frame.

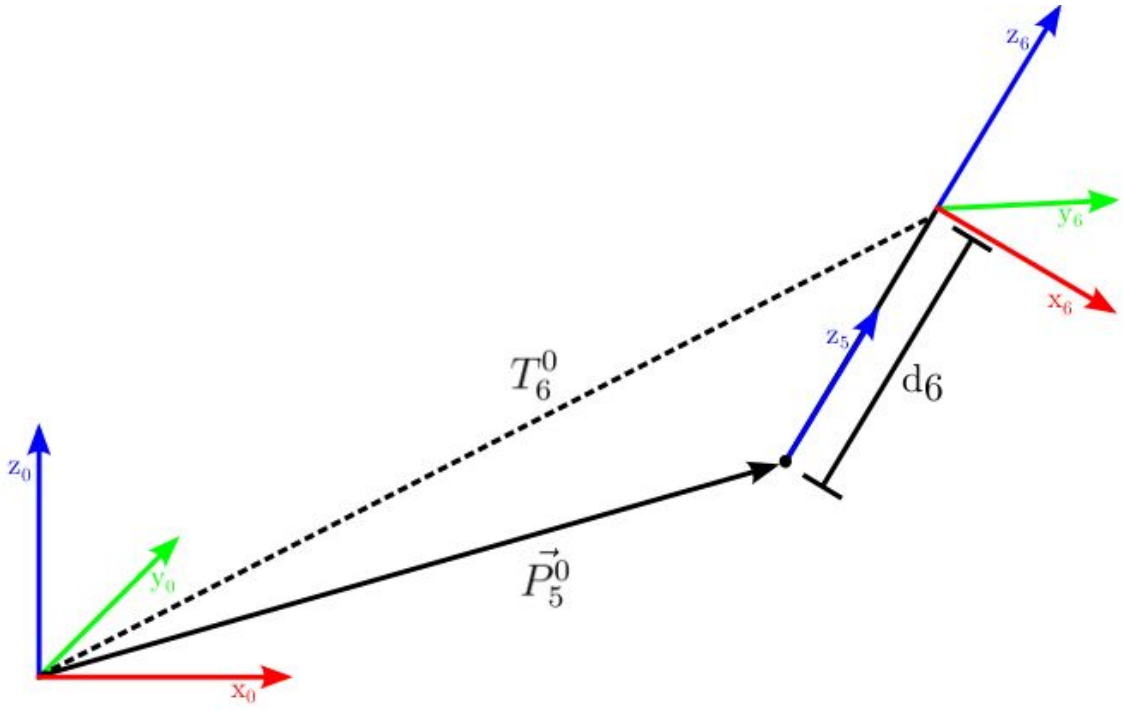


Figure 5. Finding the origin of the 5th frame

This is equivalent to the following operation:

$$\vec{P}_5^0 = T_6^0 \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3}$$

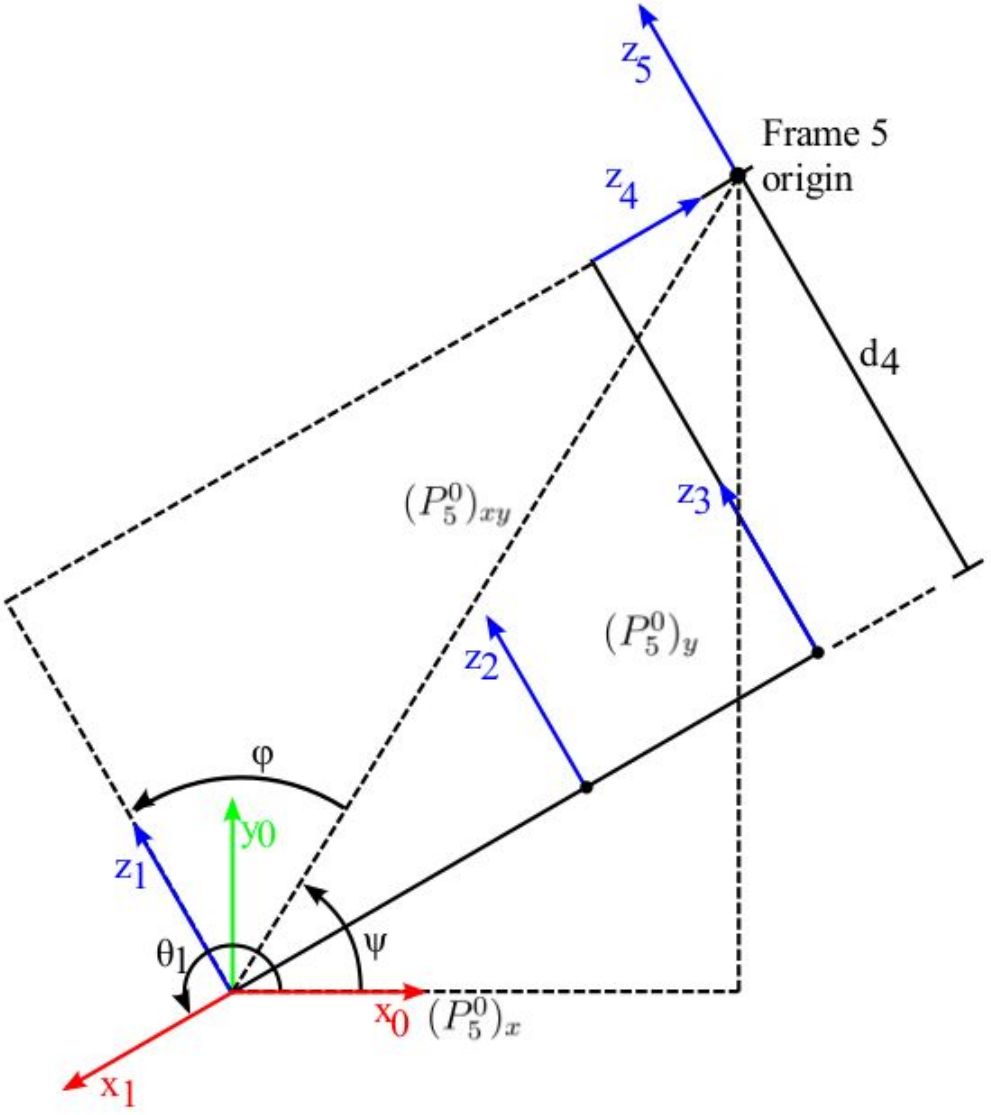


Figure 6. Finding θ_1 . Note that the rotations around the axes z_1 , z_2 , z_3 and z_4 do not change the fact that the origin of Frame 5 is in the plane that is parallel to the x_1 axis, offset by d_4 as shown.

The key to this observation is that T^0_6 is known because it is simply the desired transformation! That is, given the desired transformation, T_6^0 , we can calculate the vector from the origin of Frame 0 to the origin of Frame 5.

With the location of the 5th frame, we can draw an overhead view of the robot as shown in the above figure. From this, we can see that $\theta_1 = \psi + \varphi + \pi/2$ where

$$\psi = \operatorname{atan2}\left((P_5^0)_y, (P_5^0)_x\right) \tag{4}$$

$$\phi = \pm \arccos\left(\frac{d_4}{(P_5^0)_{xy}}\right) = \pm \arccos\left(\frac{d_4}{\sqrt{(P_5^0)_x{}^2 + (P_5^0)_y{}^2}}\right) \tag{5}$$

The two solutions for θ_1 above correspond to the shoulder being either "left" or "right,". Note that Equation 5 has a solution in all cases except that $d_4 > (P_5^0)_{xy}$. You can see from the figure this happens when the origin of the 3rd frame is close to the z axis of frame 0. This forms an unreachable cylinder in the otherwise spherical workspace of the UR5 (as shown in the below figure taken from the UR5 manual).

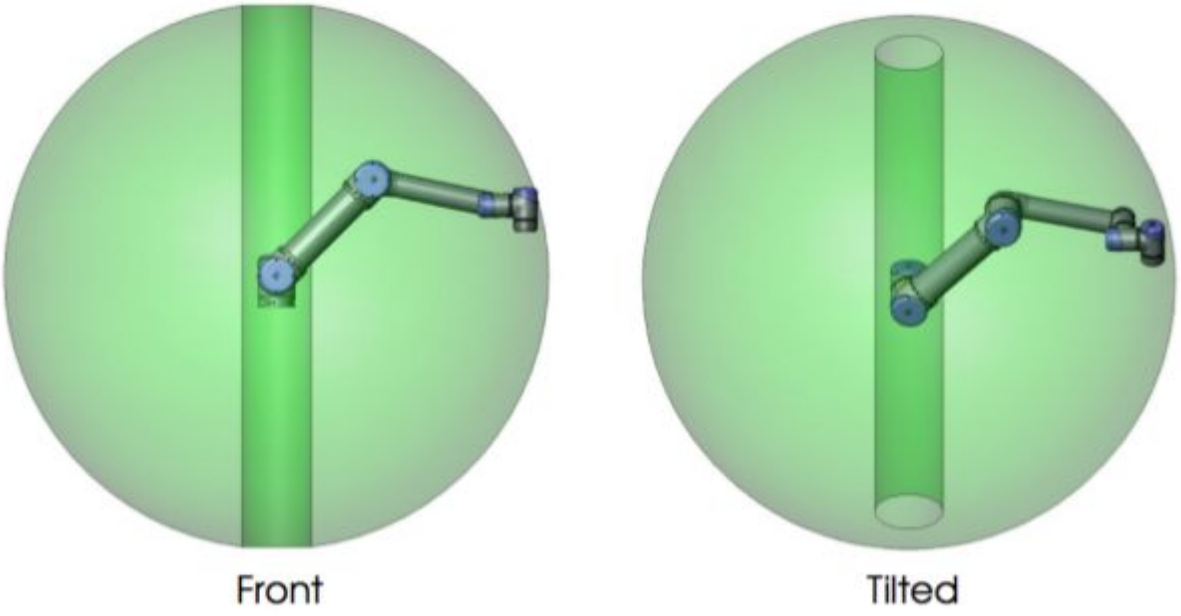


Figure 7. The Workspace of the UR5

Knowing θ_1 , we can now solve for θ_5 . Once again we draw an overhead view of the robot, but this time we consider the location of the 6th frame with respect to the 1st.
We can see that,

$$(P_6^1)_z = d_6 \cos(\theta_5) + d_4, \text{ where } (P_6^1)_z = (P_6^0)_x \sin(\theta_1) - (P_6^0)_y \cos(\theta_1)$$

Solving for θ_5,

$$\theta_5 = \pm \arccos\left(\frac{(P_6^1)_z - d_4}{d_6}\right) \tag{6}$$

Once again, there are two solutions. These solutions correspond to the wrist being "down" and "up."

Ignoring translations between frames, z_1 can be represented with respect to frame 6 as a unit vector defined with spherical coordinates. We can find the x and y components of this vector by projecting it onto the x-y plane and then onto the x
or y axes.

Next we find transformation from frame 6 to frame 1,

$$T_1^6 = ((T_1^0)^{-1}\ T_6^0)^{-1} \tag{7}$$

Remembering the structure of the first three columns of the homogeneous transformation T_1^6 (Equation 2), we can form the following equalities:

$$-\sin(\theta_6)\sin(\theta_5) = z_y \tag{8}$$

$$\cos(\theta_6)\sin(\theta_5) = z_x \tag{9}$$

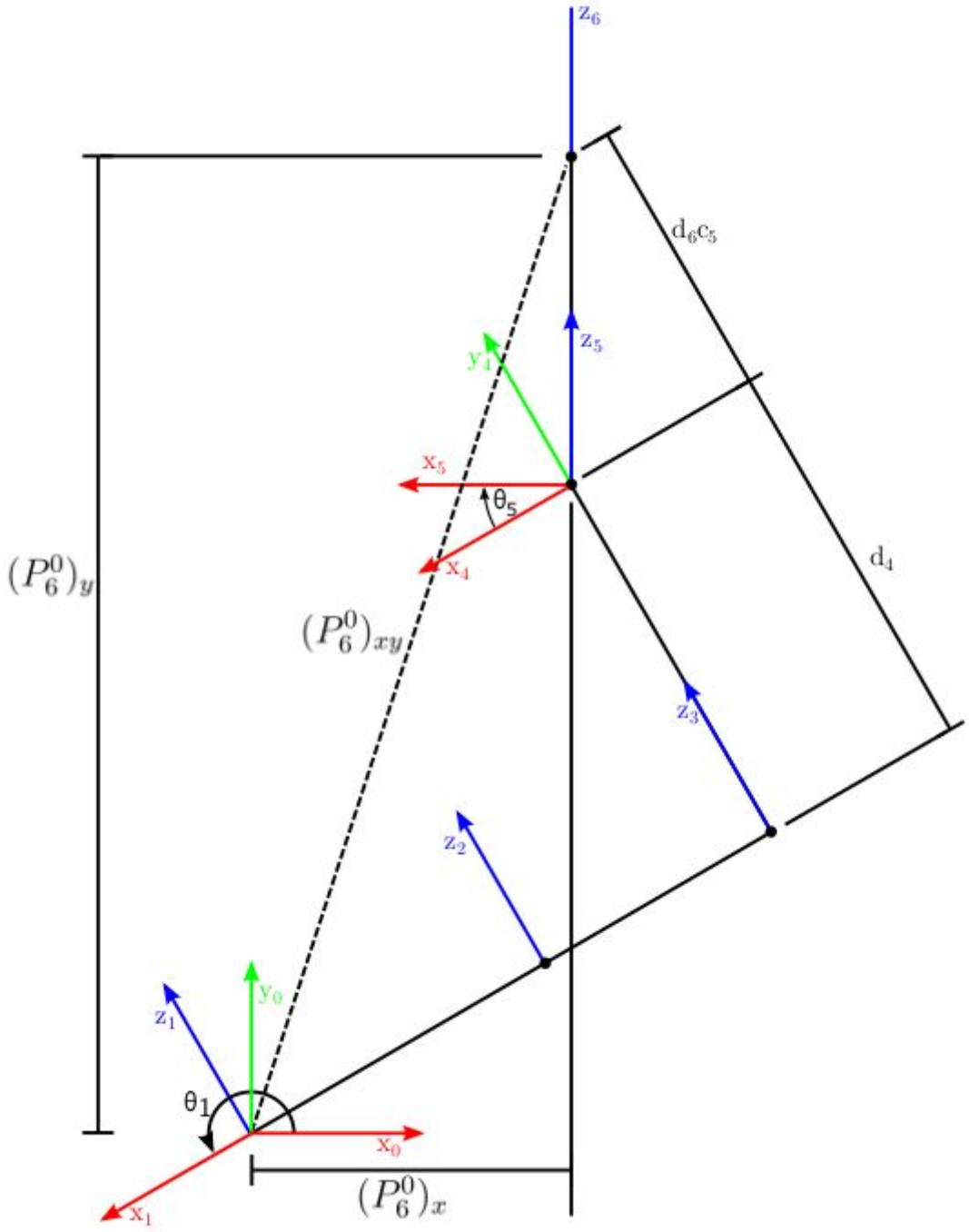


Figure 8. Finding θ_5

Solving for θ_6 ,

$$\theta_6 = \text{atan2}\left(\frac{-z_y}{\sin(\theta_5)}, \frac{z_x}{\sin(\theta_5)}\right) \tag{10}$$

Equation 10 shows that θ_6 is not well-defined when $\sin(\theta_5)=0$ or when z_x, $z_y = 0$. We can see from the below figure that these conditions are actually the same. In this configuration joints 2, 3, 4, and 6 are parallel. As a result, there are four degrees of freedom to determine the position and rotation of the end-effector in the plane, resulting in an infinite number of solutions. In this case, a desired value for q_6 can be chosen to reduce the number of degrees of freedom to three.

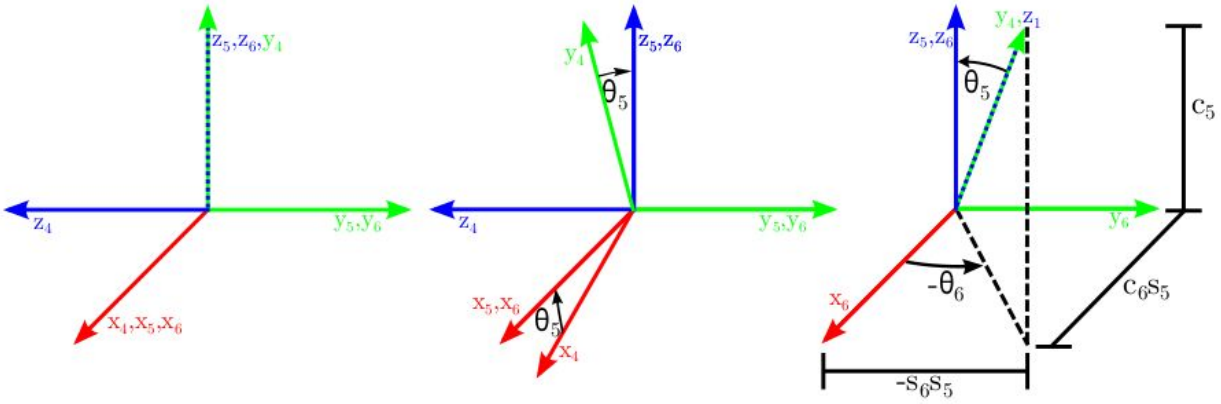


Figure 9. Finding θ_6

We can now consider the three remaining joints as forming a planar 3R manipulator. First we will find the location of frame 3 with respect to frame 1. This is done as follows:

$$T_4^1 = T_6^1\, T_4^6 = T_6^1\, (T_5^4\, T_6^5)^{-1} \tag{11}$$

$$\vec{P_3^1} = T_4^1 \begin{bmatrix} 0 \\ -d_4 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{12}$$

Now we can draw the plane containing frames 1-3, as shown in the figure below. We can see that

$$\cos(\xi) = \frac{||\vec{P_3^1}||^2 - a_2^2 - a_3^2}{2a_2a_3} \tag{13}$$

with use of the law of cosines.

$$\begin{aligned} \cos(\xi) &= -\cos(\pi - \xi) \\ &= -\cos(-\theta_3) \\ &= \cos(\theta_3) \end{aligned} \tag{14}$$

Combining 13 and 14,

$$\theta_3 = \pm \arccos\left(\frac{||\vec{P_3^1}||^2 - a_2^2 - a_3^2}{2a_2a_3}\right) \tag{15}$$

Equation 15 has a solution as long as the argument to arccos is $\in [-1, 1]$. We can see that large values of $||\vec{P_3^1}||$ will cause this the argument to exceed 1. The physical interpretation here is that the robot can only reach so far out in any direction (creating the spherical bound to the workspace).

From below figure,

$$\theta_2 = -(\delta - \epsilon) \tag{16}$$

where

$$\delta = \text{atan2}((P_3^1)_y, -(P_3^1)_x)$$

and can be found via law of sines:

$$\frac{\sin(\xi)}{||\vec{P_3^1}||} = \frac{\sin(\epsilon)}{a_3} \tag{17}$$

Combining 16 and 17,

$$\theta_2 = -\operatorname{atan2}((P_3^1)_y, -(P_3^1)_x) + \arcsin\left(\frac{a_3 \sin(\theta_3)}{||\vec{P_3^1}||}\right) \tag{18}$$

Notice that there are two solutions for θ_2 and θ_3 . These solutions are known as "elbow up" and "elbow down."

The final step to solving the inverse kinematics is to solve for θ_4 . First we want to find T_4^3 :

$$T_4^3 = T_1^3\, T_4^1 = (T_2^1\, T_3^2)^{-1} T_4^1 \tag{19}$$

Using the first column of T_4^3,

$$\theta_4 = \operatorname{atan2}(x_y, x_x) \tag{20}$$

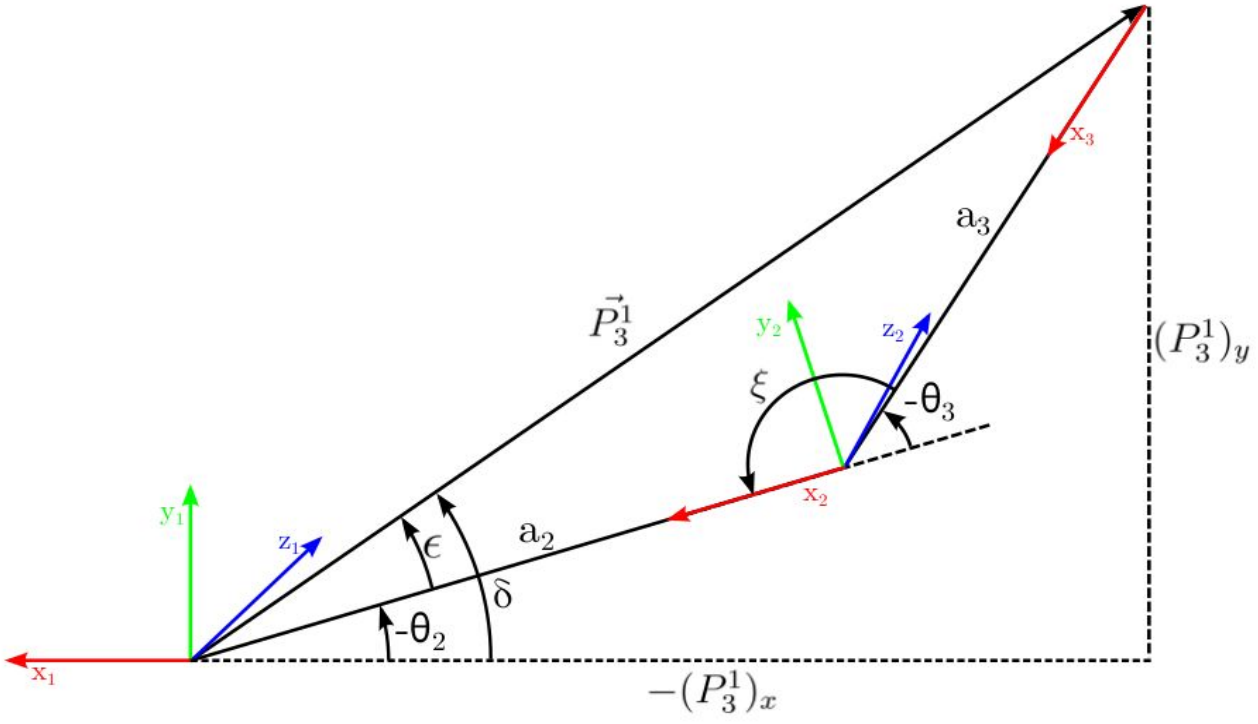
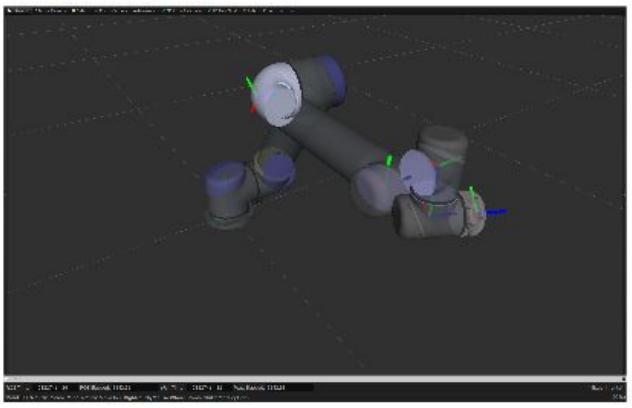
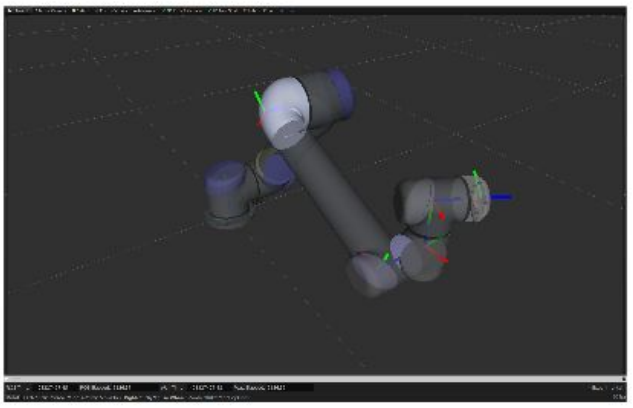


Figure 10. Finding θ_2 and θ_3

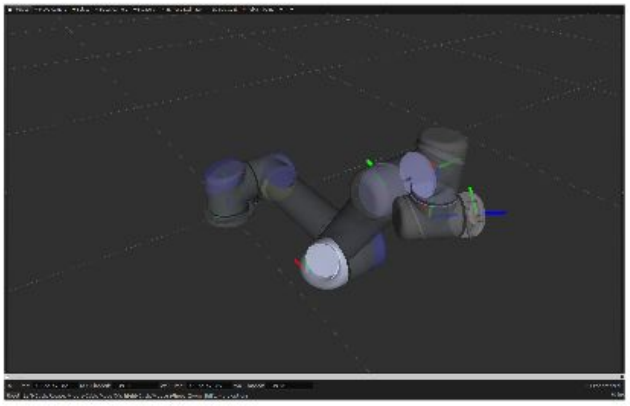
Below are figures which show the eight solutions for one end-effector position/orientation.



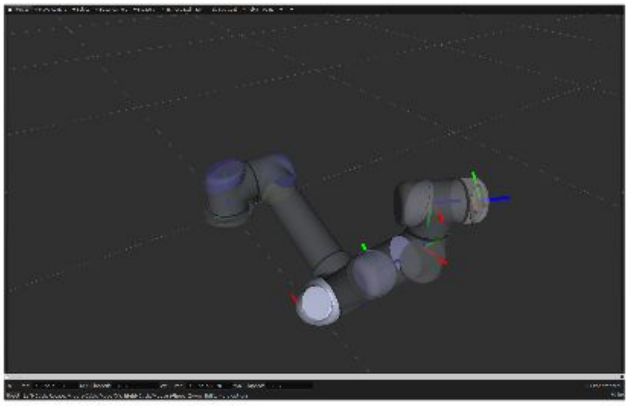
Shoulder Left, Elbow Up, Wrist Down



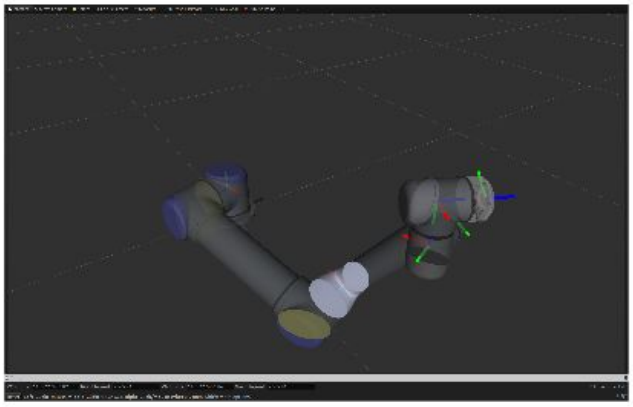
Shoulder Left, Elbow Up, Wrist Up



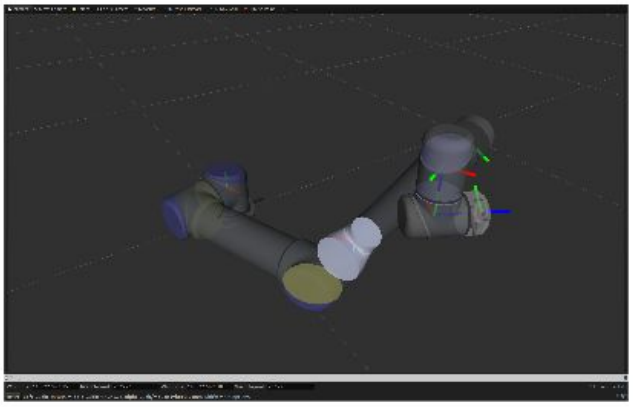
Shoulder Left, Elbow Down, Wrist Down



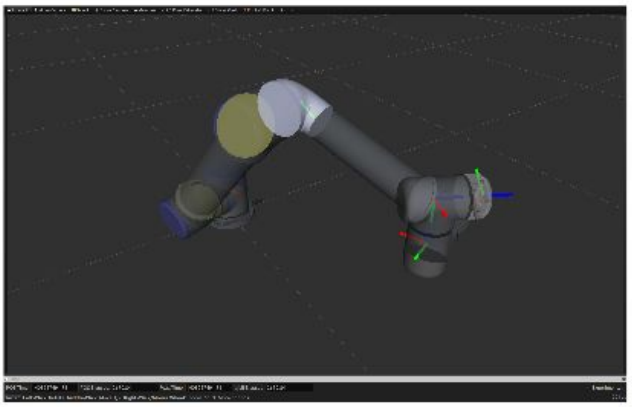
Shoulder Left, Elbow Down, Wrist Up



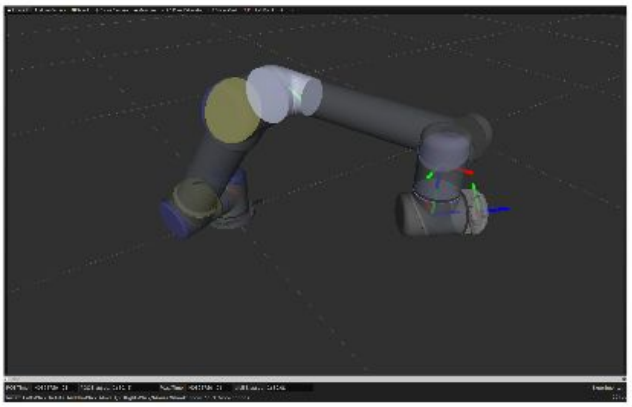
Shoulder Right, Elbow Down, Wrist Up



Shoulder Right, Elbow Down, Wrist Down



Shoulder Right, Elbow Up, Wrist Up



Shoulder Right, Elbow Up, Wrist Down

Assumptions

1. All links are rigid.
2. There is no effect of vibrations and external disturbances on movement of links.
3. The robot would not be affected by machining accuracy.
4. Flexibility effects such as bending of the links due to gravity and other loads would not be present.
5. Taking the last 3 joints to be a spherical wrist for simplified inverse kinematic solutions.
6. All the objects are placed within the workspace of the robot.
7. There is no time lag between the communication of MATLAB and V-rep.
8. Considering slow movements of the joints.
9. Object poses are known beforehand.

Implementation

Goal

UR5 is placed on a table top and is given instructions to reach various positions, grab the objects at those locations and place them accordingly. In this section, I will go through each step in the implementation process of this goal.

Approach

My working methodology for the project was as:

1. Literature Review
   Understanding original design constraints and Specifications of UR5. I went through the Technical datasheet of the company to confirm link lengths, offsets and other constraints.

2. Theoretical analysis of Forward Kinematics and Inverse Kinematics
   I used the concepts taught in class to theoretically calculate Forward and Inverse Kinematics. For FK, I will calculate end-effector positions for series of joint angles. For IK, I will calculate joint parameters for different end-effector positions. This served for Validating my Matlab scripts and Vrep simulation as well.

3. Simulation and Validation:
   I used V-rep to simulate the UR5 arm. I gave V-rep series of inputs for all the cases and compared all those with calculations made theoretically. MATLAB was used for both theoretical calculations(FK and IK) and also sending commands to move UR5 in Vrep. For FK validation, Joints angles were given and EE positions were compared while for IK, EE was given and joints angles were compared.

4. Project Simulation:
   In the end, I demonstrated my final project by simulating UR5 to pick objects and place them at various locations. MATLAB and Vrep communicated asynchronously and MATLAB gave instructions to move the robot at each step.

Matlab-Vrep Integration

The robot simulator V-REP, with integrated development environment, is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS or Blue Zero node, a remote API client, or a custom solution. This makes V-REP very versatile and ideal for multi-robot applications. Controllers can be written in C/C++, Python, Java, Lua, Matlab or Octave. I used MATLAB to simulate the virtual UR5 in the VREP Environment.

There were a few important things to note while making MATLAB communicate with Vrep. The following files need to be in the working directory, in order to run the various examples:

1. remApi.m
2. remoteApiProto.m
3. The appropriate remote API library: "remoteApi.dll" (Windows), "remoteApi.dylib" (Mac) or "remoteApi.so" (Linux)

I used Asynchronous Mode for connecting MATLAB with Vrep. This means MATLAB doesn’t wait for Vrep before sending next instruction. This worked well and I didn’t feel the need to make the software communicate in synchronous mode.

Vrep Scene Description

The scene consists of four tables, three blocks(R,G,B) and UR5 equipped with RG2 gripper. UR5 is fixed on the center table. UR5 is tasked to place the Red block on the table to its left, Green block on the table to its right and leave the Blue block on the front table itself.

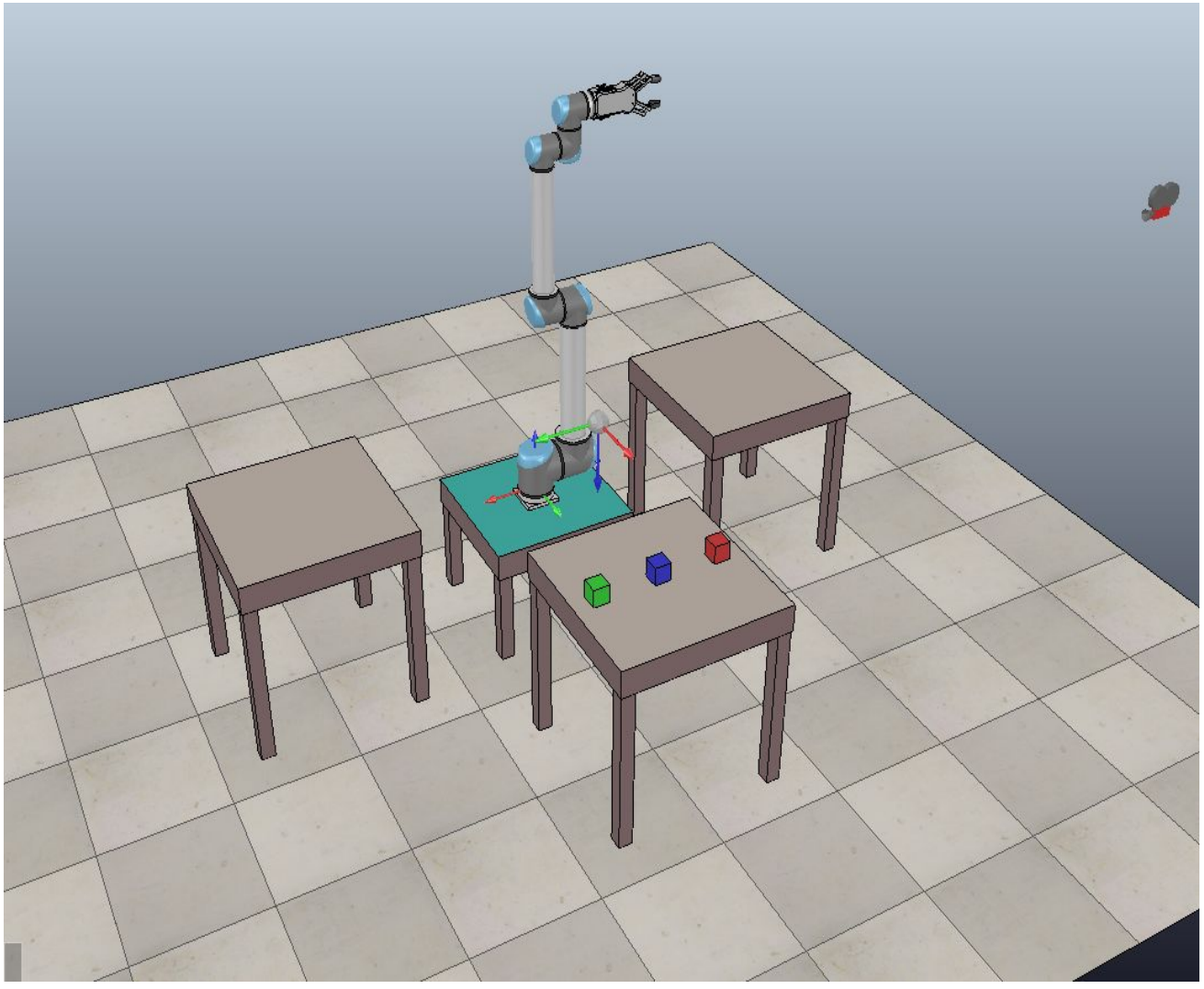


Figure 11. Vrep Scene Setup

After the first scene, Robot arm tries to place all the three blocks on top of each other. Sometimes it ends up crashing the heap of block with RG2 gripper itself while placing the blocks.

Implementation - FK & IK

For implementing the Forward and Inverse Kinematics, I have used MATLAB as the main scripting language. There were good tutorials to make MATLAB work with Vrep instead of Python working with Vrep and hence, I chose MATLAB for convenience.

Essense of all the coding lies in the script PickAndPlace.m (placed in MAIN folder). The script starts with initializing the communication with Vrep. It adds the helper functions located in various folders to its path. Then, object handles are created to control UR5 joints and RG2 gripper in the Vrep scene. The script takes the frames of the robot base, tables and cubes to track their location as well.

It was important to keep a check that we have access to all frames and handles at all times, this is implemented in the *Stream* section of the code. Once stream checks are positive, we begin our simulation. To keep the simulation manageable, I have kept the end effector position tracking threshold as 1cm. I have used Peter Corke Robotics Toolbox to create the serial linked robot object(*UR5*). I just used the toolbox to get the inverse kinematics of the robotic arm using the DH Parameter.

In the next step, we start the simulation. There are two sequences which I have implemented. In the first sequence the robot reaches for each blocks one at a time and places each of them on different tables(leaving blue block on center table). In the second sequence, robot tries to place each block on top of each other.

I have kept the code as modular as possible with many helper functions being as general and supportive of any robotic arm. We can use any other arm with different gripper in the Vrep scene as well.

Validation

The validation of Forward Kinematics was done by finding out the end effector position for two different joint variable configurations, $q = [\theta_1 , \theta_2 , \theta_3 , \theta_4 , \theta_5 , \theta_6]$

The first test was done for zero configuration , $q = [\,0, 0, 0, 0, 0, 0\,]$ and second at q = [90, 0, 90, 0, 90, 0]. Transformation matrix result and vrep results are shown in the table. As it can be seen the end-effector position and orientation obtained through calculations and the simulated system are the same as the last 1^0. The slight difference that is observed here is because of the issues in inaccurate CAD modeling and the dynamics engine used by Vrep. Even then the results seem very accurate and it can be claimed that the calculation of the Forward Kinematics of the system is validated.

-1.0000	0	0	-0.2445	-0.2500
0	0	-1.0000	1.1843	1.1700
0	-1.0000	0	0	-0.0100
0	0	0	1.0000	

Figure 12. Case for q = [0 0 0 0 0 0]. Calculated Positions(Left) vs Vrep Positions(Right)

1.0000	0	0	-0.9764	-0.9900
0	1.0000	0	0.1273	0.1200
0	0	1.0000	-0.1639	-0.1700
0	0	0	1.0000	

Figure 13. Case for q = [90 0 90 0 90 0]. Calculated Positions(Left) vs Vrep Positions(Right)

The solution obtained using the inverse position kinematics is a unique solution. The validation of the inverse kinematics was also done by selecting three different end-effector positions and orientations. Then the above formulation is used to calculate the joint variables required by the robot to attain that pose. The required joint variables are calculated using MATLAB and passed to the v-rep model. The pose of the robot is then compared to the required pose, which should be the same.

Case 1. The first test was done for the home configuration, q = [0, 0, 0, 0, 0, 0]. The results obtained in the previous subsections are used to calculate the required joint angle configurations which come out to be as expected: Passing these joint variables to the robot in vrep, the results obtained were: $(-0.2394,\ 1.2041,\ 0.0021)$.

Case 2. The first test was done for the home configuration, q = [90, 0, 90, 0, 90, 0]. The results obtained in the previous subsections are used to calculate the

required joint angle configurations which come out to be as expected: Passing these joint variables to the robot in vrep, the results obtained were: $(-0.9728,\ 0.1241,\ -0.1742)$.

The two cases used to validate the inverse kinematics were taken from the forward kinematic validation to show that the results obtained using both the formulations are consistent. It can be observed from the two cases that the inverse kinematics formulation is efficient in providing a unique solution for a required end effector pose for all the test cases, taking into account the joint constraints as mentioned in the theory section. Thus, we can safely assume that the algorithm is correct and has been validated through the results shown above.

Results

Vrep Simulation starts when MATLAB start communicating with the Vrep. Robot Arm reaches for the Red Block, grabs it, places it on the table to its left. It then reaches for the Blue block, grabs it, places it on the same table. It then reaches for the green block and places that on the table to its right.

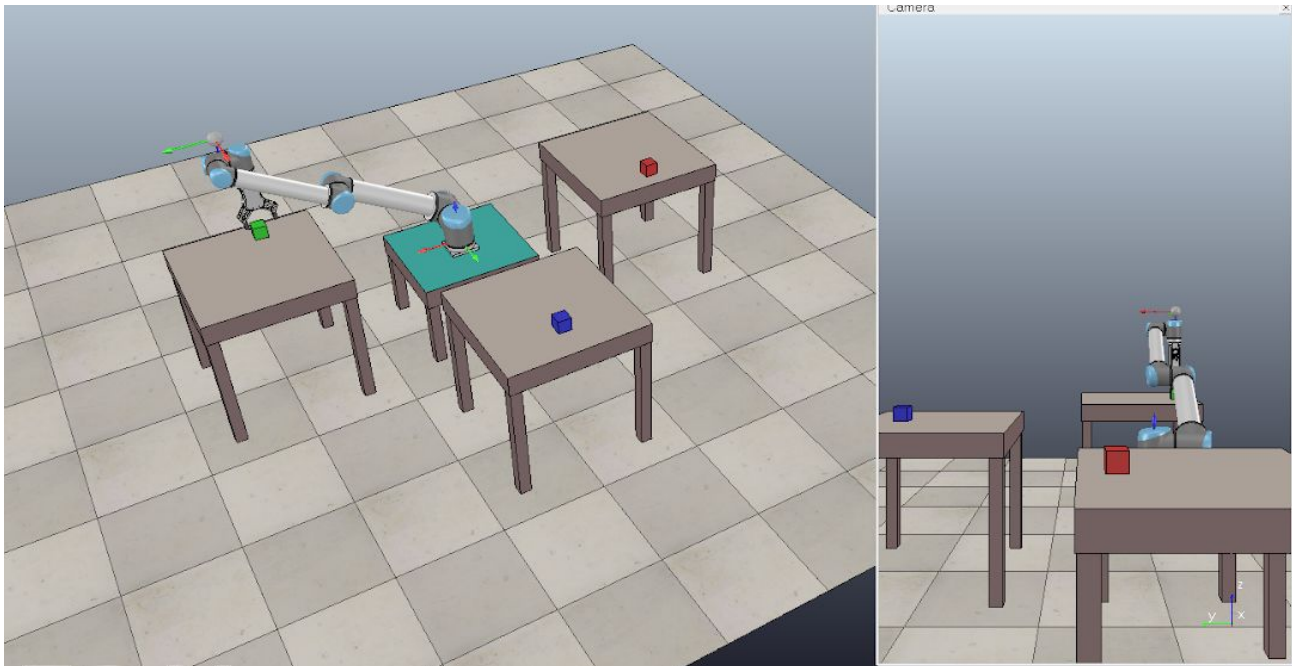


Figure 14. Scene One Demonstration Result

The scene is then restored again for second demonstration. Red block is placed on the left table first, then blue block is grabbed and dropped on to the red block on the left table. Finally, it reaches the green block, picks and drops it at the top of the two blocks. At the end of demonstration 2, the results may look like that in the below figure.

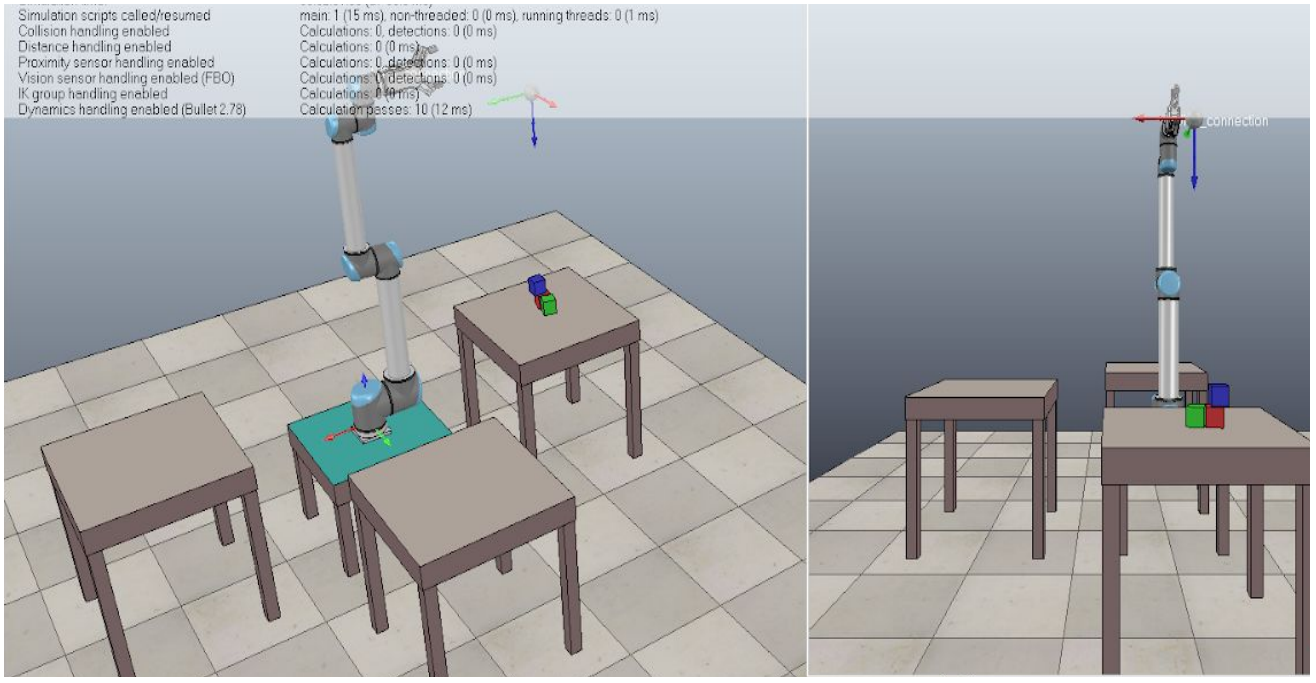


Figure 15. Scene two demonstration

The results for the two scenes is shown as video output, stored in the *Results* folder inside *Main* directory.

Applications

I demonstrated the pick and place capability of UR5 in this project. This project can be extended to any number of applications. For example, equipped with a Vision Sensor, the Arm can be used for inspecting and segregating defective pieces from an assembly line. It can be used as Welding Robot, Palletizing Robot, Painting Robot or Assembly Robot. With a robust control feedback strategy this robot application can be extended to precise operations such as Medical Robotics.

Conclusions and Future Work

The project presented a foundation approach to apply my learning of Forward and Inverse Kinematics. I started with theoretical calculations of forward and inverse kinematics, followed by validating them. Most interesting part of the project was simulating the robot in Vrep. I utilized MATLAB for scripting FK and IK for the Robot. MATLAB instructed the robot in Vrep to demonstrate the FK and IK correctness through two different scene demonstrations. I have kept the code modular, so that it can work with any kind of Robotic arm and not just UR5.

For Future work, I would like to implement a drawing robot using UR5. The robot takes in curve input from the user through Mouse over MATLAB and UR5 draws the curve in Vrep. Due to time constraints, I could not implement this in current project. This would be very fun and interesting task to implement. Another fun project I would like to implement is a Homework Bot which takes in text files and writes down it on paper with ink. More interesting thing can be taking in image files, applying *bwboundaries* function in Matlab and some pre-processing to create a curve file. These curves can be followed by the arm to draw the same hand-written document on a sheet. I will call this a *CopyBot*.

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

Appendix

Code for PickAndPlace.m

```
%% Information Section
% ENPM662 Modeling Final Project
% Author: Akshitha Pothamshetty
% Email-ID: apothams@umd.edu
% Section: 0101

%% About the Project
% Goal: In this project, I utilized a 6 DOF industrial robotic arm, UR5 to
% pick and place small known objects, similar to bin picking process. The
% project starts with the theoretical calculation of Forward Kinematics and
% Inverse Kinematics of UR5. With the assumption of slow movement of the
% joints, Dynamics of the arm were neglected. I used MATLAB and Vrep to
% demonstrate the simulation of my project. Practical implementation begins
% with making Vrep and MATLAB communicate to each other. I used MATLAB as
% the scripting language to implement FK and IK. Once, Vrep and MATLAB were
% setup, I validated my implementation using theoretical FK and IK events
% and matching them with Vrep Output. In the end, I have demonstrated the
% pick & place operation with the robot tasked to segregate between
% different blocks and placing them on different tables.


%% Step 1: Including Paths and .m Files

addpath('..'); % Go to Subdirectory, containing Main Folder, VrepConnection
and Mfunctions folder.

addpath('../VrepConnection','../Mfunctions/Robotics_Functions',...
    '../Mfunctions/URn_Functions','../Mfunctions/Vrep_Functions'); % Add to
Path all the folders.

vrep=remApi('remoteApi'); % using the prototype file
vrep.simxFinish(-1); % Close Open Connections
id=vrep.simxStart('127.0.0.1',19997,true,true,5000,5);

if (id < 0)
    disp('Failed connecting to remote API server. Exiting.');
```

```
    vrep.delete();
    return;
end

fprintf('Connection %d to remote API server open.\n', id);


%% Step 2: GetObjectHandle for UR5 Joints, Blocks and Frames.

%%% For the UR5 Joints
handles = struct('id',id);
jointNames={'UR5_joint1','UR5_joint2','UR5_joint3','UR5_joint4',...
    'UR5_joint5','UR5_joint6'};

handles.ur5Joints = -ones(1,6);
for i = 1:6
    [res, handles.ur5Joints(i)] = vrep.simxGetObjectHandle(id, ...
                     jointNames{i}, vrep.simx_opmode_oneshot_wait);
    vrchk(vrep, res);
end


%%% For the Cuboids & Tables
CuboidNames={'Left_Cuboid','Front_Cuboid','Right_Cuboid'};

TableNames={'Left_Table','Front_Table','Right_Table'};

handles.ur5Cuboids = -ones(1,3);
handles.ur5Tables = -ones(1,3);

 for i = 1:3
    [res, handles.ur5Cuboids(i)] = vrep.simxGetObjectHandle(id,...
                    CuboidNames{i}, vrep.simx_opmode_oneshot_wait);
    vrchk(vrep, res);

    [res, handles.ur5Tables(i)] = vrep.simxGetObjectHandle(id,...
                    TableNames{i}, vrep.simx_opmode_oneshot_wait);
    vrchk(vrep, res);
end

%%% For the Ref & Gripper
```

```
  % Those Handle Not Used in the Simulation
handles.ur5Ref= -1;
handles.ur5Gripper=-1;
[res, handles.ur5Ref] = vrep.simxGetObjectHandle(id, 'UR5', ...
    vrep.simx_opmode_oneshot_wait);
vrchk(vrep, res);

[res, handles.ur5Gripper] = vrep.simxGetObjectHandle(id, 'UR5_connection',
...
    vrep.simx_opmode_oneshot_wait);
vrchk(vrep, res);

%%% For the Base frame [Frame0]
handles.base=-1;
[res, handles.base] = vrep.simxGetObjectHandle(id, ...
    'Frame0', vrep.simx_opmode_oneshot_wait);
vrchk(vrep, res);

%%% For the Target frame [Frame1]
handles.Target=-1;
[res, handles.Target] = vrep.simxGetObjectHandle(id, ...
    'Frame1', vrep.simx_opmode_oneshot_wait);
vrchk(vrep, res);

%%% For the EndGripper And EndGripperJoints
handles.EndGripper= -1 ;
    [res, handles.EndGripper] = vrep.simxGetObjectHandle(id, ...
                  'RG2_openCloseJoint', vrep.simx_opmode_oneshot_wait);
    vrchk(vrep, res);

%% Stream ...

%%% Cuboids [position/orientation & Tables [position/orientation]
relativToRef = handles.base;
for i = 1:3

    %Cuboids
    res =
vrep.simxGetObjectPosition(id,handles.ur5Cuboids(i),relativToRef,...
                vrep.simx_opmode_streaming);
    vrchk(vrep, res, true);
```

```
    res =
vrep.simxGetObjectOrientation(id,handles.ur5Cuboids(i),relativToRef,...
                vrep.simx_opmode_streaming);
    vrchk(vrep, res, true);

    % Tables
    res = vrep.simxGetObjectPosition(id,handles.ur5Tables(i),relativToRef,...
                vrep.simx_opmode_streaming);
    vrchk(vrep, res, true);
    res =
vrep.simxGetObjectOrientation(id,handles.ur5Tables(i),relativToRef,...
                vrep.simx_opmode_streaming);
    vrchk(vrep, res, true);
end


%%% EndGripper & EndGripperJoints
res = vrep.simxGetJointPosition(id,handles.EndGripper,...
          vrep.simx_opmode_streaming);
vrchk(vrep, res, true);

%%% The UR5 Joints
for i = 1:6
    res = vrep.simxGetJointPosition(id, handles.ur5Joints(i),...
                vrep.simx_opmode_streaming);
    vrchk(vrep, res, true);
end


%% Start
vrep.simxGetPingTime(id);
vrep.simxStartSimulation(id, vrep.simx_opmode_oneshot_wait);

%% Simulation

% Set the threshold to check if the end effector has reached its destination
handles.threshold = 0.01;

% Set The Arm Parameters Using Peter Corke robotics toolbox
handles.ur5Robot= URnSerial_fwdtrans('UR5');
pause(0.5);
```

```
%Rest Joint for 1st time
handles.startingJoints = [0, 0, 0, 0, 0, 0];
        res = vrep.simxPauseCommunication(id, true);
        vrchk(vrep, res);
    for j = 1:6
            vrep.simxSetJointTargetPosition(id, handles.ur5Joints(j),...
                  handles.startingJoints(j),vrep.simx_opmode_oneshot);
            vrchk(vrep, res);
    end
        res = vrep.simxPauseCommunication(id, false);
        vrchk(vrep, res);
pause(1);

%% Simulating Pick and Place Operation.

ConstValue=0.25;

iTable=1;  % 1 for 'Left_Table', 2 for'Front_Table', 3 for'Right_Table'
iCuboid=1; % 1 for 'Left_Cuboid', 2 for'Front_Cuboid', 3 for'Right_Cuboid'
XYZoffset= [0 0 0]; % [ X Y Z]

%%% 1st Sequence

iCuboid=1;  XYZoffset=[0 0 ConstValue];
GoAndCatch_Cuboid( id, vrep, handles, iCuboid,XYZoffset);
iTable=1;   XYZoffset=[0 ConstValue 0];
GoAndLeave_Cuboid( id, vrep, handles, iTable ,XYZoffset);

iCuboid=2;  XYZoffset=[0 0 ConstValue];
GoAndCatch_Cuboid( id, vrep, handles, iCuboid,XYZoffset);
iTable=2;   XYZoffset=[0 ConstValue 0];
GoAndLeave_Cuboid( id, vrep, handles, iTable ,XYZoffset);

iCuboid=3;  XYZoffset=[0 0 ConstValue];
GoAndCatch_Cuboid( id, vrep, handles, iCuboid,XYZoffset);
iTable=3;   XYZoffset=[0 -ConstValue 0];
GoAndLeave_Cuboid( id, vrep, handles, iTable ,XYZoffset);

% Return to Normal Orientation.
g= eye(4,4);
g(1:3,4)=[-0.25 1 0.7301];
```

```
moveFrame( id, vrep, g, handles.ur5Cuboids(1), handles.base );
g(1:3,4)=[0 1 0.7301];
moveFrame( id, vrep, g, handles.ur5Cuboids(2), handles.base );
g(1:3,4)=[0.25 1 0.7301];
moveFrame( id, vrep, g, handles.ur5Cuboids(3), handles.base );
pause(2);

%%% 2nd Sequence
for i=1:3

    iCuboid=i;  XYZoffset=[0 0 ConstValue];
    GoAndCatch_Cuboid( id, vrep, handles, iCuboid,XYZoffset);
    iTable=1;   XYZoffset=[0 0 0.15*i];
    GoAndLeave_Cuboid( id, vrep, handles, iTable ,XYZoffset);

end


%% END ..
vrep.delete;
clear; clc;
```

References

[1] Spong, Mark W., Seth. Hutchinson, and M. Vidyasagar. Robot Modeling and Control. Hoboken, NJ: John Wiley, 2006. Print.

[2] https://www.mathworks.com/help/physmod/smlink/index.html

[3] http://petercorke.com/wordpress/toolboxes/robotics-toolbox

[4] E. Rohmer, S. P. N. Singh, M. Freese, "V-REP: a Versatile and Scalable Robot Simulation Framework," IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2013

[5] Mobile Manipulator Robots. https://en.wikipedia.org/wiki/Mobile_manipulator