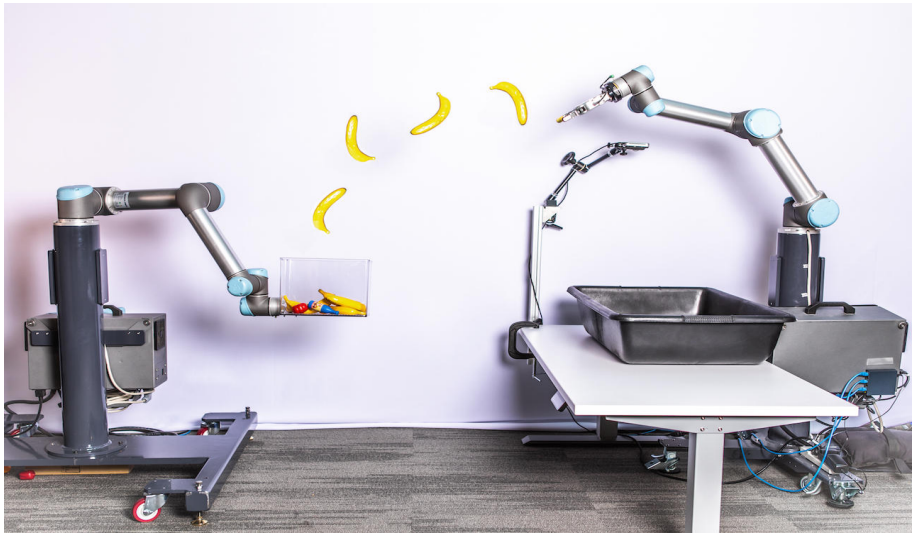
ENPM662 - PROJECT REPORT

Modeling a Tossing Robot



Author:
Umang Rastogi
116773232

Project Partner:
Govind Ajith Kumar
116699488

11^{th} December 2019

Abstract

This project report discusses about modeling of a 5-DOF robotic manipulator including a gripper. The manipulator is being to throw objects out of its workspace. The project, in general, involves modeling concepts of forward and inverse kinematics, dynamics, path planning and trajectory planning. We have only focused on the forward-inverse kinematics of the robotic arm and trajectory planning of the object. We will talk about the motivation behind the project, its applications, evaluation of various modeling parameters and their verification, and finally end with simulation of the project using V-REP.

Contents

Chapter 1

Introduction

Robotic arms have been very successful in controlled environments. They play a significant role in industrial automation, especially for assembly and production lines. Moreover, with the increasing presence of robots in our lives, their ability to pick up and throw objects will be of great use. The idea of the project was heavily inspired by the Tossingbot developed at Princeton. The ability can also be used by robots to reduce space junk. This is the reason why I want to explore the idea of a robot capable of throwing and catching various objects with the modeling project.

1.1 Motivation

The robotics community is working day in and day out to make robots perform similar to that of humans such as walking, picking up and placing things. Numerous efforts are being made to increase the human robot interaction. One of the things that we commonly do is throw things. Be it to make that touchdown or just throwing paper bowls into a dustbin, we love to throw things. A robot's ability to throw objects will definitely move us towards our goal to increase human-robot interaction. Moreover, most of the robots have a limited workspace. The object throwing ability of a robot essentially expands its workspace.

Throwing is commonly accompanied by a catcher at the other end and therefore a catching robot will have to be developed for each throwing robot unless the application involves throwing of innocuous objects. Apart from these, the numerous applications of a robot capable of throwing objects is another motivating factor to work on the project. These robots can be deployed in sports training facilities since many outdoor sports involve throwing a ball. An extensive system with feedback sensors can create endless opportunities to deploy these types of robots.

1.2 Objective

With endless possibilities for the project, we are tried to set a distinct achievable goal for the project. Since we got the idea of modeling a throwing robot from the TossingBot, we will simulate a similar scenario in which our robot will be able to throw 3 objects into 3 different baskets demonstrating a classification application of the robot. We plan on deducing all the necessary modeling parameters and verifying them as well. The final simulation will be done using V-REP and modeling will be done using MATLAB.

1.3 Assumptions

The following assumptions hold for the development of the project:

- The robotic arm is rigidly attached to the surface and the surface is immovable.
- There is no energy loss in the system, where the system consists of the robotic manipulator and the object.

- The container is out of the workspace of the robotic manipulator.
- The container is heavy enough to not to dislocate from its original position when the object thrown at it.
- The object does not deform upon hitting a rigid body such as the floor.
- Firm contact is established between the object and the gripper when the gripper reaches the position of the object.

1.4 Timeline of the Project

We got strayed-off a bit upon our initial timeline due to unexpected difficulties. However, we were able to achieve all of our current goals. The final project timeline took the following shape:

September 27^{th}	Completing the project pre-proposal
October 2^{nd}	Selecting a 6-DOF CAD model
October 10^{th}	Building a custom parts for the robotic arm on SolidWorks
October 15^{th}	Finished CAD work for the prismatic joint of the arm
October 19^{th}	Finished CAD work for the gripper of the arm
October 21^{st}	Started working on the proposal
October 25^{st}	Working on forward kinematics of the robot
October 29^{th}	Submission of project proposal
November 1^{st}	Importing the URDF model of our arm into V-REP
November 3^{rd}	Deciding on the new manipulator and gripper
November 7^{th}	Completed simulation of inverse kinematics on V-REP
November 8^{th}	Completed grasping of dynamic objects in V-REP
November 9^{th}	Designing path of the arm to follow in V-REP
November 12^{th}	Making the robotic arm move along the designed path in V-REP

November 18thnd	Started working on the Inverse kinematics of the robot
November 23nd	Throwing objects using add force method
November 24th	Force calculation for throwing the robot
November 26th	Calculation for placement of baskets to throw the objects
November 27th	Accomplished throwing the object into the basket
November 30th	Accomplished throwing two separate objects into two separate basket
December 2nd	Presentation preparation for progress review
December 4th	Presentation of project progress in class
December 5th	Completion of Simulation with three baskets for three separate objects
December 6th	Code commenting on V-REP
December 7th	Started off with report writing for final submission
December 8th	Dynamics Calculations
December 9th	Calculations completed on MATLAB
December 10th	Project Validation
December 11th	Final Project Submission

1.5 Designing the Robotic Arm

The first task at hand was to either build a custom robotic arm from scratch or use an existing one and make modifications to make it work for our application. We went ahead with the latter option to modify a 6-DOF arm with another prismatic joint and a custom gripper. We modified the manipulator using SolidWorks. The image of the model is shown in Figure 1.1. The robotic arm worked well in SolidWorks but importing the robot model into V-REP for simulation was a problem. We had to convert the CAD model into a URDF (Universal Robot Description Format) file. We used the open-source SolidWorks-to-URDF convertor for the task. After the import into V-REP, the all of the joint alignments got messed up even when we manually entered coordinates of each links. The open-source software was not well-documented to figure out the problem.To avoid any further delays, we started working on a pre-included model in V-REP known as the PhantomX-Pincher.This manipulator was also attached with a different gripper known as the RG2 gripper.

The PhantomX Pincher is one of the smallest robotic arms developed by Interbotix. It was designed to be used as the robotic manipulator on top of the Turtlebot. It weighs only 550g with a vertical and horizontal reach of 35cm and 31cm respectively. Moreover, we went ahead with the RG2 to simulate realistic picking up scenarios. You can see the difference between the original and the modified version the robotic arm in figures 1.2 and 1.3.

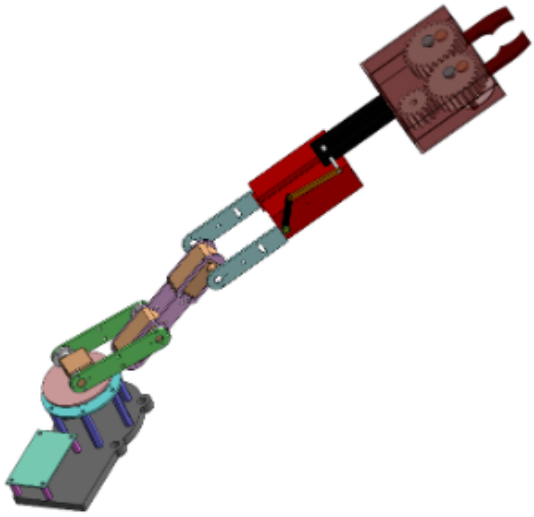


Figure 1.1: Custom designed 6-DOF Robotic Manipulator

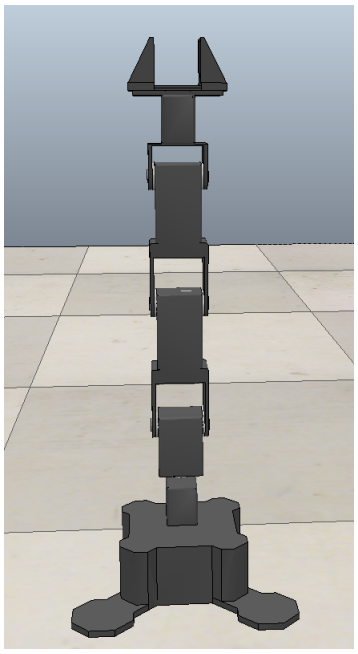


Figure 1.2: Original PhantomX Pincher

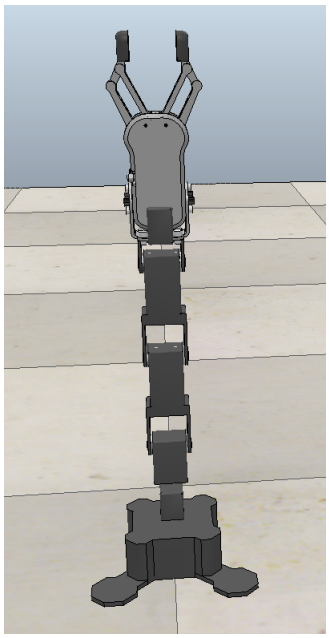


Figure 1.3: Modified PhantomX Pincher

Chapter 2

Modeling of the Robot

2.1 Inverse Kinematics

Inverse kinematics is used to determine the joint angles based on the given end-effector location. Since the last 3 axes of the robot were not coincidental and the robot had less than 6 degrees of freedom, we could not apply kinematic decoupling to simplify the inverse kinematics problem. We decided to use the inverse kinematics solver in MATLAB to find the joint angles.
The inverse kinematics solver in MATLAB requires a URDF file to evaluate which we could not find. All the URDF files for the robot available online were modified to rotate about y_0 axis in the Figure 2.1. One such robot description was available for the PhantomX Pincher arm designed to mount upon the Turtlebot. THe URDF file had to be modified to make the first joint rotate around z_0 axis. The IK was done for the following end-effector transformation:

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.3 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Joint angles: $q = [-0.0012 \quad 0.3787 \quad 2.2840 \quad -1.0927]^T$

2.2 Forward Kinematics

In contrast to inverse kinematics, forward kinematics is used to achieve an end-effector based upon a set of joint variables. Basically, if a set of joint variables are known an end-effector position can be achieved by the robotic manipulator. We did manual evaluation of the forward kinematics for the robot using the DH (Denavit Hartenberg) convention to obtain the DH table given below in Table 1. Moreover, all the frames for the deduction of the DH table are given in Figure 2.1.

Frames	θ_i	d_i	a_i	α_i
0 to 1	θ_1	l_1	0	90
1 to 2	θ_2	0	0	0
2 to 3	θ_3	0	0	0
3 to 3'	$\theta_4 + 90$	0	0	90
3'to 4	0	$l_2 + l_3 + l_4$	0	0

Table 1. DH Table for Forward Kinematics of the robot

The validation of forward kinematics and inverse kinematics of the robot can be done simultaneously by plugging in the joint variables received from inverse kinematics into final transformation matrix deduced by using the DH convention. The link lengths of the robot were obtained from the URDF used to evaluate the inverse kinematics in the previous section.
Link lengths: $L = [0.1318 \quad 0.0519 \quad 0.0519 \quad 0.0464]^T$

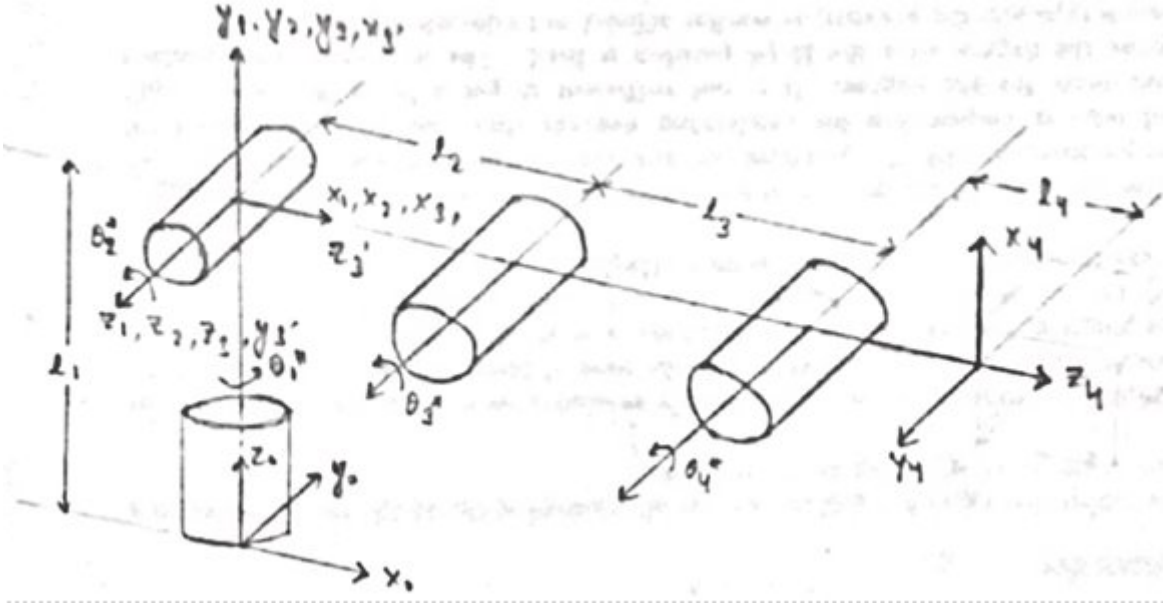


Figure 2.1: Forward Kinematics Frames

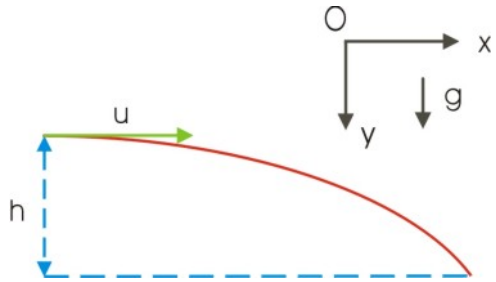


Figure 2.2: Half Projectile Motion

These link lengths and joint angles are put into their respective in the T_4^0 to finally get:

$$\begin{bmatrix} 0.998 & 0.0 & 0.0 & 0.306 \\ 0.0 & 0.998 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.998 & 0.495 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The final transformation matrix obtained is very close to the original matrix with minute errors due to the tolerances set while performing inverse kinematics.

2.3 Force Calculation

Force calculations were done to throw each object to the desired location. In V-REP, the only way to simulate throwing of a dynamic object is by adding force to be object at the end of the path. The end of the path is the point from where is object is to be thrown by the manipulator. Using the equations of motion, we derived the velocities of the object at the initial throwing position. We considered an impulsive force acted on the object to throw it into the basket placed at the desired location.

To simplify the these calculations, we planned the object trajectory such that it followed a half projectile motion as shown below in Figure 2.2.

The coordinates from which the object had to be thrown were: $[x \quad y \quad z] = [-1. \quad 0.425 \quad 0.35]$ and the final end position of the object, that is the location of the basket, was at: $[x \quad y \quad z] = [-1 \quad 0.925 \quad 0.0425]$ Using equations of motions, initial velocity of the object, u, came out out be, 4.427m/s. Now, the force was evaluated based on a V-REP forum response for imparting impulsive forces by using a time interval of 0.002 seconds. The objects mass was 0.125 grams which gives a force of 27.67N using the formula $F = ma$, where $a = u/t$.

Chapter 3

Simulation

The simulation of the project was done using V-REP. We were mainly interested in being able to simulate the scenario of throwing objects. Most of our time was put in learning to work with V-REP and trying the simulate the project. Here is the outline of the steps that we followed to simulate the robotic arm to throw objects:

- Import the CAD Models of the respective robotic elements
- Assemble the gripper onto the manipulator
- Insert appropriate dummies for the the tip of the manipulator
- Insert appropriate dummies for each targets
- Connect tips and targets using the concepts of Inverse Kinematics on V-Rep
- Design the desired path through which the Manipulator must move
- Place the dummies along the ends of the individual paths
- Import the primitive objects to be picked up
- Import and place the baskets to throw the objects into
- Apply force and torque to the object to throw the object into the basket
- Connect all the above mentioned tasks using Lua script
- Return the manipulator tip to the rest position
- Stop the simulation and bring to home position when done

The image shown in Figure 3.1 depicts the final V-REP scene that we simulated to throw 3 objects into 3 different containers. Due to the lack of tutorials, anything apart from pick and place becomes really difficult to learn on the software. We had the idea to properly execute throwing using dynamics. However, V-REP only provides add force and torque functionality to simulate a throwing condition.

The objects are placed randomly at 3 different end-effector positions. The simulation of inverse kinematics part was accomplished easily but the grasping part took a some time and iterations. The most difficult was to simulate throwing. We went for weeks just picking up the object and dropping it instead of throwing. The entire run of the simulation can be accessed on my GitHub repository.

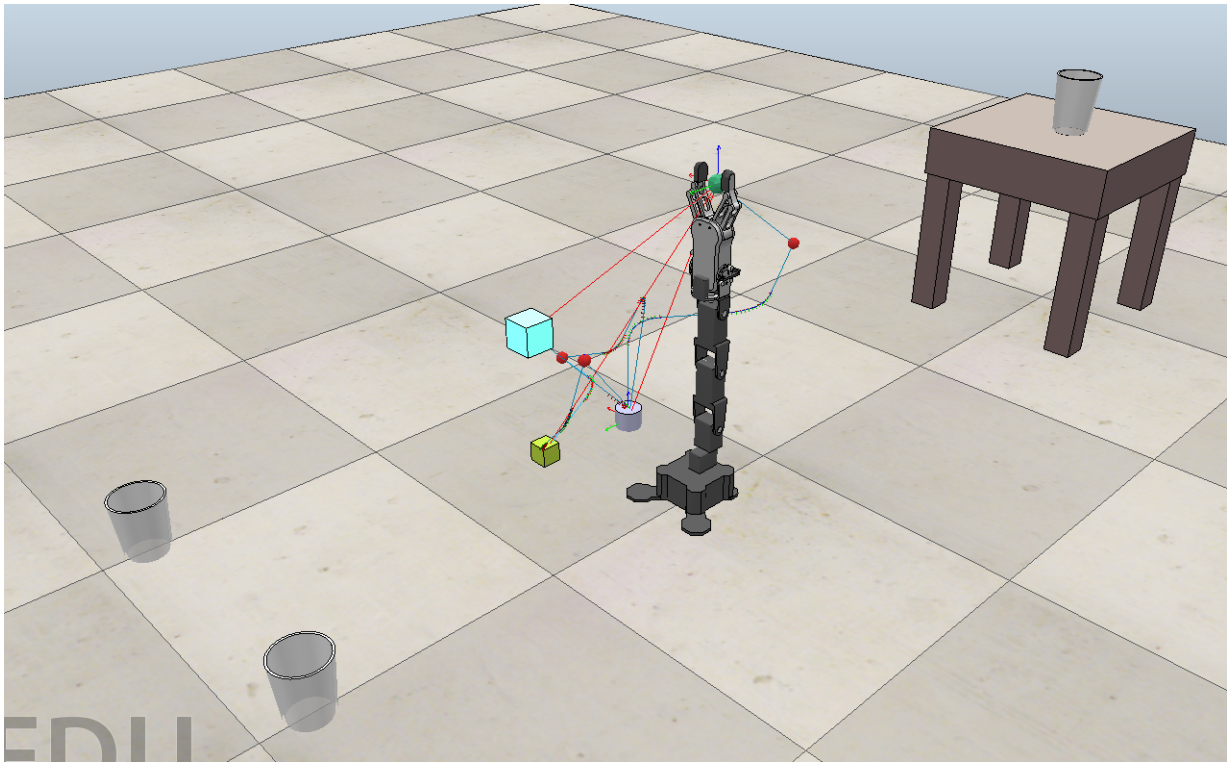


Figure 3.1: V-REP scene to throw 3 objects

Chapter 4

References

Please note that all the references are online and can be accessed via a click.

1. TossingBot
2. V-REP API List
3. Throwing an object in V-REP
4. V-REP Tutorials
5. GitHub Repository

Chapter 5

Appendix

5.1 V-REP Simulation Code

The V-REP code is written in the Lua Script. Various APIs from the standard list provided by Coppelia Robotics was used for guidance and the script was written to perform the simulation. You can use the code below or download the V-REP scene from my github repo:

```
–Initialization of the target variables
target=sim.getObjectHandle('Target')
target2=sim.getObjectHandle('Target2')
target3=sim.getObjectHandle('Target3')
–Initialization of the manipulator Velocity
manipVel=0.5
–Initialization of the gripper handle
gripper=sim.getObjectHandle('RG2_openCloseJoint')
motorVelocity=0.2 – m/s –Initialization of the motor velocity
motorForce=20 – N–Initialization of the motor force
–Initialization of the object sensor
objectSensor=sim.getObjectHandle('RG2_attachProxSensor')
–Initialization of the connector
connector=sim.getObjectHandle('RG2_attachPoint')
–Initialization of the object handles
object=sim.getObjectHandle('throwObject')
object2=sim.getObjectHandle('throwObject2')
object3=sim.getObjectHandle('throwObject3')
–for path variables
path = sim.getObjectHandle("Path")
path1 = sim.getObjectHandle("Path1")
path2 = sim.getObjectHandle("Path2")
path3 = sim.getObjectHandle("Path3")
path4 = sim.getObjectHandle("Path4")
finalReturnPath = sim.getObjectHandle("FinalReturnPath")
throw = true – assigning throw a true boolean value
–for objectThrowPosition
objectThrowPostion1 = -1.0, 0.425, 0.35
objectThrowPostion2 = -0.74, 0.475, 0.275
objectThrowPostion3 = -1.0, -0.045, 0.4
objectThrowOrientation = 0.0, 0.0, 0.0
–regular thread code is executed here
res,err=xpcall(threadedFunction,function(err) return debug.traceback(err) end)
if not res then
sim.addStatusbarMessage('Lua runtime error : '..err)
end
–main function that runs the simulation
```

```
threadedFunction=function()

    –code to signal the simulation to wait for 1 second
    sim.wait(1)
    –to get the position of the target
    position=sim.getObjectPosition(target,-1)
    sim.wait(1) –delay added so that we can see the motion clearly
    – code to move the manipulator to the position
    sim.moveToPosition(target,-1,position,nil,manipVel*0.05)
    sim.wait(1)
    – code to make the manipulator move along the path, named path
    sim.followPath(target,path,0.5,0.015,manipVel*0.4)
    sim.setObjectPosition(object, -1, objectThrowPostion1)
    – Code to set the orientation of the thrown object
    sim.setObjectOrientation(object, -1, objectThrowOrientation)
    sim.addForceAndTorque(object, 0, 27.5, 1.225 , 0.0, 0.0, 0.0)
    –adding the force and torque to throw the object into the basket
    sim.setIntegerSignal('RG2_open',1)
    –change 1 to 0 to close the gripper
    sim.wait(1)
    – to make the manipulator move along path 1
    sim.followPath(target,path1,0.5,0.015,manipVel*0.4)
    –change 1 to 0 to close the gripper
    sim.setIntegerSignal('RG2_open',0)
    –picking up the second cuboid here
    sim.wait(1)
    – to make the manipulator move along path 2
    sim.followPath(target,path2,0.5,0.015,manipVel*0.4)
    sim.setObjectPosition(object2, -1, objectThrowPostion2)
    sim.setObjectOrientation(object2, -1, objectThrowOrientation)
    sim.addForceAndTorque(object2, 16.5, 29.5, 1.225 , 0.0, 0.0, 0.0)
    sim.setIntegerSignal('RG2_open',1)
    – to make the manipulator move along path 3
    sim.followPath(target,path3,0.5,0.015,manipVel*0.4)
    –change 1 to 0 to close the gripper
    sim.setIntegerSignal('RG2_open',0)
    sim.wait(1)
    – to make the manipulator move along path 4
    sim.followPath(target,path4,0.5,0.015,manipVel*0.4)
    sim.setObjectPosition(object3, -1, objectThrowPostion3)
    sim.setObjectOrientation(object3, -1, objectThrowOrientation)
    sim.addForceAndTorque(object3, 0, -20.6, 25.5 , 0.0, 0.0, 0.0)
    –change 1 to 0 to close the gripper
    sim.setIntegerSignal('RG2_open',1)
    sim.wait(1)
    – to make the manipulator move along the final return path
    sim.followPath(target,finalReturnPath,0.5,0.015,manipVel*0.4)
    sim.wait(1)
    sim.addStatusbarMessage('—THE END——')
    sim.wait(1) end
```

5.2 MATLAB Codes

5.2.1 Forward Kinematics Code

Comment the red lines before running the code in MATLAB clear variables;
clc;
Define symbolic variables
syms t1 t2 t3 t4 len1 len2 len3 len4 real
Define tranformation matrices
$T_{01} = [cos(t1)\ 0\ sin(t1)\ 0;\ sin(t1)\ 0\ -cos(t1)\ 0;\ 0\ 1\ 0\ len1; 0\ 0\ 0\ 1]$
$T_{12} = [cos(t2)\ -sin(t1)\ 0\ 0; sin(t1)\ cos(t1)\ 0\ 0; 0\ 0\ 1\ 0; 0\ 0\ 0\ 1];$
$T_{23} = [cos(t3)\ -sin(t3)\ 0\ 0; sin(t3)\ cos(t3)\ 0\ 0; 0\ 0\ 1\ 0; 0\ 0\ 0\ 1];$
$T_{33prime} = [-sin(t4)\ 0\ cos(t4)\ 0; cos(t4)\ 0\ sin(t4)\ 0; 0\ 1\ 0\ 0; 0\ 0\ 0\ 1];$
$T_{3prime4} = [1\ 0\ 0\ 0; 0\ 1\ 0\ 0; 0\ 1\ 0\ len2 + len3 + len4; 0\ 0\ 0\ 1];$
$T_{34} = T_{33}prime * T_3prime_4;$
Get transformation matrices w.r.t base frame
$T_{02} = T_{01} * T_{12}$
$T_{03} = T_{01} * T_{12} * T_{23}$
$T_{04} = T_{01} * T_{12} * T_{23} * T_{34}$

5.2.2 Inverse Kinematics Code

You will have to download the phantomx rst master model from my GitHub repo:
clear variables;
clc;
pincher = importrobot('phantomx$_rst - master/turtlebot_arm_description/urdf/pincher_arm.urdf'$);
$ik = inverseKinematics('RigidBodyTree', pincher);$
$initialguess = pincher.homeConfiguration;$
$tform = [1000.3; 0100.0; 0010.5; 0001];$
$weights = [0.005, 0.05, 0.005, 0.001, 0.001, 0.001];$
[configSoln,solnInfo] = ik('gripper$_link', tform, weights, initialguess$);
for i = 1:4
configSoln(i)
end
show(pincher, configSoln);