THE DEPARTMENT OF AUTOMOTIVE ENGINEERING

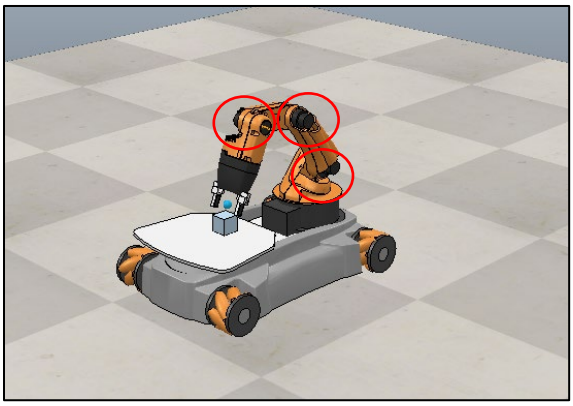
CLEMSON UNIVERSITY

AuE 8220: Autonomy: Mobility and Manipulation, Fall 2022

Homework #4: Mobile-Manipulator Kuka YouBot Control using CoppeliaSim-Matlab Interface

Assigned on: October 13th, 2022, Due: October 21st 2022, 1:00 PM

Problem 1



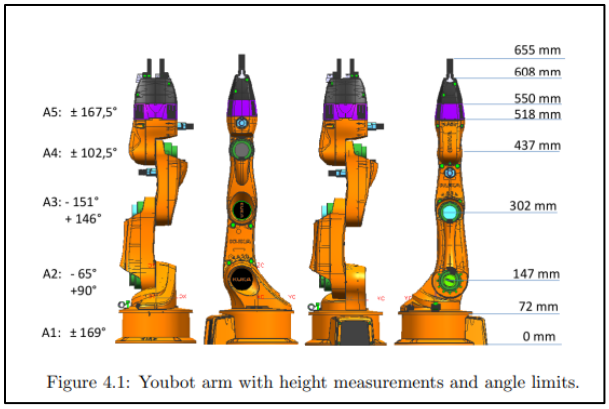


Figure 4.1: Youbot arm with height measurements and angle limits.

Image Source: https://www.merlin.uzh.ch/contributionDocument/download/6684#:~:text=The%20arm%20of%20the%20KUKA,to%200.5%20kg%20in%20weight.

Joints marked on the Kuka Arm in the diagram represent θ_1, θ_2, θ_3 respectively. Base joint and the gripper joints also rotate and, translate but for this assignment you would be working only on planar problems.

1.A) Given a base model of Kuka YouBot as shown, make the **mobile platform** travel 5 m in a straight line at a speed of 0.1 m/s.
1.B) Now, once it reaches the destination, make the **mobile platform** rotate on the spot at an angular rate of 0.5 rad/s.

1.C) For the same model, make the robot Kuka arm end-effector travel in a straight line
1.D) Next, make the Kuka arm end-effector travel a semi-circular trajectory of radius maximum possible radius (Planar condition). (Hint: Refer https://www.researchgate.net/publication/341813494_Low-Cost_Automation_for_Gravity_Compensation_of_Robotic_Arm to get an idea on the kinematics of the arm)

In separate graphs for all problems 1.A through 1.D, plot (i) X vs t (ii) Y vs t (iii) X vs Y. Problems 1.C and 1.D: plot all the joint angles vs t in a single graph for each problem. You can use CoppeliaSim or Matlab to obtain your graphs.

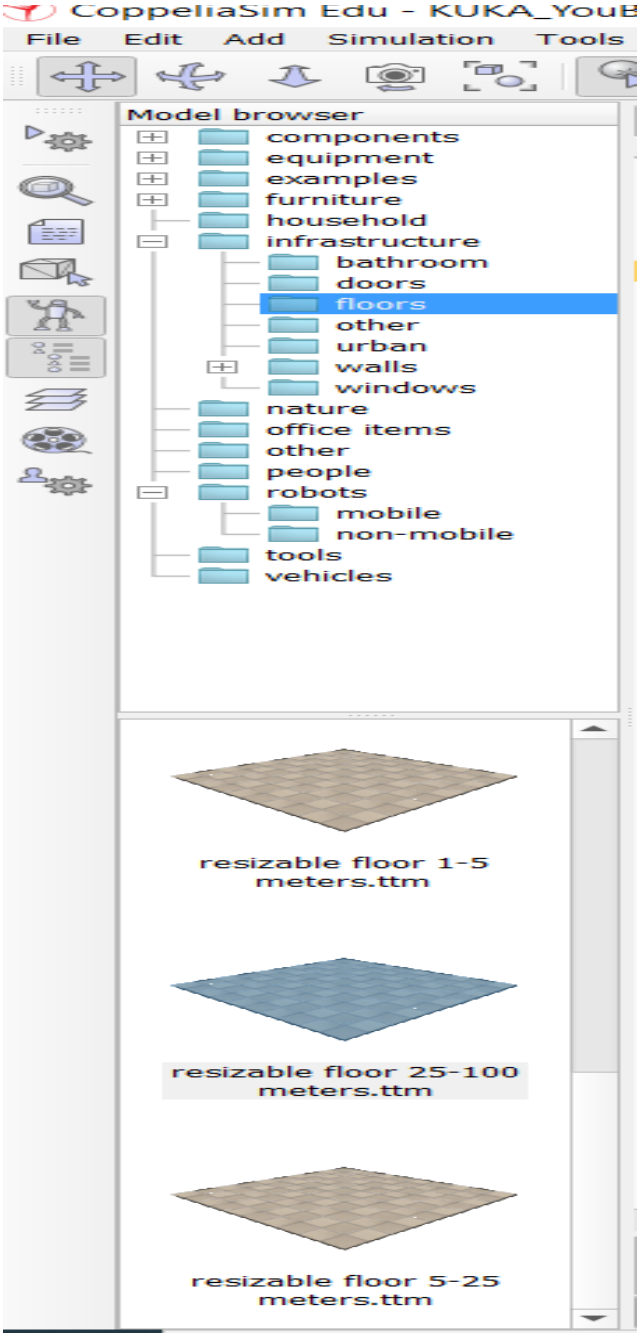
Bonus points: Additional bonus points would be awarded if you can integrate the use of Peter Corke's Robotics Toolbox functions.

Solution:

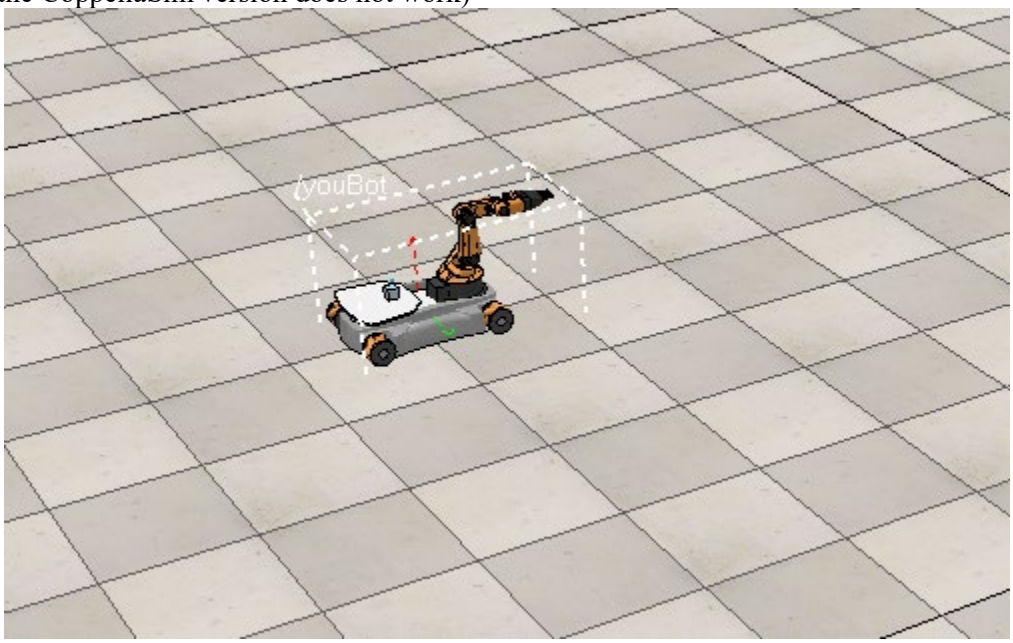
Step 1:
Open CoppeliaSim and add this line in the main script:
simRemoteApi.start(19999)

This ensures connection of Matlab with CoppeliaSim

Step 2:
Import resizable floor and robot as shown:



Similarly go to robots -> mobile -> Kuka YouBot (Note: grab the model from the older Vrep versions and load it as the CoppeliaSim version does not work)



Step 3:

Add these lines to Matlab script at the start to establish a connection that enables to send command from Matlab to CoppeliaSim.

```
%Basic commands to connect with vrep-matlab
vrep = remApi('remoteApi');
vrep.simxFinish(-1);

clientID = vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
```

Step 4:

Get joint handles as per the given names or your own custom names:

```
if (clientID>-1) % To verify the connection
    disp('Connected');
    % Getting all the required joint angles
    [r, floor] = vrep.simxGetObjectHandle(clientID, 'World',
vrep.simx_opmode_blocking);
    [r, jo] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint0',
vrep.simx_opmode_blocking);
    [r, j1] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint1',
vrep.simx_opmode_blocking);
    [r, j2] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint2',
vrep.simx_opmode_blocking);
    [r, j3] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint3',
vrep.simx_opmode_blocking);
    [r, j4] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint4',
vrep.simx_opmode_blocking);
    [r, gripper1] = vrep.simxGetObjectHandle(clientID, 'youBotGripperJoint1',
vrep.simx_opmode_blocking);
```

```
    [r, gripper2] = vrep.simxGetObjectHandle(clientID, 'youBotGripperJoint2',
vrep.simx_opmode_blocking);
    [r, w1] = vrep.simxGetObjectHandle(clientID, 'rollingJoint_fr',
vrep.simx_opmode_blocking);
    [r, w2] = vrep.simxGetObjectHandle(clientID, 'rollingJoint_fl',
vrep.simx_opmode_blocking);
    [r, w3] = vrep.simxGetObjectHandle(clientID, 'rollingJoint_rr',
vrep.simx_opmode_blocking);
    [r, w4] = vrep.simxGetObjectHandle(clientID, 'rollingJoint_rl',
vrep.simx_opmode_blocking);
    [~, robot] = vrep.simxGetObjectHandle(clientID, 'youBot',
vrep.simx_opmode_blocking);
```

Step 5:

Calculate linear and angular velocity as per the mobile base dimensions (The velocity calculations differ for Mecanum wheels)

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{R} \begin{bmatrix} 1 & 1 & -(L_1 + L_2) \\ 1 & -1 & (L_1 + L_2) \\ 1 & -1 & -(L_1 + L_2) \\ 1 & 1 & (L_1 + L_2) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_0 \end{bmatrix} \quad (8)$$

```
% Putting while condition so that the commands are continuously looped
    w = 0.5*300.46*1e-3; % Half-track width (m)
    l = 0.5*471*1e-3; % Half-wheelbase (m)
    r = 0.05;
    omega = 0.5;
    v_x = 0;
    v_y = 0;
    u = 1/r*[1,1,l+w;1,-1,-l-w;1,-1,l+w;1,1,-l-w]*[v_x;v_y;omega];
```

Step 6:

Sending the desired wheel angular rates as per the command apis:

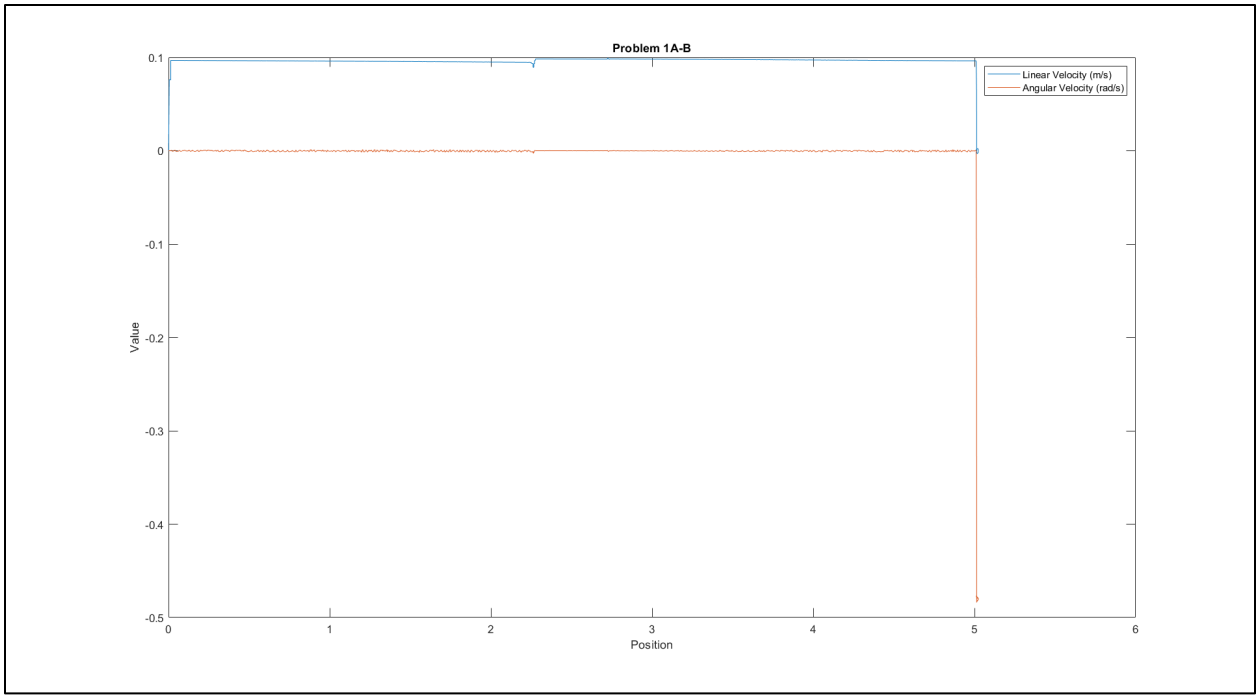
```
i = 0;
    while true
        % Position of Youbot wrt world frame
        i = i+1;
        [~,robot_linvel,robot_angvel] =
vrep.simxGetObjectVelocity(clientID,robot,vrep.simx_opmode_streaming);
        [r, bot_position] = vrep.simxGetObjectPosition(clientID,robot,-
1,vrep.simx_opmode_streaming);
        if bot_position(2) <=5
            [r] = vrep.simxSetJointTargetVelocity(clientID, w1, -2,
vrep.simx_opmode_streaming);
            [r] = vrep.simxSetJointTargetVelocity(clientID, w2, -2,
vrep.simx_opmode_streaming);
            [r] = vrep.simxSetJointTargetVelocity(clientID, w3, -2,
vrep.simx_opmode_streaming);
```

```
            [r] = vrep.simxSetJointTargetVelocity(clientID, w4, -2,
vrep.simx_opmode_streaming);
            vel(i) = robot_linvel(2);
            pos(i) = bot_position(2);
            ang(i) = robot_angvel(1);

        else
            [r] = vrep.simxSetJointTargetVelocity(clientID, w1, u(1),
vrep.simx_opmode_streaming);
            [r] = vrep.simxSetJointTargetVelocity(clientID, w2, u(2),
vrep.simx_opmode_streaming);
            [r] = vrep.simxSetJointTargetVelocity(clientID, w3, u(3),
vrep.simx_opmode_streaming);
            [r] = vrep.simxSetJointTargetVelocity(clientID, w4, u(4),
vrep.simx_opmode_streaming);
            vel(i) = robot_linvel(2);
            pos(i) = bot_position(2);
            ang(i) = robot_angvel(1);

            %Rotating base at an angular velocity of 0.5 rad/s, jittery
            %movements because of vrep-matlab interface
            %[r] = vrep.simxSetJointTargetVelocity(clientID, w1, u(1),
vrep.simx_opmode_streaming);
            %[r] = vrep.simxSetJointTargetVelocity(clientID, w2, u(2),
vrep.simx_opmode_streaming);
            %[r] = vrep.simxSetJointTargetVelocity(clientID, w3, u(3),
vrep.simx_opmode_streaming);
            %[r] = vrep.simxSetJointTargetVelocity(clientID, w4, u(4),
vrep.simx_opmode_streaming);
        end
    end
end
```

Following are the results obtained:



The readings won't be exact due to motion disturbance in the system.

Problem 1 C-D

For enabling the arm rotation we make use of Peter Corke's toolbox functions to get inverse kinematics of the system.

We are concerned only with a planar condition so just links 2-4 of the Kuka Arm are of interest.

Step 1:

Define link using PCRTB.

```
% Concerned with links 1-3 only
% Using PCRTB
L(1) = Link([0 0 0.155 0]);
L(2) = Link([0 0 0.135 0]);
L(3) = Link([0 0 0.2175 0]);
Kukamanip = SerialLink(L, 'name', 'Planar KUKA youBot Manipulator');
```

Step 2:

Defining x and y coordinates for both straight line and circular trajectories:

```
theta = 0:1:180;
%x = 0.485*cosd(theta); % For semi-circular trajectory

%z_circ = 0.485*sind(theta);
youBotManipulator = SerialLink(L, 'name', 'Planar KUKA youBot Manipulator');
qz = [0 pi/2 -pi/2 0];
%youBotManipulator.plot(qz);
z = 0.4008:-0.01:0;
```

```
x = 0.35*ones(length(z));% For straight line trajectory
theta2 = zeros(length(z)); % Pre defining vectors is computationally more efficient
theta3 = zeros(length(z));
theta4 = zeros(length(z));
```

Looping through each point and using ikine function to find joint angles for the trajectory required:

```
for i=1:length(x)
    T = transl(x(i),0,z(i));
    q = youBotManipulator.ikine(T,'q0',qz,'mask',[1 1 1 0 0 0]);
    T2 = youBotManipulator.fkine(q);
    p = transl(T2);
    youBotManipulator.plot(q);
    hold on
    x_ee(i) = p(1);
    z_ee(i) = p(3);
    theta2(i) = q(2);
    theta3(i) = q(3);
    theta4(i) = q(4);
end
```

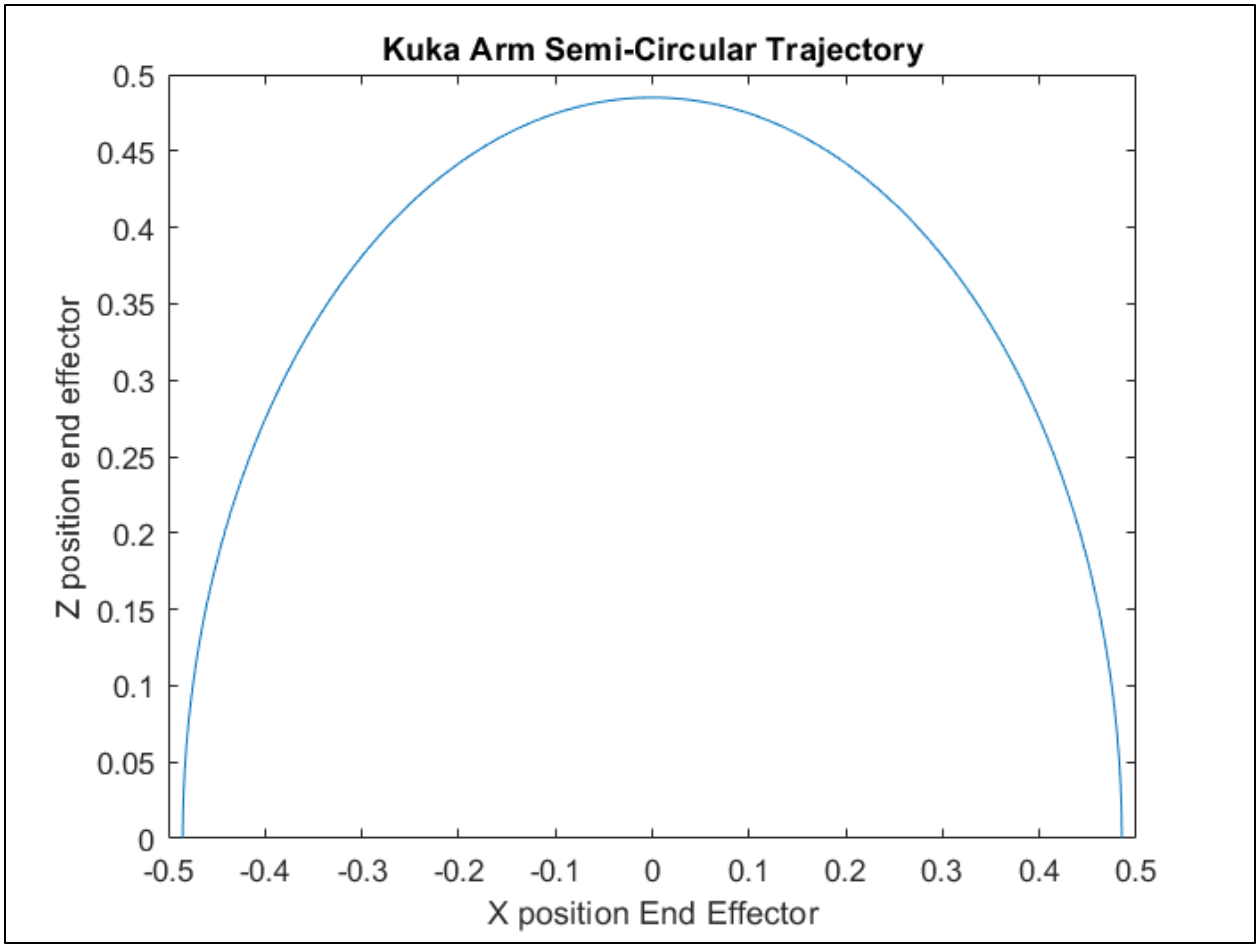
Step 3:
Applying vrep-matlab api functions to give joint anlge commands calculated previously using ikine.

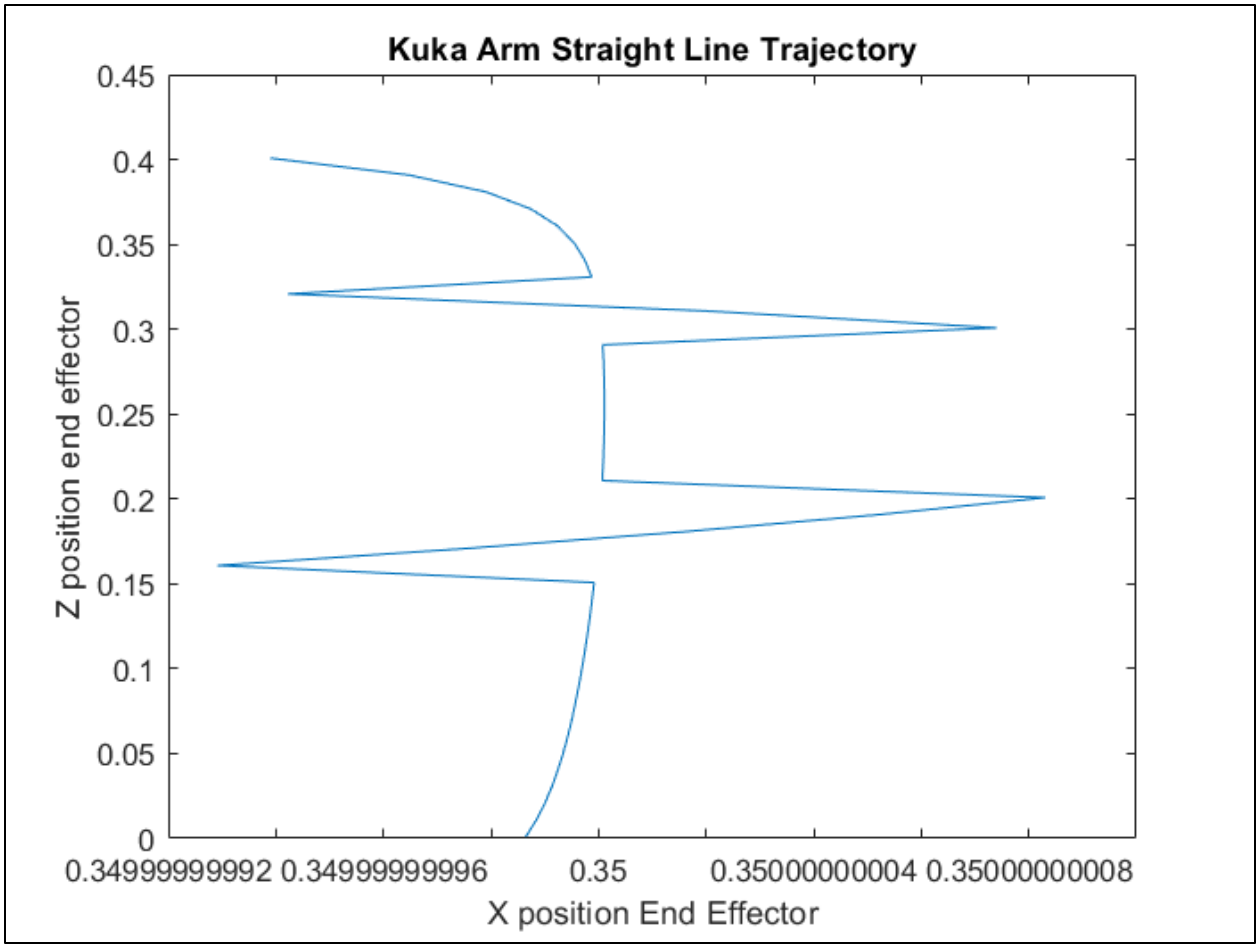
```
vrep = remApi('remoteApi');
vrep.simxFinish(-1);
clientID = vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
if (clientID>-1)
    disp('Connected');
    [r, jo] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint0',
vrep.simx_opmode_blocking);
    [r, j1] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint1',
vrep.simx_opmode_blocking);
    [r, j2] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint2',
vrep.simx_opmode_blocking);
    [r, j3] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint3',
vrep.simx_opmode_blocking);
    [r, j4] = vrep.simxGetObjectHandle(clientID, 'youBotArmJoint4',
vrep.simx_opmode_blocking);
    [r, gripper1] = vrep.simxGetObjectHandle(clientID, 'youBotGripperJoint1',
vrep.simx_opmode_blocking);
    [r, gripper2] = vrep.simxGetObjectHandle(clientID, 'youBotGripperJoint2',
vrep.simx_opmode_blocking);
    [r, w1] = vrep.simxGetObjectHandle(clientID, 'rollingJoint_fr',
vrep.simx_opmode_blocking);
    [r, w2] = vrep.simxGetObjectHandle(clientID, 'rollingJoint_fl',
vrep.simx_opmode_blocking);
    [r, w3] = vrep.simxGetObjectHandle(clientID, 'rollingJoint_rr',
vrep.simx_opmode_blocking);
    [r, w4] = vrep.simxGetObjectHandle(clientID, 'rollingJoint_rl',
vrep.simx_opmode_blocking);
```

```
    [~, robot] = vrep.simxGetObjectHandle(clientID, 'youBot',
vrep.simx_opmode_blocking);
    z1 = zeros(length(z_circ));
    x1 = zeros(length(z_circ));
    y1 = zeros(length(z_circ));
    [~] = vrep.simxSetJointPosition(clientID, jo, 0, vrep.simx_opmode_streaming);
    [~] = vrep.simxSetJointPosition(clientID, j1, pi/2, vrep.simx_opmode_streaming);
    [~] = vrep.simxSetJointPosition(clientID, j2, 0, vrep.simx_opmode_streaming);
    [~] = vrep.simxSetJointPosition(clientID, j3, 0, vrep.simx_opmode_streaming);
    [~] = vrep.simxSetJointPosition(clientID, j4, 0, vrep.simx_opmode_streaming);
    for i = 1:length(z1)
        [~] = vrep.simxSetJointPosition(clientID, jo, 0, vrep.simx_opmode_streaming);
        [~] = vrep.simxSetJointPosition(clientID, j1, -theta2(i)+pi/2,
vrep.simx_opmode_streaming);
        [~] = vrep.simxSetJointPosition(clientID, j2, -theta3(i),
vrep.simx_opmode_streaming);
        [~] = vrep.simxSetJointPosition(clientID, j3, -theta4(i),
vrep.simx_opmode_streaming);
        [~] = vrep.simxSetJointPosition(clientID, j4, 0, vrep.simx_opmode_streaming);
        [~, endeffector_pos] = vrep.simxGetObjectPosition(clientID,gripper1,-
1,vrep.simx_opmode_streaming);
        z1(i) = endeffector_pos(3);
        x1(i) = endeffector_pos(1);
        y1(i) = endeffector_pos(1);

    end
end
```

Note: Depending upon the orientation, joint angles would either be negative or positive. Also, CoppeliaSim and Matlab world-frame orientations defaults are different hence pi/2 has been added to match them. Following are the results:


Kuka Arm Semi-Circular Trajectory
Z position end effector
X position End Effector
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5



The disturbances occur due to ikine using a numerical solution approach.
Joint angles of interest can now be extracted in order to analyze joint accelerations as well.