Kuka KR210 Pick and Place Project

Introduction

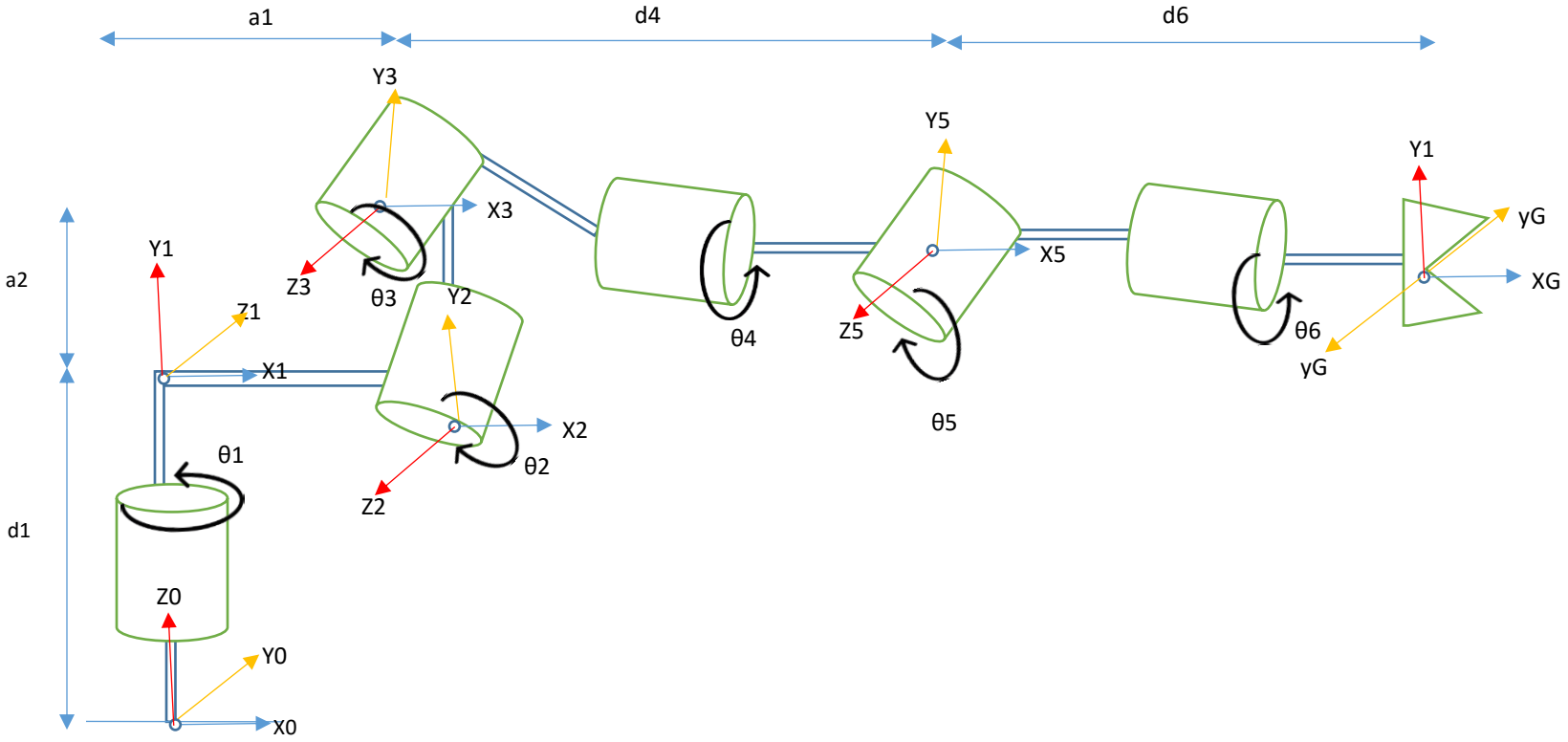
This is the project two (2) of Nanodegree Robotics Software Engineering at Udacity. The project uses Gazebo Kuka KR210 serial manipulator with six (6) degree of freedom (6 DoF) to pick up a target cylinder from a shelf and place it into a bin. This project was achieved using forward and inverse kinematics approach. In forward kinematics, basically we know all the joint angles and use them to calculate the position and orientation of the end-effector relative to the base frame. While inverse kinematics is the opposite of the forward kinematics where the position of the end effector is known and objective is to calculate the joint angle of the manipulator.

Denavit Hartenberg Method:

This is kind of a shortcut method of finding homogenous transform matrix. Denavit Hartenbeg is kind of industry standard that is faster than rotation method.

Denavit Hartenberg Diagram

This diagram shows how the links and the joints of Kuka KR210 arm are connected together based on Denavit Hartenberg convention. The arm consists of six (6) revolute joints. The first three joints determine the position of end the effector while seconds three of joints make up spherical wrist and determine the orientation of the end effector.



Denavit Hartenberg Parameters

Each row in DH parameter table is to transform from link i-1 frame to i frame. I have used it to create homogenous transformation matrices between the joints of the Kuka KR2010 arm.

Values from urdf file

	X	**y**	**z**
J0	0	0	0.33
J1	0.35 0	0	0.42
J2	0	0	1.2
J3	0.96	0	-0.54
J4	0.54	0	0
J5	0.193	0	0

Finding Link lengths and link offsets

Joint1
- a0 = 0; since it is base link
- d1 = link_2(Z) = 0.42+0.33= 0.75

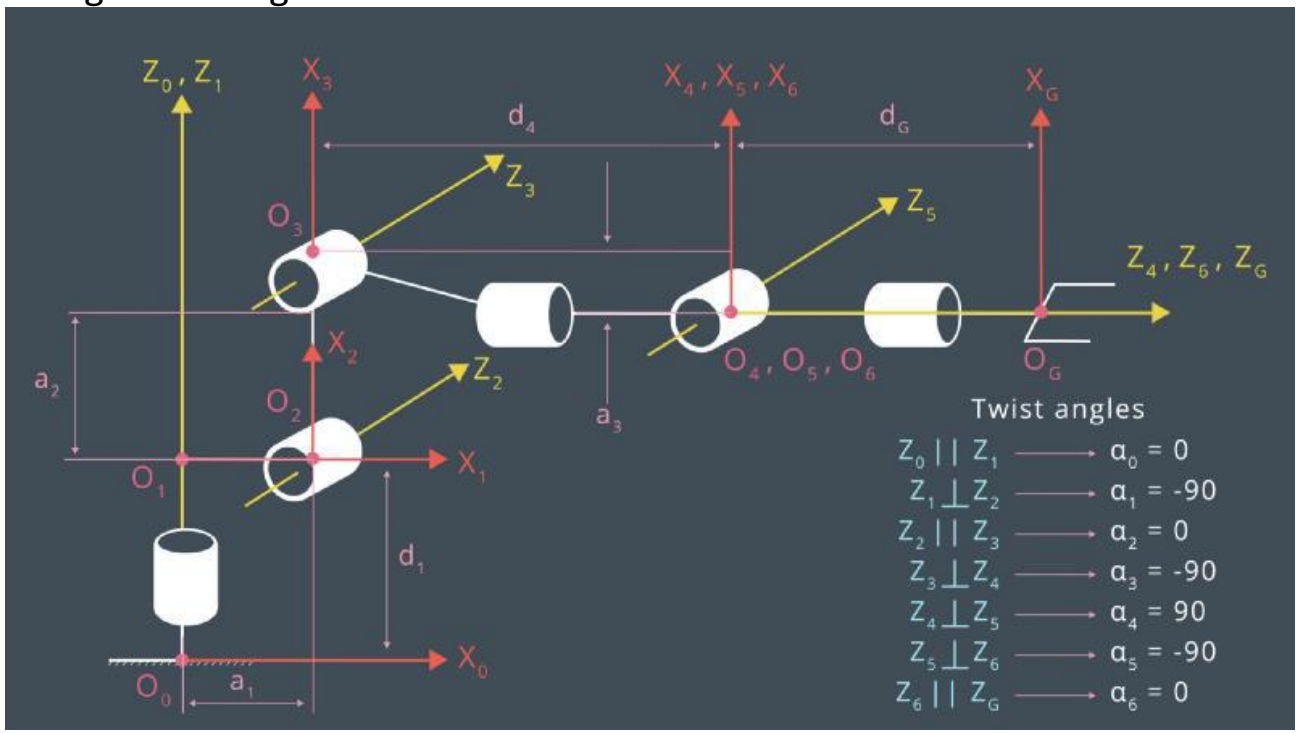
joint2
- a1= link_2(x) = 0.35
- d2 = 0; sinxe x1 and x2 are perpendicular

joint3

- a2 = link_3(z) = 1.25
- d3=0; since x2 and x3 are perpendicular

Finding twist angles

Twist angles are given below:



```
# Create Modified DH parameters
DH = {
alpha0: 0, a0: 0, d1: 0.75, q1: q1,
alpha1: -pi/2, a1: 0.35, d2: 0, q2: q2 - pi/2,
alpha2: 0, a2: 1.25, d3: 0, q3: q3,
alpha3: -pi/2, a3: -0.054, d4: 1.5, q4: q4,
alpha4: pi/2, a4: 0, d5: 0, q5: q5,
alpha5: -pi/2, a5: 0, d6: 0, q6: q6,
alpha6: 0, a6: 0, d7: 0.303, q7: 0}
```

Homogenous Transformations

Here, I defined homogeneous transform from i-1 to frame i. Each link composed of four individual transforms, two rotations and two translations performed in order.

$$ {}^{i-1}_{i}T = R_X(\alpha_{i-1})\, D_X(a_{i-1})\, R_Z(\theta_i)\, D_Z(d_i) $$

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Here, I defined a method that perform the homogeneous transforms between the adjacent links. The method accept q, d, a, and alpha for joint angle, link offset, link length and twist angle between the joints from each row in the DH table and return the transform matrix.

```
    ### Creates Homogeneous Transform Matrix from DH parameters
    def homogeneous_transform(q, d, a, alpha):
        TM = Matrix(
    [[cos(q),            -sin(q),           0,             a          ],
    [ sin(q)*cos(alpha), cos(q)*cos(alpha), -sin(alpha), -sin(alpha)*d],
    [ sin(q)*sin(alpha), cos(q)*sin(alpha),  cos(alpha),  cos(alpha)*d],
    [                 0,                 0,           0,             1]]
        )
       return TM
```

Individual joints transformation matrices

This is the method call for the individual joints transformation matrices, we only need to build the homogeneous transformation with the symbols and then substitute them with the corresponding DH parameters:

```
# Create individual transformation matrices
T0_1 = homogeneous_transform(q1, d1, a0, alpha0).subs(DH)
T1_2 = homogeneous_transform(q2, d2, a1, alpha1).subs(DH)
T2_3 = homogeneous_transform(q3, d3, a2, alpha2).subs(DH)
T3_4 = homogeneous_transform(q4, d4, a3, alpha3).subs(DH)
```

```
T4_5 = homogeneous_transform(q5, d5, a4, alpha4).subs(DH)
T5_6 = homogeneous_transform(q6, d6, a5, alpha5).subs(DH)
T6_G = homogeneous_transform(q7, d7, a6, alpha6).subs(DH)
```

The transformation matrix can calculated by substituting the DH parameters from the above table above into this function:

```
T0_1 = [[cos(θ1),      -sin(θ1),    0,     0],
       [sin(θ1),       cos(θ1),    0,     0],
       [       0,              0,    1,  0.75 ],
       [       0,              0,    0,     1]]
```

```
T1_2 = [[sin(θ2),       cos(θ2),    0,      0.35],
        [      0,             0,    1,         0],
        [cos(θ2),      -sin(θ2),    0,         0],
        [       0,             0,    0,         1]]

T2_3 = [[cos(θ3),      -sin(θ3),    0,      1.25],
        [sin(θ3),       cos(θ3),    0,         0],
        [      0,             0,    1,         0],
        [      0,             0,    0,         1]]

T3_4 = [[cos(θ4,       -sin(θ4),    0,     -0.054],
        [     0,             0,    1,         1.5],
        [-sin(θ4),     -cos(θ4),    0,           0],
        [      0,             0,    0,           1]]
```

```
T4_5 = [[cos(θ5,       -sin(θ5),     0,       0],
       [     0,              0,    -1,       0],
       [sin(θ5,        cos(θ5),     0,       0],
       [      0,             0,      0,       1]]
```

```
T5_6 = [[cos(θ6),       -sin(θ6),    0,     0],
       [      0,              0,    1,     0],
       [ -sin(θ6),     -cos(θ6),     0,     0],
       [       0,              0,      0,     1]]
```

Lastly, this is the transformation matrix of the end effector. I has no rotation but it has an offset in the Z direction.

```
T6_G = [[  1,         0,    0,    0],
       [   0,         1,    0,    0],
       [   0,         0,    1,    0.303],
       [   0,         0,    0,    1]]
```

Generalized homogeneous transform

I finally created the overall transform from the base frame to the end effector by composing individual link transform.

$$ {}^{0}_{N}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T \ldots {}^{N-1}_{N}T $$

```
# Generalized homogeneous transform
T0_G = T0_1 * T1_2 * T2_3 * T3_4 * T4_5 * T5_6 * T6_G.
```

```
T0_G = [[1.0,   0,   0, 2.153],
        [0,   1.0,   0,     0],
        [0,     0,   1.0, 1.946],
        [0, 0,  0,          1]]
```

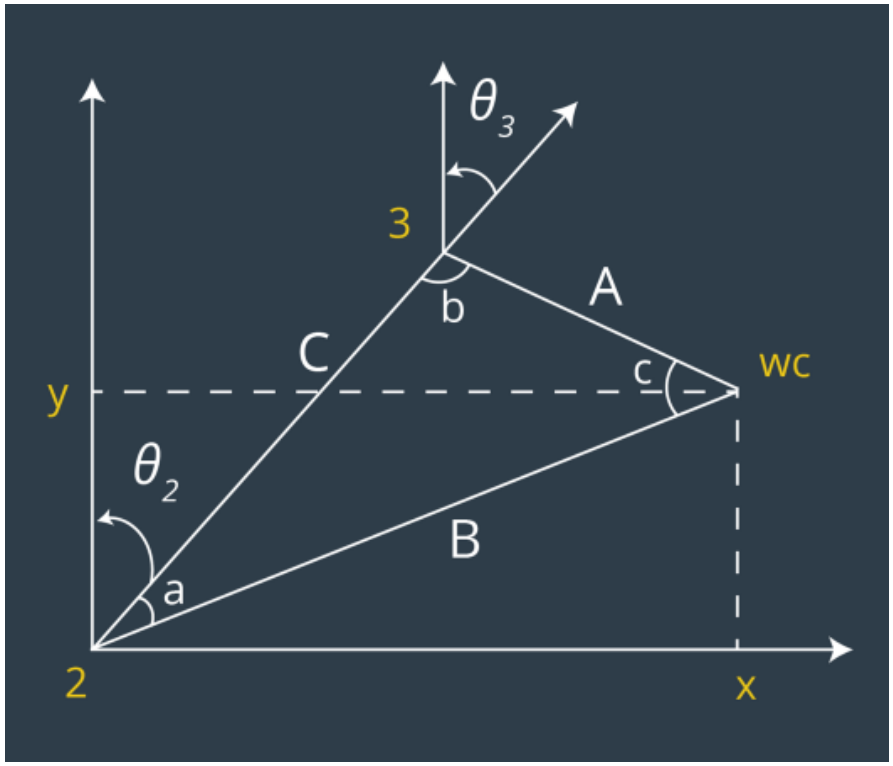
This indicates that with neutral joint angles, the gripper is at location 2.153 in the x-axis, 0 in the y-axis, and 1.946 in the z-axis.

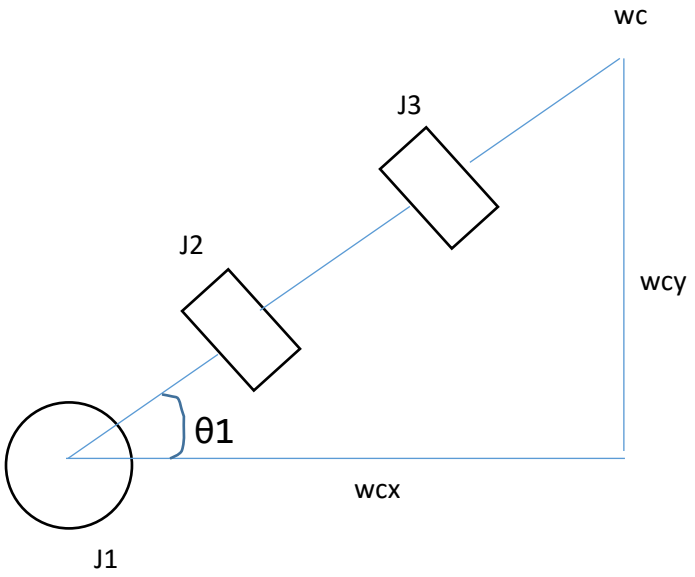
This matrix includes a correction that account for orientation difference between the gripper and the arm’s base. This correction required rotating the gripper’s coordinate frame around the z-axis by 180 degrees and around y-axis by -90 degrees.

Inverse Kinematic Calculations

Inverse kinematics is the opposite of the forward kinematics where position and orientation of end effector is known, the goal is to find joints angles of the manipulator. Inverse kinematics problem is more difficult than the forward kinematics and is much more useful to us roboticist because typically this is the kind problem we need to solve with our robot. For example, in this Kuka KR210 Pick and Place project, we know that we need the gripper to become located where cylinder is on shelf. In order to place our end effector in our targeted location we need to figure out what joint angles will place the end effector in that location. In fact, inverse kinematics problem is so difficult that we only solve it if make a couple of assumptions; the first three joints determine the position of the end effector and the second three joints of the manipulator determine the orientation of end effector and yet the second three joints make up the spherical joints. In addition, the inverse kinematics problem was resolved analytically rather than numerically by finding the first 3 joint angles from the pose position and also the remaining 3 wrist joint angles from the pose orientation as demonstrated below.

Build Inverse Kinematics





θ1 = atan2(wcy,wcx)

Here is the code the code that performed the magic of inverse kinematics:

Here I have created rotation matrix to correct the difference between the URDF and the DH reference frames for the end-effector by a rotation by 180^0 follow by -90 on the z-axis and y-axis respectively.

```
# Compensate for rotation discrepancy between DH parameters and Gazebo
rot_corr = R_z.subs(y, pi) * R_y.subs(p, -pi/2)
```

Here I calculated joint angles

```
# Calculate joint angles using Geometric IK method
# Calculating the Rotation Matrix for the Gripper
r, p, y = symbols('r p y')
rot_x = rotate_x(r)
rot_y = rotate_y(p)
rot_z = rotate_z(y)
Rrpy = rot_z * rot_y * rot_x * rot_corr
Rrpy = Rrpy.subs({'r': roll, 'p': pitch, 'y': yaw})

# Calculate the wrist center position
nx = Rrpy[0,2]
ny = Rrpy[1,2]
nz = Rrpy[2,2]
```

```
# d7: 0.303
wx =  px - 0.303 * nx
wy =  py - 0.303 * ny
wz =  pz - 0.303 * nz
```

```
# Theta 1 from above is a simple atan2 in x and y
theta1 = atan2(wy, wx)
# Calculate Radius from above (will be used later)
r = sqrt(wx**2+wy**2) - 0.35 # a1: 0.35
```

```
# Calculating Theta 2 and 3 using cosine law. A, B and C are sides of the triangle
A = 1.5014 # d4
B = sqrt(r**2+(wz-0.75)**2) # d1: 0.75
C = 1.25 # a2

# a corresponds to angle alpha
a = acos((B**2 + C**2 - A**2) / (2*B*C))
theta2 = pi/2 - a - atan2(wz-0.75, r) # d1: 0.75

# b corresponds to angle beta
b = acos((A**2 + C**2 - B**2) / (2*A*C))
theta3 = pi/2 - (b + 0.036)
```

```
# Calculating Euler angles from orientation
R0_3 = T0_1[0:3,0:3] * T1_2[0:3,0:3] * T2_3[0:3,0:3]
R0_3 = R0_3.evalf(subs={'q1':theta1, 'q2':theta2, 'q3':theta3})
# R3_6 = R0_3.inv("LU")*Rrpy # calculate inverse of R0_3:
```

After finding R3_6 I used below matrix to substitute for theta4, theta5 and theta6. However, I used tangent because is more robust than cosine and sine.

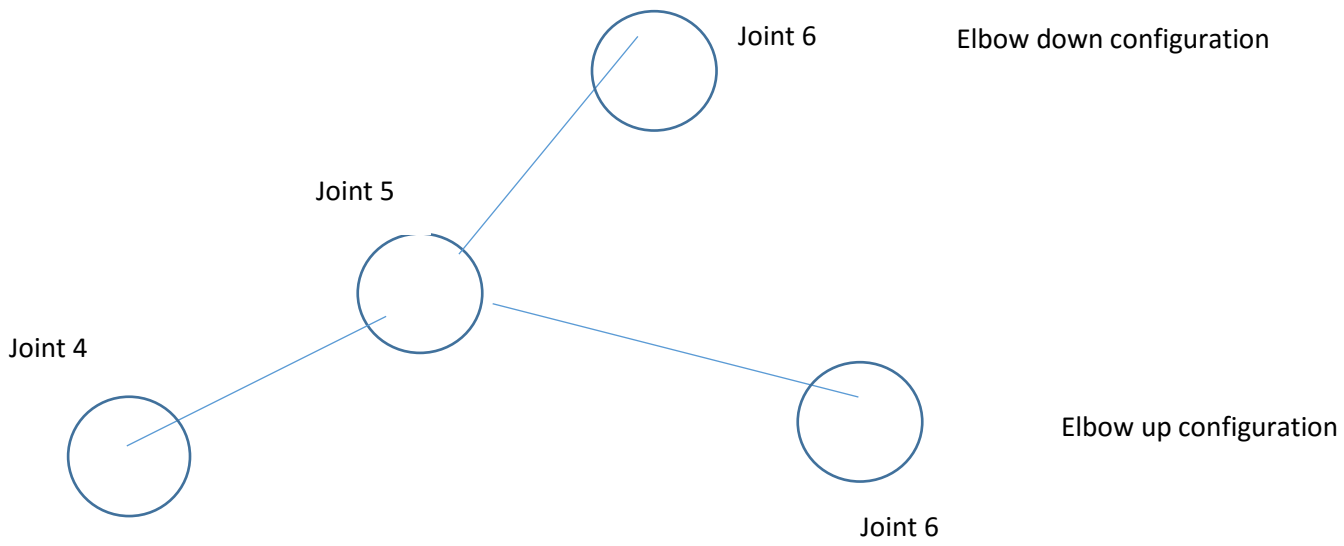
```
R3_6 = [-S04C05C06 – C04S06     S04C05C06 – C04S06       -S04S05

         C04C05C06 – S04S06   C04C05S06 – S04C06          C04S05

                  -S05C06        S05S06 + C05                C05]
```

```
theta4 = atan2(R3_6[2,2], -R3_6[0,2])
theta5 = atan2(sqrt(R3_6[0,2]**2 + R3_6[2,2]**2), R3_6[1,2])
theta6 = atan2(-R3_6[1,1], R3_6[1,0])
```

In addition I used condition while choosing theta 4 and 6 values because there will be multiple solution depending whether elbow is down or up of the spherical wrist as illustrated in below diagram:



Running the Simulator

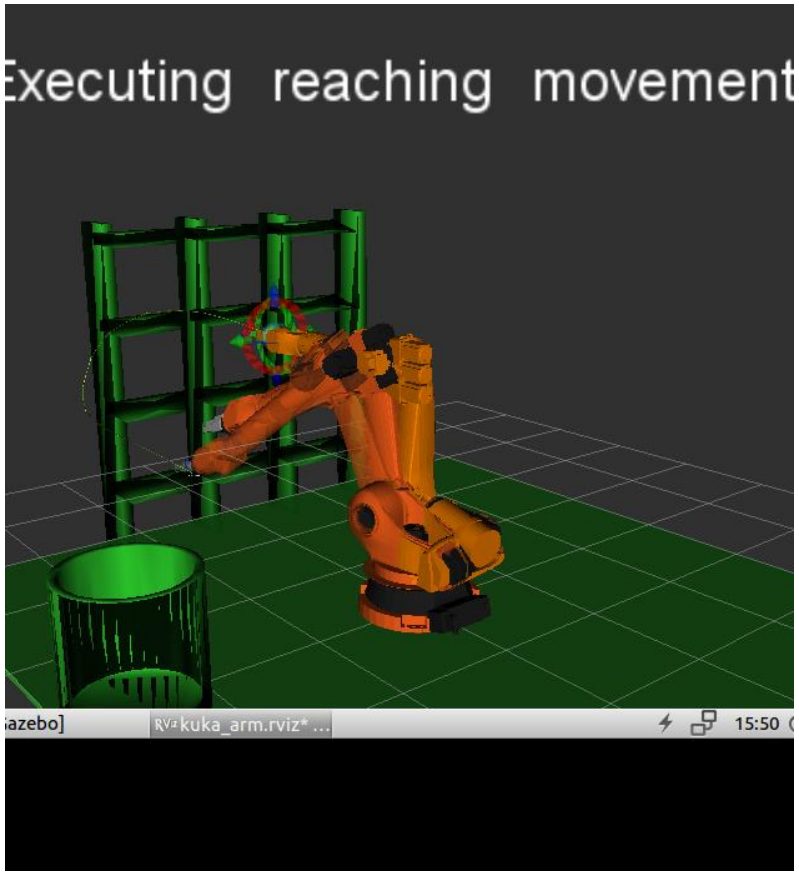
To run the IK_server.py code, change demo flag to false in *inverse_kinematics.launch file under ~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/launch/*

```
$ cd ~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/scripts
$ ./safe_spawner.sh
```

To run the inverse kinematic code, run the following code in a new terminal:

```
$ rosrun kuka_arm IK_server.py
You must now press Next in the Rviz window to have each step proceed.
```


Reached target location
azebo]
kuka_arm.rviz* ...
15:49


Executing reaching movement
azebo]
kuka_arm.rviz* ...
15:50


aying plan to drop-off locatic
azebo
kuka_arm.rviz* ...
15:54


Reached drop-off location
azebo]
kuka_arm.rviz* ...
15:55


Releasing target object
azebo]
kuka_arm.rviz* ...
15:56

References:

Fernando Jaruche Nunes(Oct 11, 2017) Udacity Robotics ND Project 2 – Robotic Arm: Pick & Place. Retrieve from **https://medium.com/@fernandojaruchenunes/udacity-robotics-nd-project-2-robotic-arm-pick-place-71af9c9ba519**

Pick & Place Walkthrough. Retrieve from **https://docs.google.com/presentation/d/1b_3jjBGiQnQYG_q6X2BCusohP7zUdrcyu_XSJE5ozT4/edit?usp=sharing**