In the handle_calculate function, I first implemented the forward kinematics code by defining the four DH symbols and creating the DH matric based on the kuka robot.

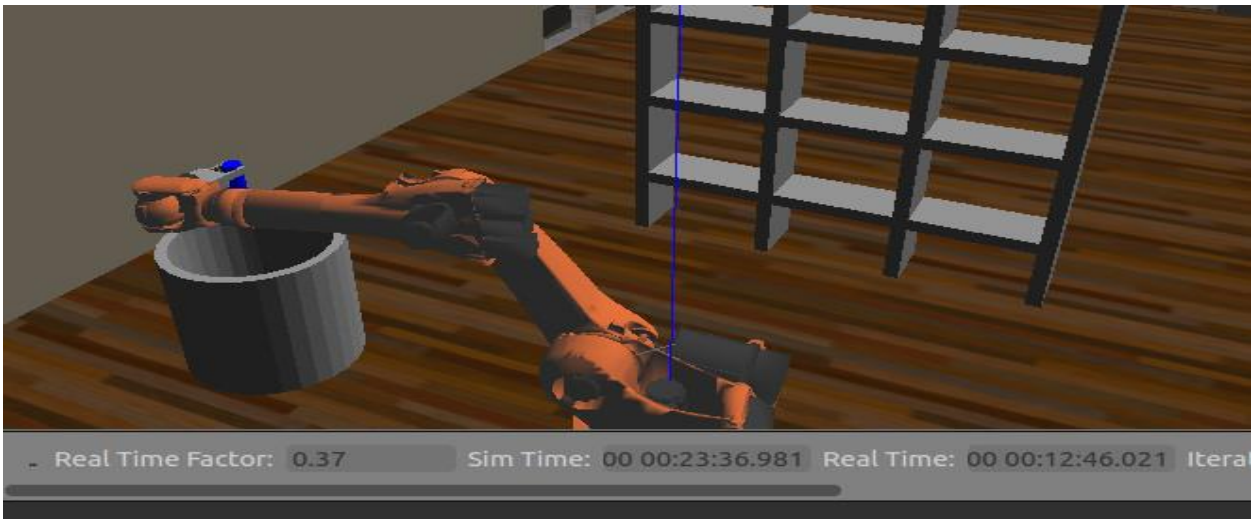
```
22 def handle_calculate_IK(req):
23     rospy.loginfo("Received %s eef-poses from the plan" % len
   (req.poses))
24     if len(req.poses) < 1:
25         print "No valid poses received"
26         return -1
27     else:
28         joint_trajectory_list = []
29         ### Your FK code here
30         # Create symbols
31         q1, q2, q3, q4, q5, q6, q7 = symbols('q1:8')
32         d1, d2, d3, d4, d5, d6, d7 = symbols('d1:8')
33         a0, a1, a2, a3, a4, a5, a6 = symbols('a0:7')
34         alpha0, alpha1, alpha2, alpha3, alpha4, alpha5, alpha6 =
   symbols('alpha0:7')
35
36         # Create Modified DH parameters
37         s = {alpha0:     0,   a0:      0,  d1:  0.75, q1: q1,
38              alpha1: -pi/2,   a1:   0.35,  d2:     0, q2: q2-pi/2,
39              alpha2:     0,   a2:   1.25,  d3:     0, q3: q3,
40              alpha3: -pi/2,   a3: -0.054,  d4:  1.50, q4: q4,
41              alpha4:  pi/2,   a4:      0,  d5:     0, q5: q5,
42              alpha5: -pi/2,   a5:      0,  d6:     0, q6: q6,
43              alpha6:     0,   a6:      0,  d7: 0.303, q7:  0}
44
```

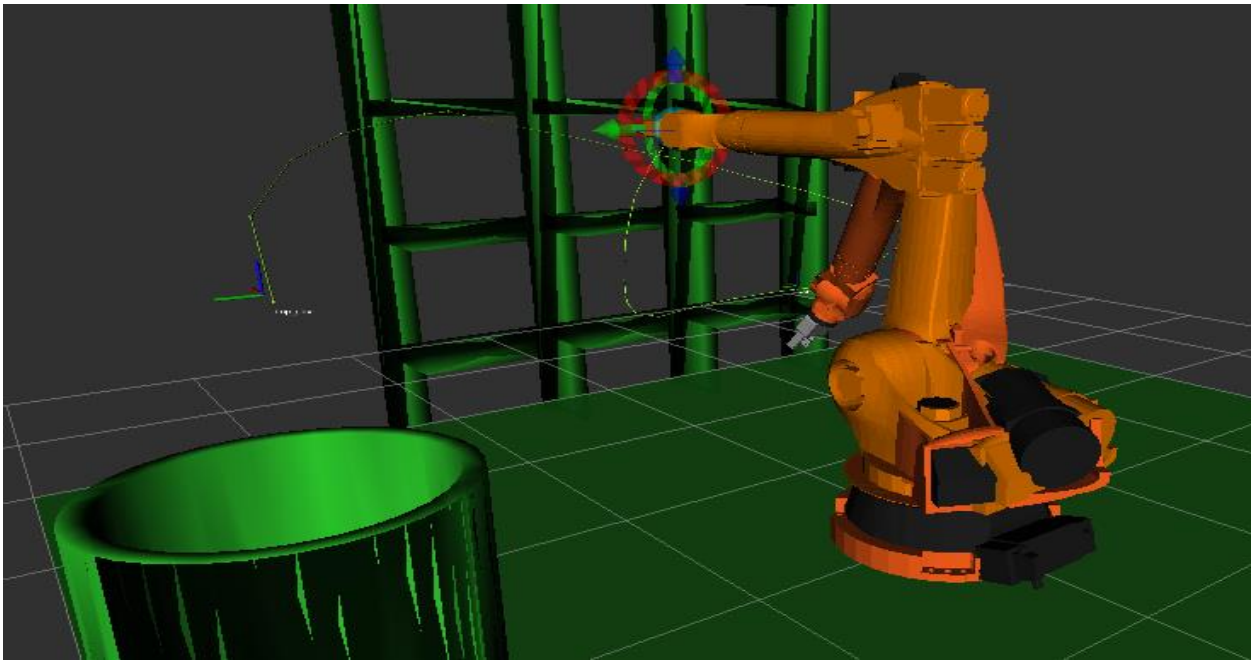
I then made a function to calculate the DH transforms and created matrices for each link.

```
# Define Modified DH Transformation matrix
def TF_Matrix(alpha, a, d, q):
    TF = 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 TF

# Create individual transformation matrices
T0_1  = TF_Matrix(alpha0, a0, d1, q1).subs(s)
T1_2  = TF_Matrix(alpha1, a1, d2, q2).subs(s)
T2_3  = TF_Matrix(alpha2, a2, d3, q3).subs(s)
T3_4  = TF_Matrix(alpha3, a3, d4, q4).subs(s)
T4_5  = TF_Matrix(alpha4, a4, d5, q5).subs(s)
T5_6  = TF_Matrix(alpha5, a5, d6, q6).subs(s)
T6_EE = TF_Matrix(alpha6, a6, d7, q7).subs(s)

T0_EE = T0_1 * T1_2 * T2_3 * T3_4 * T4_5 * T5_6 * T6_EE
```

Then, the inverse kinematic equations were performed using the rotation matrices. R_x, R_y, and R_z.

I extracted the end effector position and orientation and calculated the roll, pitch, and yaw. Rotation discrepancies were then calculated to find the wrist center. The theta values were then calculated using geometry.





The robot paths are accurate, but it is not the optimal work path. There is a lot of unnecessary movement. I think this is due to there being multiple solutions to calculating the path. The end effector also rotates. For real world applications, the end effector should try to keep the object upright and not rotate the object.