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. TF_Matrix is the general transformation matrix that is used to solve the individual transformation matrices.

```
# 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
```

The individual matrix resultants from the function TF_Matrix is shown below:

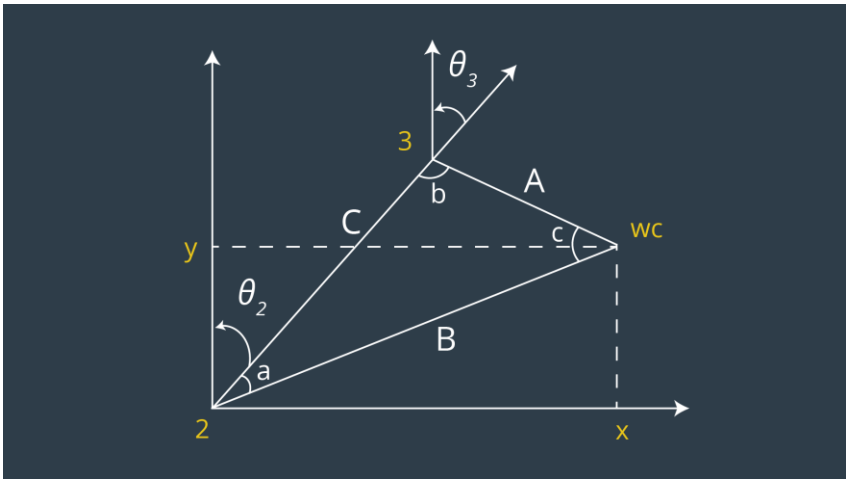
$$
{}_2T^3 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & 1.25 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}_3T^4 = \begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & -0.054 \\ 0 & 0 & 1 & 1.5 \\ -\sin(\theta_4) & -\cos(\theta_4) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

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

```
        ### Your IK code here
        # Compensate for rotation discrepancy between DH parameters and Gazebo

        Rot_Dis =  R_z.subs(y, radians(180)) * R_y.subs(p, radians(-90))
        R_EE = R_EE * Rot_Dis
        R_EE = R_EE.subs({'r':roll, 'p': pitch, 'y': yaw})

        EE = Matrix([[px],
                     [py],
                     [pz]])

        WC = EE - (0.303) * R_EE[:,2]

        # Calculate joint angles using Geometric IK method
        theta1 = atan2(WC[1], WC[0])

        # Calculate sides of the triangle to find angles
        s_a = 1.501
        s_b = sqrt(pow((sqrt(WC[0] * WC[0] + WC[1] * WC[1]) - 0.35), 2) + pow((WC[2] - 0.75), 2))
        s_c = 1.25

        # Calculate angles of triangle to find thetas 0-3
        a_a = acos((s_b * s_b + s_c * s_c - s_a * s_a) / (2 * s_b * s_c))
        a_b = acos((s_a * s_a + s_c * s_c - s_b * s_b) / (2 * s_a * s_c))
        a_c = acos((s_a * s_a + s_b * s_b - s_c * s_c) / (2 * s_a * s_b))
        theta2 = pi / 2 - a_a - atan2(WC[2] - 0.75, sqrt(WC[0] * WC[0] + WC[1] * WC[1]) - 0.35)
        theta3 = pi / 2 - (a_b + 0.036)

        # Calculate theta 3-6 using matrix multiplication
        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") * R_EE

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


        # Populate response for the IK request
        # In the next line replace theta1,theta2...,theta6 by your joint angle variables
        joint_trajectory_point.positions = [theta1, theta2, theta3, theta4, theta5, theta6]
        joint_trajectory_list.append(joint_trajectory_point)
    rospy.loginfo("length of Joint Trajectory List: %s" % len(joint_trajectory_list))
    return CalculateIKResponse(joint_trajectory_list)
```

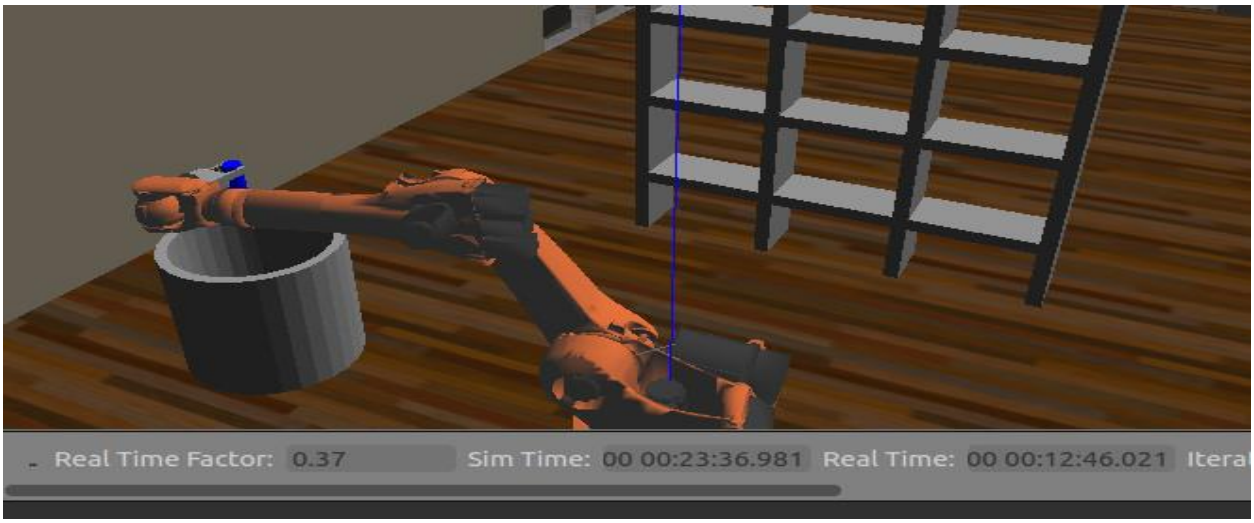
I extracted the end effector position and orientation and calculated the roll, pitch, and yaw. Rotation discrepancies were then calculated to find the wrist center. The theta values were then calculated using geometry of 2D projections shown below.

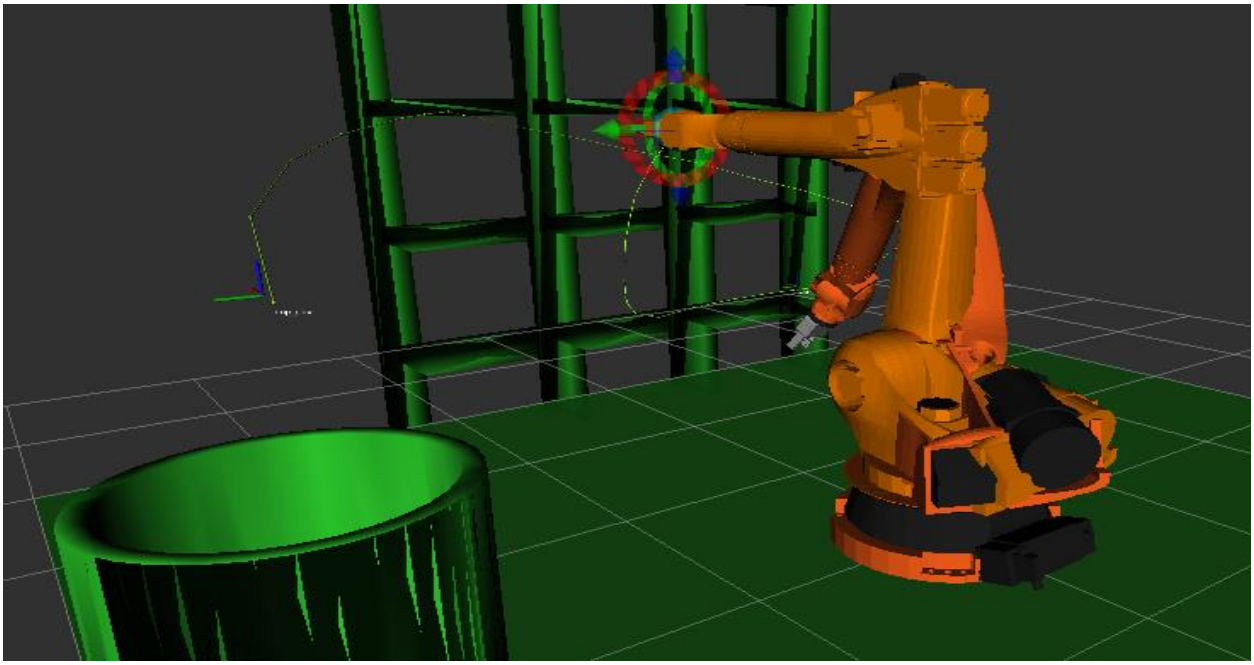


Theta 0-3 was calculated using geometry given the wrist centers. Theta 3-6 were found using matrix multiplication of LU decomposition to obtain R3_6:

$$ {}_3\mathbf{R}^6 = \begin{bmatrix} -s(\theta_4)s(\theta_6) + c(\theta_4)c(\theta_5)c(\theta_6) & -s(\theta_4)c(\theta_6) - s(\theta_6)c(\theta_4)c(\theta_5) & -s(\theta_5)c(\theta_4) \\ s(\theta_5)c(\theta_6) & -s(\theta_5)s(\theta_6) & c(\theta_5) \\ -s(\theta_4)c(\theta_5)c(\theta_6) - s(\theta_6)c(\theta_4) & s(\theta_4)s(\theta_6)c(\theta_5) - c(\theta_4)(\theta_6) & s(\theta_4)s(\theta_5) \end{bmatrix} $$

In Gazebo, the robot arm was able to pick and place the object.





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.