```
#!/usr/bin/env python

# Copyright (C) 2017 Udacity Inc.
#
# This file is part of Robotic Arm: Pick and Place project for Udacity
# Robotics nano-degree program
#
# All Rights Reserved.

# Author: Harsh Pandya

# import modules
import rospy
import tf
from kuka_arm.srv import *
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from geometry_msgs.msg import Pose
from mpmath import *
from sympy import *
import numpy as np


def handle_calculate_IK(req):
    rospy.loginfo("Received %s eef-poses from the plan" % len(req.poses))
    if len(req.poses) < 1:
        print "No valid poses received"
        return -1
    else:

        ### Your FK code here
        # Create symbols
        q1,q2,q3,q4,q5,q6,q7 = symbols('q1:8')
        d1,d2,d3,d4,d5,d6,d7 = symbols('d1:8')
        a0,a1,a2,a3,a4,a5,a6 = symbols('a0:7')
        alpha0,alpha1,alpha2,alpha3,alpha4,alpha5,alpha6 = symbols('alpha0:7')

        # Create Modified DH parameters
        s = {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}

        # 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(alph
a)*d],
                       [sin(q)*sin(alpha), cos(q)*sin(alpha),  cos(alpha),  cos(alph
a)*d],
                       [                0,                 0,           0,
   1]])
            return TF

        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_G = TF_Matrix(alpha6, a6, d7, q7).subs(s)

        R_z = Matrix([[    cos(np.pi), -sin(np.pi),             0, 0],
                      [    sin(np.pi),  cos(np.pi),             0, 0],
                      [             0,           0,             1, 0],
                      [             0,           0,             0, 1]])
        R_y = Matrix([[ cos(-np.pi/2),           0, sin(-np.pi/2), 0],
                      [             0,           1,             0, 0],
                      [-sin(-np.pi/2),           0, cos(-np.pi/2), 0],
                      [             0,           0,             0, 1]])
```

```
        T0_G = simplify(T0_1*T1_2*T2_3*T3_4*T4_5*T5_6*T6_G*R_z*R_y)

        # Create individual transformation matrices
        # Extract rotation matrices from the transformation matrices
        ###

        # Initialize service response
        joint_trajectory_list = []
        for x in xrange(0, len(req.poses)):
            # IK code starts here
            joint_trajectory_point = JointTrajectoryPoint()

            # Extract end-effector position and orientation from request
            # px,py,pz = end-effector position
            # roll, pitch, yaw = end-effector orientation
            px = req.poses[x].position.x
            py = req.poses[x].position.y
            pz = req.poses[x].position.z

            (roll, pitch, yaw) = tf.transformations.euler_from_quaternion(
                [req.poses[x].orientation.x, req.poses[x].orientation.y,
                    req.poses[x].orientation.z, req.poses[x].orientation.w])

            ### Your IK code here
            # Compensate for rotation discrepancy between DH parameters and Gazebo
            r, p, y = symbols('r p y')

            ROT_x = Matrix([[     1,        0,       0],
                            [      0, cos(r),-sin(r)],
                            [      0, sin(r), cos(r)]])
            ROT_y = Matrix([[ cos(p),       0, sin(p)],
                            [      0,       1,      0],
                            [-sin(p),       0, cos(p)]])
            ROT_z = Matrix([[ cos(y),-sin(y),       0],
                            [ sin(y), cos(y),       0],
                            [      0,       0,      1]])

            ROT_EE = ROT_z*ROT_y*ROT_x
            ROT_Error = ROT_z.subs(y, radians(180)) * ROT_y.subs(p, radians(-90))

            ROT_EE = ROT_EE * ROT_Error
            ROT_EE = ROT_EE.subs({'r':roll, 'p':pitch, 'y':yaw})

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

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

            theta1 = atan2(WC[1],WC[0])
            side_a = 1.501
            side_b = sqrt(pow((sqrt(WC[0]*WC[0]+WC[1]*WC[1])-0.35),2) + pow((WC[2] -
 0.75), 2))
            side_c = 1.25

            angle_a = acos((side_b*side_b + side_c*side_c - side_a*side_a)/(2*side_b
*side_c))
            angle_b = acos((side_a*side_a + side_c*side_c - side_b*side_b)/(2*side_a
*side_c))
            angle_c = acos((side_a*side_a + side_b*side_b - side_c*side_c)/(2*side_a
*side_b))

            theta2 = pi/2 - angle_a - atan2(WC[2] - 0.75, sqrt(WC[0]*WC[0] + WC[1]*W
C[1]) -0.35)
            theta3 = pi/2 -(angle_b + 0.036)

            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(method="LU") * ROT_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])
            # Calculate joint angles using Geometric IK method
            # Populate response for the IK request
            # In the next line replace theta1,theta2...,theta6 by your joint angle v
ariables
            joint_trajectory_point.positions = [theta1, theta2, theta3, theta4, thet
a5, theta6]
            joint_trajectory_list.append(joint_trajectory_point)

        rospy.loginfo("length of Joint Trajectory List: %s" % len(joint_trajectory_l
ist))
        return CalculateIKResponse(joint_trajectory_list)


def IK_server():
    # initialize node and declare calculate_ik service
    rospy.init_node('IK_server')
    s = rospy.Service('calculate_ik', CalculateIK, handle_calculate_IK)
    print "Ready to receive an IK request"
    rospy.spin()

if __name__ == "__main__":
    IK_server()
```