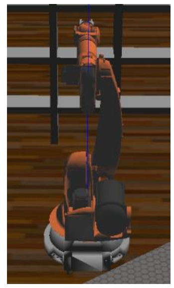
PICK AND PLACE PROJECT. IAN SUAREZ- NANODEGREE.

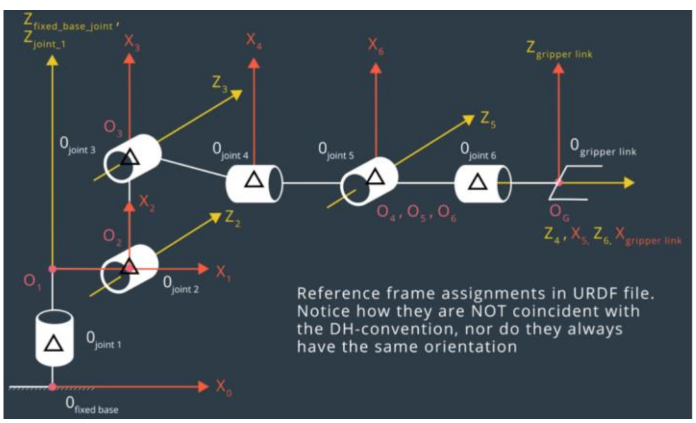


In this Project in udacity, i have to determine de math's needed to calculate de exact movement angles of the kuka robot. In order to do that, there are three principal steps.

1. Construct a denavith haterberg diagram.
2. Elaborate a denaivht haterberg table.
3. Obtain the foward kinematics.
    a. Calculate rotation matrix.
    b. Homogeneous transforms.
    c. Transformatrion matrix.
4. Calculate the inverse kinematics for the joints.

Then for the Project to work inside ros, its need to transform that kinematic ecuation into code in order to calculate the angles inside a inverse kinematics server.

The Denavit Haterberg Diagram.



The Denavit Haterberg Table.

n	theta	d	a	alpha
0	-	-	0	0
1	α1	0,75	0,35	-π/2
2	α2	0	1,25	0
3	α3	0	-0,054	-π/2
4	α4	1,5	0	π/2
5	α5	0	0	-π/2
6	α6	0,303	0	0

From the URDF file its posible to extract the parameters.

Modified DH PARAMETERS				
i	alpha{i-1}	a{i-1}	d{i}	q1
0	0	0	0,75	q1
1	-π/2	0,35	0	q2- pi/2
2	0	1,25	0	q3
3	-π/2	-0,54	1,5	q4
4	-π/2	0	0	q5
5	-π/2	0	0	q6
6	0	0	0,303	0

Individual transformation matrix

T01			
COS(Q1)	-SIN(Q1)	0	0
SIN(Q1)	COS(Q1)	0	0
0	0	1	0,75
0	0	0	1

T23			
COS(Q3)	-SIN(Q3)	0	1,25
SIN(Q3)	COS(Q3)	0	0
0	0	1	0
0	0	0	1

T12			
SIN(Q2)	COS(Q2)	0	0,35
0	0	1	0
COS(Q2)	-SIN(Q2)	0	0
0	0	0	1

T45			
COS(Q5)	-SIN,(Q5)	0	0
0	0	-1	0
SIN(Q4)	COS,(Q4)	0	0
0	0	0	1

T34			
COS(Q4)	-SIN(Q4)	0	-0,054
0	0	1	1,5
-SIN(Q4)	-COS,(Q4)	0	0
0	0	0	1

T56			
COS(Q6)	-SIN,(Q6)	0	0
0	0	1	0
-SIN,(Q6)	-COS,(Q6)	0	0
0	0	0	1

T6G			
1	0	0	0
0	1	0	0
0	0	1	0,303
0	0	0	1

This matrix are obtained by replacing the values of the DH table into the following matrix

```
# 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
```

Inverse Kinematic Ecuation Obtain.

$$\boldsymbol{theta1 = arctan(yc, xc)\{x, y\ coordinates\ for\ wirst\ center\}}$$

$$\boldsymbol{a = 1.50\ thi\ is\ obtain\ from\ the\ robot\ link\ dimensions}$$

$$\boldsymbol{c = 1.25}$$

$$\boldsymbol{b = \sqrt{(\left((\sqrt{yc^2} + xc^2) - 0.35\right)^2 + (zc^1 - 0.75)^2)}}$$

$$\boldsymbol{angle\ a = arcos\ (\frac{b^2 + c^2 - a^2}{2}) * 1/bc}$$

$$\boldsymbol{angle\ b = arcos\left(\frac{a^2 + c^2 - b^2}{2}\right) * \frac{1}{ac}}$$

$$\boldsymbol{angle\ c = arcos\ (\frac{a^2 + b^2 - c^2}{2}) * 1/ab}$$

IK_Server.Py Analisys.

First of all we import all the modules for do the math, define symbols, use matrix, the transformation matrix etc.

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

# 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 symbols, cos, sin, pi, simplify, pprint, tan, expand_trig, sqrt, trigsimp,
atan2
from sympy.matrices import Matrix
from numpy.linalg import inv
```

Then we créate the denavith haterberg table.

```
### Your FK code here
# Create symbols
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')
q1, q2, q3, q4, q5, q6, q7 = symbols('q1:8')
#
# 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:        q1,
      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}
```

Once i did that, the next step was to define the foward kinematics transformation from the base to the gripper, and also the matrix needed for determine the rotation in de three axis.

```
# function for rotation in X
def Rot_x(q):
    Rx = Matrix([[ 1,       0,        0],
                [  0,  cos(q),  -sin(q)],
                [  0,  sin(q),   cos(q)]])
    return Rx
    # function for rotation in y

def Rot_y(q):
    Ry = Matrix([[ cos(q),  0,  sin(q)],
                [       0,  1,       0],
                [ -sin(q),  0, cos(q)]])
    return Ry
    # function for rotation in z

def Rot_z(q):
    Rz = Matrix([[ cos(q), -sin(q),  0],
                [  sin(q),  cos(q),  0],
                [       0,       0,  1]])
    return Rz
# function for transformation matrix
def TRAFO(alpha, a, d, q):
    TRA = 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,
```

So now its time to obtain each of the transformation for every joint.

```
# Create individual transformation matrices
 # Create individual transformation matrices, this is the foward kinematics code
T0_1=TRAFO(alpha0, a0, d1, q1).subs(dh)
T1_2=TRAFO(alpha1, a1, d2, q2).subs(dh)
T2_3=TRAFO(alpha2, a2, d3, q3).subs(dh)
T3_4=TRAFO(alpha3, a3, d4, q4).subs(dh)
T4_5=TRAFO(alpha4, a4, d5, q5).subs(dh)
T5_6=TRAFO(alpha5, a5, d6, q6).subs(dh)
T6_G=TRAFO(alpha6, a6, d7, q7).subs(dh)
# transformation matrix form base to gripper. pure foward kinematics here!!
T0_G=T0_1 * T1_2 * T2_3 * T3_4 * T4_5 * T5_6 * T6_G
```

From now on the objective its to apply the inverse kinematics ecuation into code for calculate the angles of the joints. First of all its important to determine the center of the wirst and the rotation error. For do that we get the position from the poses msg and apply the rotation matrix i define earlier wich will give me the Rotation of the end efecctor and also the rotation error. I do that by conversión of gradians and grades.

```
# 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])

r, p, y = symbols('r p y')
R_ee = Rot_z(yaw) * Rot_y(pitch) * Rot_x(roll)
Rerror = Rot_z(pi) * Rot_y(-pi/2)
R_ee = R_ee * Rerror
```

Once i have the error of rotation and had get the end efector position i can make a vector where i put that end efector poition and began to calculate the inverse kinematics.

```
EE=Matrix([[px],[py],[pz]]) ##EE point locatioN
WC= EE - (0.303)*R_ee[:,2]
R0_3 = T0_1[0:3, 0:3]*T1_2[0:3, 0:3]*T2_3[0:3, 0:3]
theta1=atan2(WC[1],WC[0])
side_a=1.50
side_b_xy=sqrt(WC[0]*WC[0]+WC[1]*WC[1])-0.35
side_b_z=WC[2]-0.75
side_b=sqrt(pow((side_b_xy),2) + pow((side_b_z), 2))
side_c=1.25
```

The wirst center will be defined by the end efector point location times de rotation error i find before. then by using trigonometric we calcule the sides of the triangle and using cosine laws wwe determine the angles needed for the joints.

```
#Angles using cosine laws
angle_a=acos((   side_b * side_b + side_c * side_c - side_a * side_a )/( 2 * side_b * side_c ))
angle_b=acos(( - side_b * side_b + side_c * side_c + side_a * side_a )/( 2 * side_a * side_c ))
angle_c=acos((   side_b * side_b - side_c * side_c + side_a * side_a )/( 2 * side_a * side_b ))
### Thetas
theta2= pi/2 - angle_a - atan2(side_b_z, side_b_xy)
theta3= pi/2 - angle_b + 0.036 #sag in link 4
R0_3 = R0_3.evalf(subs={q1: theta1, q2: theta2, q3: theta3})
R3_6 = R0_3.T * R_ee

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])
```

Robot picking the object.

Robot placing the object.