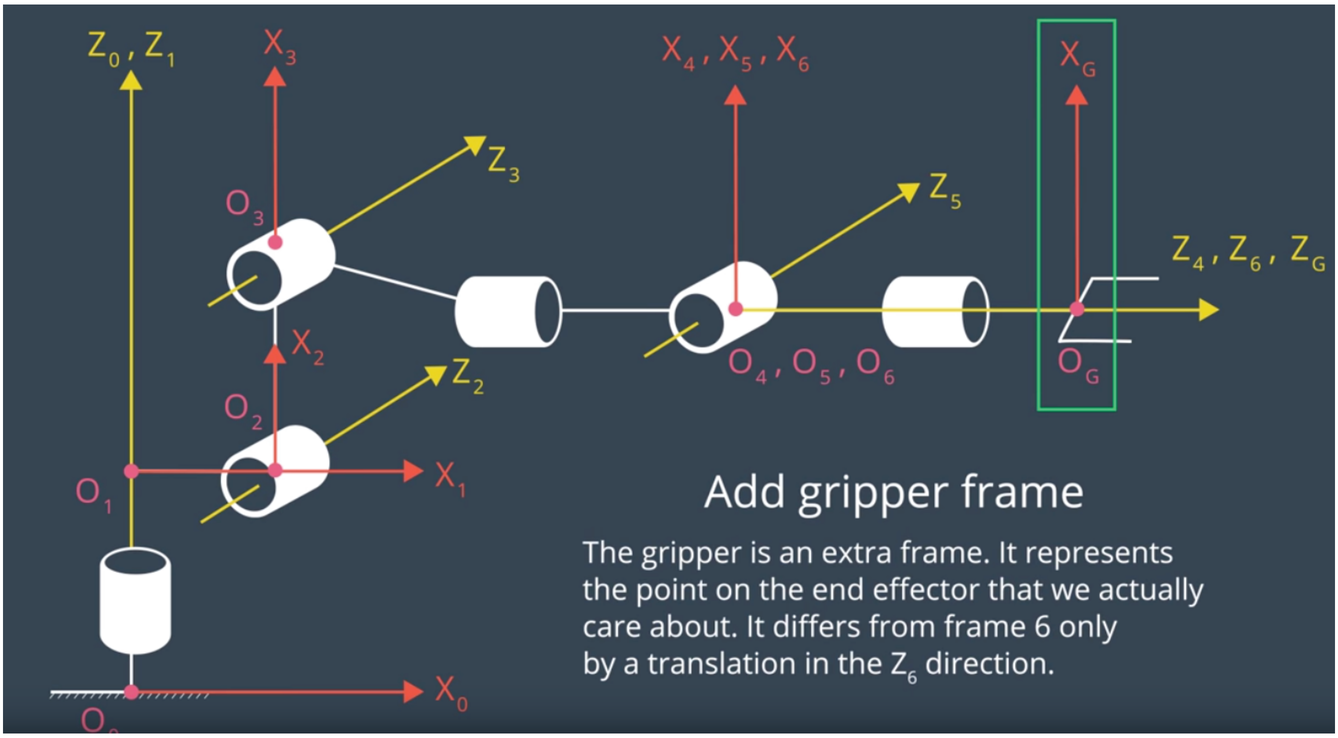
# Project - 2 : Pick and Place

*Arda Kaya* *Udacity Robotics Nanodegree* *ardakayaa@gmail.com*

## Modified DH-Parameters

Here is the diagram with reference frames being define to follow DH method to find the parameters for the transformation matrices:

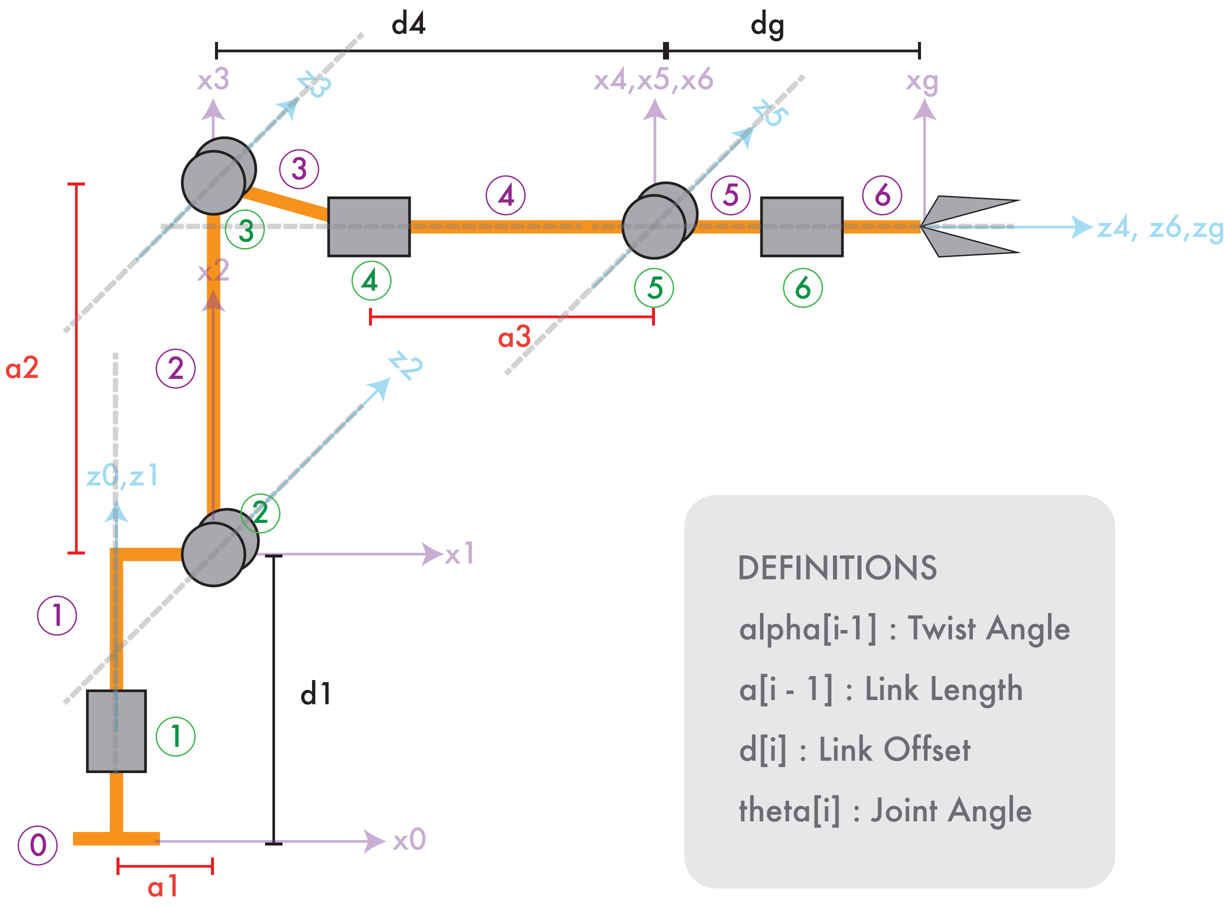



To define the homogeneous transformation between joints in this robot arm, I applied the Denavit–Hartenberg method with the convention as follow:

- $\alpha_{i-1}$ (twist angle) = angle between $\hat{Z}_{i-1}$ and $\hat{Z}_i$ measured about $\hat{X}_{i-1}$ in a right-hand sense.
- $a_{i-1}$ (link length) = distance from $\hat{Z}_{i-1}$ to $\hat{Z}_i$ measured along $\hat{X}_{i-1}$ where $\hat{X}_{i-1}$ is perpendicular to both $\hat{Z}_{i-1}$ to $\hat{Z}_i$
- $d_i$ (link offset) = signed distance from $\hat{X}_{i-1}$ to $\hat{X}_i$ measured along $\hat{Z}_i$. Note that this quantity will be a variable in the case of prismatic joints.
- $\theta_i$ (joint angle) = angle between $\hat{X}_{i-1}$ to $\hat{X}_i$ measured about $\hat{Z}_i$ in a right-hand sense. Note that this quantity will be a variable in the case of a revolute joint.

This table is the result of all the parameters being define according to the DH convention above:

| i | alpha [ i - 1 ] | a [ i - 1 ] | d [ i ] | theta [ i ] |
|---|---|---|---|---|
| **1** | 0 | 0 | 0.75 | q1 |
| **2** | - pi / 2 | 0,35 | 0 | q2 - pi / 2 |
| **3** | 0 | 1,25 | 0 | q3 |
| **4** | - pi / 2 | -0,54 | 1,50 | q4 |
| **5** | pi / 2 | 0 | 0 | q5 |
| **6** | - pi / 2 | 0 | 0 | q6 |
| **g** | 0 | 0 | 0,303 | 0 |




DEFINITIONS

alpha[i-1] : Twist Angle

a[i - 1] : Link Length

d[i] : Link Offset

theta[i] : Joint Angle

# TRANSFORMATION MATRICES ABOUT EACH JOINT

T01

| | | | |
|---|---|---|---|
| cos(q1) | - sin(q1) | 0 | 0 |
| sin(q1) | cos(q1) | 0 | 0 |
| 0 | 0 | 1 | 0.75 |
| 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 |

T23

| | | | |
|---|---|---|---|
| cos(q3) | - sin(q3) | 0.0 | 1.25 |
| sin(q3) | cos(q3) | 0 | 0 |
| 0 | 0 | 1 | 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 |

T45

| | | | |
|---|---|---|---|
| cos(q5) | - sin(q5) | 0 | 0 |
| 0 | 0 | - 1 | 0 |
| sin(q5) | cos(q5) | 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 |

Note: I have used following code to get these matrices.

```
DH_Table = {alpha0:    0.,   a0:      0.,   d1:  0.75,   q1: q1,
                 alpha1:-pi/2.,   a1:   0.35,    d2:     0.,  q2:-pi/2. + q2,
                 alpha2:    0.,   a2:   1.25,    d3:     0.,  q3: q3,
                 alpha3:-pi/2.,   a3: -0.054,    d4:  1.50,   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 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 transformations Matrixis
       T0_1 = TF_Matrix(alpha0, a0, d1, q1).subs(DH_Table)
       T1_2 = TF_Matrix(alpha1, a1, d2, q2).subs(DH_Table)
       T2_3 = TF_Matrix(alpha2, a2, d3, q3).subs(DH_Table)
       T3_4 = TF_Matrix(alpha3, a3, d4, q4).subs(DH_Table)
       T4_5 = TF_Matrix(alpha4, a4, d5, q5).subs(DH_Table)
       T5_6 = TF_Matrix(alpha5, a5, d6, q6).subs(DH_Table)
       T6_EE = TF_Matrix(alpha6, a6, d7, q7).subs(DH_Table)

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

I have multiplied individual matrices to create transformation matrix from base link to gripper .

# THE TRANSFORMATION MATRIX BETWEEN THE GRIPPER FRAME AND THE BASE FRAME GIVEN THE POSITION AND ORIENTATION OF THE GRIPPER

Rotation X

```
Matrix([[1      ,      0,       0 ],
        [0      , cos(r), -sin(r)],
        [0      , sin(r),  cos(r)]])
```

Rotation Y

```
Matrix([[cos(p) ,      0,  sin(p)],
        [0      ,      1,     0  ],
        [-sin(p),      0,  cos(p) ]])
```

Rotation Z

```
Matrix([[cos(y) , -sin(y),      0],
        [sin(y) ,  cos(y),      0],
        [     0,       0,       1]])
```

Note: I have used following code to get these matrices.

```
# 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 ])

        # Find EE rotation Matrix
        # Define RPY rotation matrices
        # http://planning.cs.uiuc.edu/node102.html

        r, p, y  = symbols('r p y')

        ROT_x = Matrix([[1      ,      0,       0],
                        [0      , cos(r), -sin(r)],
                        [0      , sin(r),  cos(r)]]) #roll

        ROT_y = Matrix([[cos(p) ,      0,  sin(p)],
                        [0      ,      1,       0],
                        [-sin(p),      0,  cos(p)]]) #pitch

        ROT_z = Matrix([[cos(y) , -sin(y),      0],
                        [sin(y) ,  cos(y),      0],
                        [     0,       0,       1]]) #yaw

        ROT_EE = ROT_z * ROT_y * ROT_x

        # More Information can be found in KR210 FK section
        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]
```

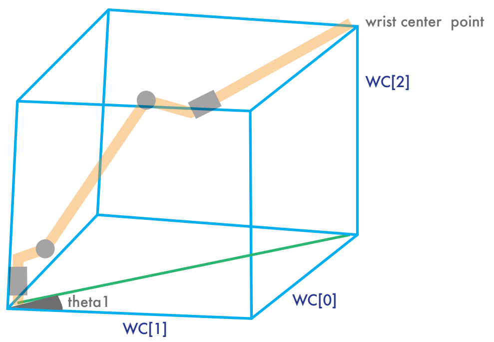
I have multiplied gripper's roll, pitch and yaw matrices to get its rotation matrix.

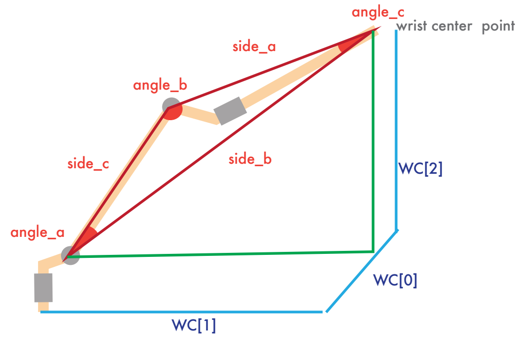
To get the wrist center I have used this equation.

$$^{0}r_{WC/0} = {}^{0}r_{EE/0} - d \cdot {}^{0}_{6}\mathrm{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} - d \cdot {}^{0}_{6}\mathrm{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

# CALCULATING THE INDIVIDUAL JOINT ANGLES

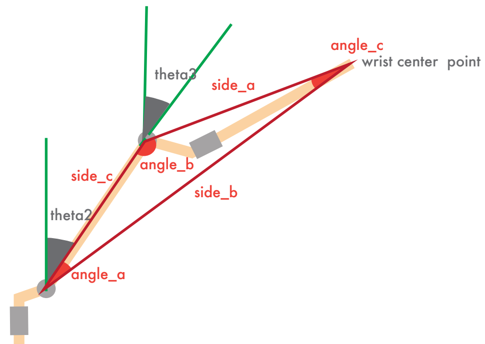
## Solving for theta_1, theta_2 and theta_3



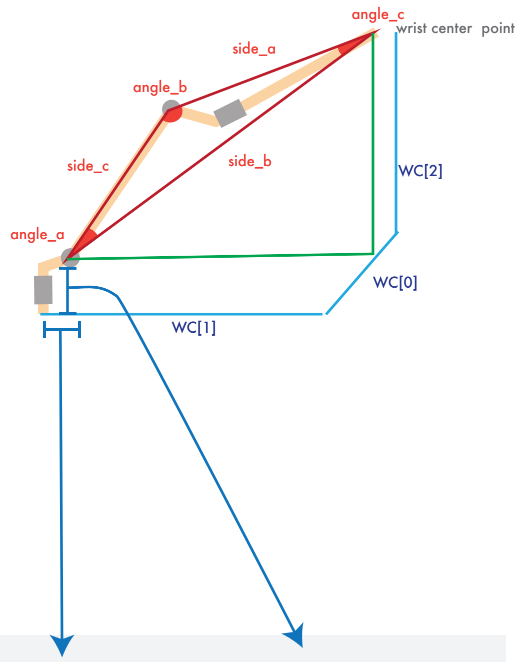




The easiest angle was theta1, as I already have the wrist center position.

```
theta1 = atan2(WC[1] , WC[0])
```




To calculate theta2 and theta3, I used a triangle that you can see on the right. Length of side_a and side_c are already given, but I had to find length of side_b.




To calculate the length of side_b I have used wrist center position and The Pythagoras Triplets.

I have used the code below to calculate side_b.

```
side_b = sqrt(pow((sqrt(WC[0] * WC[0] + WC[1] * WC[1]) - 0.35), 2) + pow((WC[2] - 0.75), 2))
```

After calculating side_a, side_b and side_c , I have used the Cousine Rule to get angle_a, angle_b and angle_c.

```
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))
```

Then, I subtracted the angles from pi/2 to get theta2 and theta3.

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

You can find the full code to calculate theta_2 and theta_3 in below.

```
# SSS triangle for theta2 and theta3
    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] * WC[1]) - 0.35)
    theta3 = pi / 2 - (angle_b + 0.036)
```

## Solving for theta_4, theta_5 and theta_6

In this second part of the decouple problem, I can define the rotation between joint_3 and the end effector as following:

```
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") * ROT_EE
```

And the last three joints can be solved as following:

```
# EULER angles from rotation Matrix
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])
```

# Conclusion

The arm successfully calculate and pick up the object closely the the reference trajectory in Rviz.

These are my results from IK_debug.py in test_case_number = 1.

Total run time to calculate joint angles from pose is 0.8585 seconds

Wrist error for x position is: 0.00000046
Wrist error for y position is: 0.00000032
Wrist error for z position is: 0.00000545
Overall wrist offset is: 0.00000548 units

Theta 1 error is: 0.00093770
Theta 2 error is: 0.00181024
Theta 3 error is: 0.00205031
Theta 4 error is: 0.00172067
Theta 5 error is: 0.00197873
Theta 6 error is: 0.00251871

End effector error for x position is: 0.00002010
End effector error for y position is: 0.00001531
End effector error for z position is: 0.00002660
Overall end effector offset is: 0.00003668 units