Laura Curioso
06 Oct 17

Project 2: Robotic Arm Pick and Place Writup

DH Table Creation for Inverse Kinematics

In the project notes, an outline on how to do forward kinematics was provided. The base of the robot was used as the start point and the DH table was created with the wrist center (WC) acting as the end effector/final position. For the Kuka robot, there are 6 joints that were described.

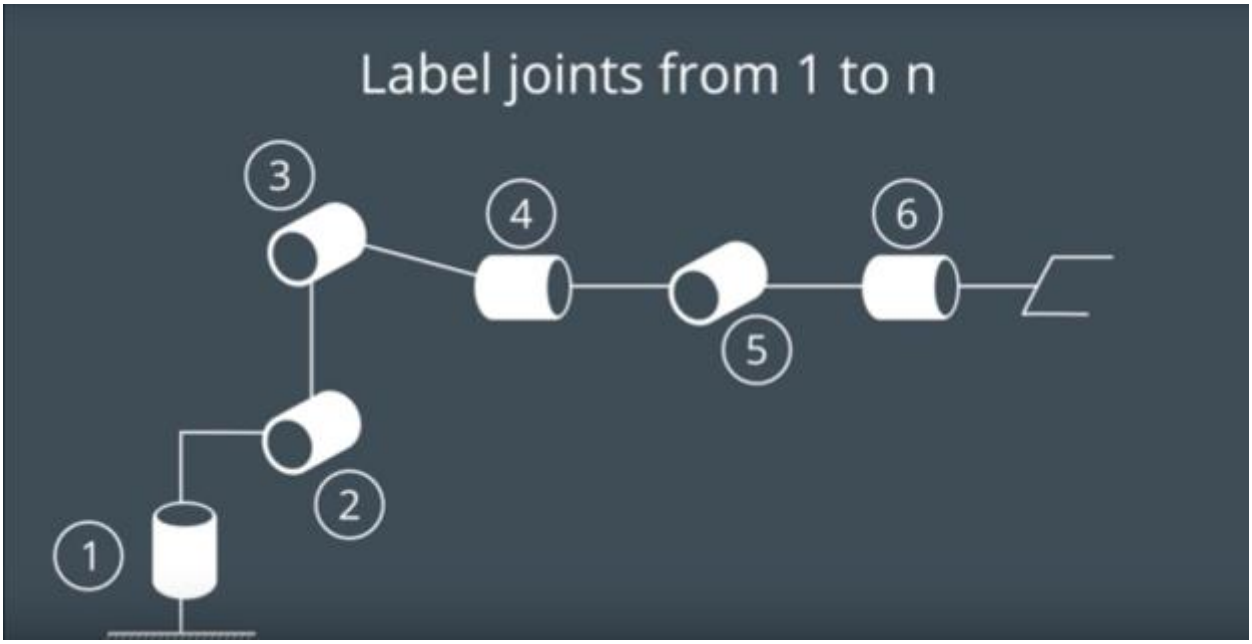

Figure 1: Joints 1-6 Starting at Kuka Arm Base

For z angle declaration, Z+ is pointing up and y+ is pointing “right”. In turn the notation clockwise is negative an counter clockwise is positive according to this assignment and by going by the right hand rule.

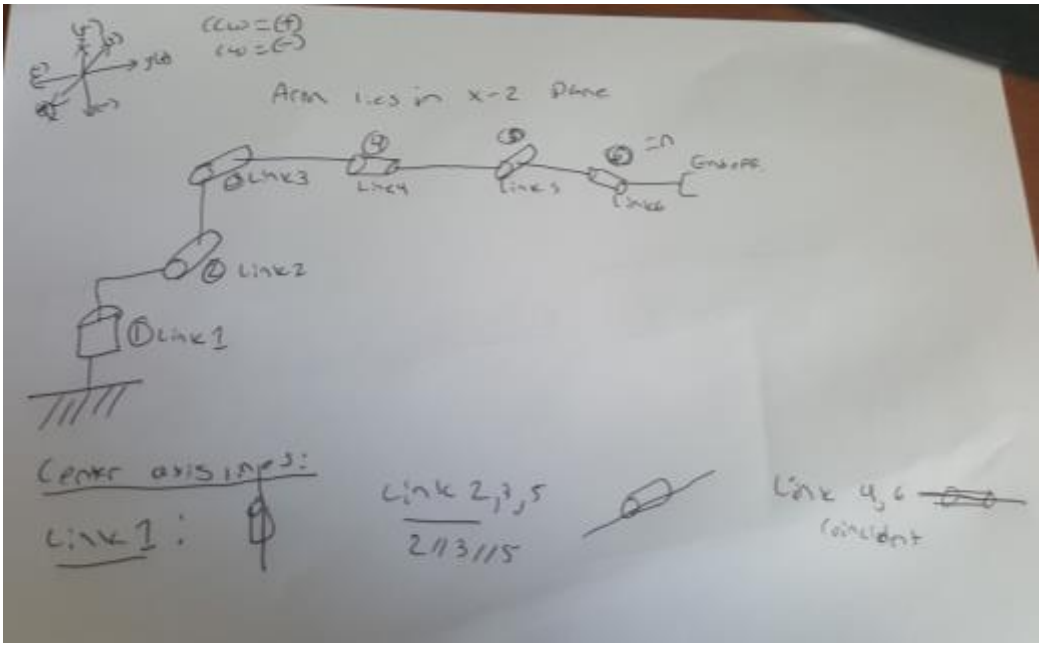

Figure 2: Link and Axis Centerline Declarations

Links 2, 3, and 5 are parallel and links 4 and 6 are coincident. Figure 2 shows the body diagram of the kuka arm along with the declared links and corresponding centerlines of each axis.

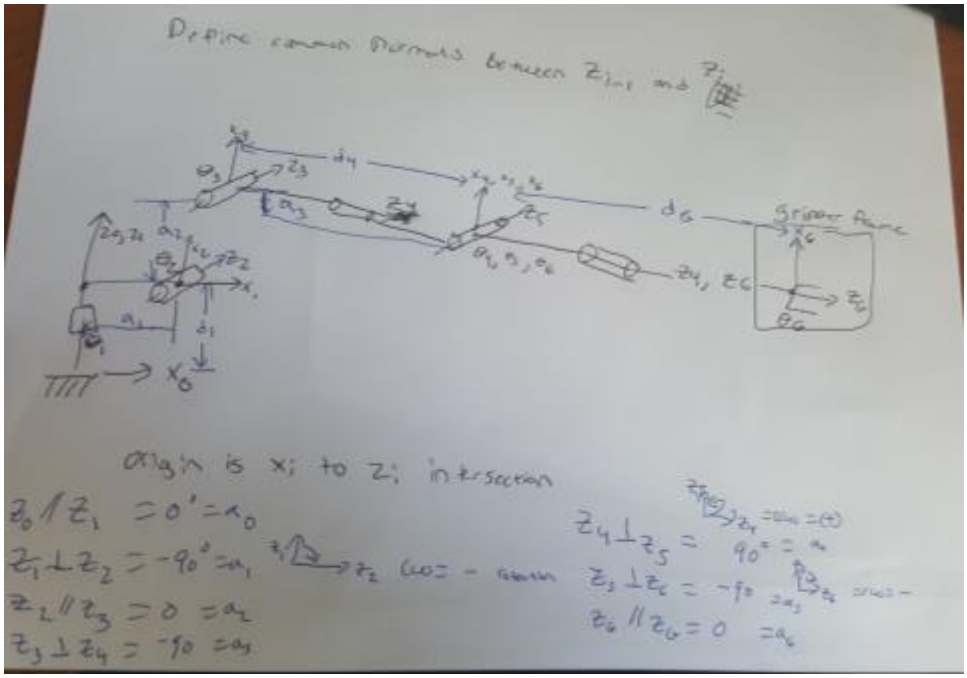

Figure 3: Defined Common Normals: Theta Link Rotation References

From these declarations one can see the non-zero attributes that need to be included in the DH table. Joint angles, assigned as variables q1 through q7 are identified. There are 7 link offsets defined as d1 through d7 and link lengths are defined as a0 through a6. Alpha twist angles are defined the same way as in the notes from alpha 0 thorough alpha 6. As described in the

forward kinematics video, joint 2 is offset by -90 degrees. X1 is not parallel to X2 when theta 2 = 0. To represent this offset in radians this translates to -π/2 (where 2π = 360 degrees). For non-zero values, the link distances are within the URDF file. When going from joint 1 to joint 2, you have to translate points from the 0.35m in the x direction to 0.42 meters in the z direction so a1 is set to 0.35m. a2 is set to 1.25m as it relates joint 2 to joint 3. Since the joint 2 is coincident to joint 3, d2 is set to zero while q2 is set to the --π/2. For alpha 3, a3 is set to -.05. D4 is set to 1.5. a4, a5, a6 are 0 with d7=dG being set to 0.303.

```
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: 1.5, 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}
```

From this table, you apply forward kinematics to translate the twist and joint angles relative to the gripper end effector frame. Taken from homogeneous transforms in lesson 2,

$$\begin{bmatrix} r_{Ax} \\ r_{Ay} \\ r_{Az} \\ \hline 1 \end{bmatrix} = \begin{bmatrix} r_{11}x + r_{12}y + r_{13}z + r_{BAx} \\ r_{21}x + r_{22}y + r_{23}z + r_{BAy} \\ r_{31}x + r_{32}y + r_{33}z + r_{BAz} \\ \hline 1 \end{bmatrix}$$

Homogeneous transform matrix is below with variable “alpha” defined:

```
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
```

To check the transforms to ensure the calculated rotations are correct, individual transforms for each of the links are created. DH table is now substituted for each row including the relationship between link 6 and the gripper frame alphas. This is referenced in the walkthrough video.

```
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_EndEff = TF_Matrix(alpha6, a6, d7, q7).subs(DH_Table)
```

To calculate the end effector link transform from the base frame, we multiply all the transformations together:

T0_EndEff = T0_1*T1_2*T2_3*T3_4*T4_5*T5_6*T6_EndEff.

Rotation component matrix:

[1,0,0], [0,1,0], [0,0,1]

From the pose notes, we define px, py, and pz to be:

```
px = req.poses[x].position.x
py = req.poses[x].position.y
pz = req.poses[x].position.z
```

To extract the end effector orientation using the Euler equation:

```
(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])
```

Compensate for rotation discrepancy between DH parameters and Gazebo by setting r,p,y as symbols for simplification.

```
r,p,y = symbols ('r p y')
# Calculate joint angles using Geometric IK method
```

Then manipulate each matrix to rotate it to the correct plane for roll, pitch, and yaw.

```
#roll rotation matrix manipulation
Rot_x = Matrix([[1, 0, 0], [0, cos(r), -sin(r)], [0, sin(r), cos(r)]])

#pitch rotation matrix manipulation

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

#yaw rotation matrix manipulation
```

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

We then multiply rotation matrices together to get end effector matrix:

Rot_EE = Rot_z * Rot_y * Rot_x

To calculate the rotation error, we convert the matrix to radians to go from polar coordinates to rectangular coordinates:

Rot_Error = Rot_z.subs(y, radians(180)) * Rot_y.subs(p, radians (-90))

To correct for error to match URDF File we multiply the roll, pitch, yaw matrix by the calculated rotation error:

```
Rot_EE = Rot_EE * Rot_Error
Rot_EE = Rot_EE.subs({'r': roll, 'p' : pitch, 'y': yaw})

#new end effector position:
EE = Matrix([[px], [py], [pz]])
```

From here we can calculate the wrist center calculation that was shown in the notes:

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

To calculate joint angle variables for each theta, we refer to the new wrist center calculation and use the inverse kinematics notes to give us theta 1-6.

For theta1, we project Zc onto the ground plane:

$$\theta_1 = atan2(y_c, x_c)$$

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

For going to links 2 and 3, we need to figure out the angle of the shortest path which is the hypotenuse because of the wrist center in relation to the base:

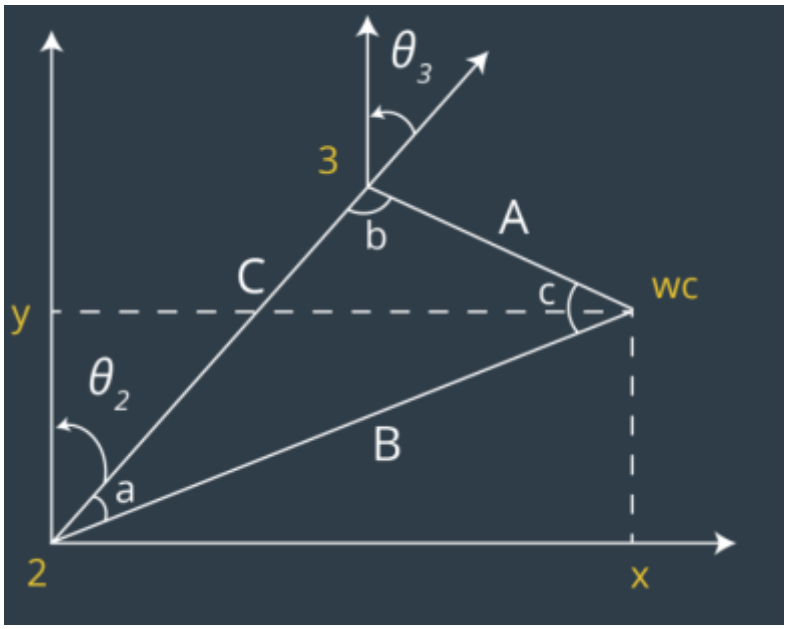

Figure 4: Geometry Outline of Wrist Center to Base

```
#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))
```

$$\theta_2 = atan2(s, r)$$

```
 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)
```

R0_3 is the resultant of matrix of T0_1 * T1_2 *T2_3 (alpha 0-2).

```
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})
```

From the notes, we know that R0_6 = Rrpy. This Homogeneous RPY rotation can be substituted with calculated values for joints 1-3 by pre-multiplying them by Rrpy:

```
R3_6 = R0_3.inv("LU") *Rot_EE
```

We then compute theta 4-6 by using 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])
```

From here we then enter the calculated thetas into the joint directory point positions:

```
joint_trajectory_point.positions = [theta1, theta2, theta3, theta4, theta5, theta6]
joint_trajectory_list.append(joint_trajectory_point)
```

Possible Improvements:

To reduce time taken to calculate each pose to determine the path from the Kuka arm to the object to the bin, one can load in the calculated values from a file. Another method is to not have the calculations done for each iteration. Another way is to have the arm move in 1 axis at a time (do calculations for only that axis).