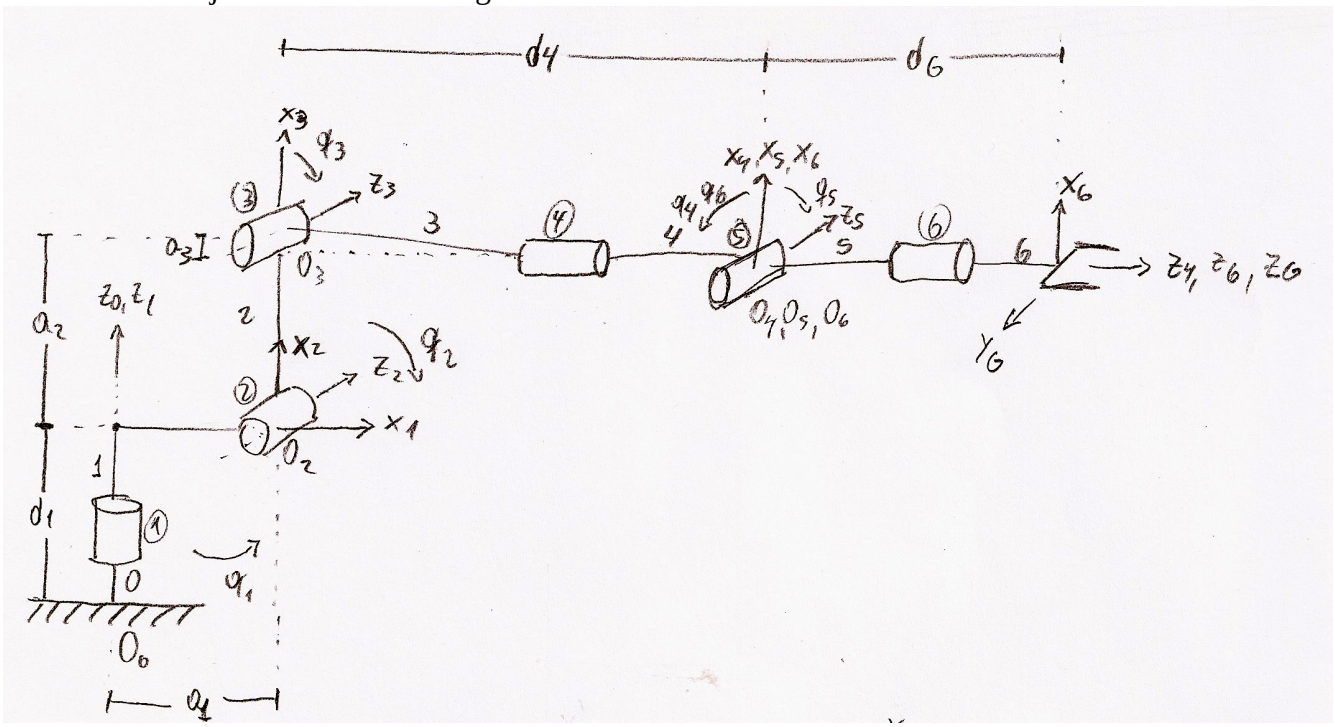
Project: Kinematics Pick & Place

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Kinematic Analysis

1 . *Run the forward_kinematics demo and evaluate the kr210.urdf.xacro file to perform kinematic analysis of Kuka KR210 robot and derive its DH parameters.*

Analyzing the axes pose and the kr210.urdf.xacro file, I was able to determine the coordinate frames for each joint as seen in the figure.



From here, I could derive the DH parameter table where most of the parameters came directly from .xacro file, but some of them were calculated as follow:

d1 = 0.33 + 0.42 = 0.75

d4 = 0.96 + 0.54 = 1.5

d7 = dG = 0.193 + 0.11 = 0.303

therefore, the DH parameter table is the following:

Links	α_{i-1}	a_{i-1}	d_{i-1}	θ_i
0 → 1	0	0	0.75	q1
1 → 2	$-\pi/2$	0.35	0	$-\pi/2$ + q2
2 → 3	0	1.25	0	q3
3 → 4	$-\pi/2$	-0.054	1.5	q4
4 → 5	$\pi/2$	0	0	q5
5 → 6	$-\pi/2$	0	0	q6
6 → EE	0	0	0.303	0

2. Transformation matrices

I wrote the transformation matrix from ***i-1*** to ***i*** independently in the code and then substitute the parameter value.

```
T0_1 = Matrix([[                cos(q1),                -sin(q1),            0,               a0],
               [ sin(q1)*cos(alpha0), cos(q1)*cos(alpha0), -sin(alpha0), -sin(alpha0)*d1],
               [ sin(q1)*sin(alpha0),  cos(q1)*sin(alpha0), cos(alpha0),  cos(alpha0)*d1],
               [                   0,                   0,            0,               1]])
T0_1 = T0_1.subs(s)
```

then I made the composition of homogeneous transformation matrices in order to obtain the complete homogeneous transformation matrix from frame 0 to the EE

```
# Composition of Homogeneous Transformation matrix
T0_2 = T0_1 * T1_2  # base_link to link_2
T0_3 = T0_2 * T2_3  # base_link to link_3
T0_4 = T0_3 * T3_4  # base_link to link_4
T0_5 = T0_4 * T4_5  # base_link to link_5
T0_6 = T0_5 * T5_6  # base_link to link_6
T0_G = T0_6 * T6_G  # base_link to gripper
```

but the orientation of the gripper frame in the DH convention is different from what we extract from the .xacro file, so we need to make a couple of rotations of our frame in order to obtain the same orientation. First 180º around the Z axe, and then -90º around the Y axe.

```
# Gripper orientation correction
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]])
R_corr = R_z * R_y
```

so the total homogeneous transformation matrix is:

```
# Total homogeneous transform
T_total = T0_G * R_corr
```

3. Inverse Kinematics

As the robotic arm has six dof and has a spherical joint for the gripper, we can decouple the IK problem in inverse position and inverse orientation.

3.1 Inverse Position

For the IP, first we need to calculate the wrist center, having the desired position of the EE relative to ***frame 0*** $0_{r EE/0}$, and the vector from the WC to the EE $0_{r EE/WC}$, we can calculate the WC position relative to the frame 0 as:

$$0_{r WC/0} = 0_{r EE/0} - 0_{r EE/WC}$$

Due to I considered the gripper frame correction in the FK, the vector $0_{r EE/WC}$ is the rotation of the vector [d, 0, 0], therefore:

$$0_{r WC/0} = 0_{r EE/0} - d_7 \cdot {}^0_6R \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} - d_7 \cdot {}^0_6R \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

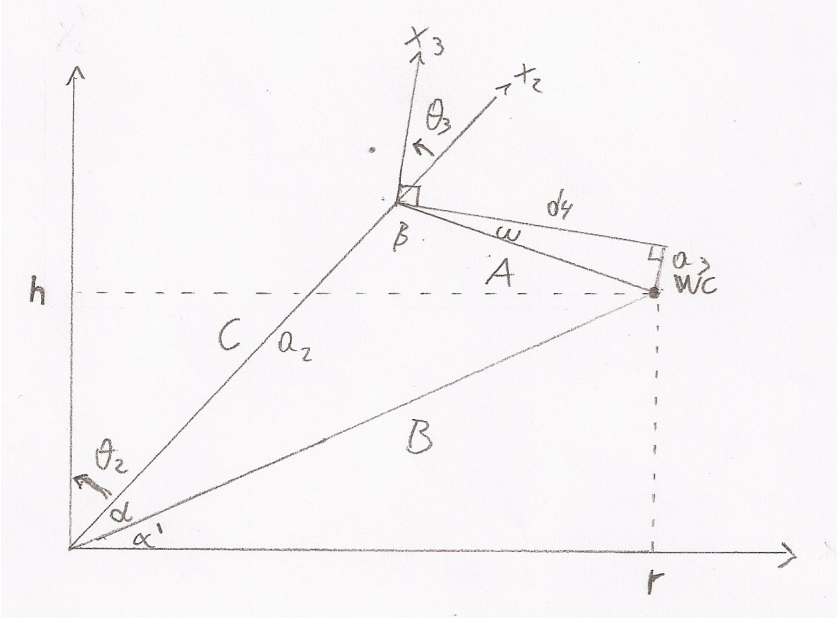
where ${}^0_6R$ correspond to a composition of rotations using Euler angles. The convention used is the x-y-z extrinsic rotations, then:

$$Rrpy = Rot(Z, yaw) \cdot Rot(Y, pitch) \cdot Rot(X, roll)$$

so having the WC it's easy to obtain θ_1:

$$\theta_1 = \arctan\left(\frac{w_y}{w_x}\right)$$

Now for calculating theta 2 and theta 3, we have the following diagram:



Using geometry, trigonometric and cosine law, we can calculate the angles in steps:

$$r = \sqrt{wc_x^2 + wc_y^2} - a_1$$
$$h = wc_z - d_1$$
$$A = \sqrt{d_4^2 + a_3^2}$$
$$B = \sqrt{r^2 + h^2}$$
$$\alpha' = \arctan\left(\frac{h}{r}\right)$$
$$\alpha = \arccos\left(\frac{a_2 + B^2 - A^2}{2\,a_2\,B}\right)$$
$$\theta_2 = \frac{\pi}{2} - \alpha - \alpha'$$
$$\omega = \arctan\left(\frac{a_3}{d_4}\right)$$
$$\beta = \arccos\left(\frac{A^2 + a_2^2 - B^2}{2\,A\,a_2}\right)$$
$$\theta_3 = \frac{\pi}{2} - \beta - \omega$$

where a_i and d_i are parameters from the DH table.

3.1 Inverse Orientation

For the IO problem, I based my code in the analysis of the IK for the kuka KR210, so I just needed to solve:

$$R_{3-6} = R_{0-3}^{-1} \cdot Rrpy$$

using the FK, I could compute R_{0-3}^{-1} in terms of *q1*, *q2* and *q3* and use it in the code*:*

```
inv_R0_3 = Matrix([[ sin(q2 + q3) * cos(q1),  sin(q1) * sin(q2 + q3),  cos(q2 + q3)],
                   [ cos(q1) * cos(q2 + q3), sin(q1) * cos(q2 + q3),  -sin(q2 + q3)],
                   [                -sin(q1),                cos(q1),               0]])
```

and R_{3-6} in the same way:

```
R3_6 = Matrix([[  -sin(q5)*cos(q4),   sin(q4)*cos(q6) + sin(q6)*cos(q4)*cos(q5),  -sin(q4)*sin(q6) + cos(q4)*cos(q5)*cos(q6)],
               [           cos(q5),                                  sin(q6),                                 sin(q5)*cos(q6)],
               [   sin(q4)*sin(q5),  -sin(q4)*sin(q6)*cos(q5) + cos(q4)*cos(q6),  -sin(q4)*cos(q5)*cos(q6) - sin(q6)*cos(q4)]])
```

from here, I could obtain θ_4 , θ_5 y θ_6 :

$$\theta_4 = \arctan\left(\frac{r_{31}}{-r_{11}}\right)$$

$$\theta_5 = \arctan\left(\frac{\sqrt{(r_{11}^2 + r_{31}^2)}}{r_{21}}\right)$$

$$\theta_6 = \arctan\left(\frac{r_{22}\sqrt{(r_{11}^2 + r_{31}^2)}}{r_{23}}\right)$$

4. Code implementation

The code is basically the implementation of the previous analysis of the FK and IK.

All the forward kinematics was implemented in a function outside the *CalculateIK* service, to just do it once at the beginning of execution.

The total transformation matrix was stored in a global variable to use it later for error calculation.

Also, as I mention before, I calculated the inverse of the rotation matrix from ***frame 0*** to ***frame 3,*** `inv_R0_3`, and the rotation matrix between ***frame 3*** and ***frame 6,*** `R3_6`, for use it in the IK solver. It's worth to notice that this rotation matrix includes the EE frame correction.

After some test of the code, I realized that the gripper was doing a lot of rotations, but the path was OK. The problem is that the function *atan2* gives angles between $[-\pi, \pi]$ so when the calculated angle is less than π , let say $\pi - \delta$ with $0 < \delta < \pi/2$, and the next desired angle is greater than π , the result from *atan2* is a negative angle, so the joint is forced to rotate in the other direction instead of doing the shortest path. In order to get rid of this, I implemented the following code (in this case, for θ_4):

```
if (theta4 - theta4_prev) > np.pi:
    theta4 = theta4 - 2 * np.pi
elif (theta4 - theta4_prev) < -np.pi:
    theta4 = theta4 + 2 * np.pi
```

where theta4_prev is the previous value of theta4. Together with that, I had to implement the limits for the joints:

```
if theta4 > 6.109:
    theta4 = theta4 - 2 * np.pi
elif theta4 < -6.109:
    theta4 = theta4 + 2 * np.pi
```

where the limits were extracted from the URDF.

For θ_5 , there is not need for the previous correction because the equation for it only gives angles between $[0,\pi]$. If we take $\theta_5<0$ we need to re-calculate θ_6 and check which solution is the fastest.

There are more than one solution for the orientation problem, but this worked fine with the corrections, so I didn't explore others.

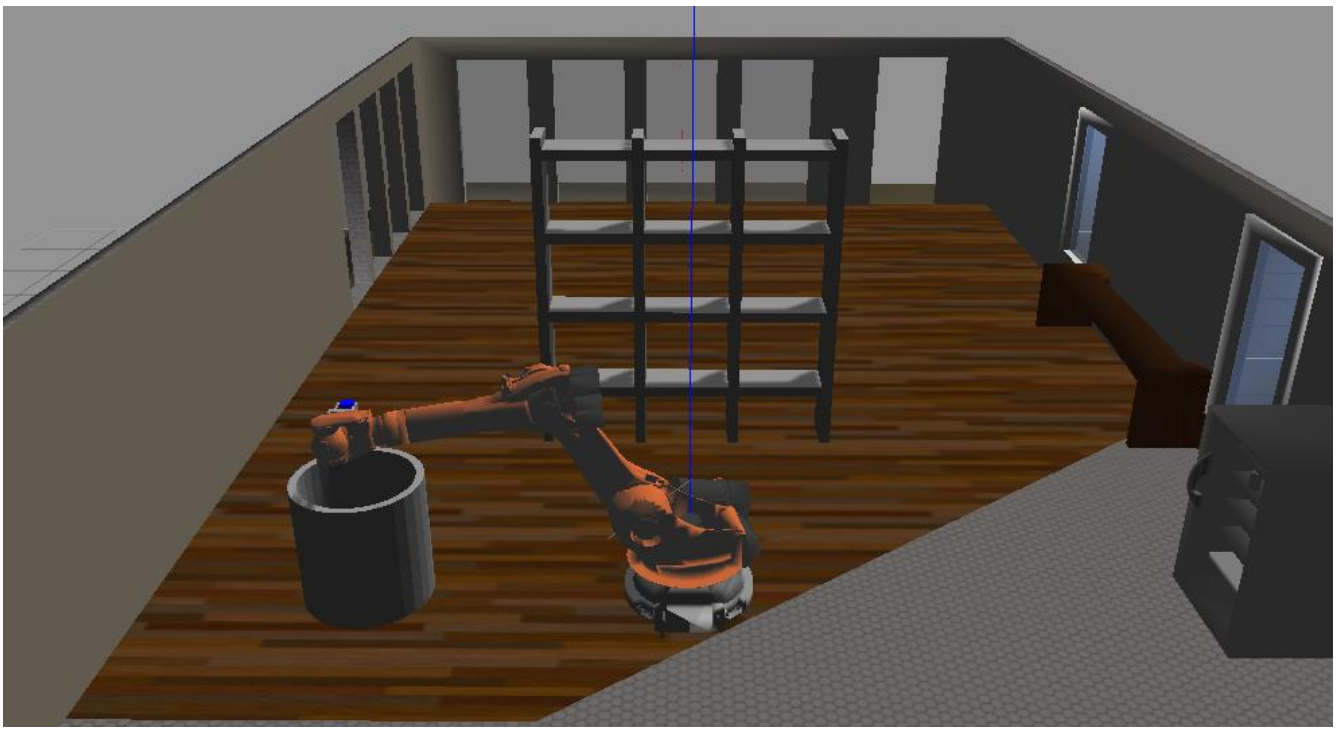
About the results, using *IK_debug.py* was very helpful for testing. There I could find some errors in the code and corrected them.

Using the total homogeneous transform matrix, I could compute the error of the EE position evaluating the matrix with the calculated joint angles. Also I compared the $Rrpy$ matrix with the rotational matrix from ***frame 0*** to ***frame 6*** (including orientation correction), because they should by ideally the same. Some of the results are this:

```
robond@udacity:~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/scripts$ ./IK_server.py
Calculating...
Ready to receive an IK request

[INFO] [1504658453.758376, 1119.155000]: Received 10 eef-poses from the plan
[INFO] [1504658454.000074, 1119.215000]: length of Joint Trajectory List: 1
EEx error = 0.00000000
EEy error = 0.00000000
EEz error = 0.00000000
offset = 0.00000000
('Rotational matrix error = ', Matrix([
[                          0, -3.55449002071095e-14, 1.65213186085866e-14],
[                          0,    1.0991207943789e-14,  -1.47437753155053e-7],
[-6.1232170575691e-17,     1.4743775315505e-7,    1.0991207943789e-14]]))

[INFO] [1504658455.734852, 1119.752000]: length of Joint Trajectory List: 2
EEx error = 0.00000000
EEy error = 0.00000000
EEz error = 0.00000000
offset = 0.00000000
('Rotational matrix error = ', Matrix([
[-1.11022302462516e-16, 0.00317883909786799,  0.00525743282243831],
[                          0,  -0.0078844298944255,      0.118692356388353],
[-6.59194920871187e-17,    -0.118756665521622, -0.00773943109551545]]))

[INFO] [1504658457.106301, 1120.177000]: length of Joint Trajectory List: 3
EEx error = 0.00000000
EEy error = -0.00000000
EEz error = 0.00000000
offset = 0.00000000
```

('Rotational matrix error = ', Matrix([
[0, 0.0102431593623888, 0.0166746290158728],
[1.38777878078145e-17, -0.0264438469415571,   0.213563257538401],
[-7.63278329429795e-17, -0.213866463843686, -0.0256077232210206]]))

here we can see that the EE error is 0, but the rotational matrix has a small error in some of its elements whereas others has 0 error. This is for the fact that there are different solutions for the inverse orientation problem, and they are affected by the precision of the calculations. However the robotic arm is capable of follow the planned trajectory and put the cylinders in the bin.



Attached to this document is a video showing the robotic arm doing the pick and place.