Project: Kinematics Pick & Place

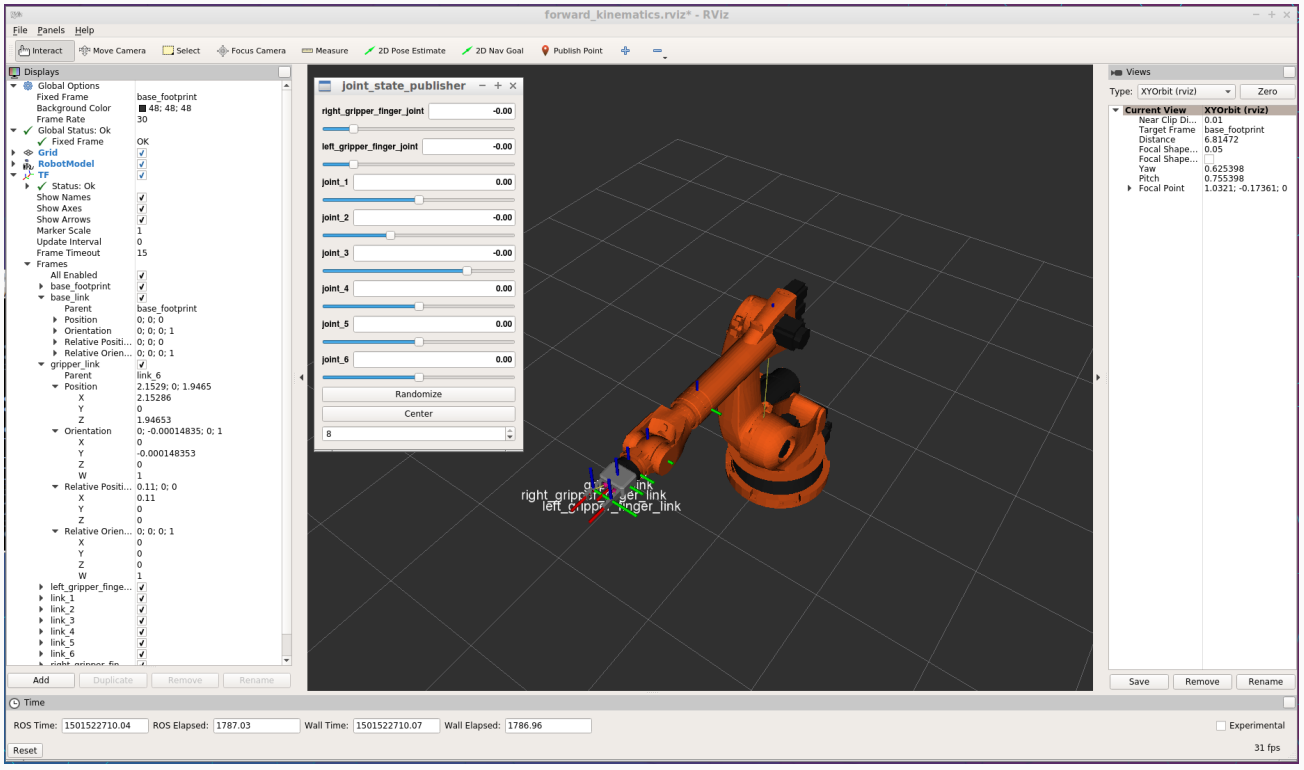
Steps to complete the project:

1. Set up your ROS Workspace.
2. Download or clone the project repository into the ***src*** directory of your ROS Workspace.
3. Experiment with the forward_kinematics environment and get familiar with the robot.
4. Launch in demo mode.
5. Perform Kinematic Analysis for the robot following the project rubric.
6. Fill in the `IK_server.py` with your Inverse Kinematics code.

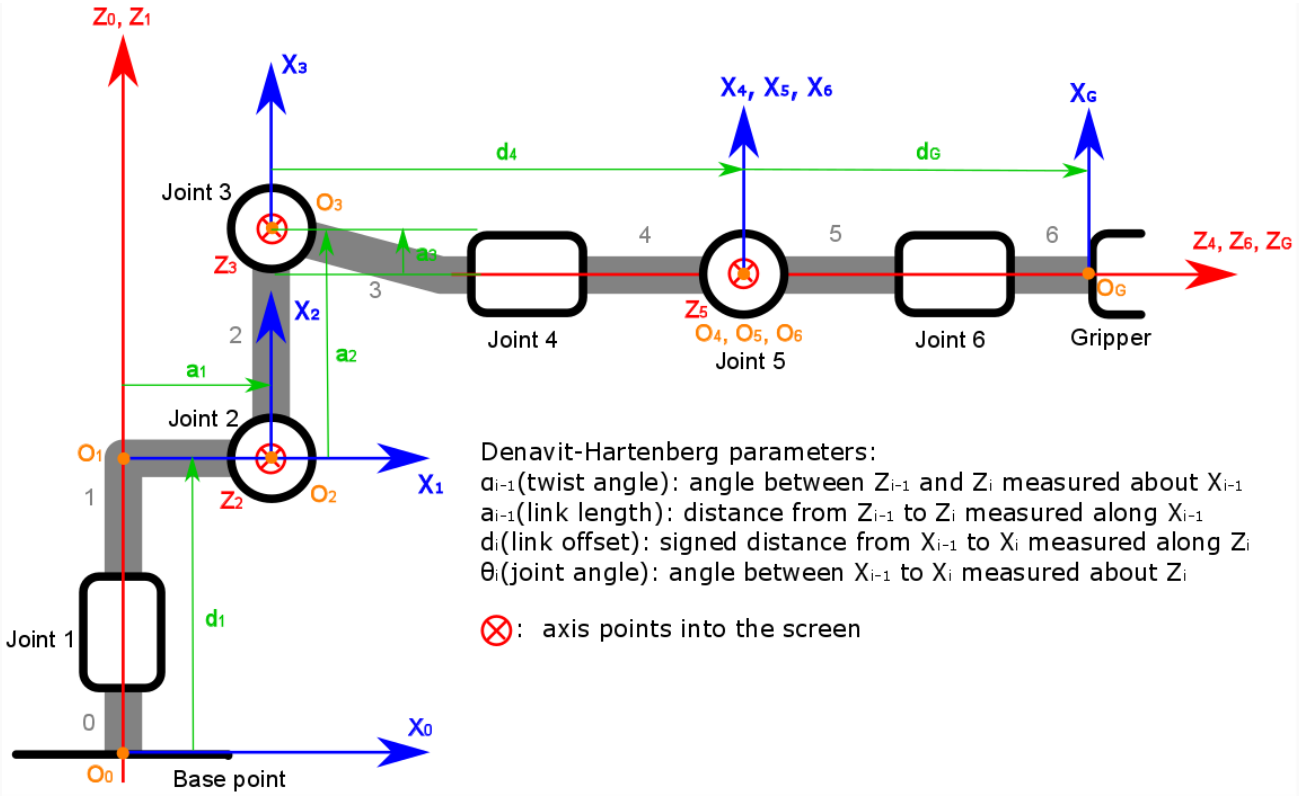
Kinematic Analysis

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.

Run command `roslaunch kuka_arm forward_kinematics.launch` and the screenshot is



The following figure shows the basic structure of Kuka KR210 robot.



Eight frames associate to the base point, six joints and the gripper. The DH parameter are assigned by using the algorithm in the course. Some of the DH parameter are shown in the figure.

One can derive the DH parameter through the demo, or from the file `kr210.urdf.xacro`. First of all, let us read the initial state (relative positions and orientations) of the joints through the demo,

Joint	X	Y	Z	Roll-Pitch-Yaw
1	0	0	0.33	0
2	0.35	0	0.42	0
3	0	0	1.25	0
4	0.96	0	-0.054	0
5	0.54	0	0	0
6	0.193	0	0	0
G	0.11	0	0	0

Then we can derive the DH parameter

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0.75	q1
2	-π/2	0.35	0	q2 - π/2

i				
3	0	1.25	0	q3
4	-π/2	-0.054	1.5	q4
5	π/2	0	0	q5
6	-π/2	0	0	q6
G	0	0	0.303	0

where qi, i= 1, 2, …, 6 are independent variables of Kuka KR210 robot. So the degrees of freedom of the robot is 6. Given a set of qi will capture a unique state of the robot. The $-\pi/2$ deficit between q2 is because axis X1 and X2 are orthogonal in the initial state.

2. Using the DH parameter table you derived earlier, create individual transformation matrices about each joint. In addition, also generate a generalized homogeneous transform between base_link and gripper_link using only end-effector(gripper) pose.

The individual transformation from frame i-1 to i by using the DH parameter is given as

$$T_i^{i-1} = R_X(\alpha_{i-1}) \cdot D_X(a_{i-1}) \cdot R_Z(\theta_i) \cdot D_Z(d_i)$$

where R_X and R_Z are rotations around axis X and Z, respectively, while D_X and D_Z are translations around axis X and Z, respectively. Practically, R and D are all homogeneous transformations, i.e. 4×4 matrices.

$$R = \begin{pmatrix} r_{3\times3} & 0_{3\times1} \\ 0 & 1 \end{pmatrix}, \quad D = \begin{pmatrix} I_{3\times3} & p_{3\times1} \\ 0 & 1 \end{pmatrix}$$

where r is a 3d rotation matrix, I is the identity matrix and p is a 3-vector.

Moreover, between frame G and frame gripper_link there is an extra rotation, which is

$$T_g^G = R_Y\left(-\frac{\pi}{2}\right) \cdot R_Z(\pi) = \begin{pmatrix} r_g^G & 0 \\ 0 & 1 \end{pmatrix}, \quad \text{where} \quad r_g^G = \begin{pmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Then the homogeneous transform between base_link and gripper_link is given as

$$T_g^0 = T_1^0 \cdot T_2^1 \cdot \ \cdots \ \cdot T_G^6 \cdot T_g^G$$

A vector V_{Og} from the origin O to the gripper can be obtained by using T_g^0 to transform the gripper pose in the gripper_link frame $V_g = (0, 0, 0, 1)^T$:

$$V_{Og} = T_g^0 \cdot V_g$$

3. Decouple Inverse Kinematics problem into Inverse Position Kinematics and inverse Orientation Kinematics; doing so derive the equations to calculate all individual joint angles.

The KUKA KR210 robot can be decomposed into two parts. The part with the first three joints (joint 1, 2 and 3) capture mainly the position kinematics of the robot, while the other part with the rest joints (joint 4, 5 and 6), called the spherical wrist, capture mainly the orientation kinematics of the robot. The center of the spherical wrist is called the wrist center (WC). It sits at the joint 5 and coincides to the origins of frame 4, 5 and 6.

In order to obtain q1, q2, …, q6 from the pose of the gripper, we follow the steps below:

step 1: Split the pose of the gripper to frame 0 into two parts.

In the problem of inverse kinematics, what we know in prior is the pose of the gripper to the origin, which is given by T_g^0

$$T_g^0 = \begin{pmatrix} r_g^0 & p_g \\ 0 & 1 \end{pmatrix}$$

where r_g^0 gives the orientation and p_g gives the position of the gripper. Remind the decomposition $T_g^0 = T_6^0 \cdot T_G^6 \cdot T_g^G$. We also have

$$T_g^0 = T_6^0 \cdot T_G^6 \cdot T_g^G = \begin{pmatrix} r_6^0 & p_{\text{wc}} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & d_G \hat{z} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} r_g^G & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} r_6^0 \cdot r_g^G & p_{\text{wc}} + d_G \; r_6^0 \cdot \hat{z} \\ 0 & 1 \end{pmatrix}$$

Comparing the above two equations, one can immediately get

$$r_6^0 = r_g^0 \cdot (r_g^G)^{-1} = r_g^0 \cdot r_g^G$$
$$p_{\text{wc}} = p_g - d_G r_g^0 \cdot (r_g^G)^{-1} \cdot \hat{z} = p_g - d_G r_g^0 \cdot \hat{x}$$

where $\hat{z} \equiv (0, 0, 1)^T$, $\hat{x} \equiv (1, 0, 0)^T$ and $(r_g^G)^{-1} = r_g^G$. p_{wc} is the position of the WC to frame 0.

q1 to q6 will be obtained as functions of p_{wc} and r_6^0.

step 2: obtain q1, q2, q3 such that WC's pose is $p_{\text{wc}} \equiv (p_x, p_y, p_z)$.

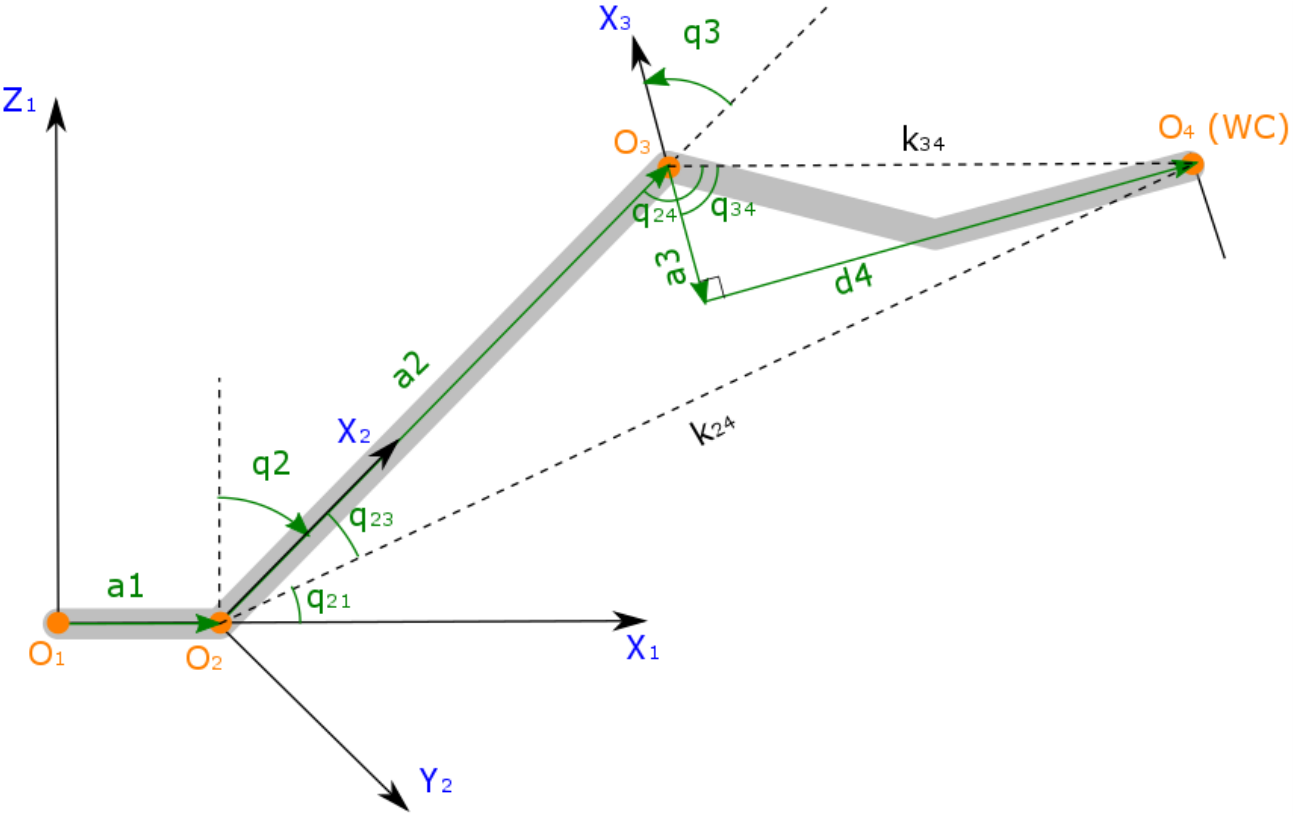
q1 is quite easy to get

$$q_1 = \text{atan2}(p_y, p_x)$$

We then move our reference frame to frame 1. The pose of WC in the new frame is then

$$(p_{\text{wc1}},\ 1)^T \equiv (p_{x1}, 0,\ p_{z1},\ 1)^T \equiv [T_1^0]^{-1} \cdot (p_{\text{wc}},\ 1)^T$$

The system in frame 1 is shown in the following figure



Let us get q3 first:

$$q_{34} = \text{atan2}(d_4, a_3)$$

$$k_{34} = \sqrt{d_4^2 + a_3^2}$$

$$k_{24} = \sqrt{(p_{x1} - a_1)^2 + p_{z1}^2}$$

Then

$$\cos(q_{24}) = \frac{a_2^2 + k_{34}^2 - k_{24}^2}{2a_2 k_{34}}$$

$$q_{24} = \text{atan2}\left(\sqrt{1 - \cos^2(q_{24})},\ \cos(q_{24})\right)$$

Then

$$q_3 = -(q_{24} - q_{34})$$

Here we assume that q_{24} from k_{34} to a_2 clock-wise is always not bigger than π. The minus sign in front of the above equation is because q3 is negative if the arm rotates anti-clock-wise from its initial state in our convention.

Then q_2:

$$\cos(q_{23}) = \frac{a_2^2 + k_{24}^2 - k_{34}^2}{2a_2 k_{24}}$$

$$q_{23} = \text{atan2}\left(\sqrt{1 - \cos^2(q_{23})},\ \cos(q_{23})\right)$$

$$q_{21} = \text{atan2}(p_{z1},\ p_{x1} - a_1)$$

Then

$$q_2 = \frac{\pi}{2} - q_{23} - q_{21}$$

step 3: carry out q4, q5 and q6.

By using p1, p2, p3, we can obtain T_3^0, then T_6^3 then can be given as

$$T_6^3 = [T_3^0]^{-1} T_6^0 \equiv \begin{pmatrix} r_{11} & r_{12} & r_{13} & k_1 \\ r_{21} & r_{22} & r_{23} & k_2 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

While T_6^3 can also be obtained by using T_4^3, T_5^4 and T_6^5

$$T_6^3 = \begin{pmatrix} -\sin q_4 \sin q_6 + \cos q_4 \cos q_6 \cos q_5 & -\sin q_4 \cos q_6 - \cos q_4 \sin q_6 \cos q_5 & -\cos q_4 \sin q_5 & -0.054 \\ \sin q_5 \cos q_6 & -\sin q_5 \sin q_6 & \cos q_5 & 1.5 \\ -\sin q_4 \cos q_6 \cos q_5 - \sin q_6 \cos q_4 & \sin q_4 \sin q_6 \cos q_5 - \cos q_4 \cos q_6 & \sin q_4 \sin q_5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Comparing the above two equations, we will get

$$\begin{aligned} q_6 &= \text{atan2}(-r_{22},\ r_{21}) \\ q_4 &= \text{atan2}(r_{33},\ -r_{13}) \\ q_5 &= \text{atan2}\left(\sqrt{r_{21}^2 + r_{22}^2},\ r_{23}\right) \end{aligned}$$

Here we assume that q5 is from 0 to π.

Project Implementation

1. Fill in the `IK_server.py` file with properly commented python code for calculating Inverse Kinematics based on previously performed Kinematic Analysis. Your code must guide the robot to successfully complete 8/10 pick and place cycles. Briefly discuss the code you implemented and your results.

Basically I followed the previous section of Kinematics Analysis.

First define global constants: the identity matrix

```
I = Matrix([[1, 0, 0],
            [0, 1, 0],
            [0, 0, 1]])
```

and the rotation rGg between frame G and frame gripper_link

```
rGg = Matrix([[0, 0,  1],
              [0, -1, 0],
              [1, 0,  0]])
```

and DH parameter

```
# Create Modified DH parameters
s = {'alpha0': 0,     'a0': 0,      'd1': 0.75,  'theta1': q1,
     'alpha1': -pi/2, 'a1': 0.35,   'd2': 0,     'theta2': q2 - pi/2,
     'alpha2': 0,     'a2': 1.25,   'd3': 0,     'theta3': q3,
     'alpha3': -pi/2, 'a3': -0.054, 'd4': 1.5,   'theta4': q4,
     'alpha4': pi/2,  'a4': 0,      'd5': 0,     'theta5': q5,
     'alpha5': -pi/2, 'a5': 0,      'd6': 0,     'theta6': q6,
     'alpha6': 0,     'a6': 0,      'dG': 0.303, 'thetaG': 0}
```

Then define the functions of transform.

Rotations:

```
def rot_x(q):
    '''
    rotation around axis X
    '''
    R_x = Matrix([[ 1,              0,        0],
                  [ 0,         cos(q),  -sin(q)],
                  [ 0,         sin(q),  cos(q)]])

    return R_x

def rot_y(q):
    '''
    rotation around axis Y
    '''
    R_y = Matrix([[ cos(q),        0,  sin(q)],
                  [      0,        1,       0],
                  [-sin(q),        0, cos(q)]])

    return R_y

def rot_z(q):
    '''
    rotation around axis Z
    '''
    R_z = Matrix([[ cos(q),  -sin(q),       0],
                  [ sin(q),   cos(q),       0],
                  [      0,        0,       1]])

    return R_z
```

translations:

```
def p_x(d):
    '''
    translation along axis X
    '''
    t = Matrix([[d], [0], [0]])
    return t

def p_y(d):
    '''
    translation along axis Y
    '''
    t = Matrix([[0], [d], [0]])
    return t

def p_z(d):
    '''
    translation along axis Z
    '''
    t = Matrix([[0], [0], [d]])
    return t
```

and Homogeneous transformations

```
# homogeneous transform: put rotation and translation together
def H(R, p=Matrix([[0], [0], [0]])):
    '''
    R is rotation matrix
    p is translation vector
    '''
    D = R.row_join(p)
    D = D.col_join(Matrix([[0,0,0,1]]))
    return D

# homogeneous transform between neighbouring frames
def T(alpha, a, theta, d):
    T = H(rot_x(alpha)) * H(I, p_x(a)) * H(rot_z(theta)) * H(I, p_z(d))
    return T
```

Then the homogeneous transformation between joints can be defined as

```
# Homogeneous transform between frames
T01 = T(s['alpha0'], s['a0'], s['theta1'], s['d1'])
T12 = T(s['alpha1'], s['a1'], s['theta2'], s['d2'])
T23 = T(s['alpha2'], s['a2'], s['theta3'], s['d3'])
T34 = T(s['alpha3'], s['a3'], s['theta4'], s['d4'])
T45 = T(s['alpha4'], s['a4'], s['theta5'], s['d5'])
T56 = T(s['alpha5'], s['a5'], s['theta6'], s['d6'])
T6G = T(s['alpha6'], s['a6'], s['thetaG'], s['dG'])
TGg = H(rGg)
#
# transform from base_point to G
T0g = T01 * T12 * T23 * T34 * T45 * T56 * T6G * TGg
```

Finally we define the function for the inverse kinematics, which calculate the joint angles.

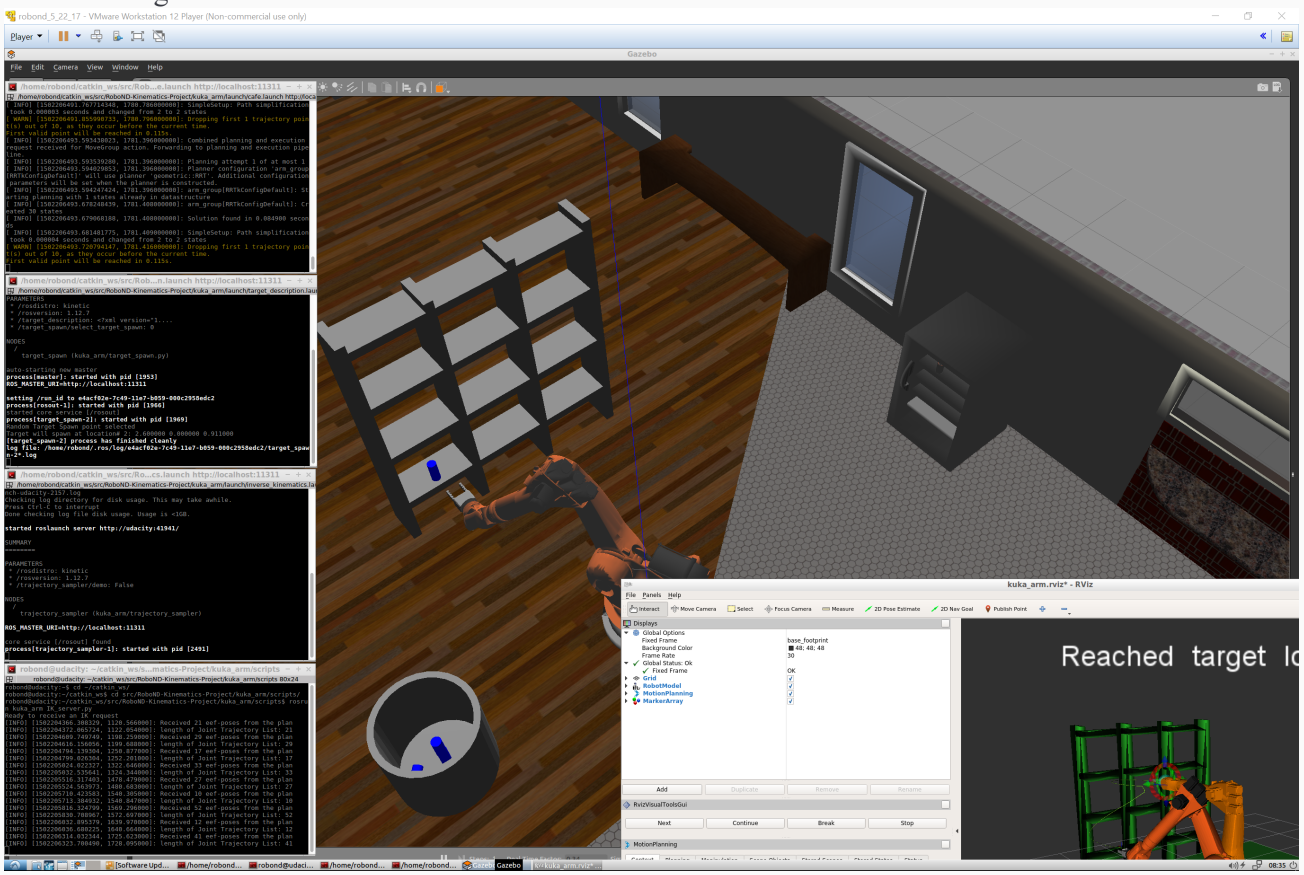
```
##########################
# Inverse Kinematics functions
##########################
def pose_wc(p_g, o_g):
    '''
    get the orientation and position of wrist center (WC) on frame 6
    inputs:
    p_g: numpy array, [x, y, z] position of gripper to origin
    o_g: numpy array, [x, y, z, w] quaternion of gripper to origin
    '''
    # get homogeneous transform of rotation from quaternion og
    T_0g = tf.transformations.quaternion_matrix(o_g)
    # identify pg with the translation part
    T_0g[:3, 3] = p_g

    # np array rGg
    r_Gg = np.array(rGg).astype(np.float32)
    # r06: the 3 rotation from frame 0 to frame 6
    r_06 = T_0g[:3, :3].dot(r_Gg)
    # pwc: position of WC
    x = np.array([[1.], [0.], [0.]])
    p_wc = np.array(p_g).reshape(3,1) - s['dG'] * T_0g[:3,:3].dot(x)
    # return T_0g, r_06, p_wc
    return T_0g, r_06, p_wc

def P_wc1(p_wc):
    '''
    get position of WC to frame 1
    p_wc: position of WC to frame 0
    '''
    _, T_01 = Q_1(p_wc)
    # get inverse matrix of T_01
    T_01_inv = np.linalg.inv(T_01)
    # 4-vector of p_wc
    p_wc_4 = np.append(p_wc, 1).reshape(4, 1)
    # p_wc1: WC position to frame 1
    p_wc1 = T_01_inv.dot(p_wc_4)
    # p_wc1 = p_wc1[:3]
    return p_wc1

def Q_1(p_wc):
```

```
    '''
    calculate q1
    inputs:
    p_wc: position of WC to frame 0
    '''
    q_1 = atan2(p_wc[1], p_wc[0]).evalf()
    T_01 = np.array(T(s['alpha0'], s['a0'], s['theta1'], s['d1']).evalf(subs={q1: q_1})).astype(np.float32)
    return q_1, T_01

def Q_2(p_wc1):
    '''
    calculate q2
    inputs:
    p_wc1: WC position to frame 1, numpy array shape=(4, 1),
    '''
    k34 = sqrt(s['d4'] ** 2 + s['a3'] ** 2)
    k24 = sqrt((p_wc1[0] - s['a1']) ** 2 + p_wc1[2] ** 2)
    cosq23 = (s['a2']**2 + k24**2 - k34**2) / (2 * s['a2'] * k24)
    q23 = atan2(sqrt(1 - cosq23**2), cosq23)

    q21 = atan2(p_wc1[2], p_wc1[0]-s['a1'])

    q_2 = (pi/2 - q23 - q21).evalf()
    T_12 = np.array(T(s['alpha1'], s['a1'], s['theta2'], s['d2']).evalf(subs={q2: q_2})).astype(np.float32)
    return q_2, T_12

def Q_3(p_wc1):
    '''
    calculate q3
    inputs:
    p_wc1: WC position to frame 1, numpy array shape=(4, 1),
    '''
    q34 = atan2(s['d4'], s['a3'])
    k34 = sqrt(s['d4'] ** 2 + s['a3'] ** 2)
    k24 = sqrt((p_wc1[0] - s['a1']) ** 2 + p_wc1[2] ** 2)

    cosq24 = (s['a2']**2 + k34**2 - k24**2) / (2 * s['a2'] * k34)
    q24 = atan2(sqrt(1 - cosq24**2), cosq24)

    q_3 = (-q24 + q34).evalf()
    T_23 = np.array(T(s['alpha2'], s['a2'], s['theta3'], s['d3']).evalf(subs={q3: q_3})).astype(np.float32)
    return q_3, T_23

def Q_456(r_03, r_06):
    '''
    calculate q4, q5, q6
    inputs:
    r_03: rotation from frame 3 to frame 0, numpy array, shape=(3,3)
    r_06: rotation from frame 6 to frame 6, numpy array, shape=(3,3)
    '''
    r_03_inv = np.linalg.inv(r_03)
    r_36 = r_03_inv.dot(r_06)

    r21, r22, r23 = r_36[1, :]
    r13, r33 = r_36[[0,2], 2]

    q_4 = atan2(r33, -r13).evalf()
    q_5 = atan2(sqrt(r21**2 + r22**2), r23).evalf()
    q_6 = atan2(-r22, r21).evalf()
    T_34 = np.array(T(s['alpha3'], s['a3'], s['theta4'], s['d4']).evalf(subs={q4: q_4})).astype(np.float32)
    T_45 = np.array(T(s['alpha4'], s['a4'], s['theta5'], s['d5']).evalf(subs={q5: q_5})).astype(np.float32)
    T_56 = np.array(T(s['alpha5'], s['a5'], s['theta6'], s['d6']).evalf(subs={q6: q_6})).astype(np.float32)
    return q_4, q_5, q_6, T_34, T_45, T_56
```

The inverse kinematics procedure is then given as

```
T_0g, r_06, p_wc = pose_wc(p_g, o_g) # get the pose of WC
q_1, T_01 = Q_1(p_wc) # get q1
p_wc1 = P_wc1(p_wc) # get position of WC to frame 1
q_2, T_12 = Q_2(p_wc1) # get q2
q_3, T_23 = Q_3(p_wc1) # get q3
T_03 = T_01.dot(T_12).dot(T_23) # transform from base_point to joint 3
r_03 = T_03[:3, :3] # rotation part of T_03
q_4, q_5, q_6, T_34, T_45, T_56 = Q_456(r_03, r_06) # get q4, q5, q6
```

After the implementation, the arm can pick up targets and drop off into the dustbin. Here is a screenshot when running the code.



PS

I think the course is a bit unfriendly to beginners, and a bit unorganized. Some of the part is unclear, especially when using math. It would be better if some of the equations can have derivations.