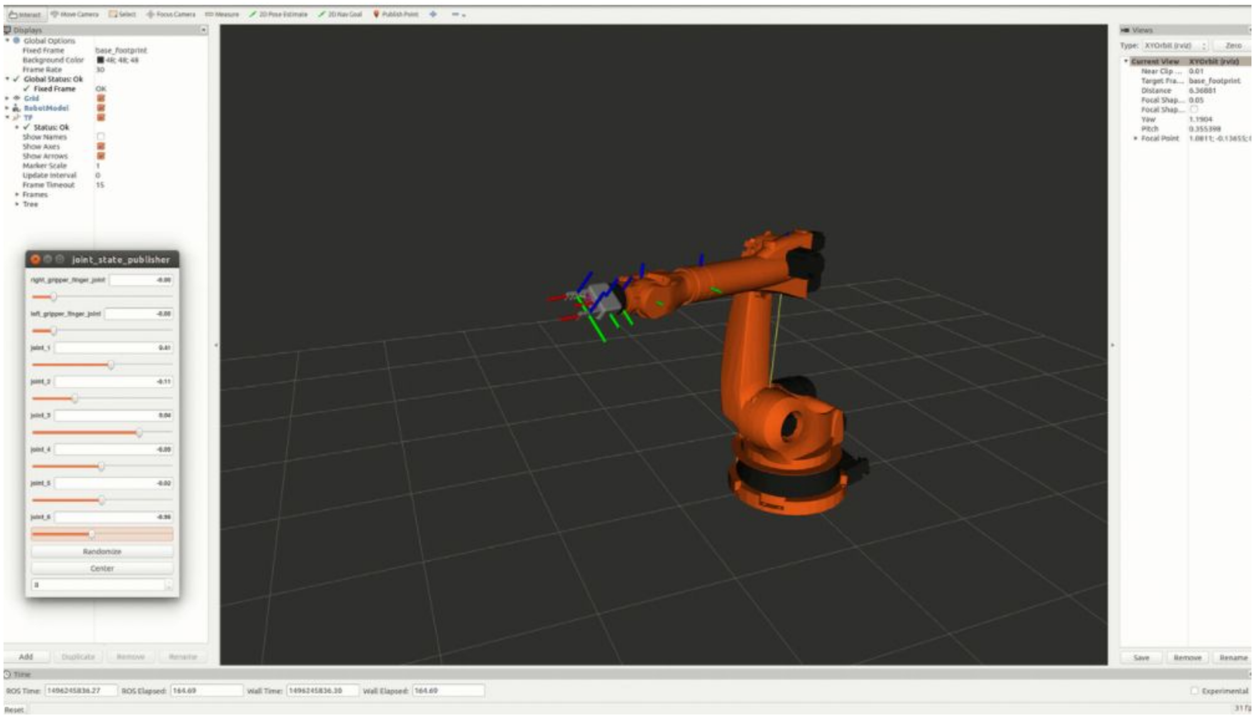
# Kinematic Analysis:

## Kuka KR210 six degree of freedom serial manipulator



## Modified DH-Parameters:

By reading kr210.urdf.xacros files, we could derive its DH parameters as shown in Figure 1. The DH parameters are defined as follows:

- $\alpha_{i-1}$ (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.

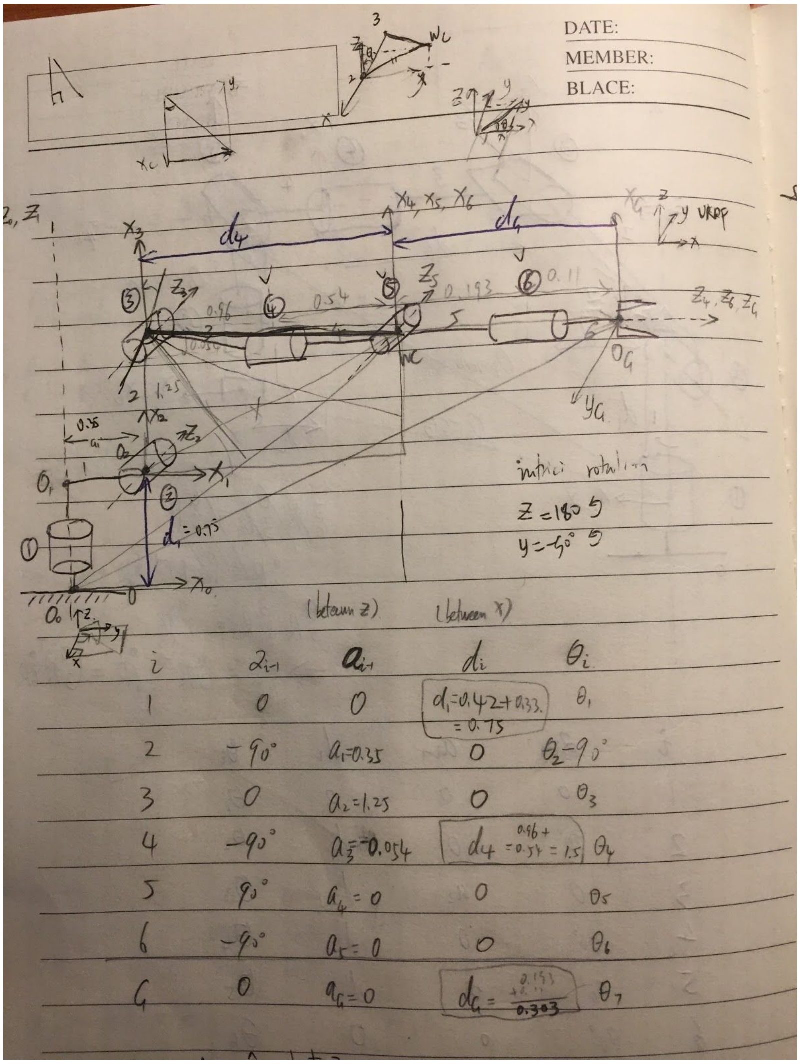


Figure 1. Robot links assignments, joint rotations, and DH parameters table.

## Individual Transformation Matrices About Each Joint

The transformation matrix from frame i-1 to  frame i is given by equation,

$$ {}^{i-1}_{\ \ \ i}T = R(x_{i-1},\alpha_{i-1})\,T(x_{i-1},a_{i-1})\,R(z_i,\theta_i)\,T(z_i,d_i) $$

This individual homogeneous transformation matrix means that we first rotate about x[i-1] by alpha[i-1]. Then, translate along x[i-1] by a[i-1]. Next, rotate about resulting axis z[i] by theta[i]. Finally, translate along axis z[i] by d[i]. We will use sympy to write out the symbolic transformation matrices about each joint using the DH table shown in Figure 1 and total homogeneous transformation matrix from base_link to gripper_linker. The symbolic transform matrix for individual frame is given by,

$$
{}^{i-1}_{\ \ i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Therefore, by plugging in corresponding DH parameters, we can get individual transformation matrix as follows:

```
('T0_1 is', Matrix([
[cos(q1), -sin(q1), 0,    0],
[sin(q1),  cos(q1), 0,    0],
[      0,        0, 1, 0.75],
[      0,        0, 0,    1]]))
('T1_2 is', Matrix([
[ cos(q2 - 0.5*pi), -sin(q2 - 0.5*pi), 0, 0.35],
[                0,                 0, 1,    0],
[-sin(q2 - 0.5*pi), -cos(q2 - 0.5*pi), 0,    0],
[                0,                 0, 0,    1]]))
('T2_3 is', Matrix([
[cos(q3), -sin(q3), 0, 1.25],
[sin(q3),  cos(q3), 0,    0],
[      0,        0, 1,    0],
[      0,        0, 0,    1]]))
('T3_4 is', Matrix([
[ cos(q4), -sin(q4), 0, -0.054],
[       0,        0, 1,    1.5],
[-sin(q4), -cos(q4), 0,      0],
[       0,        0, 0,      1]]))
('T4_5 is', Matrix([
[cos(q5), -sin(q5),  0, 0],
[      0,        0, -1, 0],
[sin(q5),  cos(q5),  0, 0],
[      0,        0,  0, 1]]))
('T5_6 is', Matrix([
[ cos(q6), -sin(q6), 0, 0],
[       0,        0, 1, 0],
[-sin(q6), -cos(q6), 0, 0],
[       0,        0, 0, 1]]))
('T6_EE is', Matrix([
[1, 0, 0,     0],
[0, 1, 0,     0],
[0, 0, 1, 0.303],
[0, 0, 0,     1]]))
('R3_6 is', Matrix([
[-sin(q4)*sin(q6) + cos(q4)*cos(q5)*cos(q6), -sin(q4)*cos(q6) - sin(q6)*cos(q4)*cos(q5), -sin(q5)*cos(q4)],
[                          sin(q5)*cos(q6),                           -sin(q5)*sin(q6),          cos(q5)],
[-sin(q4)*cos(q5)*cos(q6) - sin(q6)*cos(q4),  sin(q4)*sin(q6)*cos(q5) - cos(q4)*cos(q6),  sin(q4)*sin(q5)]]))
```

Figure 2: Individual transformation matrix.

In Figure 2, I also include the rotation matrix from joint 3 to joint 6. It will be easy for us to calculate theta4, theta5 and theta6 later on.

The total transformation matrix from base_link to gripper_linker is given by:
L0_EE = L0_1 * L1_2 * L2_3 * L3_4 * L4_5 * L5_6 * L6_EE
By plugging in these individual transformation matrix, we can get L0_EE to be,

```
('T0_EE is', Matrix([
[((sin(q1)*sin(q4) + sin(q2 + q3)*cos(q1)*cos(q4))*cos(q5) + sin(q5)*cos(q1)*cos(q2 + q3))*cos(q6) + (sin(q1)*cos(q4) - sin(q4)*sin(q2 + q3)*cos(q1))*sin(q6), -((sin(q1)*sin(q4) + sin(q2 + q3)*cos(q1)*cos(q4))*cos(q5) + sin(q5)*cos(q1)*c
os(q2 + q3))*sin(q6) + (sin(q1)*cos(q4) - sin(q4)*sin(q2 + q3)*cos(q1))*cos(q6), -(sin(q1)*sin(q4) + sin(q2 + q3)*cos(q1)*cos(q4))*sin(q5) + cos(q1)*cos(q5)*cos(q2 + q3), -0.303*sin(q1)*sin(q4)*sin(q5) + 1.25*sin(q2)*cos(q1) - 0.303*sin(
q5)*sin(q2 + q3)*cos(q1)*cos(q4) - 0.054*sin(q2 + q3)*cos(q1) + 0.303*cos(q1)*cos(q5)*cos(q2 + q3) + 1.5*cos(q1)*cos(q2 + q3) + 0.35*cos(q1)],
[((sin(q1)*sin(q2 + q3)*cos(q4) - sin(q4)*cos(q1))*cos(q5) + sin(q1)*sin(q5)*cos(q2 + q3))*cos(q6) - (sin(q1)*sin(q4)*sin(q2 + q3) + cos(q1)*cos(q4))*sin(q6), ((-sin(q1)*sin(q2 + q3)*cos(q4) + sin(q4)*cos(q1))*cos(q5) - sin(q1)*sin(q5)*c
os(q2 + q3))*sin(q6) - (sin(q1)*sin(q4)*sin(q2 + q3) + cos(q1)*cos(q4))*cos(q6), (-sin(q1)*sin(q2 + q3)*cos(q4) + sin(q4)*cos(q1))*sin(q5) + sin(q1)*cos(q5)*cos(q2 + q3),  1.25*sin(q1)*sin(q2) - 0.303*sin(q1)*sin(q5)*sin(q2 + q3)*cos(q4)
 - 0.054*sin(q1)*sin(q2 + q3) + 0.303*sin(q1)*cos(q5)*cos(q2 + q3) + 1.5*sin(q1)*cos(q2 + q3) + 0.35*sin(q1) + 0.303*sin(q4)*sin(q5)*cos(q1)],
[                                (-sin(q5)*sin(q2 + q3) + cos(q4)*cos(q5)*cos(q2 + q3))*cos(q6) - sin(q4)*sin(q6)*cos(q2 + q3),                                                    (sin(q5)*sin(
q2 + q3) - cos(q4)*cos(q5)*cos(q2 + q3))*sin(q6) - sin(q4)*cos(q6)*cos(q2 + q3),                                     -sin(q5)*cos(q4)*cos(q2 + q3) - sin(q2 + q3)*cos(q5),
       -0.303*sin(q5)*cos(q4)*cos(q2 + q3) - 0.303*sin(q2 + q3)*cos(q5) - 1.5*sin(q2 + q3) + 1.25*cos(q2) - 0.054*cos(q2 + q3) + 0.75],
[
                                        0,                                        0,                                        0,
                                        1]]))
```

## Decouple Inverse Kinematics Problem

The next step we will decouple inverse kinematics problem into inverse position kinematics and inverse orientation kinematics. For the inverse position kinematics, we will try to solve the wrist center coordinate. After solving the wrist center, we will target on solving the inverse orientation kinematics problem. For the inverse orientation kinematics problem, we first solve the first three joint angle and then solve the last three joint angle.

First, let's find out the position of wrist center.

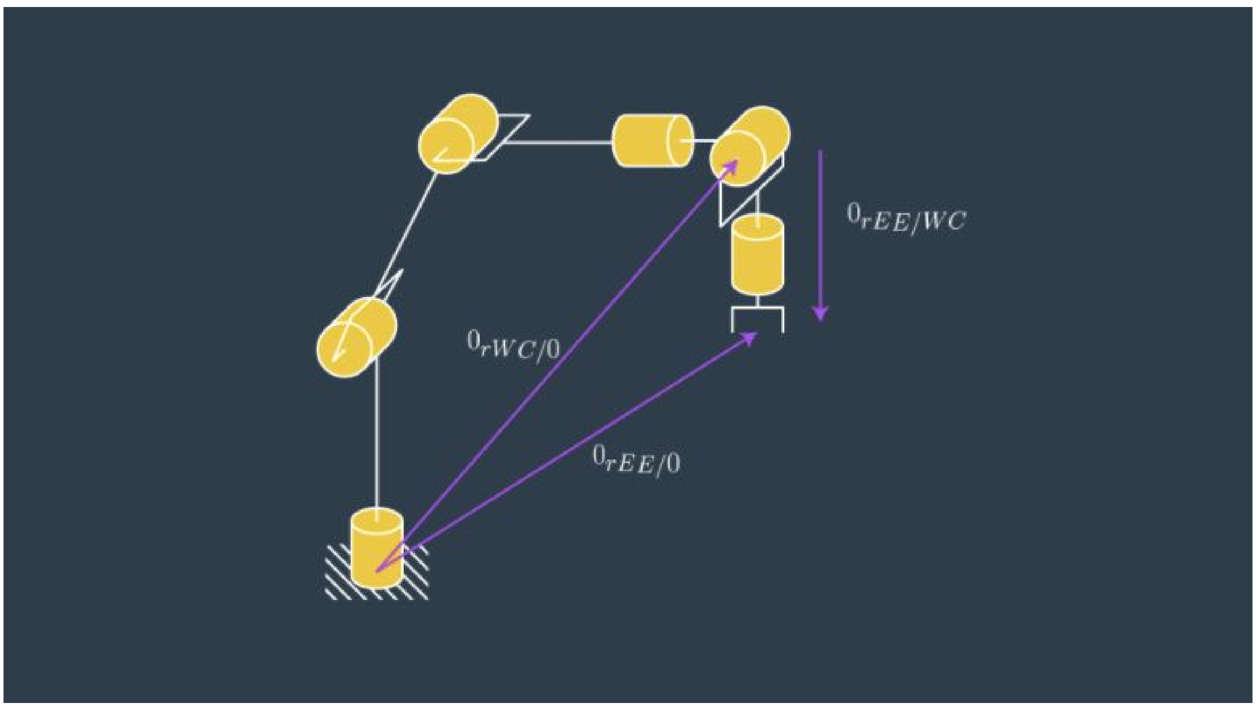



Figure 3: Inverse position kinematics for solving wrist center position

From Figure 3, we could derive the vector equation,

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

From the DH table, d is d7 = 0.303. The end effector position [px, py, pz] is given because it is inverse kinematics problem. We also know the roll, pitch, yaw angle from the base_link with respect to end effector. The roll, pitch, yaw are extrinsic rotations, so we can build the transformation matrix from the base_link to end effector. However, we need to pay attention to the difference between the gripper reference frame as defined in the URDF versus the DH parameters, so we have to perform a 180 degree counterclockwise rotation about gripper frame z axis and then a 90 degree clockwise rotation about the gripper frame y axis to form the right R0_6 matrix. The error_rotation matrix performance is shown in Figure 4.

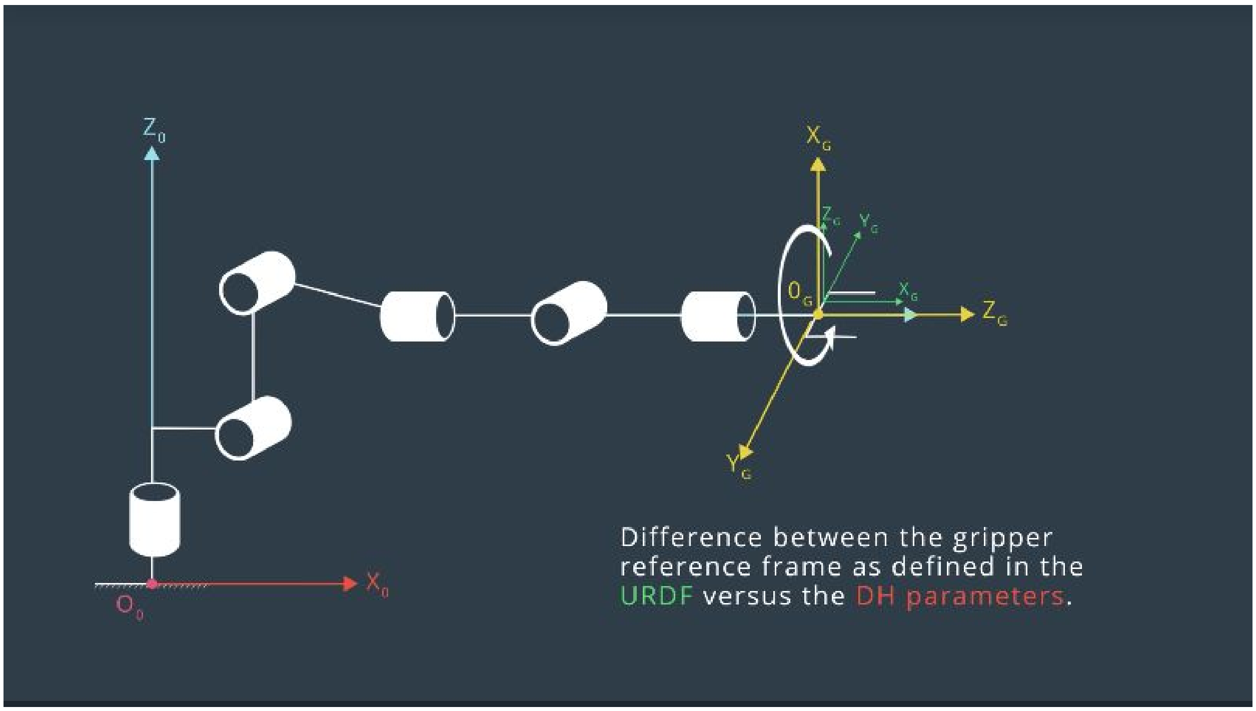



Figure 4: Error correction matrix rotation

Therefore, we first define the roll, pitch, yaw rotation matrix, then do extrinsic rotations wrt base_link frame. Finally, multiply the extrinsic rotation matrix by the error correction matrix. The procedure is shown in Figure 5.

```
RotRoll_x  = Matrix([[1,      0,      0],
                     [0, cos(R),-sin(R)],
                     [0, sin(R), cos(R)]]);
RotPitch_y = Matrix([[cos(P),  0, sin(P)],
                     [0,       1,      0],
                     [-sin(P), 0, cos(P)]]);
RotYaw_z   = Matrix([[cos(Y), -sin(Y),  0],
                     [sin(Y),  cos(Y),  0],
                     [0,            0,  1]]);
Rot_EE = RotYaw_z * RotPitch_y * RotRoll_x;
Rot_Error = RotYaw_z.subs(Y, radians(180)) * RotPitch_y.subs(P, radians(-90));
Rot_EE = Rot_EE * Rot_Error;
```

Plug the Rot_EE, Euler angles into the above equations, we can find the wrist center position.

Next, let's solve the inverse orientation kinematics problem. We will first draw out the joint rotation diagram for the first three joints shown in Figure 5.

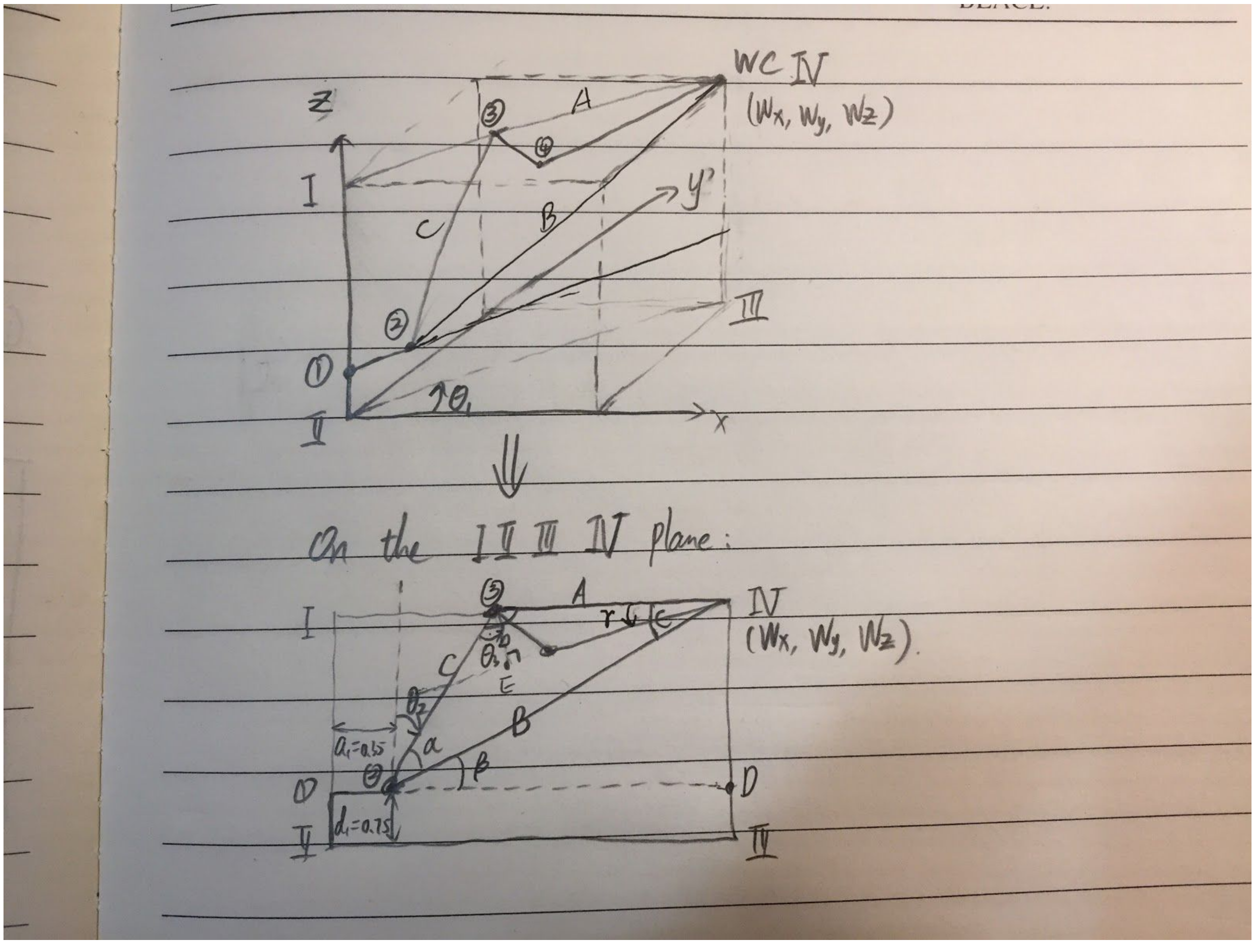
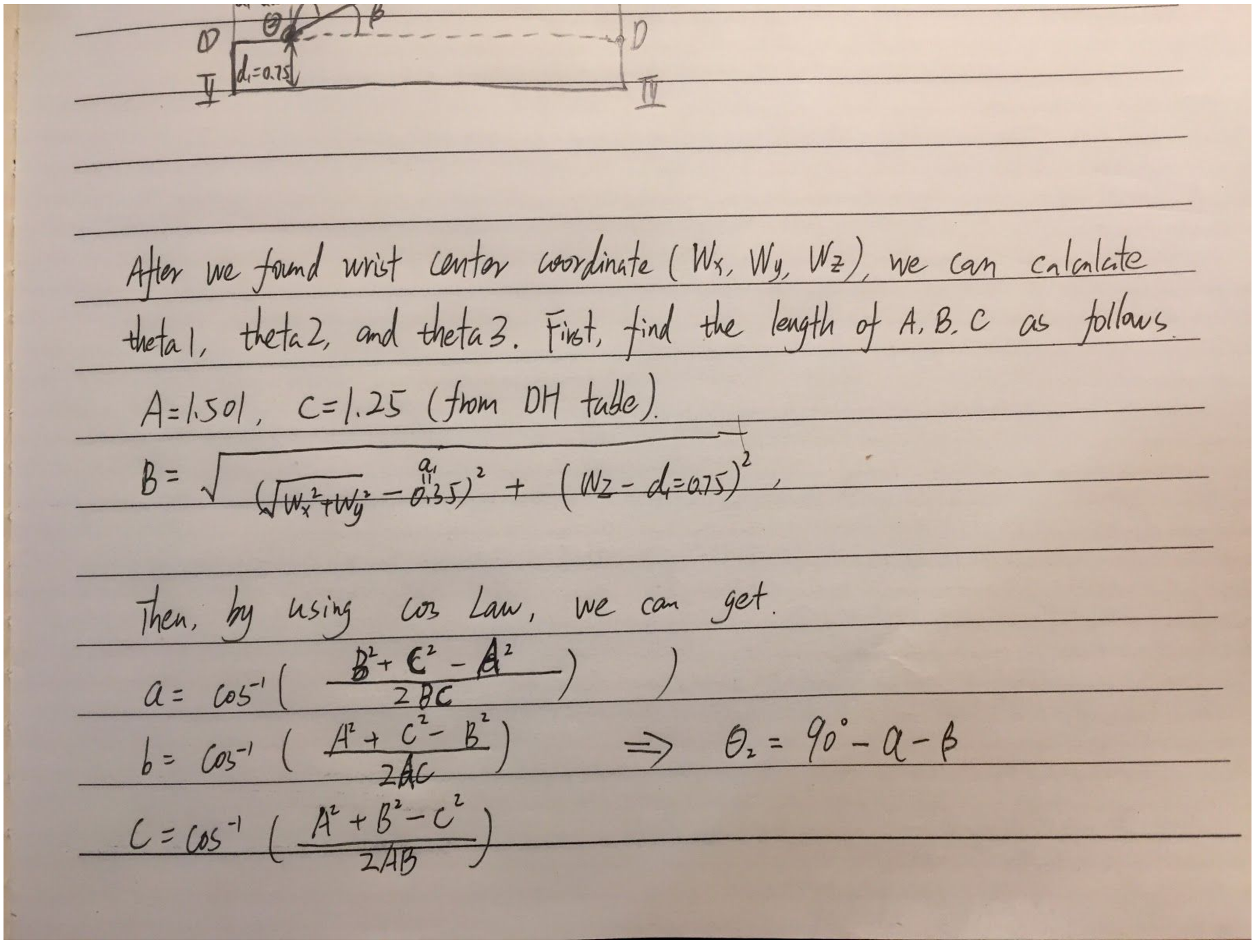



Figure 5: Joints angle for the first three joints.

The procedures for solving theta1, theta2, theta3 is shown in Figure 6 and Figure 7. Theta3 is the trickiest angle to solve. Remember theta 3 is the angle between X2 and X3 mesured about

Z3 in a right-hand sense. The direction is negative based on the picture we drawed.



$\theta$ $\beta$ $D$ $D$

$d_1 = 0.75$

After we found wrist center coordinate ($W_x$, $W_y$, $W_z$), we can calculate theta1, theta2, and theta3. First, find the length of A, B, C as follows.

$A = 1.501$, $C = 1.25$ (from DH table).

$$B = \sqrt{\left(\sqrt{W_x^2 + W_y^2} - \overset{a_1}{0.35}\right)^2 + \left(W_z - d_1 = 0.75\right)^2}$$

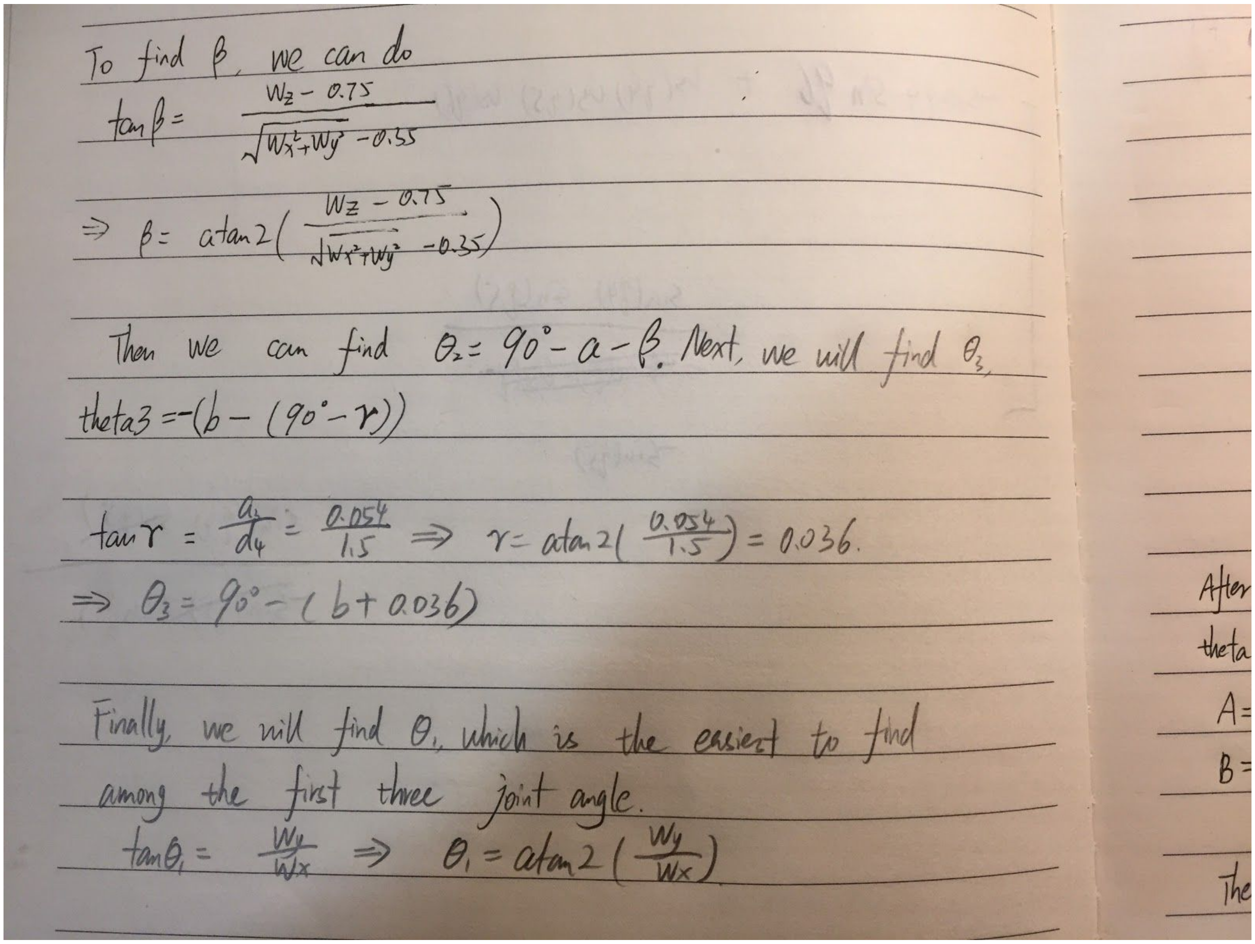
Then, by using cos Law, we can get.

$$a = \cos^{-1}\left(\frac{B^2 + C^2 - A^2}{2BC}\right)$$

$$b = \cos^{-1}\left(\frac{A^2 + C^2 - B^2}{2AC}\right) \Rightarrow \theta_2 = 90^\circ - a - \beta$$

$$c = \cos^{-1}\left(\frac{A^2 + B^2 - C^2}{2AB}\right)$$

Figure 6: Solving first three joints angles part 1.


To find $\beta$, we can do

$$\tan\beta = \frac{W_z - 0.75}{\sqrt{W_x^2+W_y^2} - 0.35}$$

$$\Rightarrow \beta = \mathrm{atan2}\left(\frac{W_z - 0.75}{\sqrt{W_x^2+W_y^2} - 0.35}\right)$$

Then we can find $\theta_2 = 90° - a - \beta$. Next, we will find $\theta_3$,

$$theta3 = -(b - (90° - \gamma))$$

$$\tan\gamma = \frac{a_3}{d_4} = \frac{0.054}{1.5} \Rightarrow \gamma = \mathrm{atan2}\left(\frac{0.054}{1.5}\right) = 0.036.$$

$$\Rightarrow \theta_3 = 90° - (b + 0.036)$$

Finally, we will find $\theta_1$, which is the easiest to find among the first three joint angle.

$$\tan\theta_1 = \frac{W_y}{W_x} \Rightarrow \theta_1 = \mathrm{atan2}\left(\frac{W_y}{W_x}\right)$$

Figure 7: Solving first three joints angles part 2.

Last, let's solve the last three joint angles theta4, theta5 and theta6. First, we need to use theta1, theta2 and theta3 that we just calculated to compute R0_3. Because we know that R0_EE = Rot_EE, both side multiple by inv(R0_3), we can get R3_6 = inv(R0_3) * Rot_EE. We already have the form of R3_6 in Figure 2, then we can compute last three joint angle by doing,

```
# Convert the rotation matrix to Euler angles using tf
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]);
```

# Project Implementation

The code below is the implementation of handling inverse kinematics problem.

```
def handle_calculate_IK(req):
    rospy.loginfo("Received %s eef-poses from the plan" % len(req.poses))
    if len(req.poses) < 1:
        print "No valid poses received"
        return -1
    else:
        ### Your FK code here
        # Create symbols
        q1, q2, q3, q4, q5, q6, q7 = symbols('q1:8');
        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');
        # Create Modified DH parameters
        s = {
            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
        }
        # 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;
        # Create individual transformation matrices
        T0_1 = TF_Matrix(alpha0, a0, d1, q1).subs(s);
        T1_2 = TF_Matrix(alpha1, a1, d2, q2).subs(s);
        T2_3 = TF_Matrix(alpha2, a2, d3, q3).subs(s);
        # T3_4 = TF_Matrix(alpha3, a3, d4, q4).subs(s);
        # T4_5 = TF_Matrix(alpha4, a4, d5, q5).subs(s);
        # T5_6 = TF_Matrix(alpha5, a5, d6, q6).subs(s);
        # T6_EE = TF_Matrix(alpha6, a6, d7, q7).subs(s);
```

```
        # T3_6 = T3_4 * T4_5 * T5_6;
        # R3_6 = T3_6[0:3, 0:3];
        # print(R3_6);

        # T0_EE = T0_1 * T1_2 * T2_3 * T3_4 * T4_5 * T5_6 * T6_EE;


        R = symbols('R');
        Y = symbols('Y');
        P = symbols('P');

        ### Build Rotation Matrix extrinsic

        RotRoll_x  = Matrix([[1,      0,      0],
                             [0, cos(R),-sin(R)],
                             [0, sin(R), cos(R)]]);
        RotPitch_y = Matrix([[cos(P),  0, sin(P)],
                             [0,       1,      0],
                             [-sin(P), 0, cos(P)]]);
        RotYaw_z   = Matrix([[cos(Y), -sin(Y),  0],
                             [sin(Y),  cos(Y),  0],
                             [0,            0,  1]]);
        Rot_EE = RotYaw_z * RotPitch_y * RotRoll_x;
        Rot_Error = RotYaw_z.subs(Y, radians(180)) * RotPitch_y.subs(P, 
radians(-90));
        Rot_EE = Rot_EE * Rot_Error;
        wc_to_ee = 0.303;
        ###
        R0_3 = T0_1[0:3, 0:3] * T1_2[0:3, 0:3] * T2_3[0:3, 0:3]
        # Initialize service response
        joint_trajectory_list = []
        for x in xrange(0, len(req.poses)):
            # 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])

            ### Your IK code here

            euler = {R: roll, P: pitch, Y: yaw};
            Rrpy = Rot_EE.subs(euler);
            nx = Rrpy[0,2];
            ny = Rrpy[1,2];
            nz = Rrpy[2,2];

            ### Calculate wrist center
            wx = px - wc_to_ee * nx;
            wy = py - wc_to_ee * ny;
            wz = pz - wc_to_ee * nz;
            # Calculate joint angles using Geometric IK method
            theta1 = atan2(wy, wx);
            # side_a = sqrt(s[a3]**2 + s[d4]**2);
            side_a = 1.501
            side_c = 1.25
            # side_c = s[a2];
            # wcx = wz - s[d1];
            wcx = wz - 0.75;
            # wcy = sqrt(wx**2 + wy**2) - s[a1];
            wcy = sqrt(wx**2 + wy**2) - 0.35;

            side_b = sqrt(wcy**2 + wcx**2);

            angle_a = acos((side_b**2 + side_c**2 -
side_a**2)/(2*side_b*side_c));
            angle_b = acos((side_a**2 + side_c**2 -
side_b**2)/(2*side_a*side_c));
            angle_c = acos((side_a**2 + side_b**2 -
side_c**2)/(2*side_a*side_b));
            theta2 = pi/2 - angle_a - atan2(wcx, wcy);
            theta3 = pi/2 - (angle_b + 0.036);
            ### Clip the joint angle for physical constrain
            theta1 = np.clip(theta1, -3.23, 3.23);
            theta2 = np.clip(theta2, -0.79, 1.48);
            theta3 = np.clip(theta3, -3.67, 1.13);
```

```
            R0_3 = R0_3.evalf(subs={q1: theta1, q2: theta2, q3: theta3});
            R3_6 = R0_3.inv("LU") * Rrpy;

            # Convert the rotation matrix to Euler angles using tf
            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]);
            # print("theta4, theta5, theta6", theta4, theta5, theta6);
            ### Clip the joint angle for physical constrain
            theta4 = np.clip(theta4, -6.11, 6.11);
            theta5 = np.clip(theta5, -2.18, 2.18);
            theta6 = np.clip(theta6, -6.11, 6.11);

            # Populate response for the IK request
            joint_trajectory_point.positions = [theta1, theta2, theta3,
theta4, theta5, theta6]
            joint_trajectory_list.append(joint_trajectory_point)

        rospy.loginfo("length of Joint Trajectory List: %s" %
len(joint_trajectory_list))
        return CalculateIKResponse(joint_trajectory_list)
```

The images below show the pick and drop processes.

Figure 8: Reaching to the target position

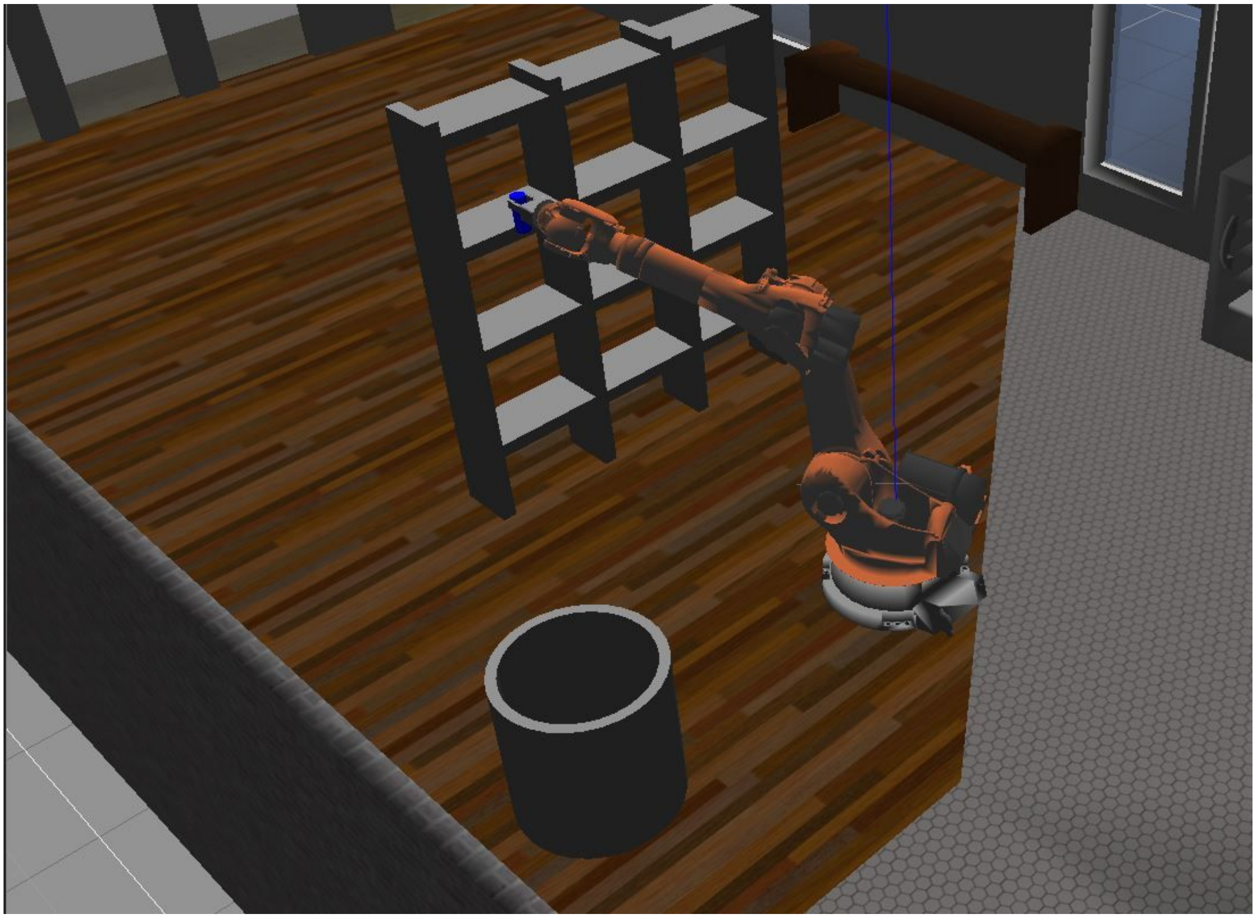


Figure 9: Retrieve the target

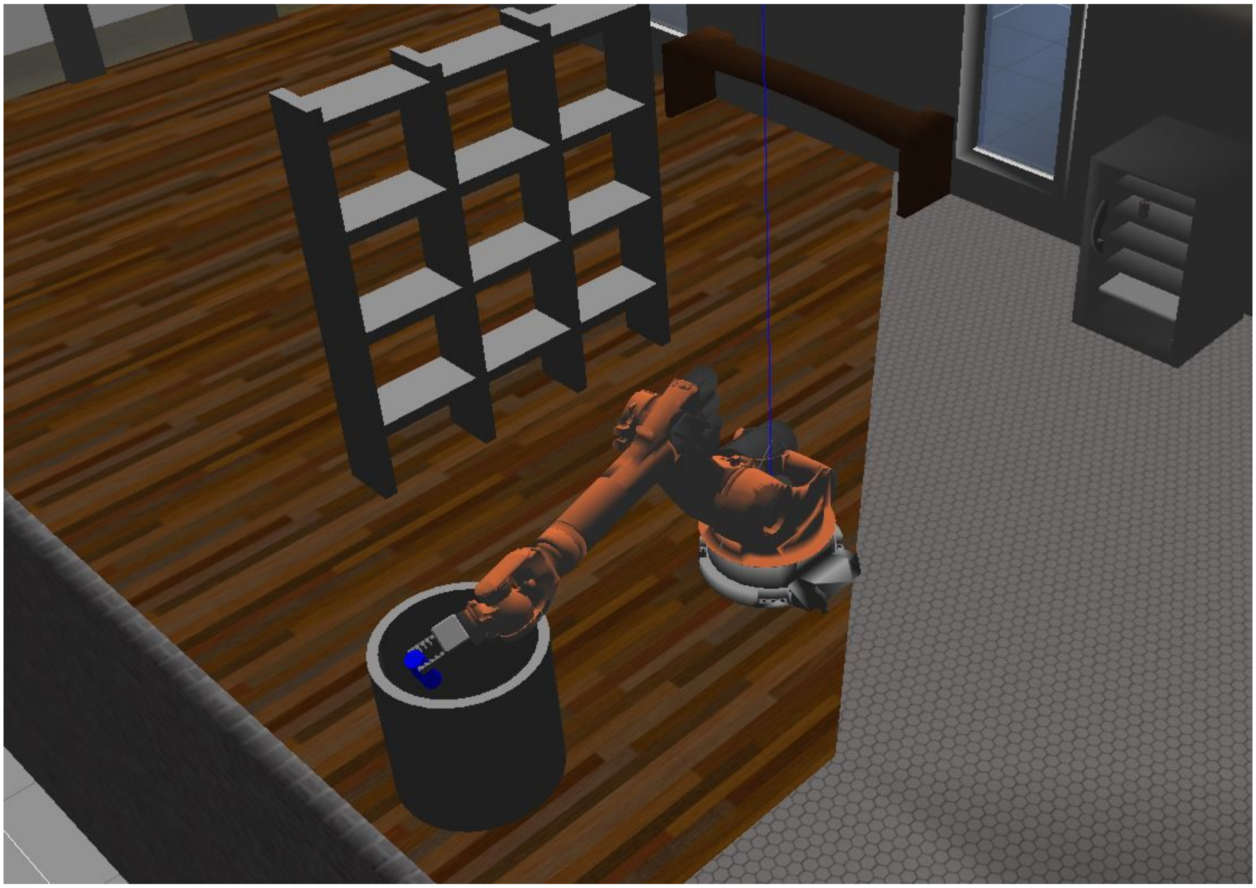


Figure 10: Drop the target