Project 2 Robotic Arm: Pick & Place

First step is to sketch the joints, links, link length, and offsets as shown in Fig. 1.

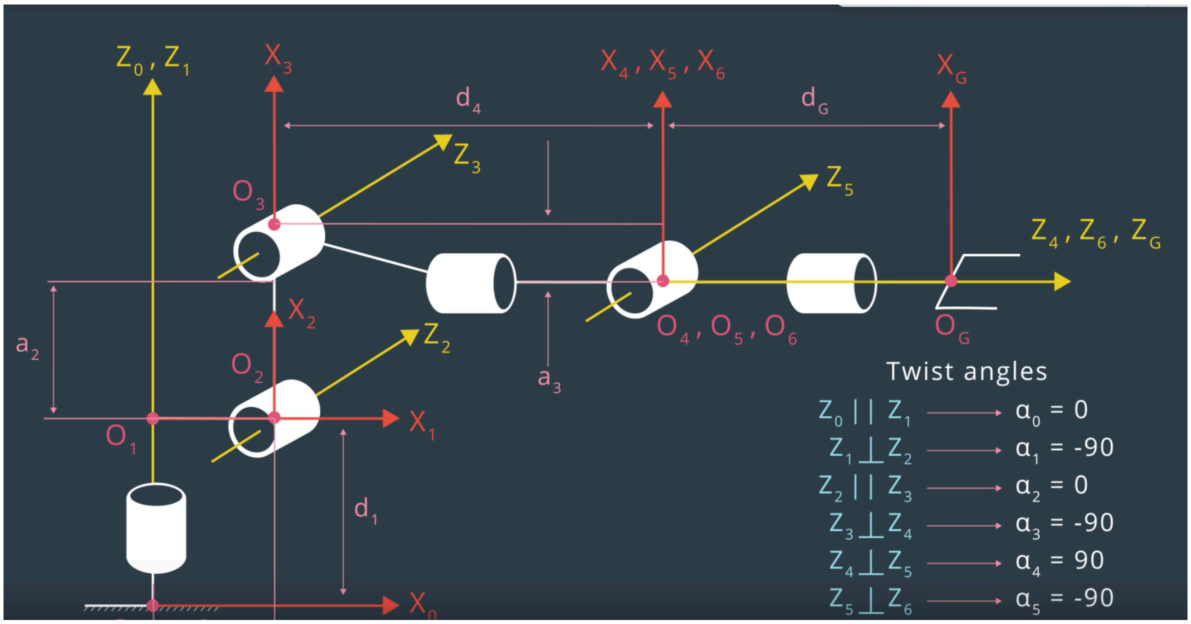


Fig. 1 Kuka KR210

α: arm twist angle
a: arm link length
d: arm link offset
θ: arm join angle

Second step is to fill out DH table as shown in table 1.

	α	a	d	θ
1	0	0	0.75	q1
2	-π/2	0.35	0	q2 - π/2
3	0	1.25	0	q3
4	-π/2	0.0536	1.5	q4
5	π/2	0	0	q5
6	-π/2	0	0	q6
7 (gripper link)	0	0	0.303	0

Table 1 DH Parameters

Then we need to perform calculations. We have a homogenous transformation matrix taking rotation again x-axis with α angle , translation on x-axis with a distance, rotation against z-axis with θ angle, and translation on z-axis with d distance between joint i-1 and i.

$$ {}^{i-1}_{\ \ i}T = R_X(\alpha_{i-1})\, D_X(a_{i-1})\, R_Z(\theta_i)\, D_Z(d_i) $$

I created a function get_DH_matrix like below by taking parameters of alpha, a, d, and q.

```
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 ]])
```

For example, transformation matrix between join 0 and 1 (T0_1) is get_DH_matrix(q1, alpha0, a0, d1).subs(s). Similarly, we can have T1_2, T2_3, T3_4, T4_5, T5_6, and T6_G. By multiplying them together from T0_1 to T6_G, we have transformation matrix from the base line to grasper. To compensate the difference between URDF and DH table, we need to rotate on z-axis for π and y-axis for -π/2 degree.

Inverse Kinematics

We can simplify it to two steps, inverse position and inverse orientation.

Inverse Position

We can obtain position by using the complete transformation matrix based on the end-effector pose.

$$ \begin{bmatrix} l_x & m_x & n_x & p_x \\ l_y & m_y & n_y & p_y \\ l_z & m_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

L, m, and n are orthonormal vectors corresponding to the end-effector orientation along X, Y, Z axes of the local coordinate frame.
Since n is the vector along the z-axis of the gripper_link, we can have the following:

$w_x = p_x - (d_6 + d_7)\ n_x$
$w_y = p_y - (d_6 + d_7)\ n_y$
$w_z = p_z - (d_6 + d_7)\ n_z$

Where,

p_x, p_y, p_z = end-effector position
w_x, w_y, w_z = wrist center position
d_6, d_7 = from DH table
“n” can be calculated from rotation matrix with correction rotation matrix.

inverse orientation

```
Rrpy = Rot(Z, yaw) * Rot(Y, pitch) * Rot(X, roll) * R_corr
```

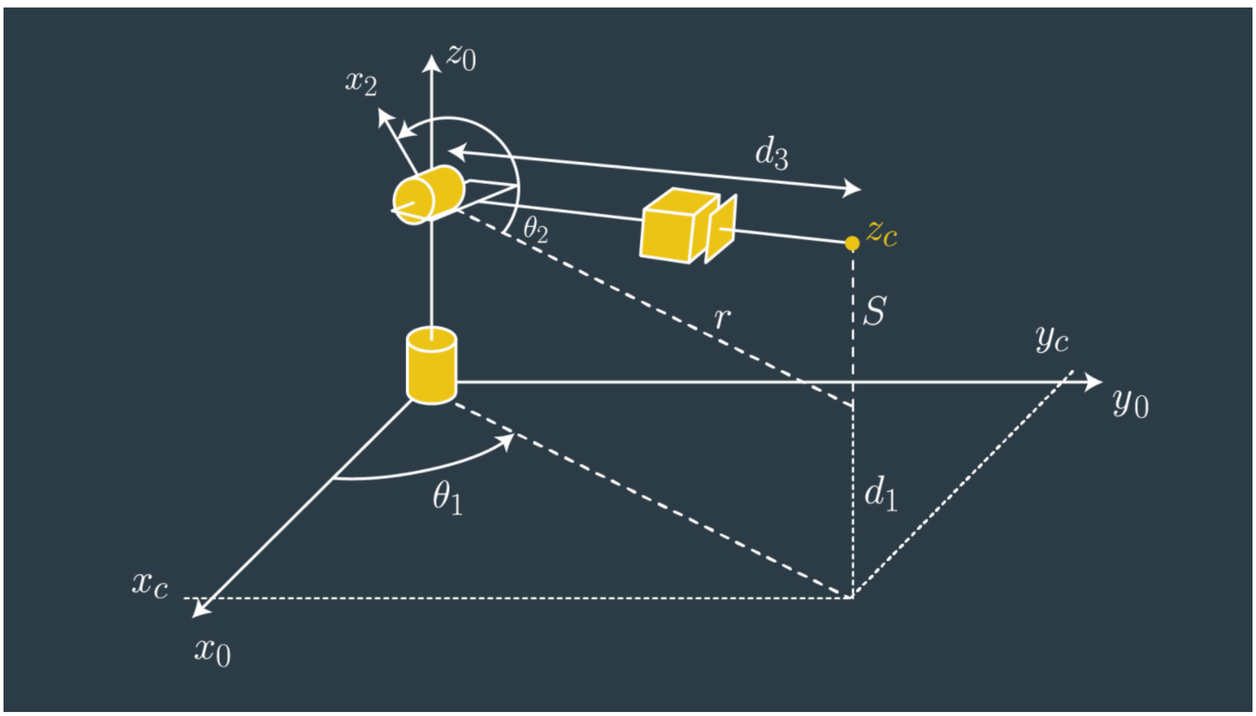


Fig 2. An RRP manipulator for three joints

$\theta_1 = \mathrm{atan2}(y_c, x_c)$

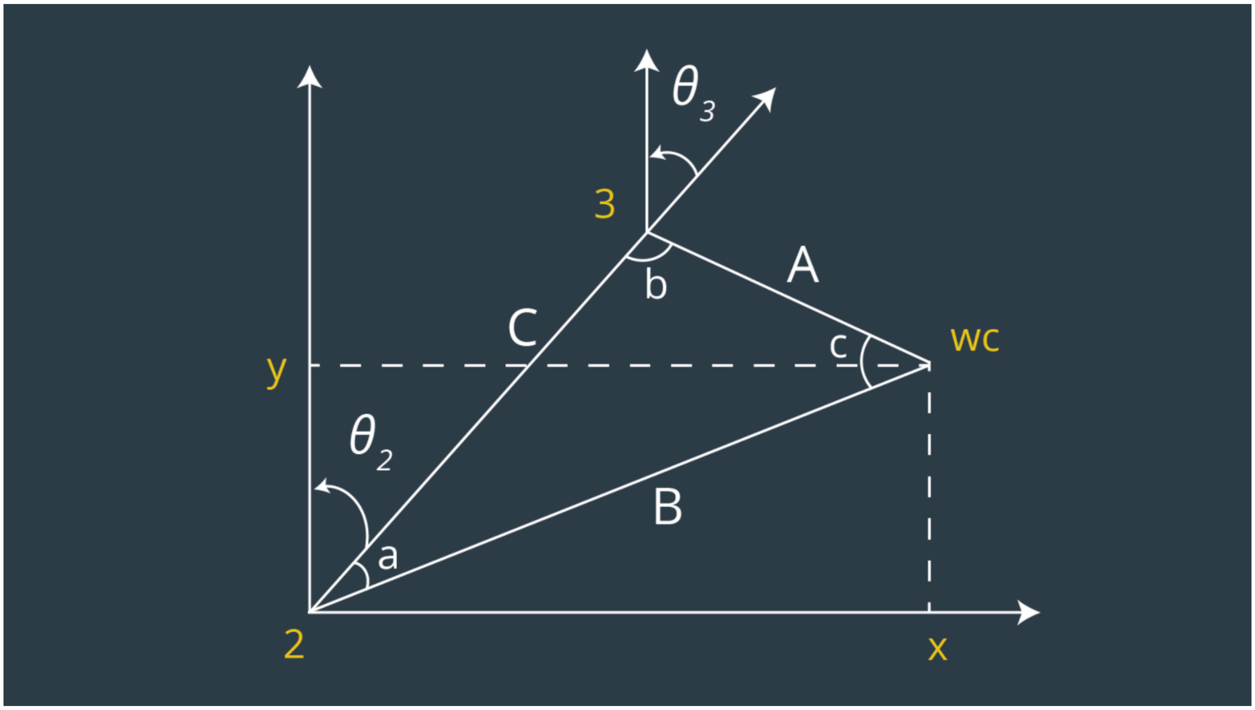


Fig 3 Illustration for θ_2, θ_3 calculation

$\theta_2 = \pi/2$ - a - atan2(wz - d1, radius)

where,

radius = sqrt(y_c^2, x_c^2) - a1

d1 = 0.75

a1 = 0.35

$\theta_3 = \pi/2$ - (b + 0.036)

where,

b = acos ((A^2 + C^2 - B^2) / (2*A*C))

Angle b is obtained from the cosine law.

0.036 accounts for sag in link4 of -0.054m

$$ {}^{3}_{6}R = \left({}^{0}_{3}R\right)^{-1} {}^{0}_{6}R = \left({}^{0}_{3}R\right)^{T} {}^{0}_{6}R $$

Find a set of Euler angles corresponding to the rotation matrix

As an orthogonal matrix, transpose matrix is equal to inverse matrix.

I use transpose matrix to save some computation.

```
R3_6 = R0_3.T * ROT_EE
```

θ_4 = atan2(R3_6[2,2], -R3_6[0,2])
θ_5 = atan2(sqrt(R3_6[0,2]^2 + R3_6[2,2]^2), R3_6[1,2])
θ_6 = atan2(-R3_6[1,1], R3_6[1,0])

As there might be more than one solution available, we need to check sin value of θ_5 to decide θ_4 and θ_6.

if sin(θ_5) < 0:
  θ_4 = atan2(-R3_6[2,2], R3_6[0,2])
  θ_6 = atan2(R3_6[1,1], -R3_6[1,0])
else:
  θ_4 = atan2(R3_6[2,2], -R3_6[0,2])
  θ_6 = atan2(-R3_6[1,1], R3_6[1,0])

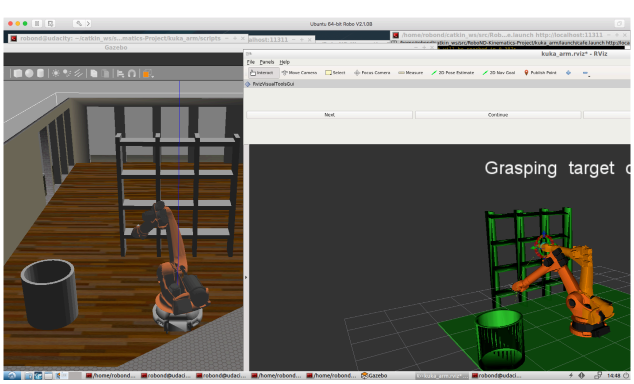
Optimization

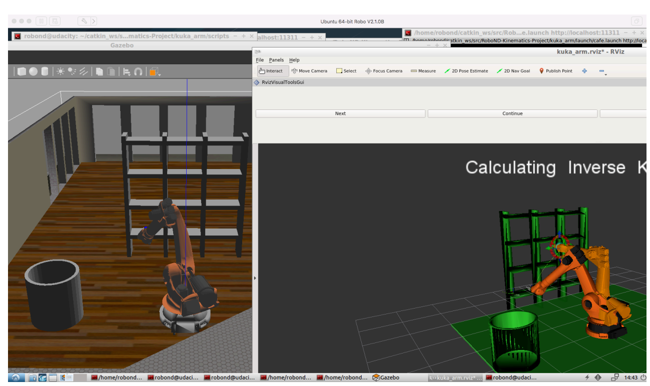
- Reuse values as much as possible instead of re-calculation
- Use transpose matrix instead of inverse matrix when they are equa
- I implement a class to encapsulate and reuse formula and values

Class CalcObject is defined to perform calculation for θ_1, θ_2 , θ_3 , θ_4, θ_5, θ_6, and WC.
First, constants are defined as in the class level.
side_a = 1.501
side_c = 1.25
side_a2 = side_a * side_a
side_c2 = side_c * side_c
side_2ac = side_a * side_c
half_pi = np.pi / 2
In the initializer of CalcObject, I calculate two instance variables, mROT_EE and mT_Total (total transformation matrix from baseline to gripper)
Function get_angles takes input of roll, pitch, yaw and EE to get θ_1, θ_2 , θ_3 , θ_4, θ_5, θ_6, and WC.

With the optimization, it might save more than 60% of computation time.

Screenshots





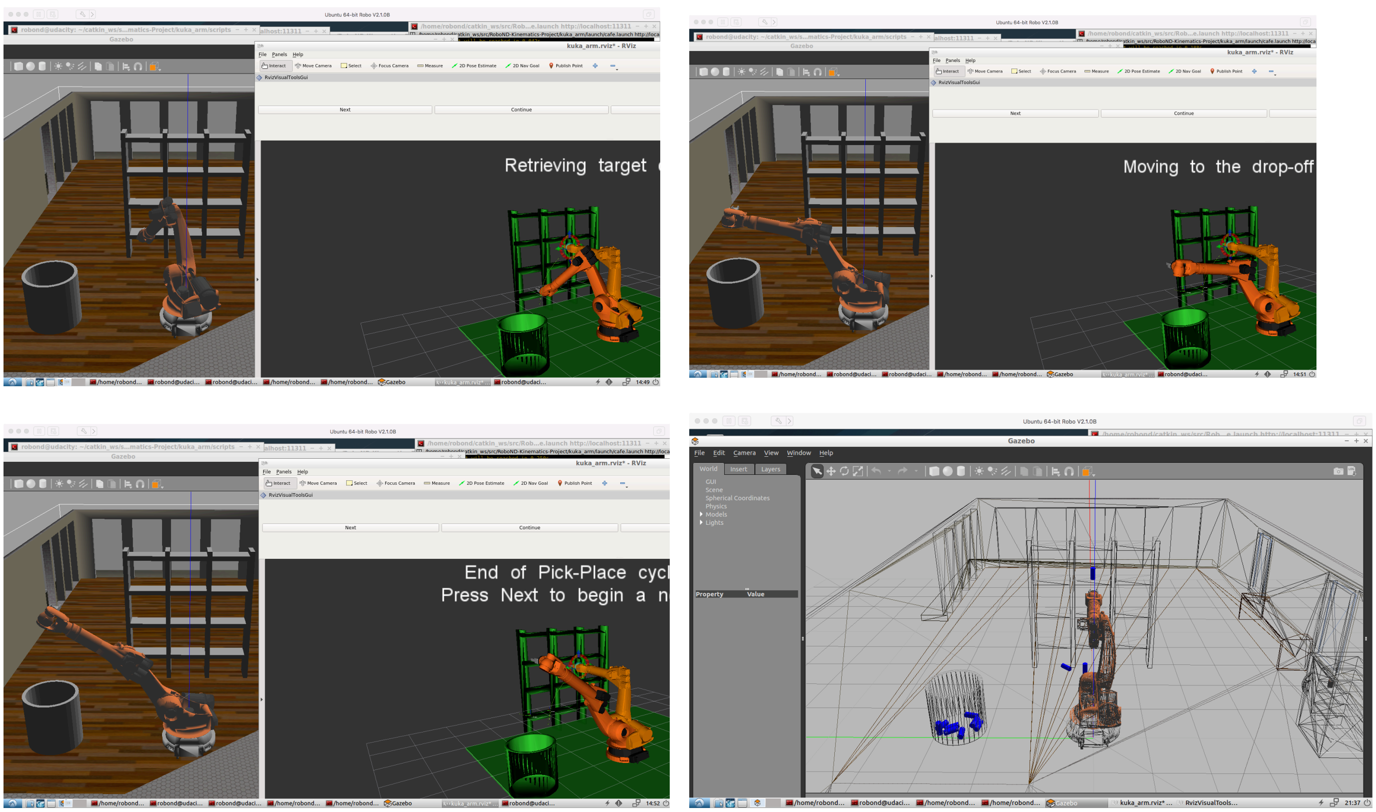


Fig 4 Screenshots of Robot Arm Movements, Completed 8/10 cylinders

Conclusion
It's a good project. Fig. 4 shows screenshots. I used the default random target in the target_description.launch. Fig 4.6 shows 8 of 10 cylinders were in the bin. Two cylinders were dropped during movement when it turned. Trajectory generated by MoveIt sometimes is not optimal. It's an area to improve.

I spent lots of time in environment. First, my VMWare Fusion 10.1 on Mac kept lock up keyboard and mouse, which needed to be reboot to get it back.
Native Ubuntu might be a better development machine for ROS, especially with Gazebo and rViz.