## Project: Kinematics Pick & Place
### Writeup Template: You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

**Steps to complete the project: **

1. Set up your ROS Workspace.
2. Download or clone the [project repository](https://github.com/udacity/RoboND-Kinematics-Project) 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](https://classroom.udacity.com/nanodegrees/nd209/parts/7b2fd2d7-e181-401e-977a-6158c77bf816/modules/8855de3f-2897-46c3-a805-628b5ecf045b/lessons/91d017b1-4493-4522-ad52-04a74a01094c/concepts/ae64bb91-e8c4-44c9-adbe-798e8f688193).
5. Perform Kinematic Analysis for the robot following the [project rubric](https://review.udacity.com/#!/rubrics/972/view).
6. Fill in the `IK_server.py` with your Inverse Kinematics code.

[//]: # (Image References)

[image1]: ./misc_images/misc1.png
[image2]: ./misc_images/misc3.png
[image3]: ./misc_images/misc2.png

## [Rubric](https://review.udacity.com/#!/rubrics/972/view) Points
### Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

### Writeup / README

#### 1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

You're reading it!

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.

Forward Kinematic Demo:

After setting up the environment, do the following to run forward kinematics demo:

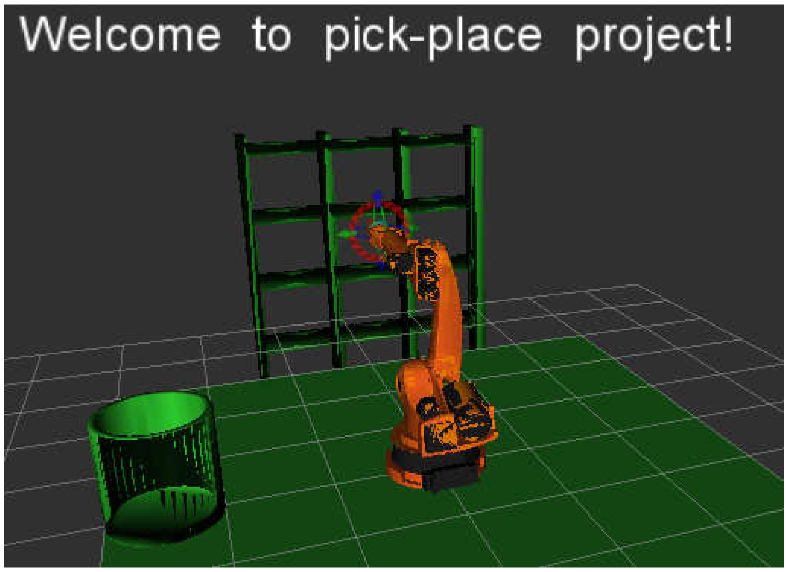
1. First make sure the simulator is set to **demo mode** by checking that the demo flag is set to `true` in `inverse_kinematics.launch` file under `~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/launch/`

2. Now project can be opened by typing following in a fresh terminal:

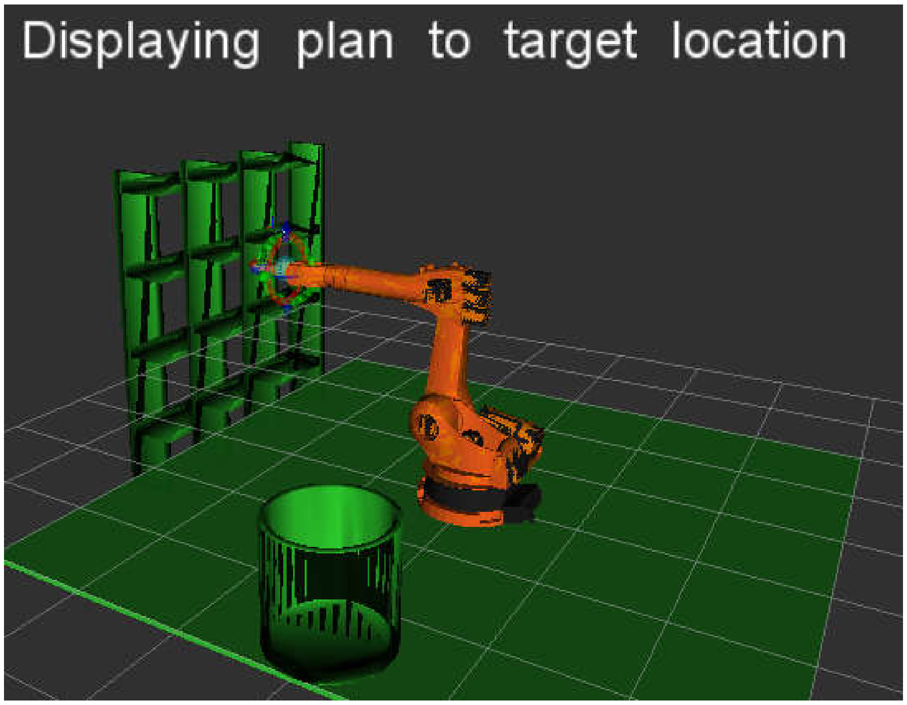
```
$ cd ~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/scripts
$ ./safe_spawner.sh
```

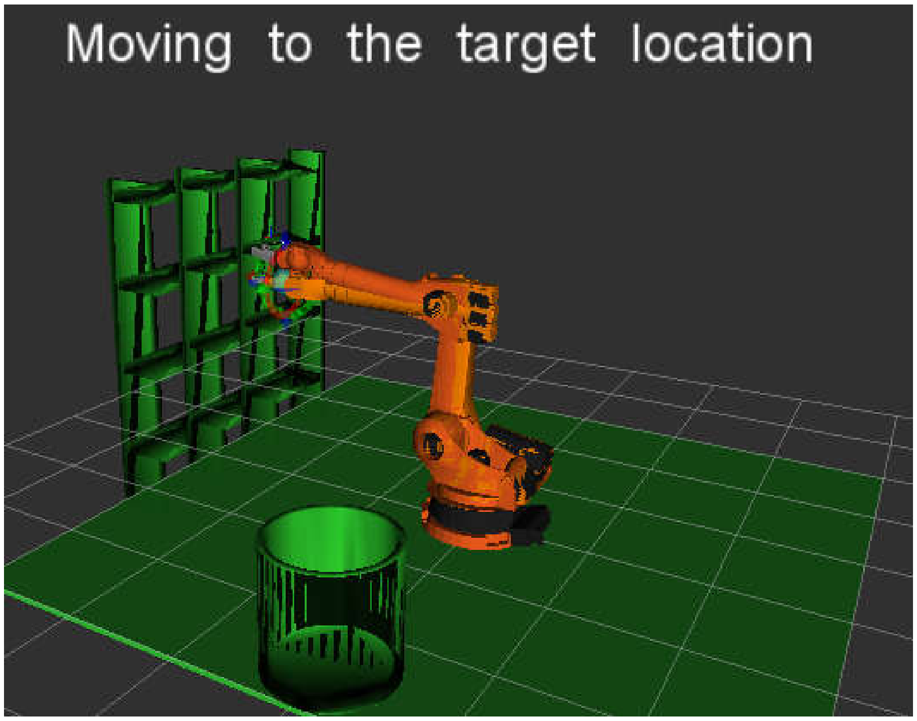
3. The demo goes in following order:

1. First the robot is in idle state, clicking `Next` will create a plan for the robot from idle position to target location and display the plan.

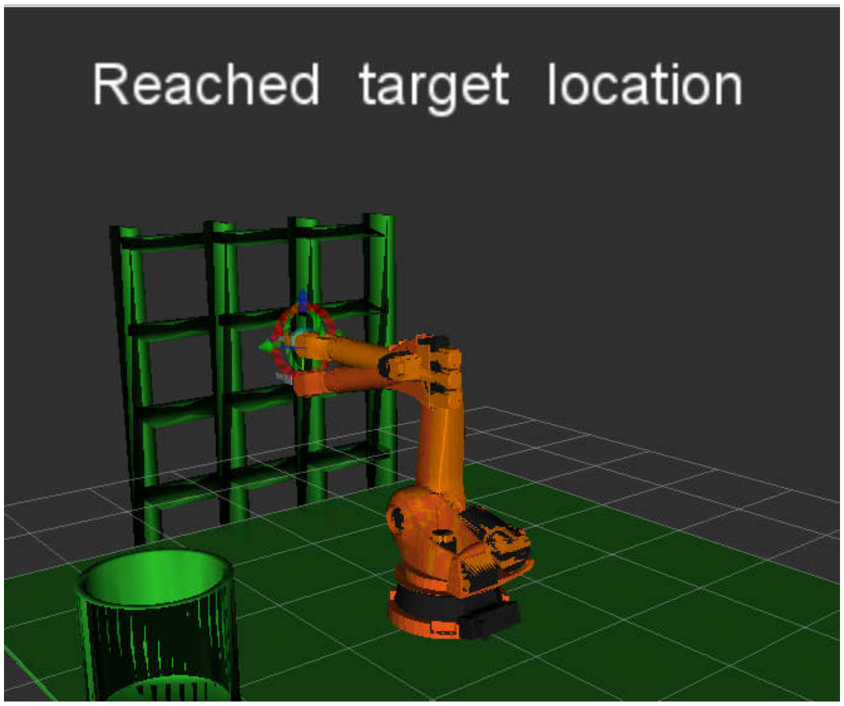


2. Clicking on `Next` again, will begin the robot motion towards the target_location. **Note that in test mode, your code is responsible to perform this motion.**

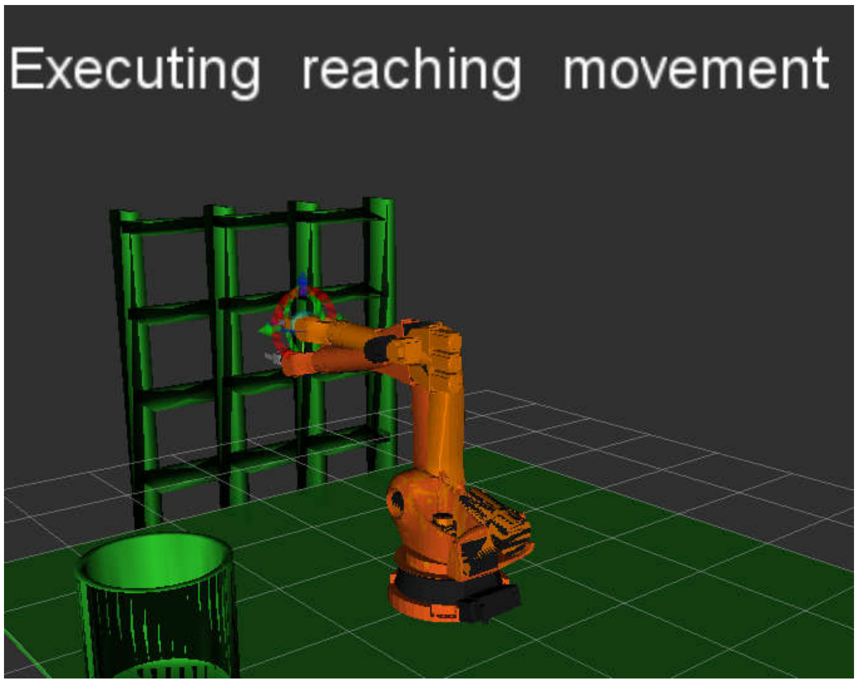


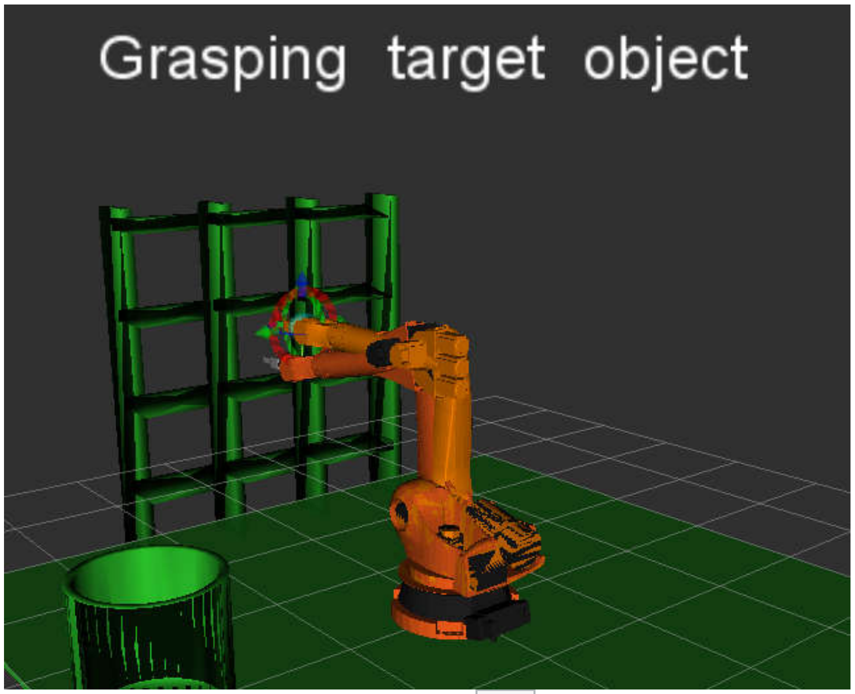


3. Once the robot reaches the target_location, hitting the Next button will perform a reaching movement.

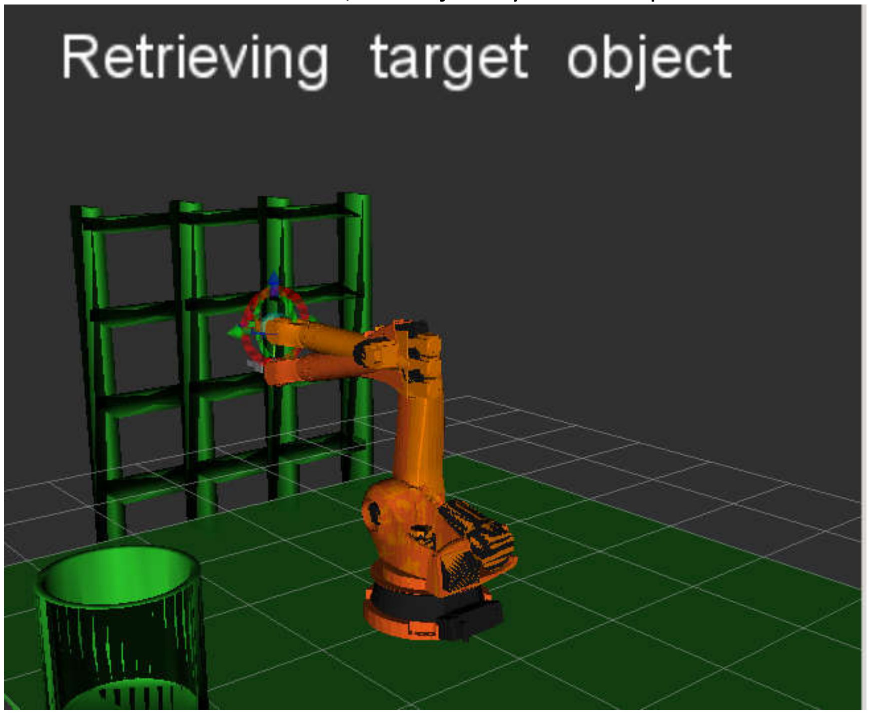


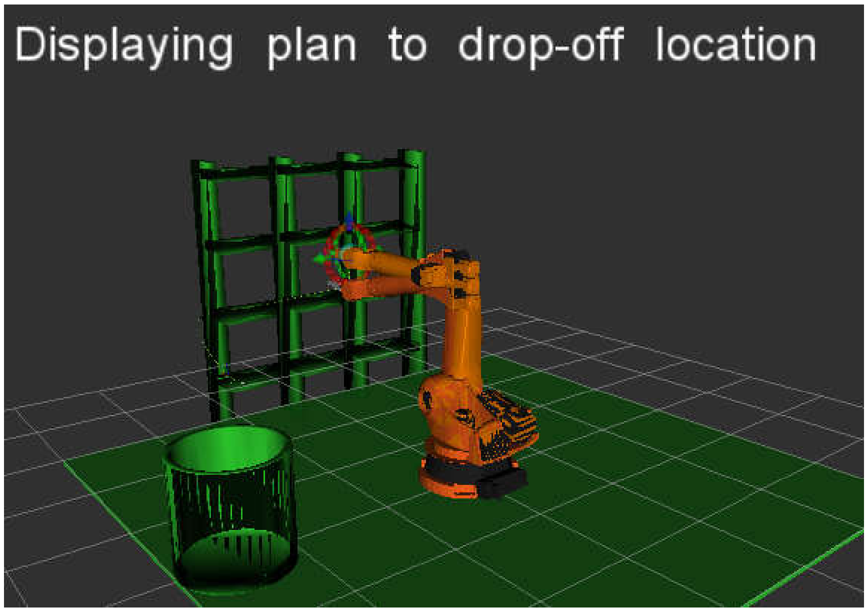
4. Next the gripper will close to grasp the target object.



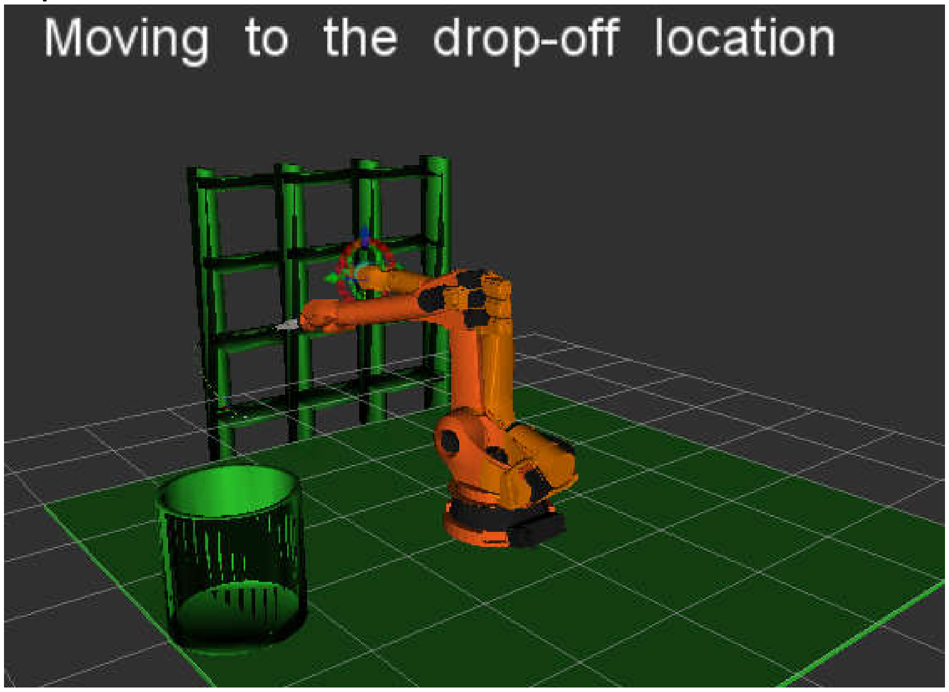


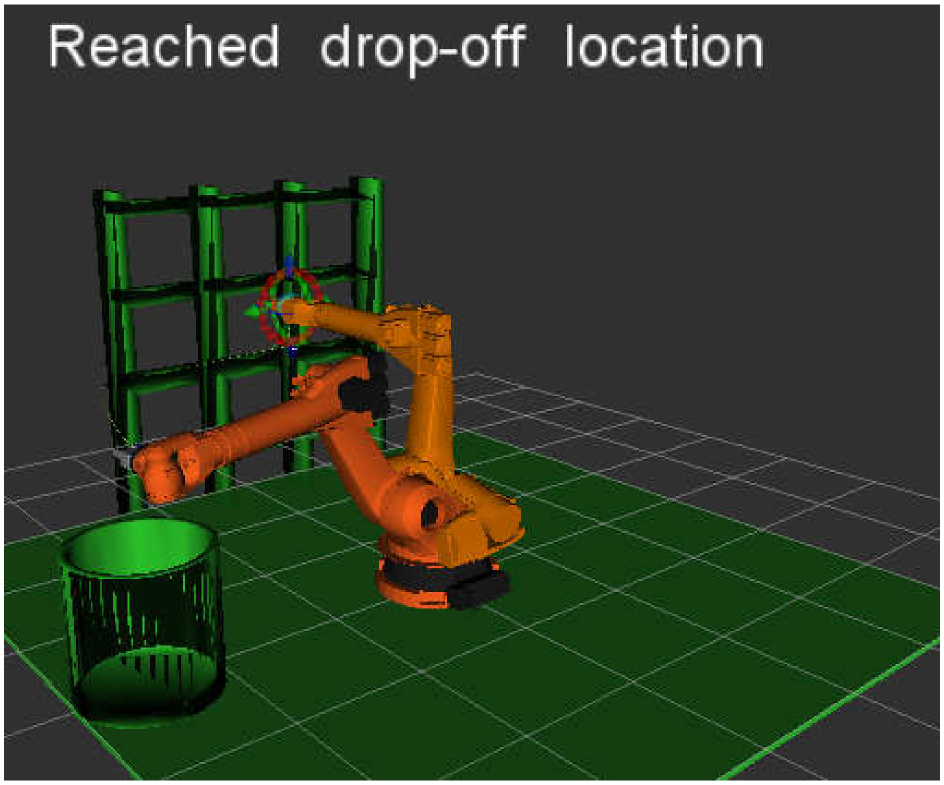
5. Once the object is grasped, we can hit Next to retract the arm and then plan for a motion towards the drop off location.
   In a similar fashion as before, the trajectory to the drop off location is displayed.



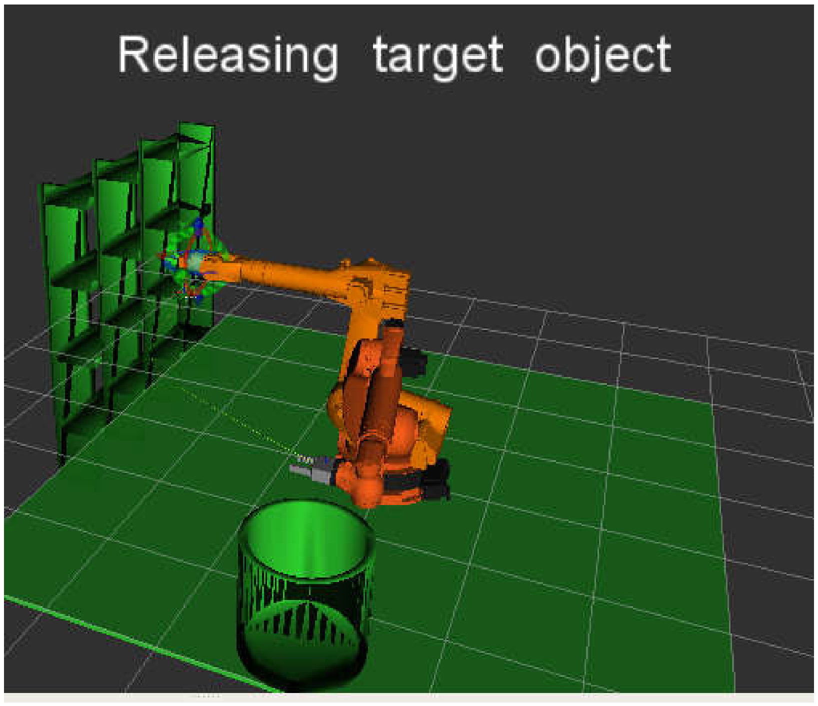


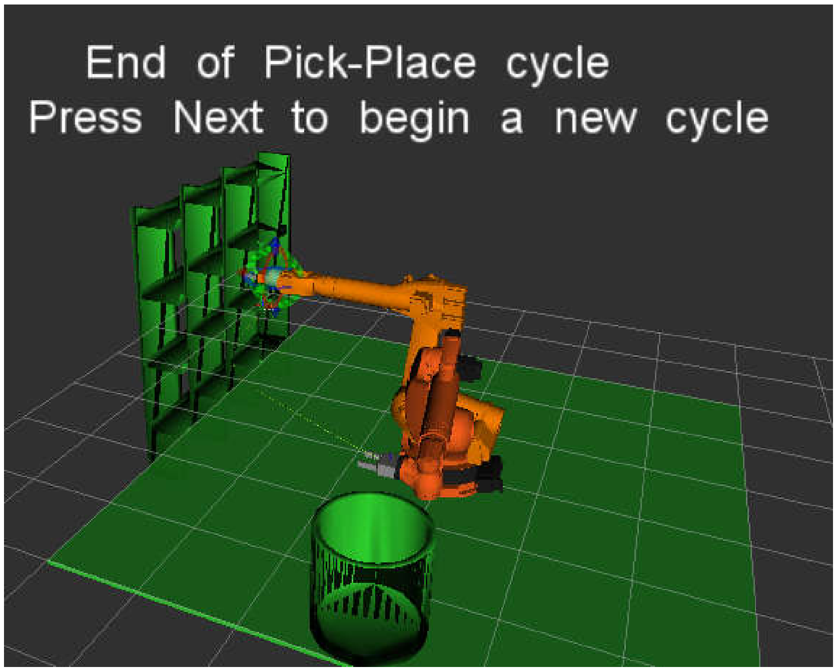
6. Next robot starts motion towards the drop off location, **again note that in test mode, your code will be responsible for this motion**.





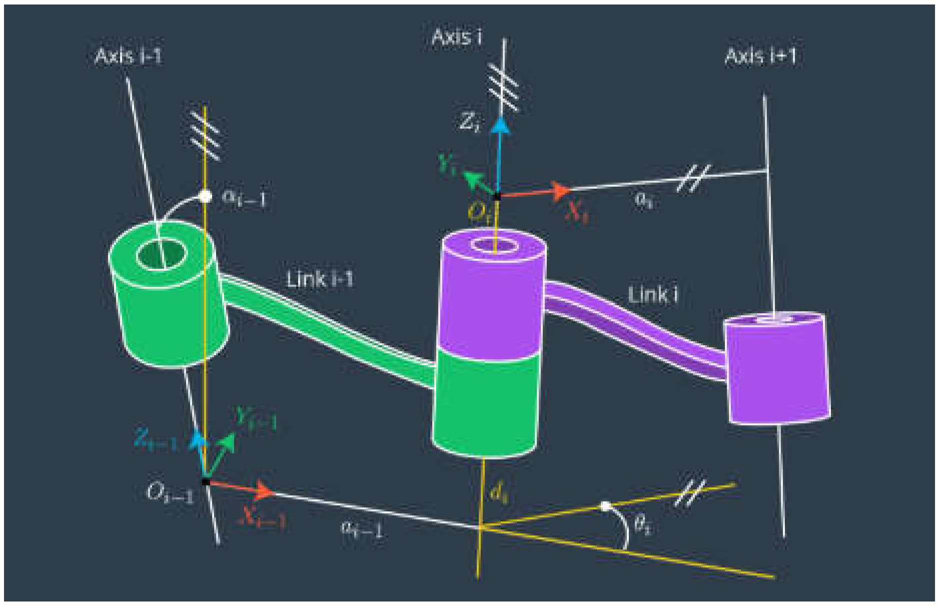
7. Once the robot reaches the drop off location, we can finally open the gripper and drop the target into the bin. This signifies the completion of a single pick and place operation cycle.





The kr210.urdf.xacro file and DH parameter Table:

We can extract the DH parameters for each joint by following the DH parameters algorithm as mentioned below:

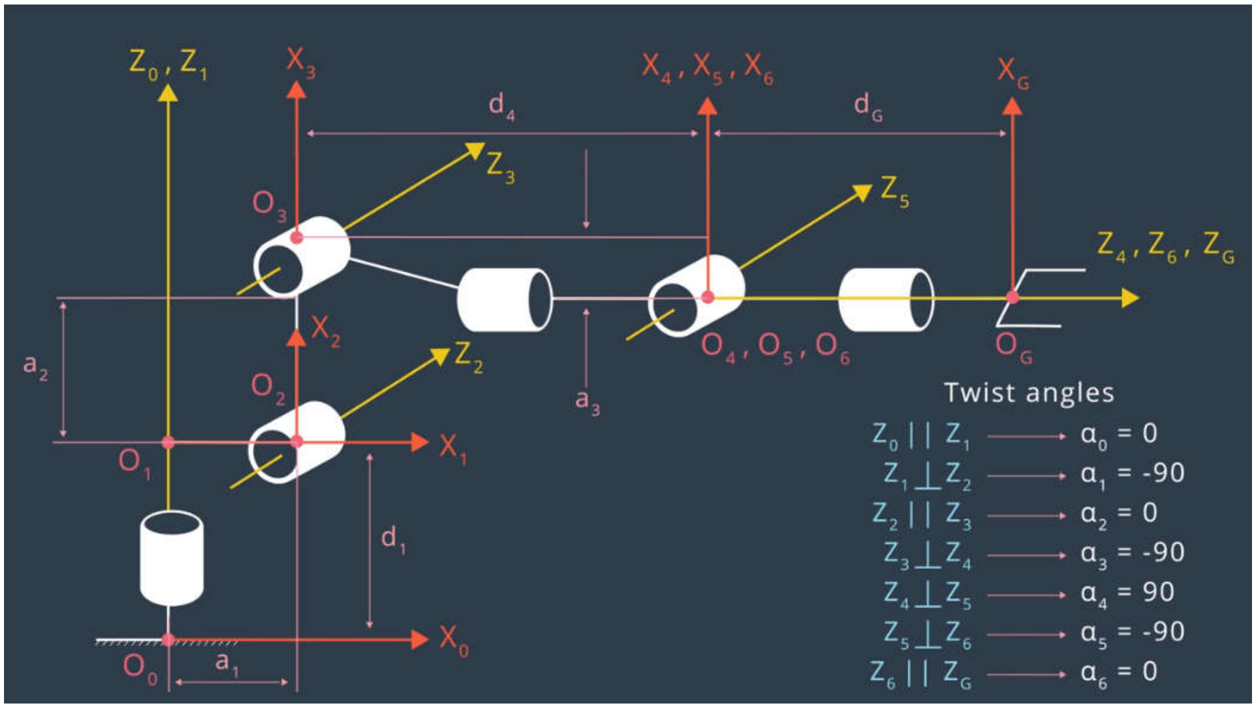


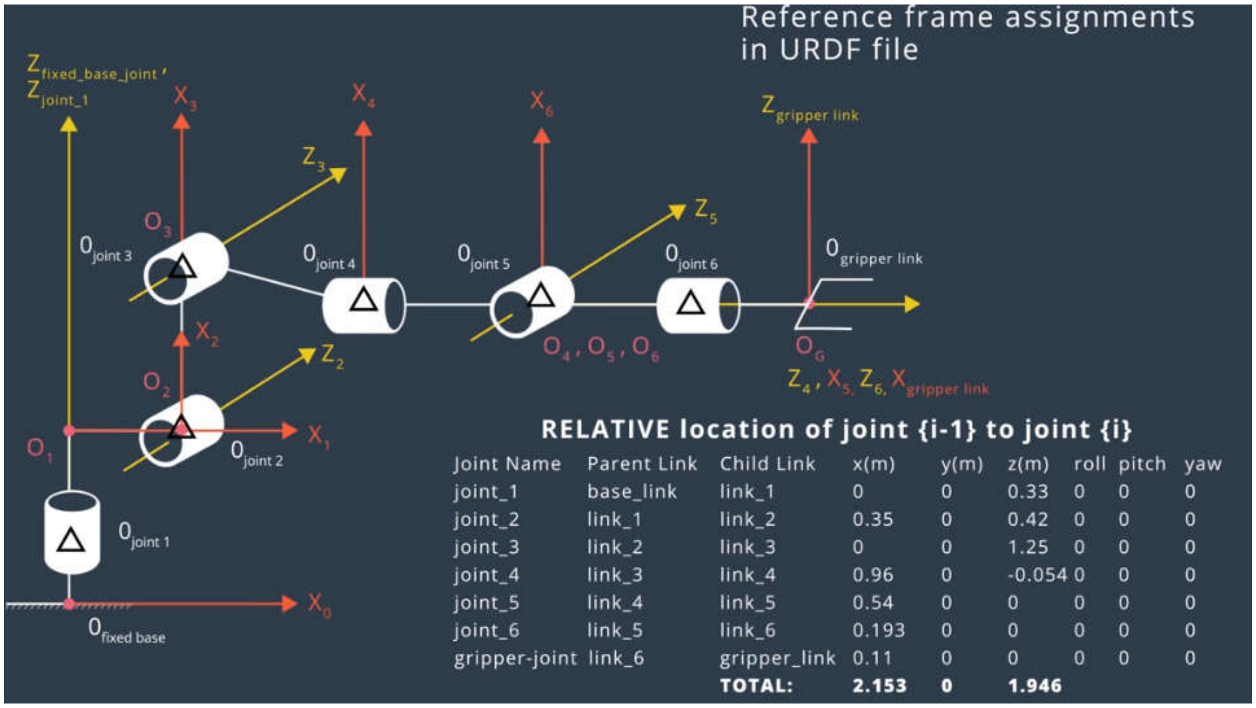
1. Label all joints from {1, 2, ... , *n*}.
2. Label all links from {0, 1, ..., n} starting with the fixed base link as 0.
3. Draw lines through all joints, defining the joint axes.
4. Assign the Z-axis of each frame to point along its joint axis.
5. Identify the common normal between each frame $\hat{Z}_{i-1}$ and frame $\hat{Z}_i$.
6. The endpoints of "intermediate links" (i.e., not the base link or the end effector) are associated with two joint axes, {i} and {i+1}. For *i* from 1 to *n-1*, assign the $\hat{X}_i$ to be ...

   - For skew axes, along the normal between $\hat{Z}_i$ and $\hat{Z}_{i+1}$ and pointing from {*i*} to {*i+1*}.
   - For intersecting axes, normal to the plane containing $\hat{Z}_i$ and $\hat{Z}_{i+1}$.
   - For parallel or coincident axes, the assignment is arbitrary; look for ways to make other DH parameters equal to zero.

7. For the base link, always choose frame {0} to be coincident with frame {1} when the first joint variable (θ_1 or d_1) is equal to zero. This will guarantee that α_0 = a_0 = 0, and, if joint 1 is a revolute, d_1 = 0. If joint 1 is prismatic, then θ_1 = 0.
8. For the end effector frame, if joint *n* is revolute, choose X_n to be in the direction of X_{n-1} when θ_n = 0 and the origin of frame {*n*} such that d_n = 0.

We can obtain diagrams showing each DH parameter by following above algorithm:



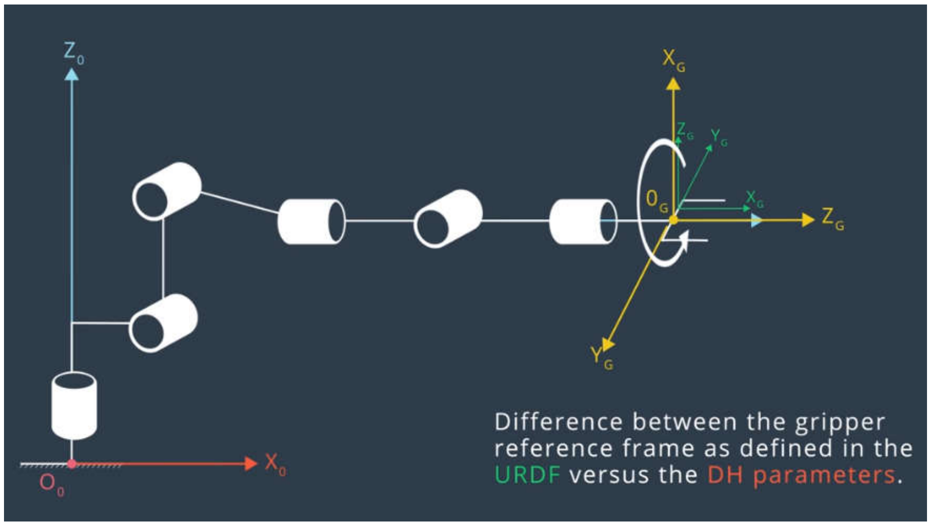


RELATIVE location of joint {i-1} to joint {i}

Joint Name	Parent Link	Child Link	x(m)	y(m)	z(m)	roll	pitch	yaw
joint_1	base_link	link_1	0	0	0.33	0	0	0
joint_2	link_1	link_2	0.35	0	0.42	0	0	0
joint_3	link_2	link_3	0	0	1.25	0	0	0
joint_4	link_3	link_4	0.96	0	-0.054	0	0	0
joint_5	link_4	link_5	0.54	0	0	0	0	0
joint_6	link_5	link_6	0.193	0	0	0	0	0
gripper-joint	link_6	gripper_link	0.11	0	0	0	0	0
		TOTAL:	**2.153**	**0**	**1.946**			

i	α_{i-1}	a_{i-1}	d_i	θ_i
$T^0{}_1$ (i=1)	0	0	0.75	q1
$T^1{}_2$ (i=2)	-pi/2	0.35	0	q2-pi/2
$T^2{}_3$ (i=3)	0	1.25	0	q3
$T^3{}_4$ (i=4)	-pi/2	-0.054	1.5	q4
$T^4{}_5$ (i=5)	Pi/2	0	0	q5
$T^5{}_6$ (i=6)	-pi/2	0	0	q6
$T^6{}_{EE}$ (i=7/EE)	0	0	0.303	0

Difference between Gripper frame (DH convention) and the URDF frame:


Z_0
X_G
Z_G
Y_G
O_G
X_G
Z_G
Y_G
O_0
X_0
Difference between the gripper reference frame as defined in the URDF versus the DH parameters.

In order to take into account the difference between the gripper reference frame as defined in the URDF versus the DH parameters, We have to do intrinsic transformation: first a rotation of 180 degree about Z-axis and then a rotation of -90 degree about Y-axis. There will be a correction matrix corresponding to this:

Rcorr = Rz * Ry;

Where

```
R_y = [ cos(-pi/2),   0,  sin(-pi/2)   ],

      [     0,          1,     0          ],

      [-sin(-pi/2),   0,    cos(-pi/2) ]



R_z = [ cos(pi),          -sin(pi),       0],

      [ sin(pi),            cos(pi),      0],

       [ 0,                      0,          1]
```

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 homogeneous transform from frame ***i-1*** to frame ***i*** is constructed as a sequence of four basic transformations, two rotations and two translations as follows:

$$ {}^{i-1}_{i}T = R_X(\alpha_{i-1})\, D_X(a_{i-1})\, R_Z(\theta_i)\, D_Z(d_i) \tag{3} $$

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

Using above DH table and above transformation Matrix we can obtain the homogeneous transformation matrix about each joint:

T^{0}_{1} (i=1) = (Transformation Matrix between base link 0 and link 1)

[cos(q1), -sin(q1) , 0, 0]

[sin(q1), cos(q1), 0, 0]

[0, 0, 1, 0.75]

[0, 0, 0, 1]

T^{1}_{2} (i=2) = (Transformation Matrix between link 1 and link 2)

[cos(q2), -sin(q2), 0, 0.35]

[0, 0 , 1, 0]

[-sin(q2), -cos(q2), 0, 0]

[0, 0, 0, 1]

T^{2}_{3} **(i=3) = (Transformation Matrix between link2 and link 3)**

$$\begin{bmatrix} \cos(q3), & -\sin(q3), & 0, & 1.25 \\ \sin(q3), & \cos(q3), & 0, & 0 \\ 0, & 0, & 1, & 0 \\ 0, & 0, & 0, & 1 \end{bmatrix}$$

T^{3}_{4} **(i=4) = (Transformation Matrix between link3 and link 4)**

$$\begin{bmatrix} \cos(q4), & -\sin(q4), & 0, & -0.054 \\ 0, & 0, & 1, & 1.5 \\ -\sin(q4), & -\cos(q4), & 0, & 0 \\ 0, & 0, & 0, & 1 \end{bmatrix}$$

T^{4}_{5} **(i=5) = (Transformation Matrix between link 4 and link 5)**

$$\begin{bmatrix} \cos(q5), & -\sin(q5), & 0, & 0 \\ 0, & 0, & -1, & 0 \\ \sin(q5), & \cos(q5), & 0, & 0 \\ 0, & 0, & 0, & 1 \end{bmatrix}$$

T^5_6 **(i=6) = (Transformation Matrix between link 5 and link 6)**

$$\begin{bmatrix} \cos(q6), & -\sin(q6), & 0, & 0 \\ 0, & 0, & 1, & 0 \\ -\sin(q6), & -\cos(q6), & 0, & 0 \\ 0, & 0, & 0, & 1 \end{bmatrix}$$

T^6_{EE} **(i=7) = (Transformation Matrix between link 6 and end effector)**

$$\begin{bmatrix} \cos(q7), & -\sin(q7), & 0, & 0 \\ \sin(q7), & \cos(q7), & 0, & 0 \\ 0, & 0, & 1, & 0.303 \\ 0, & 0, & 0, & 1 \end{bmatrix}$$

T^0_{EE} **= (Total Transformation Matrix between base link 0 and end effector)**

$$= T^0_1 * T^1_2 * T^2_3 * T^3_4 * T^4_5 * T^5_6 * T^6_{EE}$$

Taking into account the correction for orientation difference between definition of gripper link in URDF versus DH convention:

$$T^0_{EE} \text{ (corrected)} = T^0_{EE} * \text{Rcorr}$$

Debugging Forward Kinematics:

1. First make sure the simulator is set to **test mode** by checking that the demo flag is set to false in inverse_kinematics.launch file under ~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/launch/

2. From a terminal window, launch the forward kinematics demo:

```
$ roslaunch kuka_arm forward_kinematics.launch
```

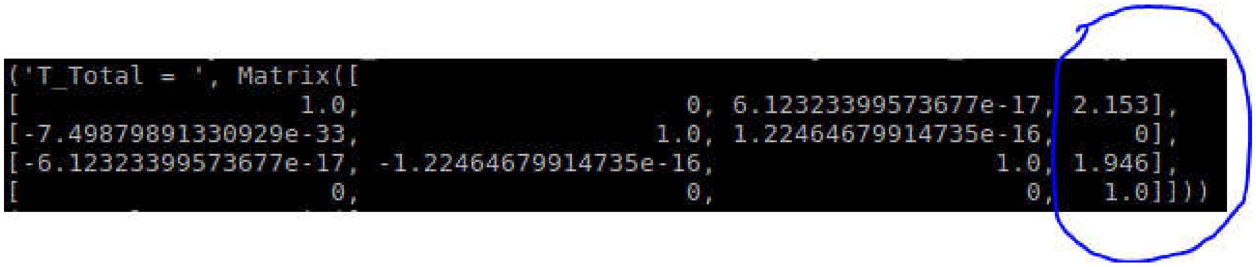
Executing the roslaunch command should spawn the robot arm in RViz along with a “joint state publisher window”.

A test code FK_kuka.py is written separately to check the python code output with the Rviz results. The python code is shared in the following folder structure:
~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/src/FK_kuka.py

Following test cases were observed:

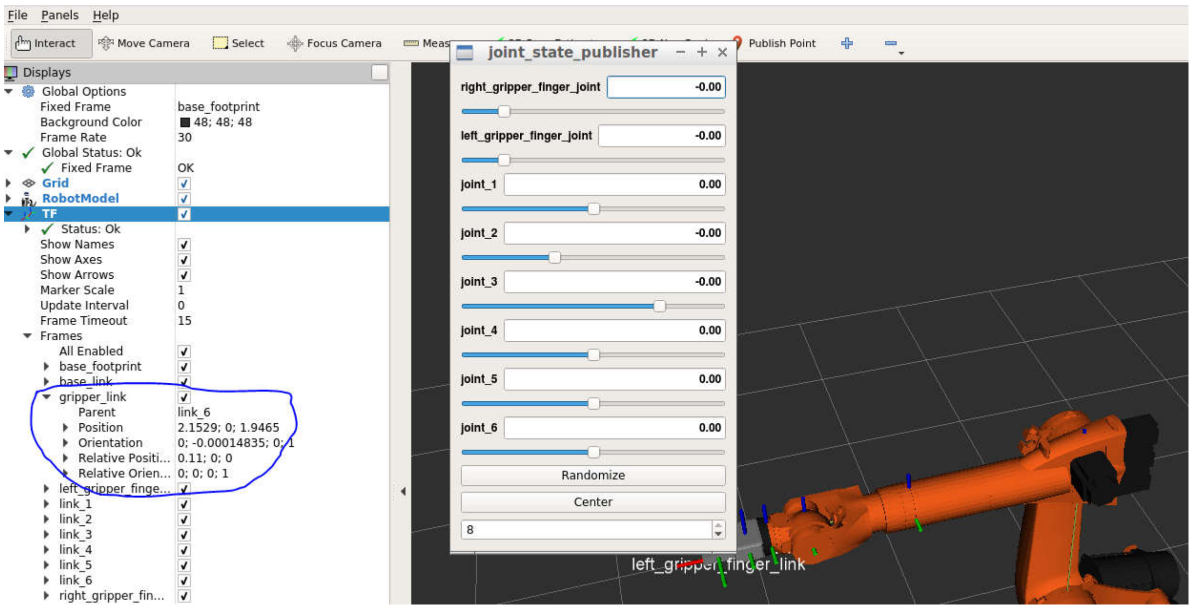
Case 1: q1=q2=q3=q4=q5=q6=0

Python code output:



```
('T_Total = ', Matrix([
[                  1.0,                     0, 6.12323399573677e-17, 2.153],
[-7.49879891330929e-33,                   1.0, 1.22464679914735e-16,     0],
[-6.12323399573677e-17, -1.22464679914735e-16,                  1.0, 1.946],
[                    0,                     0,                    0,   1.0]]))
```

Rviz Output:



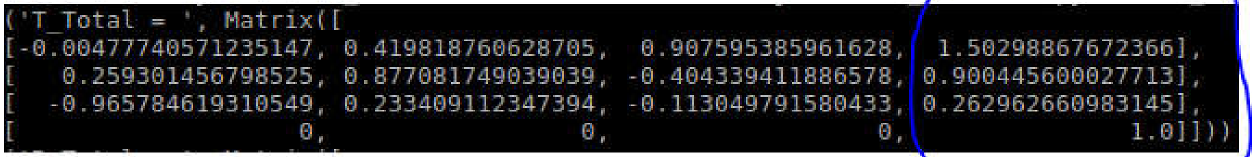
```
File  Panels  Help
Interact  Move Camera  Select  Focus Camera  Meas...  Publish Point

Displays
▾ Global Options
    Fixed Frame            base_footprint
    Background Color       ■ 48; 48; 48
    Frame Rate             30
▾ ✓ Global Status: Ok
    ✓ Fixed Frame          OK
▸ Grid                     ✓
▸ RobotModel               ✓
▾ TF                       ✓
  ▸ ✓ Status: Ok
    Show Names             ✓
    Show Axes              ✓
    Show Arrows            ✓
    Marker Scale           1
    Update Interval        0
    Frame Timeout          15
  ▾ Frames
      All Enabled          ✓
    ▸ base_footprint       ✓
    ▸ base_link            ✓
    ▾ gripper_link         ✓
        Parent             link_6
      ▸ Position           2.1529; 0; 1.9465
      ▸ Orientation        0; -0.00014835; 0; 1
      ▸ Relative Positi... 0.11; 0; 0
      ▸ Relative Orien...  0; 0; 0; 1
    ▸ left_gripper_finge... ✓
    ▸ link_1               ✓
    ▸ link_2               ✓
    ▸ link_3               ✓
    ▸ link_4               ✓
    ▸ link_5               ✓
    ▸ link_6               ✓
    ▸ right_gripper_fin... ✓

joint_state_publisher
right_gripper_finger_joint   -0.00
left_gripper_finger_joint    -0.00
joint_1                       0.00
joint_2                      -0.00
joint_3                      -0.00
joint_4                       0.00
joint_5                       0.00
joint_6                       0.00
Randomize
Center
8
```

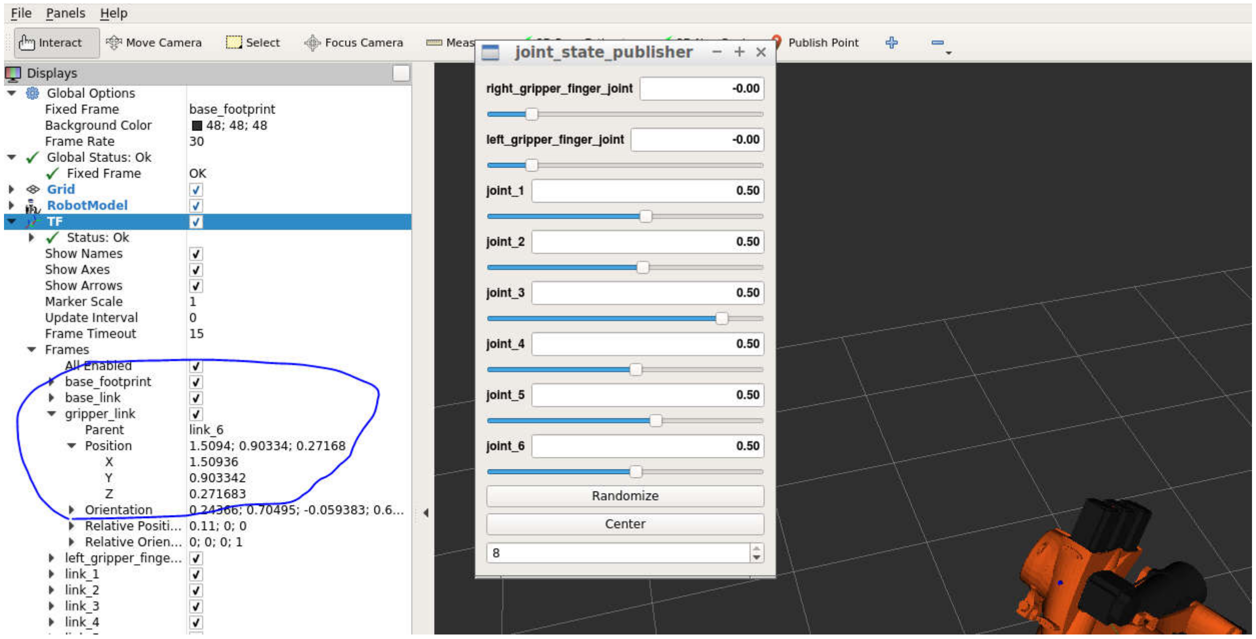
Case 2: q1= 0.5 , q2= 0.5   , q3= 0.5   , q4= 0.5   , q5= 0.5   ,q6= 0.5

Python code output:



```
('T_Total = ', Matrix([
[-0.00477740571235147, 0.419818760628705,  0.907595385961628,  1.50298867672366],
[   0.259301456798525, 0.877081749039039, -0.404339411886578, 0.900445600027713],
[  -0.965784619310549, 0.233409112347394, -0.113049791580433, 0.262962660983145],
[                   0,                 0,                  0,               1.0]]))
```

Rviz Output:



```
Global Options
  Fixed Frame            base_footprint
  Background Color       48; 48; 48
  Frame Rate             30
Global Status: Ok
  Fixed Frame            OK
Grid                     ✓
RobotModel               ✓
TF                       ✓
  Status: Ok
  Show Names             ✓
  Show Axes              ✓
  Show Arrows            ✓
  Marker Scale           1
  Update Interval        0
  Frame Timeout          15
  Frames
    All Enabled          ✓
    base_footprint       ✓
    base_link            ✓
    gripper_link         ✓
      Parent             link_6
      Position           1.5094; 0.90334; 0.27168
        X                1.50936
        Y                0.903342
        Z                0.271683
      Orientation        0.24366; 0.70495; -0.059383; 0.6...
      Relative Positi... 0.11; 0; 0
      Relative Orien...  0; 0; 0; 1
    left_gripper_finge... ✓
    link_1               ✓
    link_2               ✓
    link_3               ✓
    link_4               ✓
```

```
right_gripper_finger_joint   -0.00
left_gripper_finger_joint    -0.00
joint_1                       0.50
joint_2                       0.50
joint_3                       0.50
joint_4                       0.50
joint_5                       0.50
joint_6                       0.50
Randomize
Center
8
```

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

And here's where you can draw out and show your math for the derivation of your theta angles.

Since the last three joints in our robot are revolute and their joint axes intersect at a single point, we have a case of **spherical wrist** with joint_5 being the common intersection point and hence the **wrist center**. This allows us to kinematically decouple the IK problem into **Inverse Position** and **Inverse Orientation** problems. it is now possible to independently solve two simpler problems: first, the Cartesian coordinates of the wrist center, and then the composition of rotations to orient the end effector.

Finding Location of wrist center:

find the location of the WC relative to the base frame. Recall that the overall homogeneous transform between the base and end effector has the form:

$$
{}^{0}_{EE}T = \left[\begin{array}{ccc:c} & {}^{0}_{6}R & & {}^{0}\boldsymbol{r}_{EE/0} \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Since z4 parallel to z_6 and pointing from the WC to the EE, then this displacement is a simple translation along z_6. The magnitude of this displacement, let's call it *d*, would depend on the dimensions of the manipulator and are defined in the URDF file. Further, since r_{13}, r_{23}, and r_{33} define the Z-axis of the EE relative to the base frame, the Cartesian coordinates of the WC is,

$$
{}^{0}\boldsymbol{r}_{WC/0} = {}^{0}\boldsymbol{r}_{EE/0} - d \cdot {}^{0}_{6}R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} - d \cdot {}^{0}_{6}R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
$$

Which simplifies to,

$$\begin{bmatrix} WC_x \\ WC_y \\ WC_z \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} - 0.303 * \begin{bmatrix} r_{13} \\ r_{23} \\ r_{33} \end{bmatrix}$$

Where, WC = $\begin{bmatrix} WC_x \\ WC_y \\ WC_z \end{bmatrix}$, where WC_x, WC_y and WC_z are x, y, and z co-ordinates of the wrist center respectively.

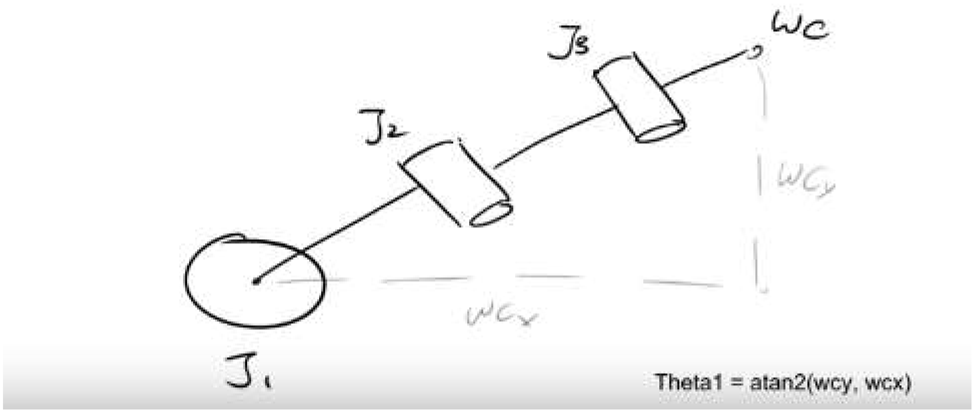
d = d_G = 0.303 = End effector displacement w.r.t to the wrist center.

end effector positions = $\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$ = obtained by deleting last row and then clipping the last column of the end effector homogeneous transformation matrix $T^0{}_{EE}$

r_{13}, r_{23}, and r_{33} define the Z-axis of the EE relative to the base frame = $\begin{bmatrix} r_{13} \\ r_{23} \\ r_{33} \end{bmatrix}$ = Which is obtained by clipping the last column of the rotation matrix $R^0{}_6$

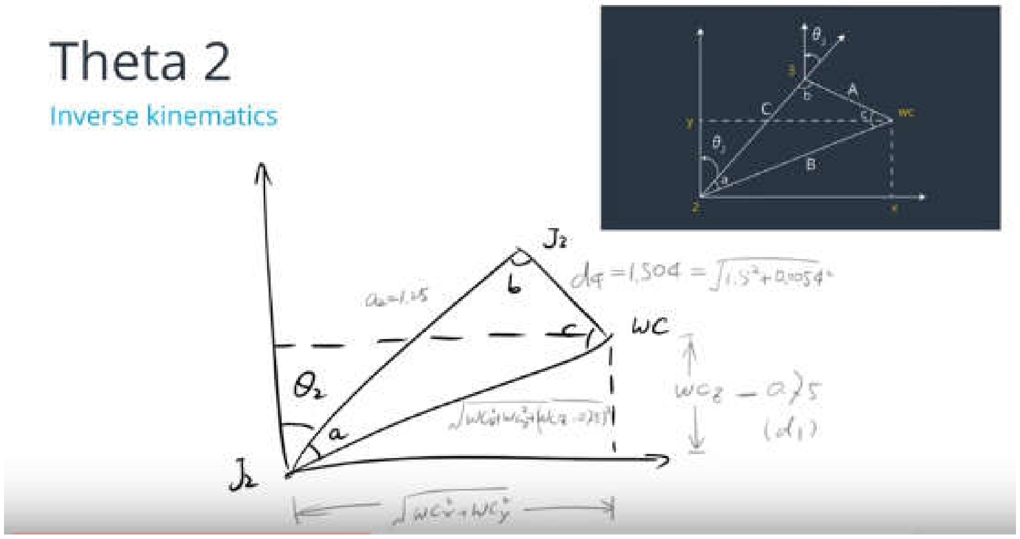
Finding find joint variables, q_1, q_2 and q_3, such that the WC has coordinates obtained from above method.

After finding the location of wrist center, q1 (theta1), q2(theta2) and q3(theta3) can be obtained using geometric properties as shown below:



$$theta1 = \tan^{-1}\left(\frac{WC_y}{WC_x}\right)$$

By looking at the triangle formed by joint2, joint 3 and the wrist center, we can get theta2 and theta3 as shown below:



Similarly, theta3 can be obtained by subtracting the angle b and the angle between joint 3 to wrist center and joint 3 to joint 4:

Length of the sides of the triangle formed by joint J2, Wrist center and joint J1 can be calculated as follow:

Length of side A = A= $\sqrt{d4^2 + a3^2} = \sqrt{(1.5)^2 + (0.054)^2} = 1.501$, where d4 and a3 are DH parameters

Length of Side C = C = a2 =1.25, where a2 is the DH parameter: refer to the DH table

Length of side B = B = $\sqrt{(\sqrt{WC_x{}^2 + WC_y{}^2} - a1)^2 + (WC_z - d1)^2}$ where a1=0.35 = x-co-ordinate of joint 2 w.r.t base frame.
d1 = 0.75 = z-co-ordinate of joint 2 w.r.t base frame

Now applying cosine in this triangle, we can get angles A, B and C as follows:

$$\cos(a) = \frac{B^2 + C^2 - A^2}{2*B*C}$$

$$\cos(b) = \frac{A^2 + C^2 - B^2}{2 * A * C}$$

$$\cos(c) = \frac{A^2 + B^2 - C^2}{2 * A * B}$$

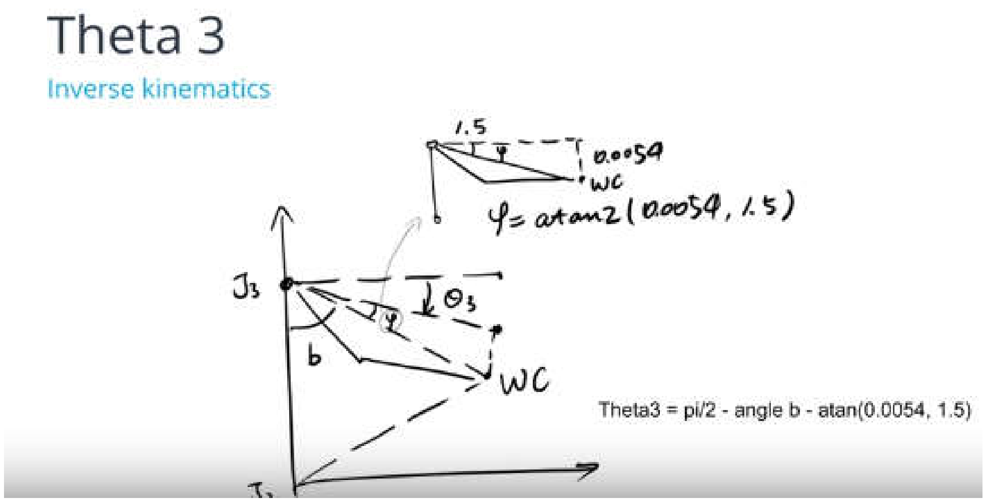
Angles a, b and c can be obtained by taking inverse cosine of above equations.

The angle that the wrist center makes with Joint 2 or with the x-axis of base frame :

$$\tan(\omega) = \frac{W_c - d1}{(\sqrt{WC_x^{\ 2} + WC_y^{\ 2}} - a1)^2}$$

Now, from the triangle theta2 and theta3 can be calculated as below:

$$theta2 = \pi / 2 - a - \omega$$


Theta 3

Inverse kinematics

Theta3 = pi/2 - angle b - atan(0.0054, 1.5)

We can calculate angle theta3 by using the above triangle:

$$theta3 = \pi / 2 - b - \phi$$

Where ϕ represents the sag between joint 3 and Wrist centre and is given by,

$$\tan \phi = \frac{0.054}{1.5} = = \frac{a4}{d3}$$

Finding out q4(theta4), q5(theta5) and q6(theta6):

Step 1: First obtain R3_6 (Rotation Matrix between Join3 to Joint 6)

This can be obtained by pre-multiplying R0_6 by inverse of R0_3 (rotation matrix between base link and joint 3). R0_3 can be obtained by forward kinematics and substituting q1,q2 and q3 by theta1,theta2 and theta3 respectively.

R3_6 = inv(R0_3) *R0_6

Step 2: Theta 4, theta5 and theta6 can be considered as Euler angle for the spherical wrist as explained in the classroom:

$$\begin{aligned} {}^{A}_{B}R_{XYZ} &= R_Z(\alpha)R_Y(\beta)R_X(\gamma) \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \end{aligned}$$

α,β,ϒ is the angle theta4,theta5 and theta6 respectively, which can be found by identifying each element of R3_6 and using the suitable trigonometric equations as shown in the below image:

$$theta4 = \tan^{-1}\left(\frac{r_{33}}{-r_{13}}\right)$$

$$theta5 = \tan^{-1}\left(\frac{\sqrt{r_{13}^{2} + r_{33}^{2}}}{r_{23}}\right)$$

$$theta6 = \tan^{-1}\left(\frac{-r_{22}}{r_{21}}\right)$$

Where r11, r12,r13,r21,r22,r23,r31,r32,r33 are corresponding elements of the Matrix R3_6

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.

Here I'll talk about the code, what techniques I used, what worked and why, where the implementation might fail and how I might improve it if I were going to pursue this project further.

Code implementation in IK_Debug.py:

FK code:

1. Get roll, pitch and yaw from Poses class
2. Get the end effector position from Poses class
3. Implementation Forward Kinematics
    a. Create symbols for DH parameters

```
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')
```

    b. Fill the DH_Table based on the urdf file entries and define the function for getting transformation matrix from DH parametrs.

```
DH_Table = {

        alpha0 : 0, a0:0, d1:0.75, q1:q1,
        alpha1 : -pi/2., a1:0.35, d2:0, q2:q2-pi/2.,
        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,
        }

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
```

c. Calculate Homogeneous transformation matrix for consecutive links based on Forward kinematics

```
# Create individual transformation matrices
  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_EE =TF_Matrix(alpha6,a6,d7,q7).subs(DH_Table)
```

d. Find out the Homogeneous transformation matrix between the END effector and the base link

```
# Composition of Homogeneous transforms
T0_2  = simplify(T0_1 * T1_2)    #base link to link 2
T0_3  = simplify(T0_2 * T2_3)    #base link to link 3
T0_4  = simplify(T0_3 * T3_4)    #base link to link 4
T0_5  = simplify(T0_4 * T4_5)    #base link to link 5
T0_6  = simplify(T0_5 * T5_6)    #base link to link 6
T0_EE = simplify(T0_6 * T6_EE)  #base link to End Effector
```

e. Find out the correction in rotational matrix to take into account the difference in convention between the DH convention and xaro file convention.

```
T_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.]])

T_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]])



T_corr = simplify(T_z * T_y)
R_corr = T_corr[0:3,0:3]
```

f. Find out the corrected Homogenous transformation matrix between the end effector and base link
   *T_total = simplify(T0_EE * T_corr)*
g. Find out the position of end effector from above transformation matrix

Inverse Kinematics code:

4. Implementation of Inverse kinematics:
   a. Get the rotation matrix from roll, pitch and yaw and multiply by correction matrix to take into account the difference in convention of DH parametrs and urdf file

```
### define function for getting rotation Matrix from euler angle

  def rotation_from_euler(r,p,y):

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

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

         R_x_r = Matrix([[1,0,0],[0,cos(r),-sin(r)],[0,sin(r),cos(r)]])

         Rot = R_z_y * R_y_p *R_x_r

         return Rot

  #Calculate rotation matrix from roll,pitch and yaw values

  RTOT_EE = rotation_from_euler(roll,pitch,yaw)


 # Compensate for rotation discrepancy between DH parameters and Gazebo

  R_corr = T_corr[0:3,0:3]

  RTOT_EE = RTOT_EE * R_corr
```

   b. Get the wrist center co-ordinates using rotation matrix obtained from roll, yaw and pitch parameters and the DH parameter for end effector offset from wrist center.

```
 #calculation of wrist centre co-rdinates


  WC_x = px-0.303*RTOT_EE[0,2]

  WC_y = py-0.303*RTOT_EE[1,2]

  WC_z = pz-0.303*RTOT_EE[2,2]
```

   c. Based on triangle created by joint 1, 2,3, and wrist center, obtain theta1, theta2 and theta3.

```
#calculation of theta1 based on the triangle shown in write up
theta1= atan2(WC_y,WC_x)
#print("theta1 = ",theta1.evalf(subs={q1: 1.5, q2: 2.5, q3: 3.0, q4: 3.14, q5: 2.14, q6: 1.14}))
#calculation of theta2 and theta3 based on the triangle mentioned in the write up
#calculation of sides of the triangle formed by joint 2,joint 3 and the wrist centre
C=1.25
A=round(sqrt(1.5**2+0.054**2),7)
B=sqrt((sqrt(WC_x**2 + WC_y**2) -0.35)**2 + (WC_z-0.75)**2)
cos_a=(B**2+C**2-A**2)/(2*B*C)
cos_b=(A**2+C**2-B**2)/(2*A*C)
angle_a = acos(cos_a)
angle_b = acos(cos_b)
theta2 = pi/2 - atan2((WC_z-0.75),(sqrt(WC_x**2+WC_y**2)-0.35)) - angle_a
theta3 = pi/2 - angle_b - atan2(0.054,1.5)
```

d. Get R0_3 i.e. rotational transformation matrix using forward kinematics and theta1, theta2, theta3 values calculated above.

```
R0_3 = T0_3[0:3,0:3]
R0_3 = R0_3.evalf(subs={q1: theta1, q2: theta2, q3: theta3})
#print(R0_3)
```

e. Get the Rotation matrix R3_6 by using linear algebra.

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

f. R3_6 is rotation matrix of a spherical joint the roll, pitch and yaw of which are theta3, theta4 and theata6 respectively.
g. Calculate theta4, thata5 and theta6 based on the trigonometric equations as mentioned in the theoretical part of the write up

```
r12 = R3_6[0,1]
r13 = R3_6[0,2]
r21 = R3_6[1,0]
r22 = R3_6[1,1]
```

```
    r23 = R3_6[1,2]

    r31 = R3_6[2,0]

    r32 = R3_6[2,1]

    r33 = R3_6[2,2]


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])
```

5. Error analysis:
    a. Find out the end effector position from theta values calculated above and the transformation matrix obtained from forward kinematics.

```
T_EE = T_total.evalf(subs={q1: theta1, q2: theta2, q3: theta3, q4: theta4, q5: theta5, q6: theta6})

print("T_EE",T_EE)

your_ee = [T_EE[0,3],T_EE[1,3],T_EE[2,3]]
```

    b. Error in end effector position is calculated as the difference between the end effector positions obtained from poses classes and the positions calculated from forward kinematics as mentioned in step a above.

```
  ee_x_e = abs(your_ee[0]-test_case[0][0][0])

  ee_y_e = abs(your_ee[1]-test_case[0][0][1])

  ee_z_e = abs(your_ee[2]-test_case[0][0][2])

  ee_offset = sqrt(ee_x_e**2 + ee_y_e**2 + ee_z_e**2)

  print ("\nEnd effector error for x position is: %04.8f" % ee_x_e)

  print ("End effector error for y position is: %04.8f" % ee_y_e)

  print ("End effector error for z position is: %04.8f" % ee_z_e)

  print ("Overall end effector offset is: %04.8f units \n" % ee_offset)
```

    c. Similarly, the error in wrist center is also calculated as the difference in wrist center position obtained from forward kinematics code and the one mentioned inside the test cases.

```
your_wc = [WC_x,WC_y,WC_z] # <--- Load your calculated WC values in this array


wc_x_e = abs(your_wc[0]-test_case[1][0])

wc_y_e = abs(your_wc[1]-test_case[1][1])

wc_z_e = abs(your_wc[2]-test_case[1][2])

wc_offset = sqrt(wc_x_e**2 + wc_y_e**2 + wc_z_e**2)

 print ("\nWrist error for x position is: %04.8f" % wc_x_e)

  print ("Wrist error for y position is: %04.8f" % wc_y_e)

  print ("Wrist error for z position is: %04.8f" % wc_z_e)

  print ("Overall wrist offset is: %04.8f units" % wc_offset)
```

d. Finally, we calculate error in theta values as the difference in theta values obtained from inverse kinematics code and the values mentioned inside each test cases.

```
t_1_e = abs(theta1-test_case[2][0])
t_2_e = abs(theta2-test_case[2][1])
t_3_e = abs(theta3-test_case[2][2])
t_4_e = abs(theta4-test_case[2][3])
t_5_e = abs(theta5-test_case[2][4])
t_6_e = abs(theta6-test_case[2][5])
print ("\nTheta 1 error is: %04.8f" % t_1_e)
print ("Theta 2 error is: %04.8f" % t_2_e)
print ("Theta 3 error is: %04.8f" % t_3_e)
print ("Theta 4 error is: %04.8f" % t_4_e)
print ("Theta 5 error is: %04.8f" % t_5_e)
print ("Theta 6 error is: %04.8f" % t_6_e)
```

All the test cases were verified using the Ik_debug code. Below is the result for each test case :

Test case 1:

```
1:[[[2.16135,-1.42635,1.55109],
        [0.708611,0.186356,-0.157931,0.661967]],
        [1.89451,-1.44302,1.69366],
        [-0.65,0.45,-0.36,0.95,0.79,0.49]]
Total run time to calculate joint angles from pose is 65.7330 seconds

Wrist error for x position is: 0.00000046
Wrist error for y position is: 0.00000032
Wrist error for z position is: 0.00000545
Overall wrist offset is: 0.00000548 units
```

```
Theta 1 error is: 0.00093770
Theta 2 error is: 0.00178634
Theta 3 error is: 0.00206504
Theta 4 error is: 2.11884713
Theta 5 error is: 0.78262546
Theta 6 error is: 0.51143787

End effector error for x position is: 0.23324990
End effector error for y position is: 0.01457186
End effector error for z position is: 0.44370056
Overall end effector offset is: 0.50148583 units
```
Test case 2:
```
2: [[[-0.56754,0.93663,3.0038],
            [0.62073, 0.48318,0.38759,0.480629]],
            [-0.638,0.64198,2.9988],
            [-0.79, -0.11, -2.33,1.94,1.14, -3.68]],


Total run time to calculate joint angles from pose is 67.4567 seconds

Wrist error for x position is: 0.00002426
Wrist error for y position is: 0.00000562
Wrist error for z position is: 0.00006521
Overall wrist offset is: 0.00006980 units

Theta 1 error is: 3.14309971
Theta 2 error is: 0.27927963
Theta 3 error is: 1.86833316
Theta 4 error is: 2.28141750
Theta 5 error is: 1.08112292
Theta 6 error is: 4.95520872

End effector error for x position is: 0.04839921
End effector error for y position is: 0.20232259
End effector error for z position is: 0.29281153
Overall end effector offset is: 0.35918728 units
```
Test case 3:
```
3:[[[-1.3863,0.02074,0.90986],
            [0.01735,-0.2179,0.9025,0.371016]],
            [-1.1669,-0.17989,0.85137],
            [-2.99,-0.12,0.94,4.06,1.29,-4.12]],


Total run time to calculate joint angles from pose is 66.2134 seconds

Wrist error for x position is: 0.00000503
Wrist error for y position is: 0.00000512
```

Wrist error for z position is: 0.00000585
Overall wrist offset is: 0.00000926 units

Theta 1 error is: 0.00136747
Theta 2 error is: 0.00329798
Theta 3 error is: 0.00339864
Theta 4 error is: 6.28213720
Theta 5 error is: 0.00287049
Theta 6 error is: 6.28227457

End effector error for x position is: 0.00000001
End effector error for y position is: 0.00000000
End effector error for z position is: 0.00000001
Overall end effector offset is: 0.00000001 units

We added two more cases by running through the forward kinematics debug steps. Wrist center is calculated with forward kinematics code and using the values of given theta1, theta2, theta3, theta4, theta5 and theta6 values obtained from joint state publisher slider bar. The code for forward kinematics debug is locate here:
~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/src$ python FK_kuka.py

Test case 4:

```
4: [[[-0.0041726,0.34072,3.5038],

   [0.85348,0.44389,0.26409, -0.0692427]],

    [-0.12229,0.23472,3.76372],

    [2.44, -0.42, -0.95,4.71,1.02, -3.04]],
```

Total run time to calculate joint angles from pose is 72.7067 seconds

Wrist error for x position is: 0.02321352

Wrist error for y position is: 0.11250136

Wrist error for z position is: 0.41513509

Overall wrist offset is: 0.43073492 units

Theta 1 error is: 0.00295024

Theta 2 error is: 0.00309978

Theta 3 error is: 0.00390123

Theta 4 error is: 5.67064024

Theta 5 error is: 1.71009514

Theta 6 error is: 5.34249501

End effector error for x position is: 0.08696512

End effector error for y position is: 0.13445040

End effector error for z position is: 0.44120251

Overall end effector offset is: 0.46936073 units

Test case 5:

```
5: [[[-0.13217, -1.5285,2.92805],
   [-0.83251,0.081294, -0.13317,0.531592]],
   [-0.215395, -1.779108,3.054832],
   [-1.85,1.03, -2.59,4.71, -1.44,1.14]]
```

```
Total run time to calculate joint angles from pose is 74.2658 seconds

Wrist error for x position is: 0.20502325
Wrist error for y position is: 0.33452043
Wrist error for z position is: 0.16777773
Overall wrist offset is: 0.42671749 units

Theta 1 error is: 0.00400354
Theta 2 error is: 1.07432180
Theta 3 error is: 1.96504170
Theta 4 error is: 3.03256068
Theta 5 error is: 2.92669600
Theta 6 error is: 4.03166825

End effector error for x position is: 0.00000000
End effector error for y position is: 0.00000001
End effector error for z position is: 0.00000001
Overall end effector offset is: 0.00000001 units
```

Implementation of Ik_server.py code:

The forward and inverse kinematics section in Ik_server.py code is written similar to what is explained in Ik_debug.py code.

Finally, I ran the simulation by following these steps:

a. Set the demo flag to 'false" inside the file inverse_kinematics.launch located inside the launch folder.
b. Change the default location form "0" to "4" in the file target_descriptions.launch located inside the launch folder.
   We need to change this location from 0 t0 9 to test each case one by one.
c. Build the catkin package by running the command "catkin_make" from catkin workspace.
d. Project can be launched by running the file safe_spawner.sh kept inside the script folder
e. Run the Ik_server by running the command "roslaunch kuka_arm Ik_server.py"
f. Press "Next" key in Rviz to go through different steps of pick and place cycles. We can also observe the respective change in positions of the robot in Gazebo file as well.
g. Validate each scenario by changing the default location in target_description.launch file located inside launch folder.

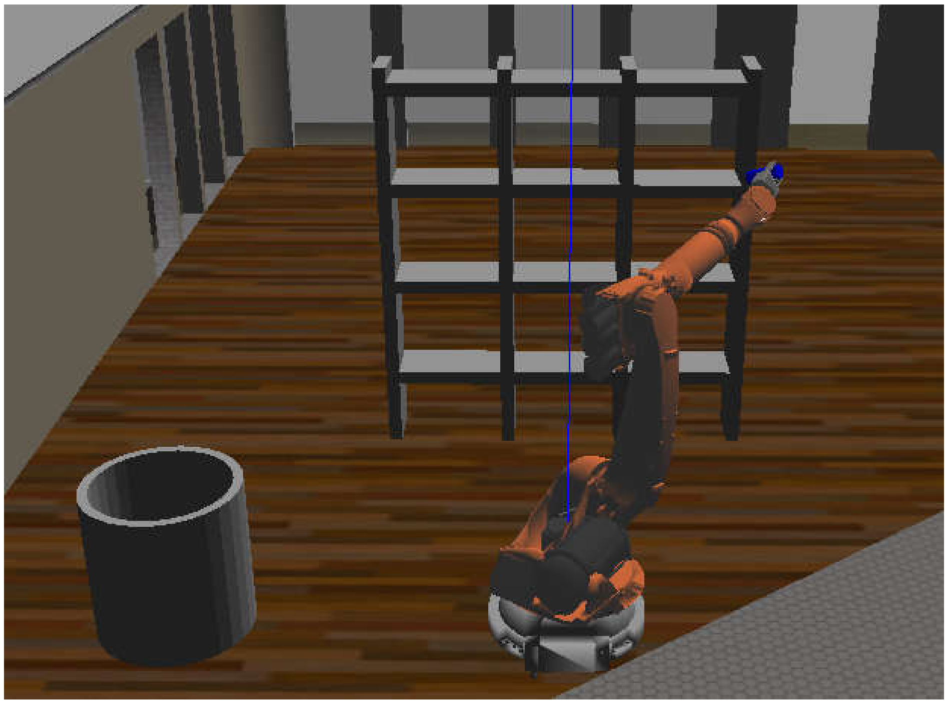
Improvements required:

1. Sometimes Gazebo hangs while running the safe_spawner file. This can be improved by trying the project in native ROS installation.
2. The RViz simulator is very slow between few steps while running the Inverse kinematics code. This also impacts the project run time. This can be improved by better implementation of functions and variables used inside the for look in IK_server.py code.
3. The error is little high in some of the test cases (see test case 5). This can be improved by using better trigonometric logic or equations for obtaining theta4, theta5 and theta6.

Some snapshots of Gazebo environment when the target location is "3"

Grasping target location:

Moving to the drop-off location:



Releasing the target: