Project: Kinematics Pick & Place

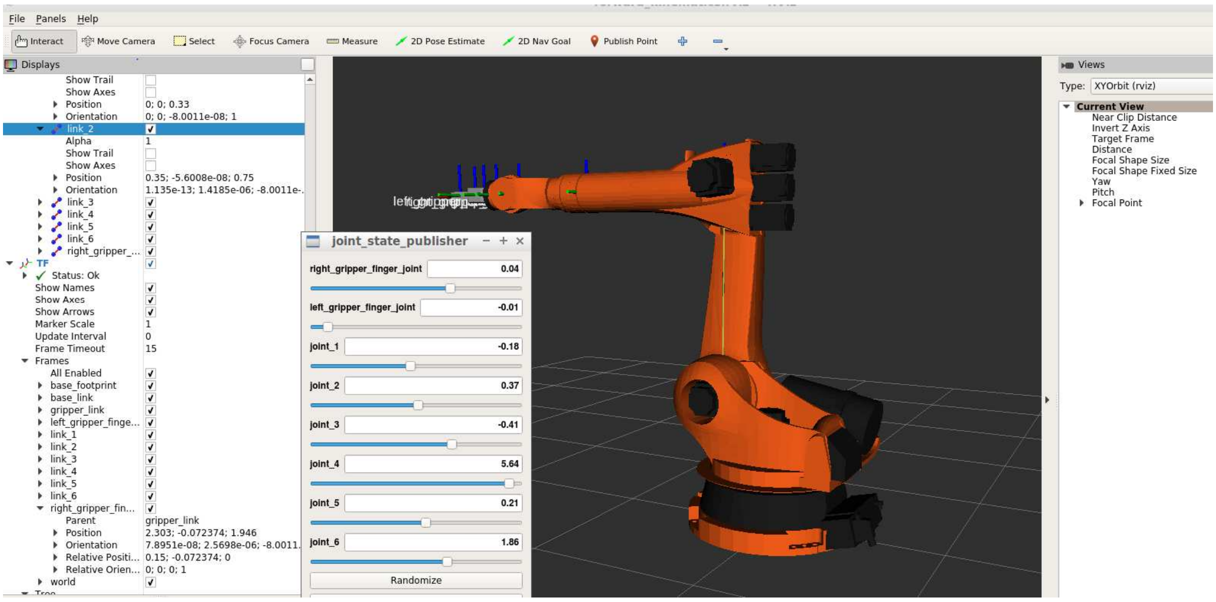
1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

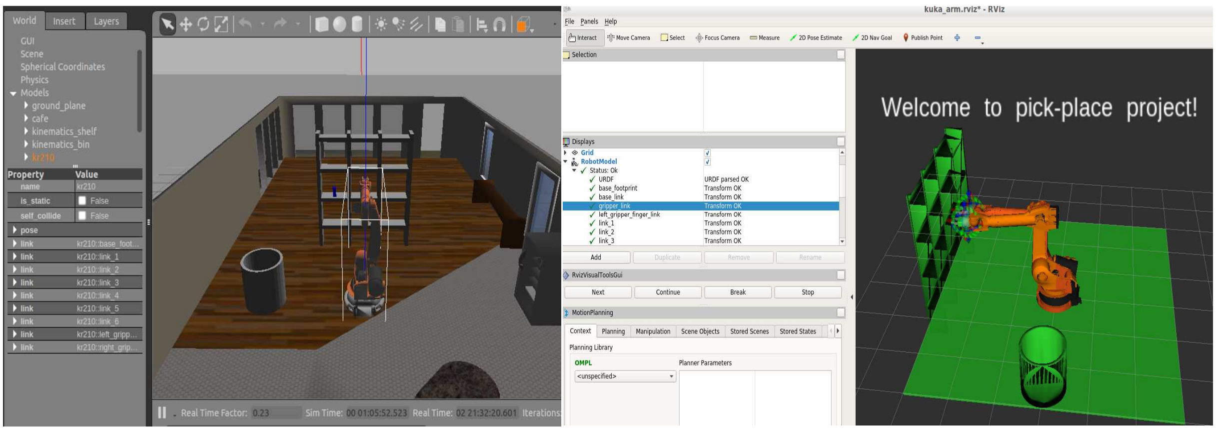
You're reading it!

Kinematic Analysis

2. 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.

First, I tested the robot motions in the demo mode by using the Gazebo and RViz tools.



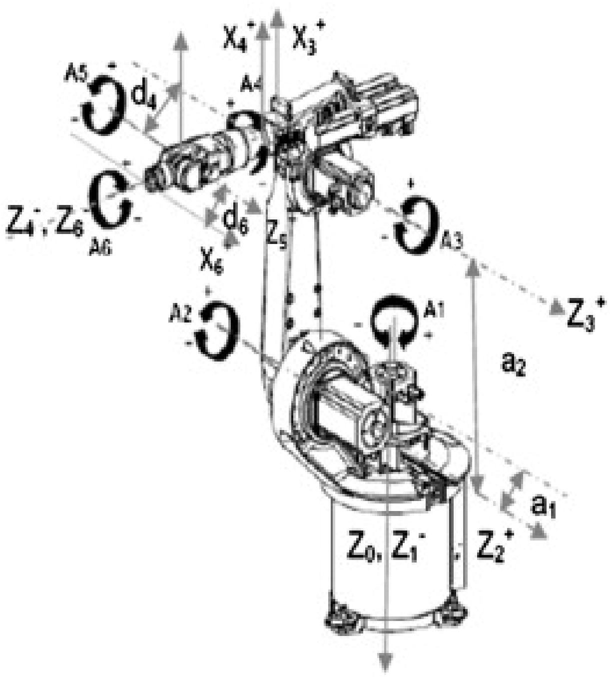


3. Using the DH parameter table you derived earlier, create individual transformation matrices about each joint. In addition, also generate a generalized homogeneous transform between base_link and gripper_link using only end-effector(gripper) pose.

Denavit-Hartenberg Parameters, Forward Kinematics

Deriving the file: kr210.urdf.xacro we can receive the following information about the joints and the links of the robot KR210.

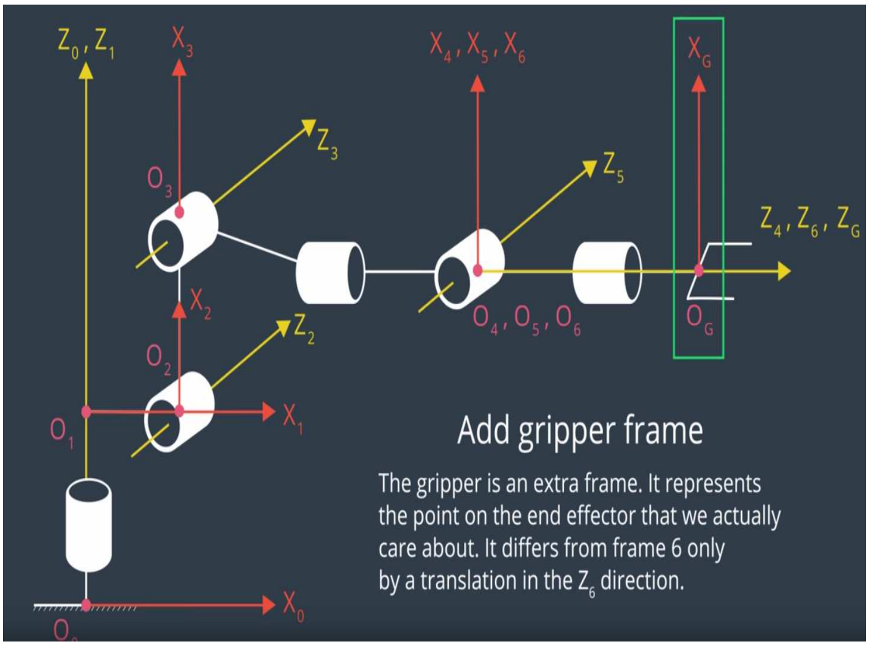
```
<joint name="joint_1" type="revolute">
  <origin xyz="0 0 0.33" rpy="0 0 0"/>
  <axis xyz="0 0 1"/>
<joint name="joint_2" type="revolute">
  <origin xyz="0.35 0 0.42" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
<joint name="joint_3" type="revolute">
  <origin xyz="0 0 1.25" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
<joint name="joint_4" type="revolute">
  <origin xyz="0.96 0 -0.054" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
<joint name="joint_5" type="revolute">
  <origin xyz="0.54 0 0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
<joint name="joint_6" type="revolute">
  <origin xyz="0.193 0 0" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
```

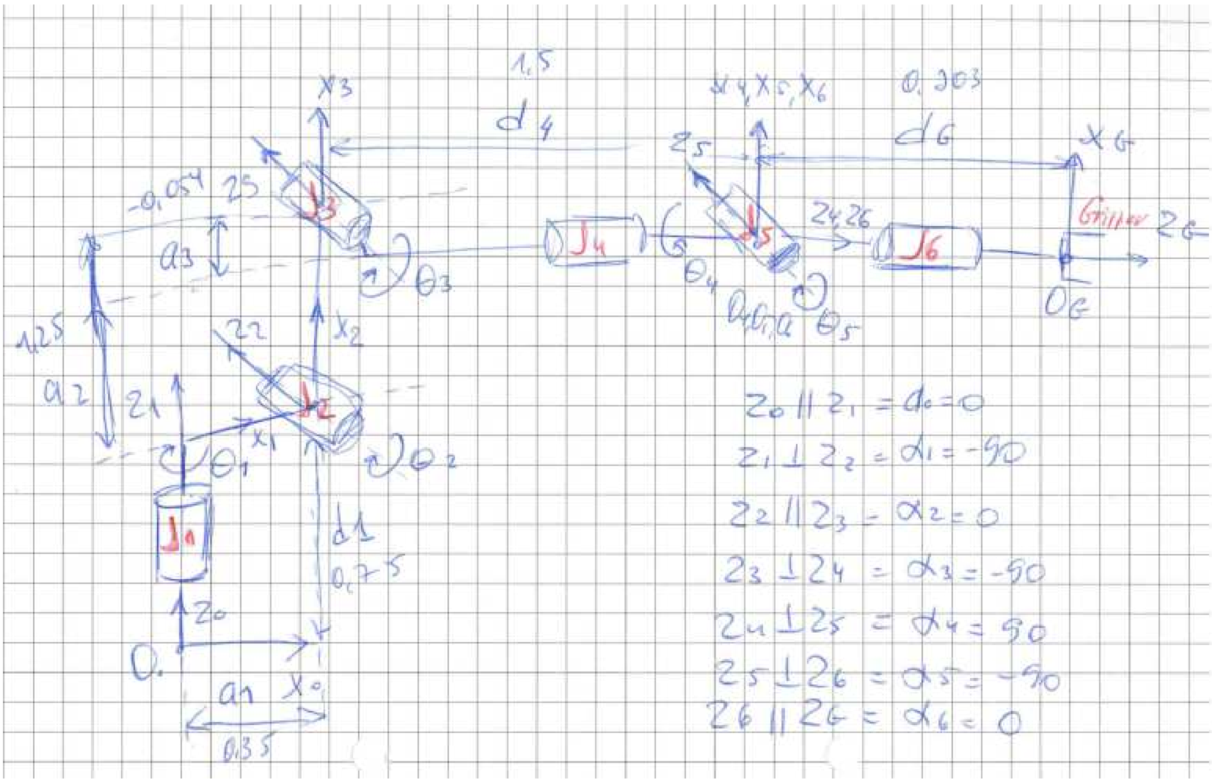


Based on this information we can fill in the table below for each robot joint:

L	X	Y	Z	Roll	Pitch	Yaw
L1	0	0	0.33	0	0	0
L 2	0.35	0	0.42	0	0	0
L3	0	0	1.25	0	0	0
L4	0.96	0	-0.054	0	0	0
L5	0.54	0	0	0	0	0
L6	0.193	0	0	0	0	0
Gripper	0.11	0	0	0	0	0

Using Denavit-Hartenberg method we can evaluate the manipulator kinematics.





I defined z-axes as the joint axes, where joints 2, 3, and 5 are all parallel and joints 4 and 6 are coincident. I defined x-axes as the common normals between zi-1 and zi, , origin of frame {i} is the intersection of xi with zi. A frame is rigidly attached to link 6 for the gripper, which frame differs from the link 6 reference frame by a translation along z6.

Definitions:

Based on the arm's specifications, I was able to derive the following parameters that were also used within the DH diagram.

Links	α (i-1)	a(i-1)	d(i-1)	θ (i)
0->1	0	0	L1	θ1
1->2	- pi/2	0.35	0.75	-pi/2 + θ2
2->3	0	1.25	0	θ3
3->4	- pi/2	-0.054	1.5	θ4
4->5	pi/2	0	0	θ5
5->6	- pi/2	0	0	θ6
6->EE	0	0	0.303	0

The 4 x 4 homogeneous transform between adjacent links is the following:

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

Where for each link four individual transforms is composed: 2 rotations and 2 translations as:

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

The transformation matrix can be calculated by substituting the DH parameters from the table above into this matrix that can solve the Forward kinematics problem:

```
T = [ [        cos(θ),       -sin(θ),       0,                 a],
      [sin(θ)*cos(α), cos(θ)*cos(α), -sin(α), -sin(α)*d],
      [sin(θ)*sin(α), cos(θ)*sin(α),  cos(α),  cos(α)*d],
      [             0,             0,       0,         1]]
```

So now I can define the homogeneous transform between adjacent links and by substituting the known constant values, create the overall transform between the base frame and the end effector.

T0_1: [[cos(θ1), -sin(θ1), 0, 0], [sin(θ1), cos(θ1), 0, 0], [0, 0, 1, 0.75], [0, 0, 0, 1]]	T1_2: [[sin(θ2), cos(θ2), 0, 0.35], [0, 0, 1, 0], [cos(θ2), -sin(θ2), 0, 0], [0, 0, 0, 1]]
T2_3: [[cos(θ3), -sin(θ3), 0, 1.25], [sin(θ3), cos(θ3), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]	T3_4: [[cos(θ4), -sin(θ4), 0, -0.054], [0, 0, 1, 1.5], [-sin(θ4), -cos(θ4), 0, 0], [0, 0, 0, 1]]
T4_5: [[cos(θ5), -sin(θ5), 0, 0], [0, 0, -1, 0], [sin(θ5), cos(θ5), 0, 0], [0, 0, 0, 1]]	T5_6: [[cos(θ6), -sin(θ6), 0, 0], [0, 0, 1, 0], [-sin(θ6), -cos(θ6), 0, 0], [0, 0, 0, 1]]

Transformation matrix for the gripper does not have a rotation, but it does have an offset in the Z direction (out forward).

T6_EE: [[1,   0,   0,   0],

[0,   1,   0,   0],

[0,   0,   1, 0.303],

[0,   0,   0,   1]]

The resulting transformation from base link to the link 6 will be:

T0_EE =T0_1*T1_2*T2_3*T3_4*T4_5*T5_6*T_EE

We will later need the T03 transformation, which is:

T0_3 =T0_1*T1_2*T2_3

T0_3: Matrix

([[sin(q2 + q3)*cos(q1), cos(q1)*cos(q2 + q3), -sin(q1), (1.25*sin(q2) + 0.35)*cos(q1)],

[sin(q1)*sin(q2 + q3), sin(q1)*cos(q2 + q3), cos(q1), (1.25*sin(q2) + 0.35)*sin(q1)],

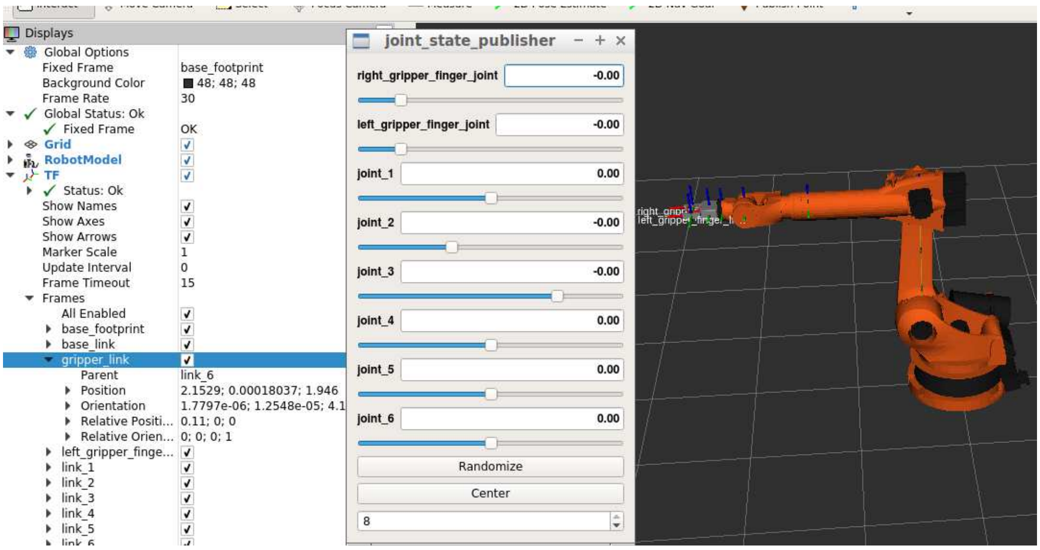
[cos(q2 + q3), -sin(q2 + q3), 0, 1.25*cos(q2) + 0.75],

[0, 0, 0, 1.000]])

Here q1, q2, q3=theta1, theta2, theta3 respectively.

The frames from DH table do not match the default orientation of the KUKA arm in RViz and gazebo. We will need to use additional rotations compensate the difference.

The forward kinematics was executed in RViz together with using the “joint state publisher window”.



Example:

Setting all the angels theta to 0, we can find the position of the end effector as calculated using the transform matrix.

print (T0_EE.evalf(subs={q1:0, q2:0, q3:0, q4:0, q5:0, q6:0}))

Matrix([[0, 0, 1.000, 2.153], [0, -1.0000, 0, 0], [1.000, 0, 0, 1.946], [0, 0, 0, 1.000]])

Using joint angles from the test cases (IK debug.py) we can also receive the position of the end effector.

print (T0_EE[0:3,3].evalf(subs={q1:-0.65, q2:0.45, q3:-0.36, q4:0.95, q5:0.79, q6:0.49}))

Calculated Matrix:

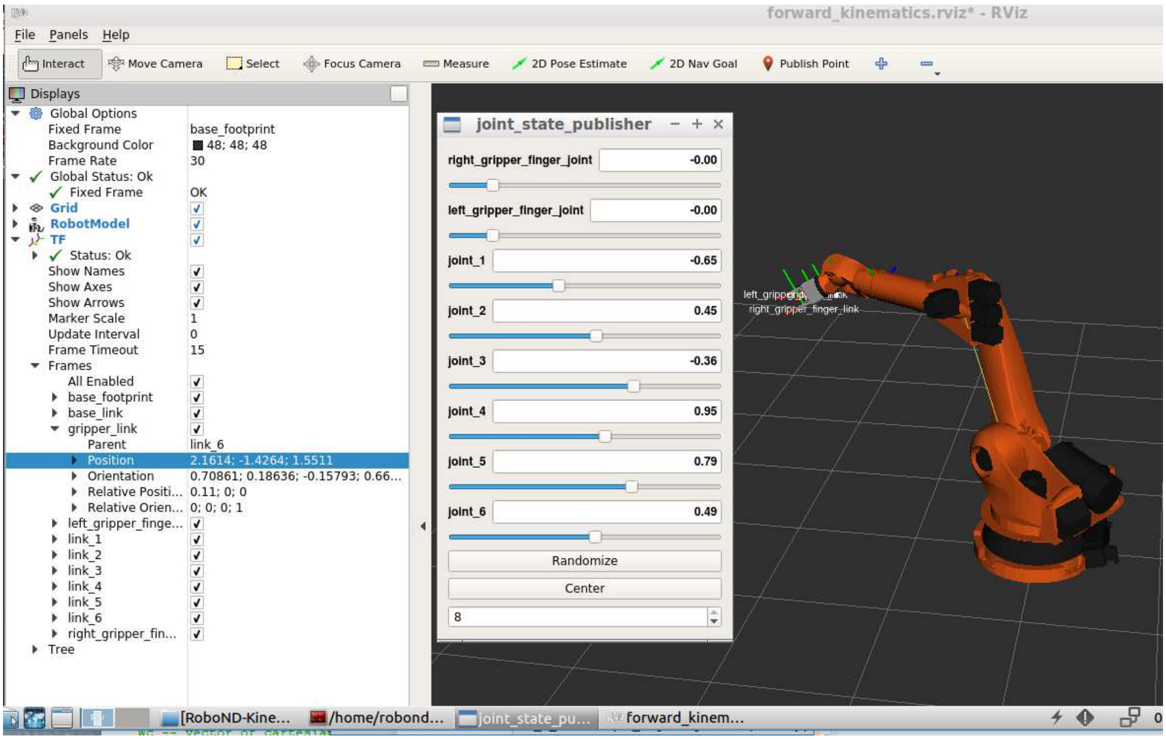
P0_EE=([[2.16298054661547], [-1.42438431475958], [1.54309861553709]])

Results from the RViz:

P_FK=([[2.1614], [-1.4264], [1.5511]])

Error would be: P_FK – P0_EE

Error: 0.00840138488080479

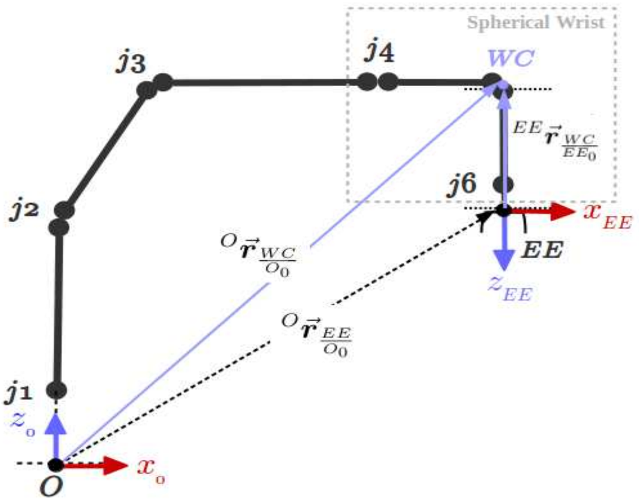


4. Decouple Inverse Kinematics problem into Inverse Position Kinematics and inverse Orientation Kinematics; doing so derive the equations to calculate all individual joint angles.

Inverse Kinematics

The problem that should be solved in Inverse kinematics is how from the end effector position and orientation find the robot joint angles theta.

To solve for the first half of the inverse kinematics problem, which are the first three joint angles, we can apply some trigonometry calculation to find these angles if the coordinates of the wrist-center are known.



How to derive wrist center location from the gripper position and orientation.

From the figure above we can see that the position vector of WC w.r.t. to EE is a simple translation along **z**. The desired position vector of WC w.r.t. to the base frame O can be found by transforming $r_{WC/EEO}$ into the base frame O using a homogeneous transform consisting of Rotation matrix R0_EE and a translation vector from O to Gripper, where d_{EE} is given in the DH table, and the column 3 vector of the Rotation Matrix describes the z-axis of Gripper relative to base frame O.

We can obtain the position of the wrist center by using the complete transformation matrix

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

where *l, m* and *n* are orthonormal vectors representing the end-effector orientation along *X, Y, Z* axes of the local coordinate frame. With known position and orientation of end-effector, we can find the position of the wrist.

Since n is the vector along the z-axis of the Gripper, we can say the following:

Wx = Px - d7*nx

Wy = Py - d7*ny

Wz = Pz - d7*nz

Where,

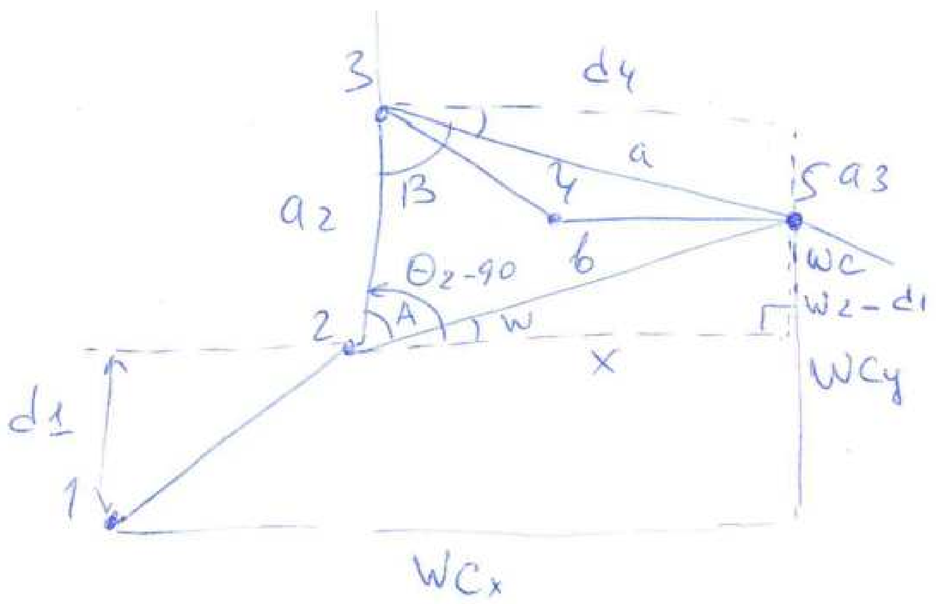
Px, Py, Pz = end-effector positions

Wx, Wy, Wz = wrist positions

D7 = from DH table

Calculation of theta 1, 2 and theta 3

Calculating theta 1 is relatively simple once the wrist center position is known. Theta 2 and theta 3 can be found using projection and the low of cosines. I have used the following Figure to get the missing angles.



x = sqrt(wx**2 + wy**2)-a1

W = arctan2(wz – d1, x)

b = sqrt((wz –d1)**2 + (x)**2)

a = sqrt(d4**2 + a3**2)

A = arccos((b**2 + a2**2 - a**2)/(2*b*a2))

B = arccos((c**2 + a**2 - b**2)/(2*a2*a))

theta1 = atan2(wy, wx)

theta2 = pi/2 - W - A

theta3 = pi/2 - B - arctan2(a3, d4)

where a1=0.35, a2 = 1.25, a3=0.054,d1=0.75, d4=1.5 (from the DH Table)

Since the last three joints of the 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.

In the orientation part of Inverse Kinematics, joint angles 4, 5, and 6 are analytically calculated from R3_6; the composition of x-y-z rotations (roll, pitch, yaw) that orients the wrist center WC. Thus, joint angles 4, 5, and 6 are the Euler angles of this composition of rotations.

Because the base_link is in fixed position where it can only allow rotation on z axis, the relationship between the end-effector and the base_link is extrinsic rotation. It means, that the rotation matrix for the end-effector could be found by knowing the roll, pitch, yaw angles between end-effector and base_link.

To transform the rotation matrix of end effector from URDF and the DH reference frame, we would need to do the correction which is 180 deg yaw and -90 deg pitch rotation.

*R_corr = Rz(pi) * Ry(-pi/2)*

```
Where    Rz= Matrix ([[cos(yaw),-sin(yaw), 0],
                      [sin(yaw), cos(yaw), 0],
                      [       0,        0,        1]])
         R_y = Matrix ([[ cos(pitch), 0, sin(pitch)],
                       [       0,      1,             0],
```

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

One such convention is the x-y-z extrinsic rotations. The resulting rotational matrix using this convention to transform from one fixed frame to another, would be:

*Rrpy = Rot(yaw) * Rot(pitch) * Rot(roll) * R_corr*

```
Rot(roll) = Matrix([[1,          0,            0],
                    [0, cos(roll), -sin(roll)],
                    [0, sin(roll),  cos(roll)]])
Rot(pitch) = Matrix([[ cos(pitch), 0, sin(pitch)],
                     [          0,    1,             0],
                     [-sin(pitch), 0, cos(pitch)]])
Rot(yaw) = Matrix([[cos(yaw),-sin(yaw), 0],
                   [sin(yaw), cos(yaw), 0],
                   [    0,         0,          1]])
```

R3_6 can be determined from R0_6 as following

*R3_6 = inv(R0_3)*R0_6 , where R0_6 can be replaced by Rrpy* R_corr*

R3_6 is the composite rotation matrix from the homogeneous transform T3_6, and R0_3 is given by,

*R0_3 = R0_1 * R1_2 * R2_3*

Since joint angles theta 1, theta 2, and theta 3 have already been calculated, R0_3 is no longer a variable as theta 1, theta 2, and theta 3 can simply be substituted in R0_1, R0_2, and R0_3 respectively, leaving theta 4, theta 5, and theta 6 as the only variables in R3_6.

*R3_6 = inv(R0_3)** Rrpy* R_corr*

```
inv(R0_3)=  [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]])
```

R3_6 =

([[-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)]])

We can now find theta 4, theta 5, and theta 6 easily using the following equations:

r12, r13 = R3_6 [0,1], R3_6 [0,2]

r21, r22, r23 = R3_6 [1,0], R3_6 [1,1], R3_6 [1,2]

r32, r33 = R3_6 [2,1], R3_6 [2,2]

theta5 = arctan2(sqrt(r13**2 + r33**2), r23)

theta4 = arctan2(r33, -r13)

theta6 = arctan2(-r22, r21)

And derive the angels since we have the numerical values of the R0_3 and R0_EE.

Example (from the debug file):

```
test_case = [[[2.16135,-1.42635,1.55109], # end effector position
              [0.708611,0.186356,-0.157931,0.661967]], # end effector orientation
              [1.89451,-1.44302,1.69366] #Wc location
              [-0.65,0.45,-0.36,0.95,0.79,0.49]] # Joint angels
```

Where

px = 2.16135, py= -1.42635, pz= 1.55109

roll = 0.708611, pitch = 0.186356, yaw = -0.157931

wx = px – d7*R0_EE[0,2]

wy = py - d7*R0_EE[1,2]

wz = pz - d7*R0_EE[2,2]

wx= 1.86730177374851

wy= -1.37952067852228

wz= 1.60722960534495

q1= -0.636279983937866

q2= 0.410401660065528

q3= -0.252653773154244

q4= 1.46960145630107

q5= 0.472001895809933

q6= -0.844444132991418

T0_EE = Matrix([[0.0365739909998582, -0.238489249459210, 0.970456192249153, 2.16135000000000], [-0.664805602129873, -0.730853695260487, -0.154552216098079, -1.42635000000000], [0.746120536213204, -0.639512121868283, -0.185279225560902, 1.55109000000000], [0, 0, 0, 1.00000000000000]])

Where

P_EE = Matrix([[2.16135000000000], [-1.42635000000000], [1.55109000000000]])

In comparisons with the RViz results: 2.1718; -1.4021; 1.5621

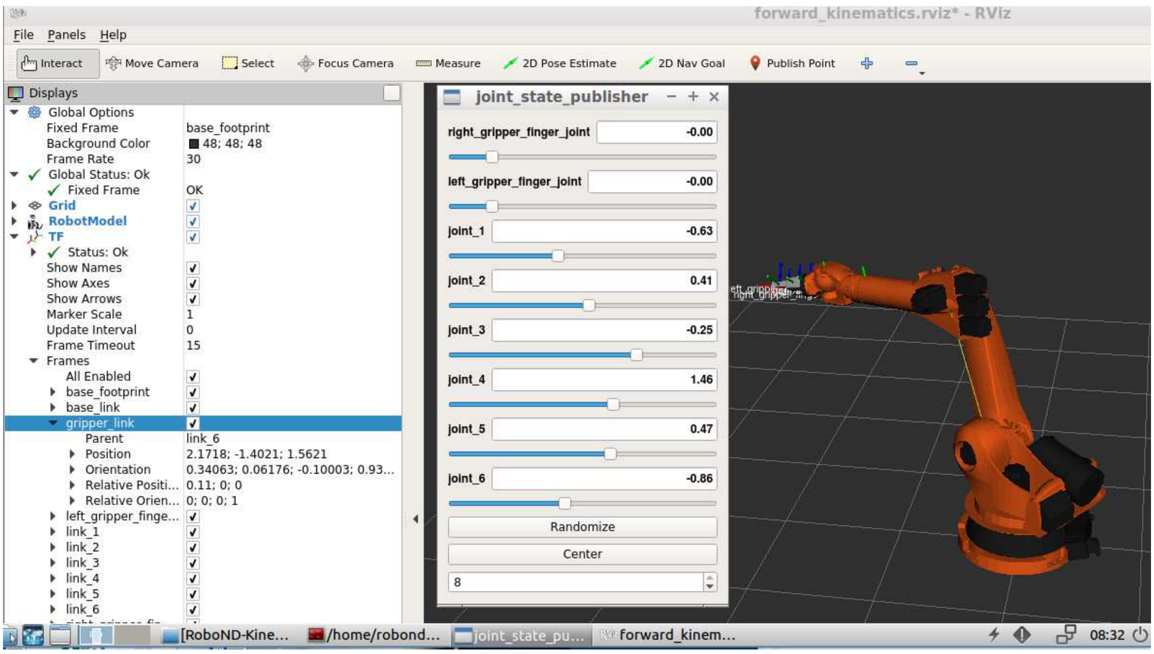
We can easily find the difference by error calculation:

P_EE = Matrix([[2.16298054661547], [-1.42438431475958], [1.54309861553709]])

P_FK = Matrix([[2.1614], [-1.4264], [1.5511]])

Error = P_FK-P_EE

Error.norm () = 0.00840138488080479

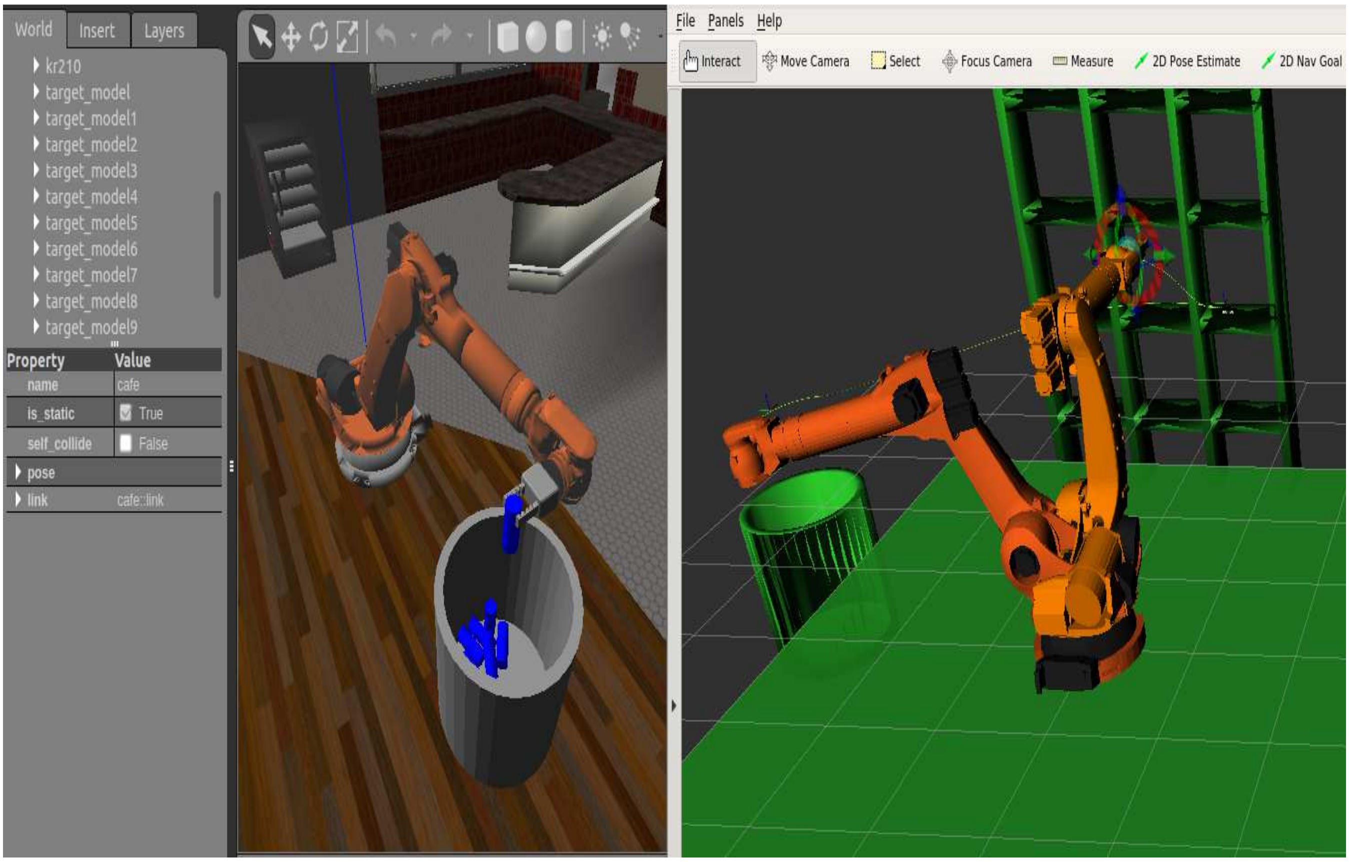


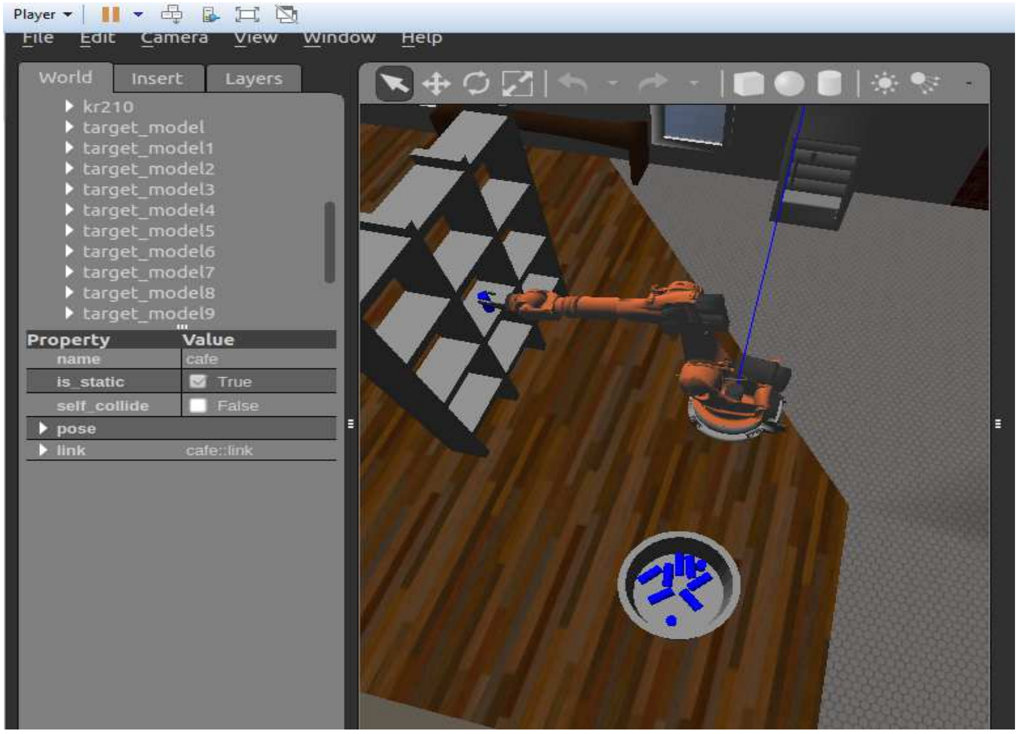
Project Implementation

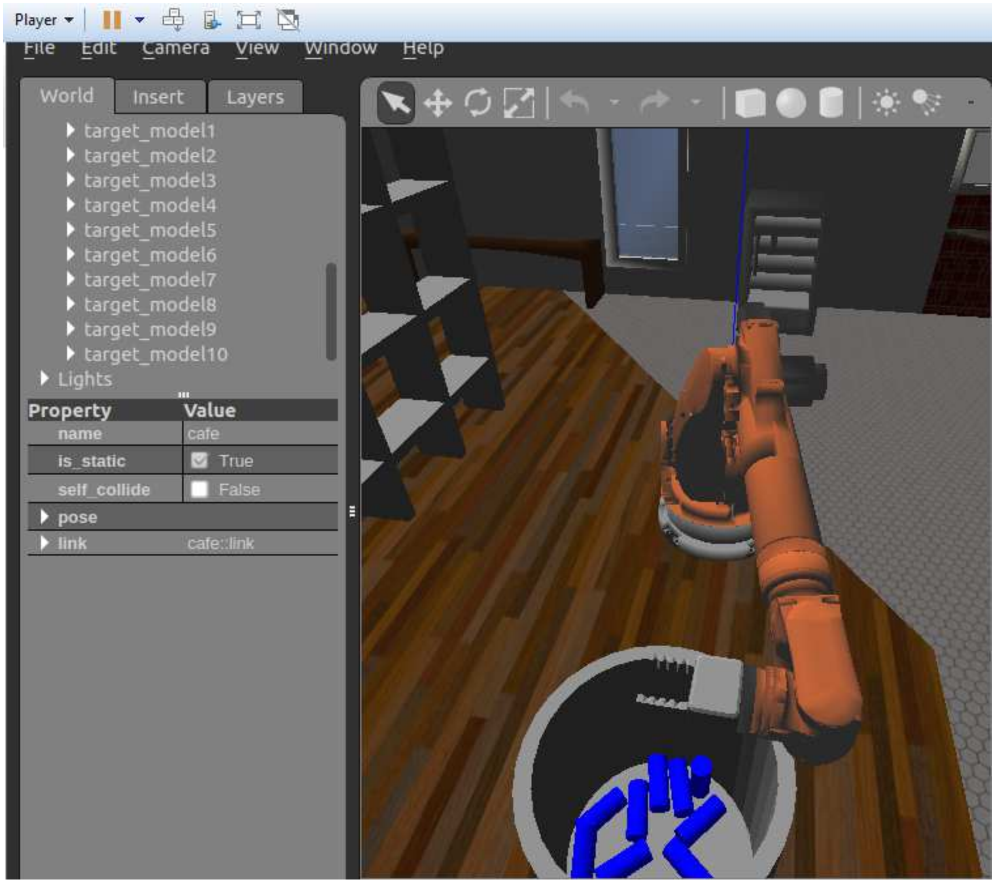
5. Fill in the `IK_server.py` file with properly commented python code for calculating Inverse Kinematics based on previously performed Kinematic Analysis. Your code must guide the robot to successfully complete 8/10 pick and place cycles. Briefly discuss the code you implemented and your results.

By switching from the demo to inverse kinematics, the robot performs the movements using the calculated inverse kinematics.

The results are quite good, but there are too many rotations performed by the robot at the wrist prior moving towards the target. Also the motions are too slow and the improvements can be conducted.




Player
File Edit Camera View Window Help
World Insert Layers
kr210
target_model
target_model1
target_model2
target_model3
target_model4
target_model5
target_model6
target_model7
target_model8
target_model9
Property Value
name cafe
is_static True
self_collide False
pose
link cafe::link


Player
File Edit Camera View Window Help
World Insert Layers
target_model1
target_model2
target_model3
target_model4
target_model5
target_model6
target_model7
target_model8
target_model9
target_model10
Lights
Property Value
name cafe
is_static True
self_collide False
pose
link cafe::link