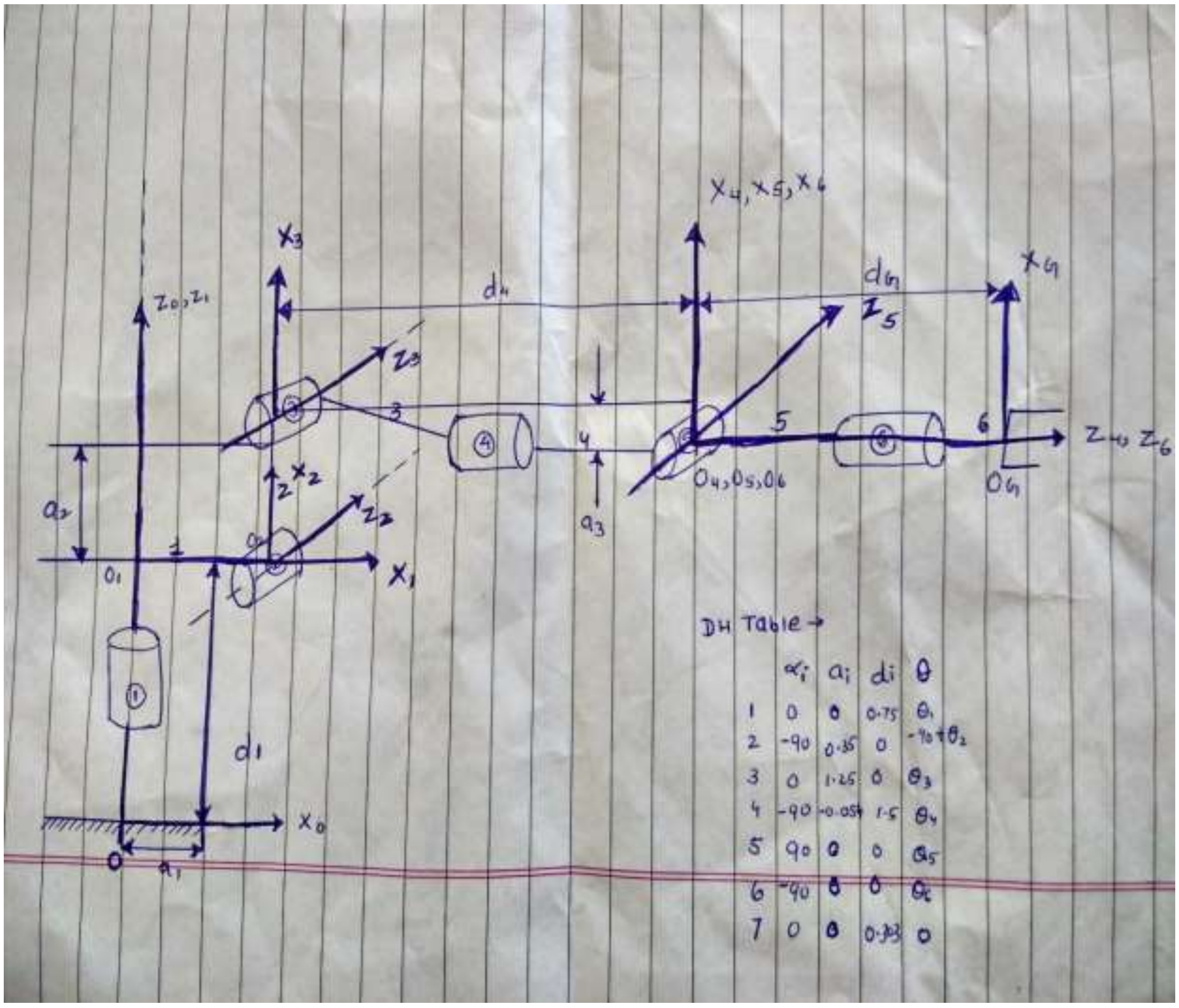
Robotic Arm: Pick & Place

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.**



DH Table →

	α_i	a_i	d_i	θ
1	0	0	0.75	θ_1
2	-90	0.35	0	$-90+\theta_2$
3	0	1.25	0	θ_3
4	-90	-0.054	1.5	θ_4
5	90	0	0	θ_5
6	-90	0	0	θ_6
7	0	0	0.303	0

Using the arm’s specification given in the URDF file and the udacity KR210 session in the classroom, I was able to derive following DH table for Kuka KR210 6 DOF manipulator.

DH table –

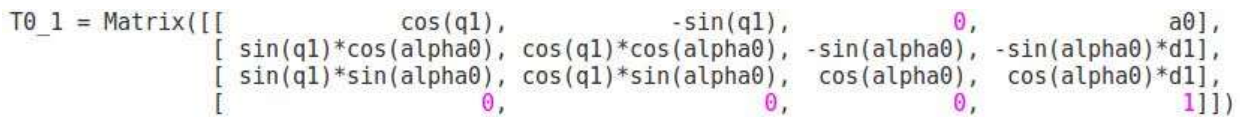
Link	Alpha (j-1)	a (i-1)	d (i-1)	Theta (i)
1	0	0	0.75	q1
2	-pi/2	0.35	0	q2
3	0	1.25	0	q3
4	-pi/2	-0.054	1.5	q4
5	pi/2	0	0	q5
6	-pi/2	0	0	q6
7	0	0	0.303	q7

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.

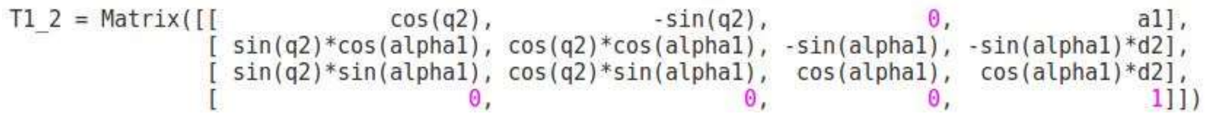
$$
{}^{i-1}_{\ \ i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

To compose a homogeneous transformation matrix of the arm, in order to determine any joint's position relative to another joint, Individual transformation matrix for each matrix must be calculated using the DH table in the above formula and multiplied (T0_1 * T1_2 * T2_3 * T3_4 * T4_5 * T5_6 * T6_EE). Then apply the orientation correction to account for the orientation difference between the gripper and the arm's base. This correction required rotating the end effector's coordinate frame around the Z axis by 180 degrees and around the Y axis by -90 degrees.

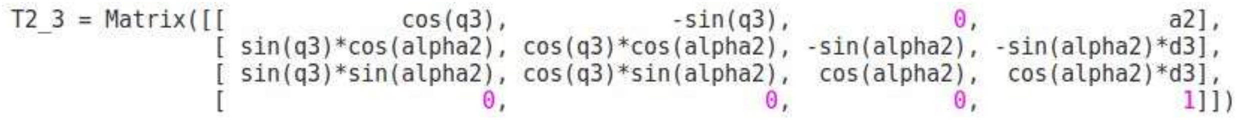
Individual transformation matrix –



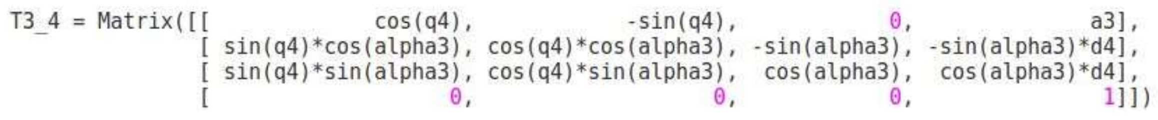
```
T0_1 = Matrix([[             cos(q1),             -sin(q1),            0,              a0],
               [ sin(q1)*cos(alpha0), cos(q1)*cos(alpha0), -sin(alpha0), -sin(alpha0)*d1],
               [ sin(q1)*sin(alpha0), cos(q1)*sin(alpha0),  cos(alpha0),  cos(alpha0)*d1],
               [                   0,                   0,            0,               1]])
```



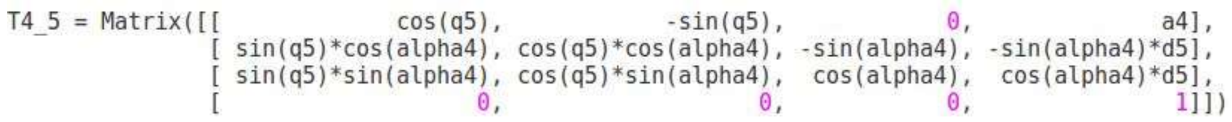
```
T1_2 = Matrix([[             cos(q2),             -sin(q2),            0,              a1],
               [ sin(q2)*cos(alpha1), cos(q2)*cos(alpha1), -sin(alpha1), -sin(alpha1)*d2],
               [ sin(q2)*sin(alpha1), cos(q2)*sin(alpha1),  cos(alpha1),  cos(alpha1)*d2],
               [                   0,                   0,            0,               1]])
```



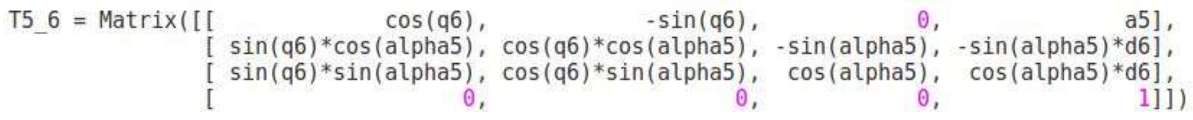
```
T2_3 = Matrix([[             cos(q3),             -sin(q3),            0,              a2],
               [ sin(q3)*cos(alpha2), cos(q3)*cos(alpha2), -sin(alpha2), -sin(alpha2)*d3],
               [ sin(q3)*sin(alpha2), cos(q3)*sin(alpha2),  cos(alpha2),  cos(alpha2)*d3],
               [                   0,                   0,            0,               1]])
```



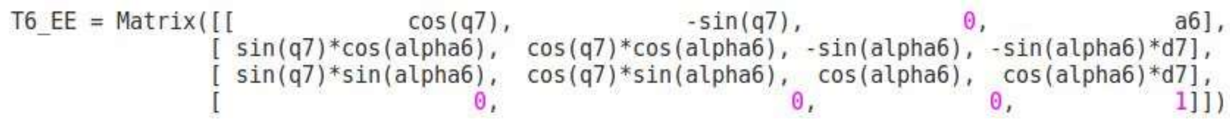
```
T3_4 = Matrix([[             cos(q4),             -sin(q4),            0,              a3],
               [ sin(q4)*cos(alpha3), cos(q4)*cos(alpha3), -sin(alpha3), -sin(alpha3)*d4],
               [ sin(q4)*sin(alpha3), cos(q4)*sin(alpha3),  cos(alpha3),  cos(alpha3)*d4],
               [                   0,                   0,            0,               1]])
```



```
T4_5 = Matrix([[             cos(q5),             -sin(q5),            0,              a4],
               [ sin(q5)*cos(alpha4), cos(q5)*cos(alpha4), -sin(alpha4), -sin(alpha4)*d5],
               [ sin(q5)*sin(alpha4), cos(q5)*sin(alpha4),  cos(alpha4),  cos(alpha4)*d5],
               [                   0,                   0,            0,               1]])
```



```
T5_6 = Matrix([[             cos(q6),             -sin(q6),            0,              a5],
               [ sin(q6)*cos(alpha5), cos(q6)*cos(alpha5), -sin(alpha5), -sin(alpha5)*d6],
               [ sin(q6)*sin(alpha5), cos(q6)*sin(alpha5),  cos(alpha5),  cos(alpha5)*d6],
               [                   0,                   0,            0,               1]])
```



```
T6_EE = Matrix([[             cos(q7),              -sin(q7),            0,              a6],
                [ sin(q7)*cos(alpha6),  cos(q7)*cos(alpha6), -sin(alpha6), -sin(alpha6)*d7],
                [ sin(q7)*sin(alpha6),  cos(q7)*sin(alpha6),  cos(alpha6),  cos(alpha6)*d7],
                [                   0,                    0,            0,               1]])
```

```
T0_2 = simplify(T0_1 * T1_2)
T0_3 = simplify(T0_2 * T2_3)
T0_4 = simplify(T0_3 * T3_4)
T0_5 = simplify(T0_4 * T4_5)
T0_6 = simplify(T0_5 * T5_6)
T0_EE = simplify(T0_6 * T6_EE)
```

```
R_corr = ROT_z.subs(y, pi) * ROT_y.subs(p,-pi/2)

ROT_EE = ROT_EE * R_corr
```

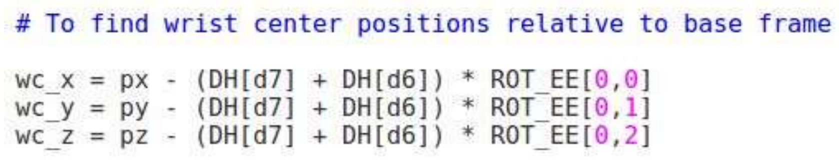
Decouple Inverse Kinematics problem into Inverse Position Kinematics and inverse Orientation Kinematics; doing so derive the equations to calculate all individual joint angles.

Given an end-effector's position (px, py, pz) and orientation (yaw, pitch, roll) the arm's wrist center can be determined using the DH parameters, along with the rotation matrices ROT_x, ROT_y and ROT_z.

```
# Define RPY Rotation matrix
r, p, y = symbols('r p y')

# Roll

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

# Pitch

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

# Yaw

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

ROT_EE = ROT_z * ROT_y * ROT_x
```

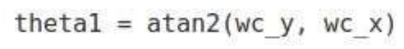
```
ROT_EE = ROT_EE. subs({'r': roll, 'p' : pitch, 'y': yaw})
```

Wrist center position is then calculated using the following formula



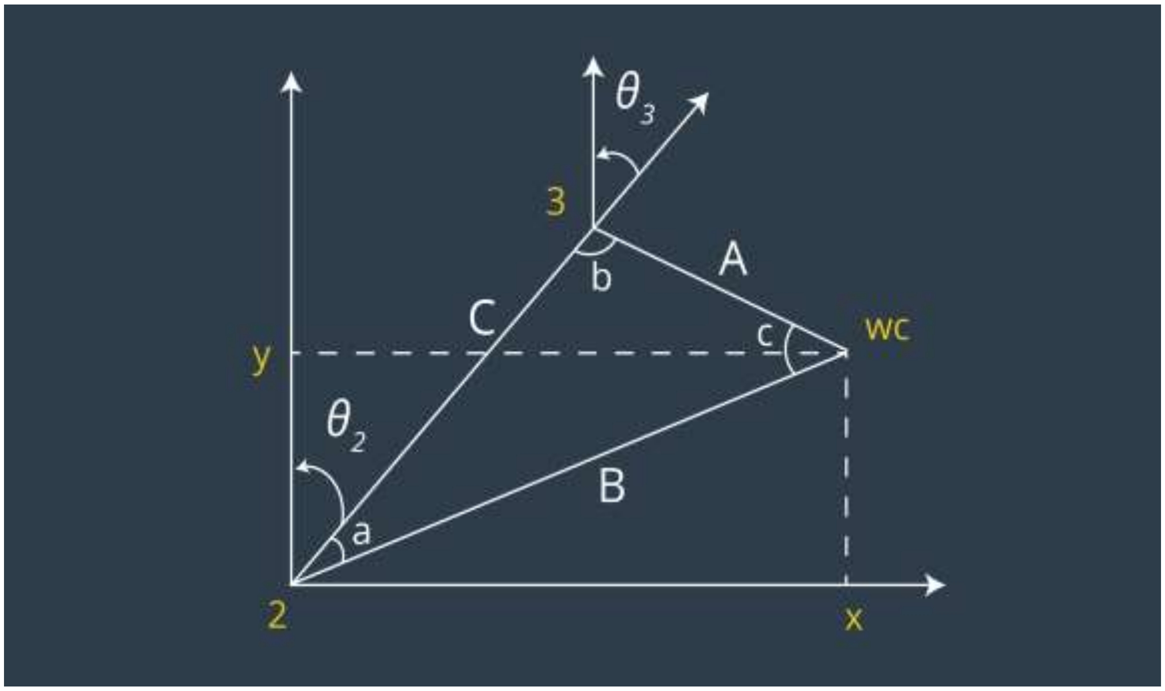
```
# To find wrist center positions relative to base frame

wc_x = px - (DH[d7] + DH[d6]) * ROT_EE[0,0]
wc_y = py - (DH[d7] + DH[d6]) * ROT_EE[0,1]
wc_z = pz - (DH[d7] + DH[d6]) * ROT_EE[0,2]
```

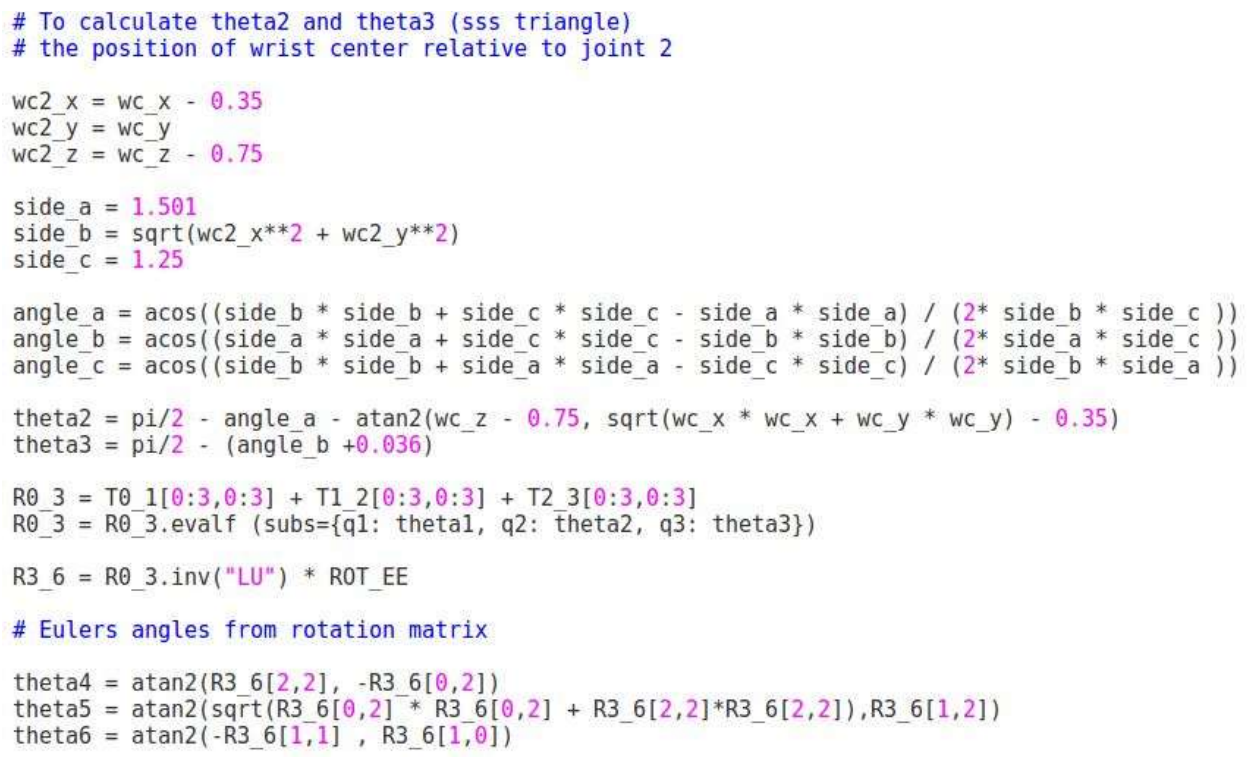
Now that we have wrist center position we can find theta1 using the following formula



```
theta1 = atan2(wc_y, wc_x)
```

Now to find theta2 and theta3, I first found out the position of wrist center relative to joint 2 and then calculated sides of triangle joining joint 2, joint3 and the wrist center (as shown in the below figure). Then using the cosine rule we can find angle a, angle b and angle c and then use it to find theta2 and theta3. Now that we have theta1, theta2 and theta3 we can derive a rotation matrix from base link to 3rd link by putting the theta values in the transformation matrix and then multiply them. Then to determine the rotation matrix from link 3 to end effector we can multiply the ROT_EE with the inverse of R0_3 (Rotation matrix from base link to 3rd link). After getting the rotation matrix from link 3 to end effector we can determine theta4, theta5, and theta6 using Euler's angle.

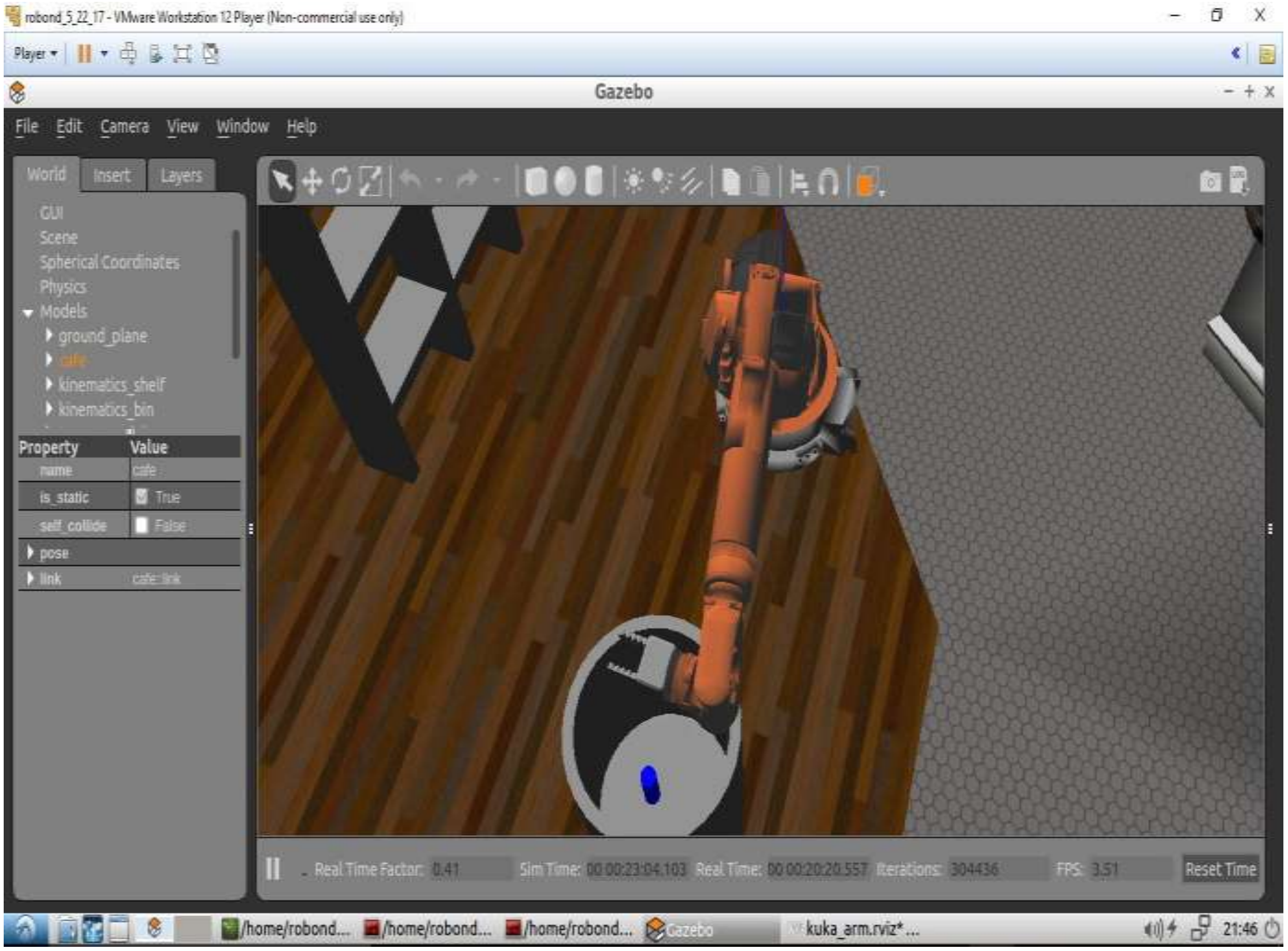




```
# To calculate theta2 and theta3 (sss triangle)
# the position of wrist center relative to joint 2

wc2_x = wc_x - 0.35
wc2_y = wc_y
wc2_z = wc_z - 0.75

side_a = 1.501
side_b = sqrt(wc2_x**2 + wc2_y**2)
side_c = 1.25

angle_a = acos((side_b * side_b + side_c * side_c - side_a * side_a) / (2* side_b * side_c ))
angle_b = acos((side_a * side_a + side_c * side_c - side_b * side_b) / (2* side_a * side_c ))
angle_c = acos((side_b * side_b + side_a * side_a - side_c * side_c) / (2* side_b * side_a ))

theta2 = pi/2 - angle_a - atan2(wc_z - 0.75, sqrt(wc_x * wc_x + wc_y * wc_y) - 0.35)
theta3 = pi/2 - (angle_b +0.036)

R0_3 = T0_1[0:3,0:3] + T1_2[0:3,0:3] + T2_3[0:3,0:3]
R0_3 = R0_3.evalf (subs={q1: theta1, q2: theta2, q3: theta3})

R3_6 = R0_3.inv("LU") * ROT_EE

# Eulers angles from rotation matrix

theta4 = atan2(R3_6[2,2], -R3_6[0,2])
theta5 = atan2(sqrt(R3_6[0,2] * R3_6[0,2] + R3_6[2,2]*R3_6[2,2]),R3_6[1,2])
theta6 = atan2(-R3_6[1,1] , R3_6[1,0])
```

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.

After running the code of IK_server.py, the robotic arm was successfully able to drop the cylinder in the bin following the required trajectory. The below image shows the successful placing of a cylinder in the bin. Future work would include improving the inverse kinematic accuracy.



Property	Value
name	cafe
is_static	✓ True
self_collide	False
pose	
link	cafe::link

Real Time Factor: 0.41 Sim Time: 00 00:23:04.103 Real Time: 00 00:20:20.557 Iterations: 304436 FPS: 3.51