Project: Kinematics Pick & Place

Steps to complete the project:

1. Set up your ROS Workspace.
2. Download or clone the project repository 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.
5. Perform Kinematic Analysis for the robot following the project rubric.
6. Fill in the `IK_server.py` with your Inverse Kinematics code.

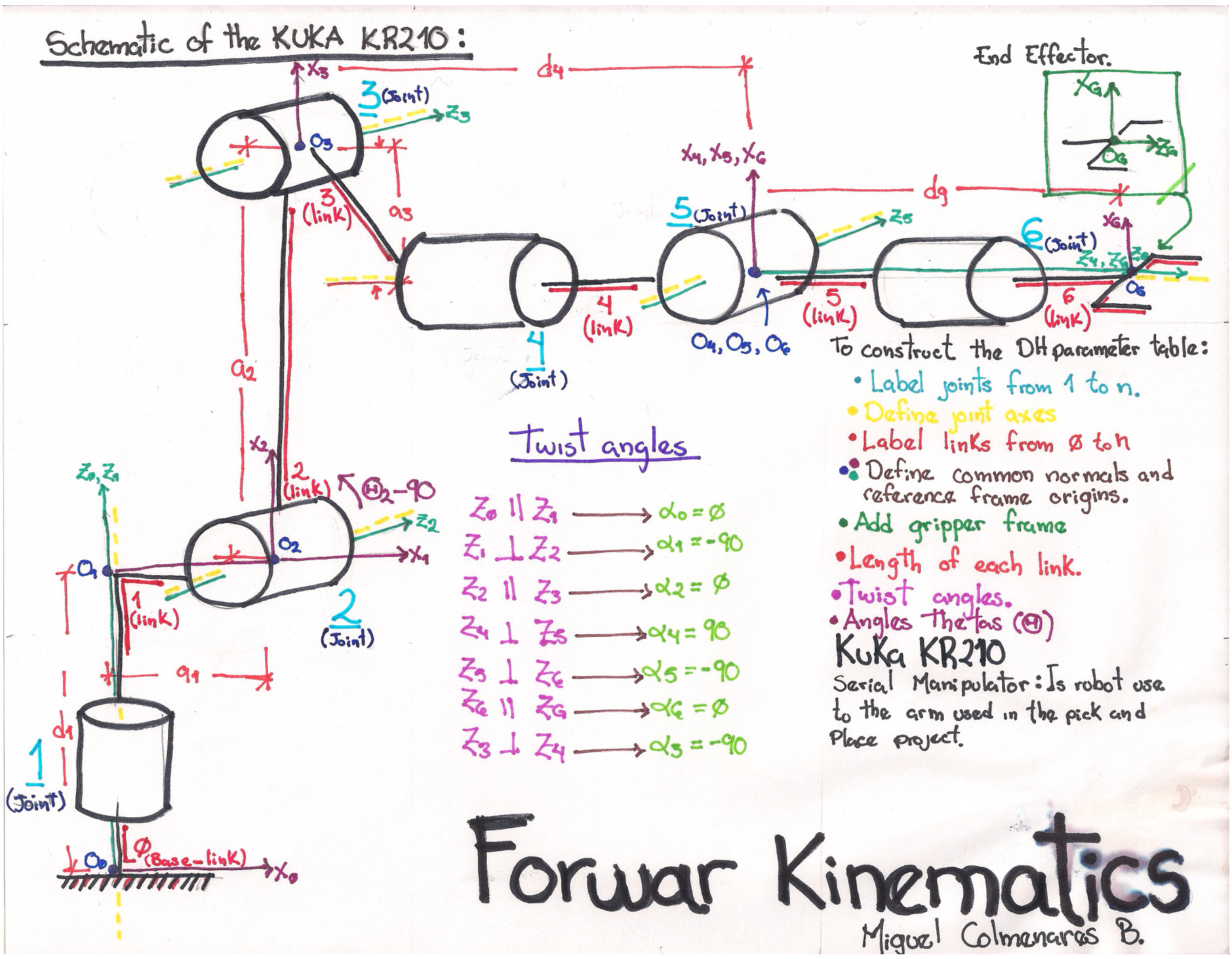
Writeup / README

You're reading it!

Hi. I am Miguel Colmenares, engineer in electronics and i want to tell you about my experience in this project.

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.



Above is an outline of the KUKA KR210 that shows the location of the DH parameters. Use colors to facilitate the understanding of the system at a glance.

To construct the DH parameter table:

- Label joint from 1 to n (Blue color).
  - In this case there are 6 joints. n=6
- Define joint axes (Yellow color).
  - The joints 2,3 & 5 are parallels, and the joints 4 & 6 are coincident.

- Label links from 0 to n (Red color).
  - link_0 = base_link.
- Define common normals and reference frame origins (Brown color).
  - The axes X (of color mulberry) define the common normals between Zi-1 and Zi.
  - The axes Z (of color aquamarine).
  - The reference frame origins Oi (of color purple) are defined as the intersection between Xi and Zi.
- Add gripper frame (Green color).
  - The important thing is the center of the clamp. For this reason, an additional (G) frame is added.
- Lenght of each link (Orange color).
  - Ai-1 is the distance from Zi-1 to Zi. Measure along of Xi-1.
  - Di is the distance from Xi-1 to Xi. Measure along of Zi.
- Twist angles (Pink color).
  - Is the angle between Zi-1 and Zi. Measure along of axes Xi-1.
  - To know the sign (+/- 90) use the technique of the right hand.
- Angles Thetas Θ (Wine color).
  - Θi is the angle between Xi-1 and Xi above Zi in the sense of the right hand.

Evaluating the kr210.urdf.xacro file you get the following information.

Joint Name	Parent link	Child link	X(m)	Y(m)	Z(m)
joint_1	base_link	link_1	0	0	0.33
joint_2	link_1	link_2	0.35	0	0.42
joint_3	link_2	link_3	0	0	1.25
joint_4	link_3	link_4	0.96	0	-0.054
joint_5	link_4	link_5	0.54	0	0
joint_6	link_5	link_6	0.193	0	0
joint_g	link_6	gripper_link	0.11	0	0
		Total:	2.153		1.946

Therefore the DH parameters are:

With i=1 : alpha(0)= 0° , a(0)=0 , d(1)=0.33+0.42 , Θ(1)= X0 ∥ X1

With i=2 : alpha(1)= -90° , a(1)=0.35 , d(2)=0 , Θ(2)= X1 ⊥ X2

With i=3 : alpha(2)= 0 , a(2)=1.25 , d(3)=0 , Θ1= X2 ∥ X3

With i=4 : alpha(3)= -90° , a(3)=-0.054 , d(4)=0.96+0.54 , Θ1= X3 ∥ X4

With i=5 : alpha(4)= 90° , a(4)=0 , d(5)=0 , Θ1= X4 ∥ X5

With i=6 : alpha(5)= -90° , a(5)=0 , d(6)=0 , Θ1= X5 ∥ X6

With i=7 : alpha(6)= 0° , a(6)=0 , d(7)=0.193+0.11 , Θ1=X6 ∥ Xg

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.

DH Parameters:

Links	alpha(i-1)	a(i-1)	d(i)	theta(i)
0->1	0	0	0,75	q1
1->2	- pi/2	0.35m	0	-pi/2 + q2
2->3	0	1.25m	0	q3
3->4	- pi/2	-0.054m	1.50	q4

Links	alpha(i-1)	a(i-1)	d(i)	theta(i)
4->5	+ pi/2	0	0	q5
5->6	- pi/2	0	0	q6
6->EE	0	0	0.303	0

DH Transformation Matrix:

The FK problem boils down to the composition of homogeneous transforms. We start with the base link and move link by link to the end effector. And we use the DH parameters to build each individual transform.

$$ {}^{0}_{N}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T \dots {}^{N-1}_{N}T $$

Recall that the total transform between adjacent links,

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

for each link is actually composed of four individual transforms, 2 rotations and 2 translations, performed in the order shown here.

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

therefore:

```
T0_1 = Rx(alpha0) Dx(a0) Rz(theta1) Dz(d1)
DH parameters: alpha0=0 , a0=0 , theta1=q1 , d1=0.75.

T0_1 = Matrix ([[          cos(theta1),           -sin(theta1),           0,             a0],
              [sin(theta1)*cos(alpha0), cos(theta1)*cos(alpha0), -sin(alpha0), -sin(alpha0)*d1],
              [sin(theta1)*sin(alpha0), cos(theta1)*sin(alpha0),  cos(alpha0),  cos(alpha0)*d1],
              [                     0,                      0,           0,             1]])

T0_1 = Matrix([[cos(q1), -sin(q1), 0,      0],
               [sin(q1),  cos(q1), 0,      0],
               [      0,        0, 1, 0.7500],
               [      0,        0, 0,      1]])
```

```
T1_2 = Rx(alpha1) Dx(a1) Rz(theta2) Dz(d2)
DH parameters: alpha1=-pi/2. , a1=0.35 , theta2=q2-pi/2. , d2=0.

T1_2 = Matrix ([[          cos(theta2),           -sin(theta2),           0,             a1],
              [sin(theta2)*cos(alpha1), cos(theta2)*cos(alpha1), -sin(alpha1), -sin(alpha1)*d2],
              [sin(theta2)*sin(alpha1), cos(theta2)*sin(alpha1),  cos(alpha1),  cos(alpha1)*d2],
              [                     0,                      0,           0,             1]])

T1_2 = Matrix([[cos(q2 - 0.5*pi), -sin(q2 - 0.5*pi), 0, 0.3500],
               [                0,                0, 1,      0],
               [-sin(q2 - 0.5*pi), -cos(q2 - 0.5*pi), 0,      0],
               [                0,                0, 0,      1]])
```

```
T2_3 = Rx(alpha2) Dx(a2) Rz(theta3) Dz(d3)
DH parameters: alpha2=0 , a2=1.25 , theta3=q3 , d3=0.

T2_3 = Matrix ([[          cos(theta3),           -sin(theta3),           0,             a2],
              [sin(theta3)*cos(alpha2), cos(theta3)*cos(alpha2), -sin(alpha2), -sin(alpha2)*d3],
              [sin(theta3)*sin(alpha2), cos(theta3)*sin(alpha2),  cos(alpha2),  cos(alpha2)*d3],
              [                     0,                      0,           0,             1]])

T2_3 = Matrix([[cos(q3), -sin(q3), 0, 1.2500],
               [sin(q3),  cos(q3), 0,      0],
```

```
               [      0,        0, 1,      0],
               [      0,        0, 0,      1]])
```

```
T3_4 = Rx(alpha3) Dx(a3) Rz(theta4) Dz(d4)
DH parameters: alpha3=-pi/2. , a3=-0.054 , theta2=q4 , d4=1.5 .

T3_4 = Matrix ([[          cos(theta4),            -sin(theta3),            0,              a3],
             [sin(theta4)*cos(alpha3), cos(theta4)*cos(alpha3), -sin(alpha3), -sin(alpha3)*d4],
             [sin(theta4)*sin(alpha3), cos(theta4)*sin(alpha3),  cos(alpha3),  cos(alpha3)*d4],
             [                      0,                      0,            0,              1]])

T3_4 = Matrix([[ cos(q4), -sin(q4), 0, -0.0540],
               [       0,        0, 1,  1.5000],
               [-sin(q4), -cos(q4), 0,       0],
               [       0,        0, 0,       1]])
```

```
T4_5 = Rx(alpha4) Dx(a4) Rz(theta5) Dz(d5)
DH parameters: alpha4= pi/2. , a4=0 , theta5=q5 , d5=0.

T4_5 = Matrix ([[          cos(theta5),            -sin(theta5),            0,              a4],
             [sin(theta5)*cos(alpha4), cos(theta5)*cos(alpha4), -sin(alpha4), -sin(alpha4)*d5],
             [sin(theta5)*sin(alpha4), cos(theta5)*sin(alpha4),  cos(alpha4),  cos(alpha4)*d5],
             [                      0,                      0,            0,              1]])

T4_5 = Matrix([[cos(q5), -sin(q5),  0, 0],
               [      0,        0, -1, 0],
               [sin(q5),  cos(q5),  0, 0],
               [      0,        0,  0, 1]])
```

```
T5_6 = Rx(alpha5) Dx(a5) Rz(theta6) Dz(d6)
DH parameters: alpha5=-pi/2. , a5=0 , theta6=q6 , d6=0.

T5_6 = Matrix ([[          cos(theta6),            -sin(theta6),            0,              a5],
             [sin(theta6)*cos(alpha5), cos(theta6)*cos(alpha5), -sin(alpha5), -sin(alpha5)*d6],
             [sin(theta6)*sin(alpha5), cos(theta6)*sin(alpha5),  cos(alpha5),  cos(alpha5)*d6],
             [                      0,                      0,            0,              1]])

T5_6 = Matrix([[ cos(q6), -sin(q6), 0, 0],
               [       0,        0, 1, 0],
               [-sin(q6), -cos(q6), 0, 0],
               [       0,        0, 0, 1]])
```

```
T6_EE = Rx(alpha6) Dx(a6) Rz(theta7) Dz(d7)
DH parameters: alpha6=0 , a6=0 , theta7=0 , d7=0.303 .

T6_EE = Matrix ([[          cos(theta7),            -sin(theta7),            0,              a6],
               [sin(theta7)*cos(alpha6), cos(theta7)*cos(alpha6), -sin(alpha6), -sin(alpha6)*d7],
               [sin(theta7)*sin(alpha6), cos(theta7)*sin(alpha6),  cos(alpha6),  cos(alpha6)*d7],
               [                      0,                      0,            0,              1]])

T6_EE = Matrix([[1, 0, 0,      0],
                [0, 1, 0,      0],
                [0, 0, 1, 0.3030],
                [0, 0, 0,      1]])
```

And now of these seven matrices I am conforming a matrix of homogeneous transformations.

```
T0_1 = simplify(T0_1 * 1   ) # base_link to link1
print("\n T0_1 = ",T0_1.evalf(subs={q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6: 0}))

T0_1 =  Matrix([[1, 0, 0,      0],
                [0, 1, 0,      0],
                [0, 0, 1, 0.7500],
                [0, 0, 0, 1.0000]])
```

```
T0_2 = simplify(T0_1 * T1_2) # base_link to link2
print("\n T0_2 = ",T0_2.evalf(subs={q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6: 0}))

T0_2 =  Matrix([[  0, 1.0,   0, 0.35],
                [  0,   0, 1.0,    0],
                [1.0,   0,   0, 0.75],
                [  0,   0,   0,  1.0]])
```

```
T0_3 = simplify(T0_2 * T2_3) # base_link to link3
print("\n T0_3 = ",T0_3.evalf(subs={q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6: 0}))
```

```
T0_3 =  Matrix([[  0, 1.0,   0, 0.35],
                [  0,   0, 1.0,    0],
                [1.0,   0,   0,  2.0],
                [  0,   0,   0,  1.0]])
```

```
T0_4 = simplify(T0_3 * T3_4) # base_link to link4
print("\n T0_4 = ",T0_4.evalf(subs={q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6: 0}))

T0_4 =  Matrix([[  0,    0, 1.0,  1.85],
                [  0, -1.0,   0,     0],
                [1.0,    0,   0, 1.946],
                [  0,    0,   0,   1.0]])
```

```
T0_5 = simplify(T0_4 * T4_5) # base_link to link5
print("\n T0_5 = ",T0_5.evalf(subs={q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6: 0}))

T0_5 =  Matrix([[  0, 1.0,   0,  1.85],
                [  0,   0, 1.0,     0],
                [1.0,   0,   0, 1.946],
                [  0,   0,   0,   1.0]])
```

```
T0_6 = simplify(T0_5 * T5_6) # base_link to link6
print("\n T0_6 = ",T0_6.evalf(subs={q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6: 0}))

T0_6 =  Matrix([[  0,    0, 1.0,  1.85],
                [  0, -1.0,   0,     0],
                [1.0,    0,   0, 1.946],
                [  0,    0,   0,   1.0]])
```

```
T0_E = simplify(T0_6 * T6_EE) # base_link to linkEE
print("\n T0_E = ",T0_E.evalf(subs={q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6: 0}))

T0_E =  Matrix([[  0,    0, 1.0, 2.153],
                [  0, -1.0,   0,     0],
                [1.0,    0,   0, 1.946],
                [  0,    0,   0,   1.0]])
```

After applying a correction of the model with respect to the gazebo.

```
R_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],
         ])

R_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],
         ])

R_corr = simplify(R_z * R_y)
```

This is the final matrix:

```
T_total = simplify(T0_E * R_corr)

print('\n T_total = ',T_total.evalf(subs={q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6: 0}))

T_total =  Matrix([[        1.0,           0, 6.1232e-17, 2.1530],
                   [-7.4987e-33,         1.0, 1.2246e-16,      0],
                   [-6.1232e-17, -1.2246e-16,        1.0, 1.9460],
                   [          0,           0,          0,    1.0]])
```

remembering that ...

$$T = \left[\begin{array}{ccc:c} & R_T & & \begin{matrix} P_x \\ P_y \\ P_z \end{matrix} \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right]$$

Therefore, the location points of our end-effector are:

Px = 2.153 ; Py = 0 ; Pz = 1.946.

All the procedure and calculation of matrices was carried out with the code of the file FWKuka210.py, which is in the zip of the project.

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

Often the last three joints in the manipulator are called "revolute joints", such a design is called a spherical wrist and the common point of intersection is called the wrist center (WC). The advantage of such a design is that it kinematically decouples the position and orientation of the end effector. Physically speaking, at six degrees of manipulation with a spherical wrist would use the first three joints to control the position of the wrist center while the last three joints would orient the end effector as needed.

After complete the DH parameter table for the manipulator, and find the location of the WC relative to the base frame derived from the overall homogeneous transform. The next step is find joint variables, q1 = nx, q2 = ny and q3 = nz, such that the WC has coordinates equal to equation of the cartesia coordinates of the WC.

Now, in order to calculate nx, ny, and nz, we need calculated the rotation matrix to correct the difference between the URDF and the DH reference frames for the end-effector.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sen\alpha \\ 0 & sen\alpha & cos\alpha \end{bmatrix}$$

$$R_y = \begin{bmatrix} cos\beta & 0 & sen\beta \\ 0 & 1 & 0 \\ -sen\beta & 0 & cos\beta \end{bmatrix}$$

$$R_z = \begin{bmatrix} cos\theta & -sen\theta & 0 \\ sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

After you obtain this correctional rotation matrix, we need to next calculate your end-effector pose with respect to the base_link.

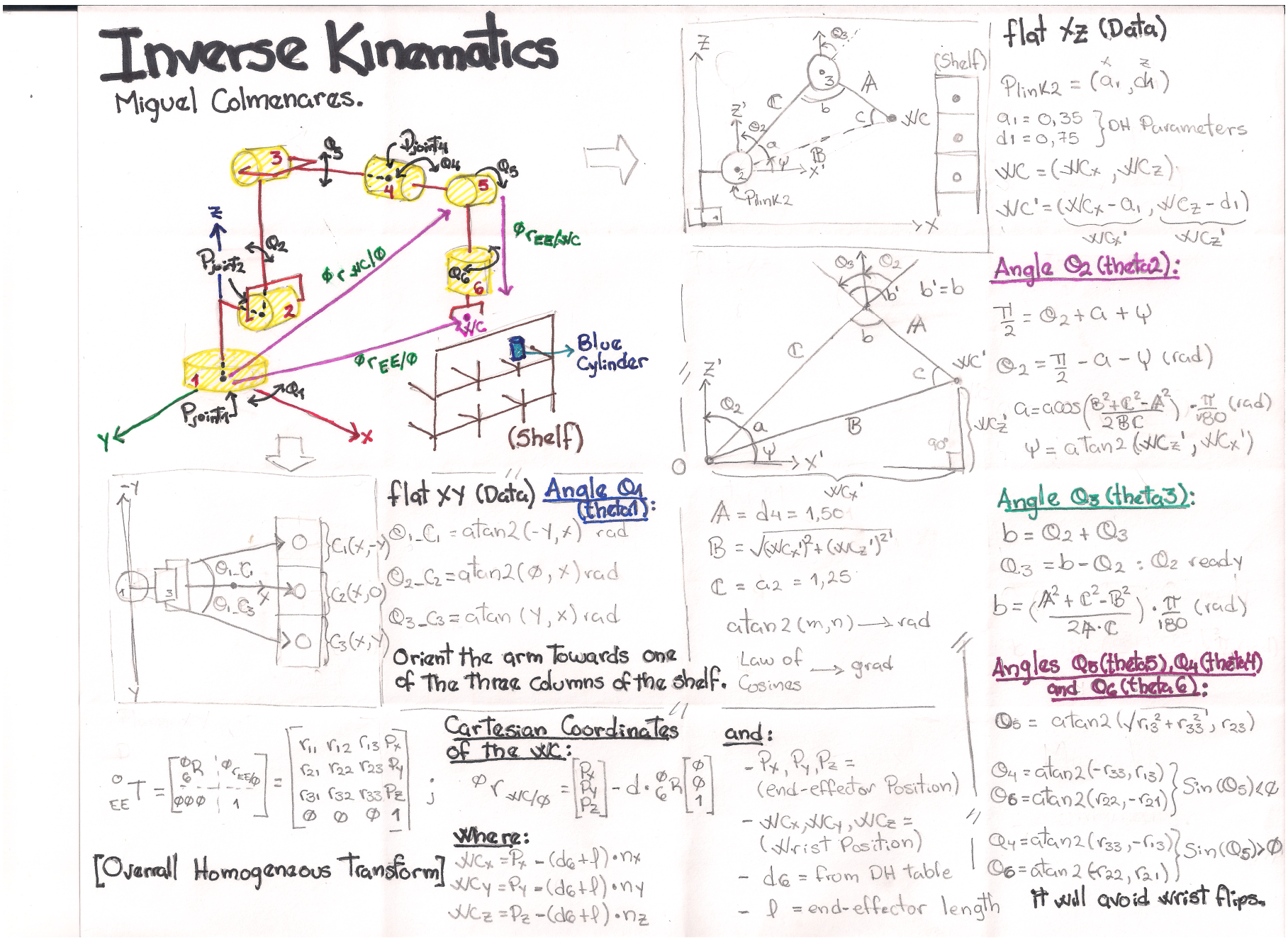
One such convention is the x-y-z extrinsic rotations. The resulting rotational matrix using this convention to transform from one fixed frame to another, would be:

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

Where R_corr is the correctional rotation matrix.

The roll, pitch, and yaw values for the gripper are obtained from the simulation in ROS. But since these values are returned in quaternions, we use the `transformations.py` module from the TF package. The euler_from_quaternions() method will output the roll, pitch, and yaw values.

We can then extract nx, ny, and nz values from this Rrpy matrix to obtain the wrist center position. Now that we have the wrist center position, now we determine the equations for first three joints.



Angle θ1 (theta1): This angle determines the XY orientation of the arm with which one of the three columns of the shelf is chosen.

therefore if we calculate theta1 for column one : θ1_C1 = atan2(-y,x) rad therefore if we calculate theta1 for column two : θ1_C2 = atan2(0,x) rad therefore if we calculate theta1 for column three: θ1_C3 = atan2(+y,x) rad

therefore theta1 = atan2(WCy ,WCx)

Angle θ2 (theta2): To determine theta2 we must first Pjoint2 at the base of the system.

Pjoint2 (a1,d1) = (0.35,0.75) ; WC = (WCx , WCz) transformations, WC'=(WCx',WCz') = (WCx-a1 , WCz-d1)

where theta2 = pi/2 - angle_a - angle_psi

angle_a = acos((side_b^2+side_c^2-side_a) / (2*side_b*side_c)) * pi/180 rad angle_psi = atan2(WCz',WCx')

Angle θ3 (theta3):

theta3 = angle_b - theta2

and the sides are:

side_a = 1.501 # igual a d3 side_b = sqrt(pow((WC[0] - 0.35),2) + pow((WC[2] - 0.75),2)) side_c = 1.25 # igual a a2

```
theta1 = atan2(WC[1],WC[0])

side_a = 1.501  # igual a d3
side_b = sqrt(pow((WC[0] - 0.35),2) + pow((WC[2] - 0.75),2))
side_c = 1.25 # igual a a2
```

```
angle_a = acos((side_b * side_b + side_c * side_c - side_a * side_a) / (2 * side_b * side_c))
angle_a = angle_a * pi / 180 # Para pasar a radianes

angle_b = acos((side_a * side_a + side_c * side_c - side_b * side_b) / (2 * side_a * side_c))
angle_b = angle_b * pi / 180 # Para pasar a radianes

theta2 = pi / 2 - angle_a - atan2(WC[2] - 0.75, WC[0] - 0.35)
theta3 = angle_b - theta2 + 0.036     # 0.036 accounts for sag in link of -0.054m
```

For the Inverse Orientation problem, we need to find values of the final three joint variables.

Using the individual DH transforms we can obtain the resultant transform and hence resultant rotation by:

`R0_6 = R0_1*R1_2*R2_3*R3_4*R4_5*R5_6`

Since the overall RPY (Roll Pitch Yaw) rotation between base_link and gripper_link must be equal to the product of individual rotations between respective links, following holds true: `R0_6 = Rrpy` where, Rrpy = Homogeneous RPY rotation between base_link and gripper_link as calculated above.

We can substitute the values we calculated for joints 1 to 3 in their respective individual rotation matrices and pre-multiply both sides of the above equation by inv(R0_3) which leads to:

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

The resultant matrix on the RHS (Right Hand Side of the equation) does not have any variables after substituting the joint angle values, and hence comparing LHS (Left Hand Side of the equation) with RHS will result in equations for joint 4, 5, and 6.

Finally, we have equations describing all six joint variables.

```
# Euler angles from rotation matrix

theta5 = atan2(sqrt(R3_6[0,2]*R3_6[0,2] + R3_6[2,2]*R3_6[2,2]),R3_6[1,2])
            if sin(theta5 ) < 0:
                theta4 = atan2(-R3_6[2,2], R3_6[0,2])
                theta6 = atan2( R3_6[1,1],-R3_6[1,0])
            else:
                theta4 = atan2( R3_6[2,2],-R3_6[0,2])
                theta6 = atan2(-R3_6[1,1], R3_6[1,0])

# It will avoid wrist flips
```

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.

The file `IK_server.py` is contained in the zip of the project. And well commented in my opinion, but the advice and suggestions are well received.

To see a test of the code enter here: https://www.youtube.com/watch?v=BVgrTOQ0oO0


Welcome to pick-place project!
Forward Kinematics
QUIRCH!
ING. MIGUEL C.
ThinkPad
SAMSUNG