Project: Kinematics pick & place

Kostas Oreopoulos

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Rubric Point 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.**

A — You are reading it

1 Kinematic Analysis

Rubric Point 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.**

A — Based on the kr210.urdf.xacro file, the forward kinematics demo and the joint diagram of Kuka KR210 as presented in lessons and following the same numbing of joints I derived the Denavit–Hartenberg parameters as show in table 1.

Table 1: Denavit – Hartenberg parameters for the Kuka KR210 arm robot

i	$\boldsymbol{\alpha}_{i-1}$	$\mathbf{a}_{i-1}$	$\mathbf{d}_i$	$\boldsymbol{\theta}_i$
1	0	0	0.75	θ_1
2	$-\pi/2$	0.35	0	θ_2
3	0	1.25	0	θ_3
4	$-\pi/2$	-0.054	1.5	θ_4
5	$-\pi/2$	0	0	θ_5
6	$-\pi/2$	0	0	θ_6
7	$-\pi/2$	0	0.303	0

Rubric Point 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.**

A — To derive the individual transformation matrices about each joint I created a small python function that takes the DH parameters for each transformation and generates the appropriate matrix. I used the code in listing 1.

Also to account for the different orientation of the frames in DH parameters and the frame as defined in the URDF file for the end effector I calculated a correction matrix that is simply an 180 degrees rotation on the z–axis, followed by one around y axis by -90 degrees. The code is available in listing 2

Listing 1: Transformation Matrix based on DH parameters

```
def dh_transformation(theta_x, d_dz, theta_z, d_dx):
    """
    Calculate the transformation matrix based on the
    Denavit - Hartenberg parameters
    :param  theta_x: α dh parameter. rotation (twist) around x axis
    :param  theta_z:  θ dh parameter. rotation around z axis
    :param  d_dz: a dh parameter. length of x_i-1 (common normal)
                  between z_i-1 and z_i
    :param  d_dx: d dh parameter. displacement of x axis
                  measured along z axis
    :return:
    """
    return Matrix([[cos(theta_z), -sin(theta_z), 0, d_dz],
                   [sin(theta_z) * cos(theta_x),
                    cos(theta_z) * cos(theta_x),
                    -sin(theta_x),
                    -sin(theta_x) * d_dx],
                   [sin(theta_z) * sin(theta_x),
                    cos(theta_z) * sin(theta_x),
                    cos(theta_x),
                    cos(theta_x) * d_dx],
                   [0, 0, 0, 1]])
```

Listing 2: Correction Matrix

```
def correction_matrix():
"""
A matrix to transform the end effector frame such as to align
with the URDF specs
"""
r_z = Matrix([[cos(pi), -sin(pi), 0, 0],
[sin(pi), cos(pi), 0, 0],
[0, 0, 1, 0],
[0, 0, 0, 1]])
r_y = Matrix([[cos(-pi / 2), 0, sin(-pi / 2), 0],
[0, 1, 0, 0],
[-sin(-pi / 2), 0, cos(-pi / 2), 0],
[0, 0, 0, 1]])
return simplify(r_z * r_y)
```

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

A — By design the end effector of the robot arm is rigidly attached to the robots wrist center, extending by 0.303m. That means that the orientation of the end effector is the same as the one of the wrist center, which means that $R_7^6 = Identity$ and $R_7^3 = R_6^3$.
We can also calculate the position of the wrist center by the equation $\vec{WC} = \vec{EE} - R_6^0 * (0.303, 0, 0)^T$, where all quantities are expressed in the 0th frame.

Our next step is to calculate R_3^0, which will by the equation $R_6^3 = (R_3^0)^{-1} * R_6^0 = (R_3^0)^T * R_6^0$, will also give us R_6^3. From that step and using what we learned in Lesson 2.18 we can extract the Euler angles of the wrist center, concluding our Inverse Kinematics task.

To find the first three angles of the IK problem we will use figure 1. In it, Point A is joint 2, point B is joint 3 and point C is joint 4. The following equations hold

$$\gamma = pi/2 - theta_3 + \text{initial angle} \tag{1}$$

$$\text{initial angle} = \arctan(-0.054, 1.5) \tag{2}$$

$$\cos(\gamma) = \frac{\left(|\vec{AC}|^2 - |\vec{AB}|^2 - |\vec{BC}|^2\right)}{-2|\vec{AB}||\vec{BC}|} \tag{3}$$

$$\gamma = \arctan(\sqrt{1 - cos^2(\gamma)}, cos(\gamma)) \tag{4}$$

From 1 and 4 we get $theta_3$. Next we find $theta_2$ which is α in the figure. We also note that angles $E\hat{A}H$ and $E\hat{C}B$ are equal because they have their corresponding sides perpendicular. By that fact the following equations hold:

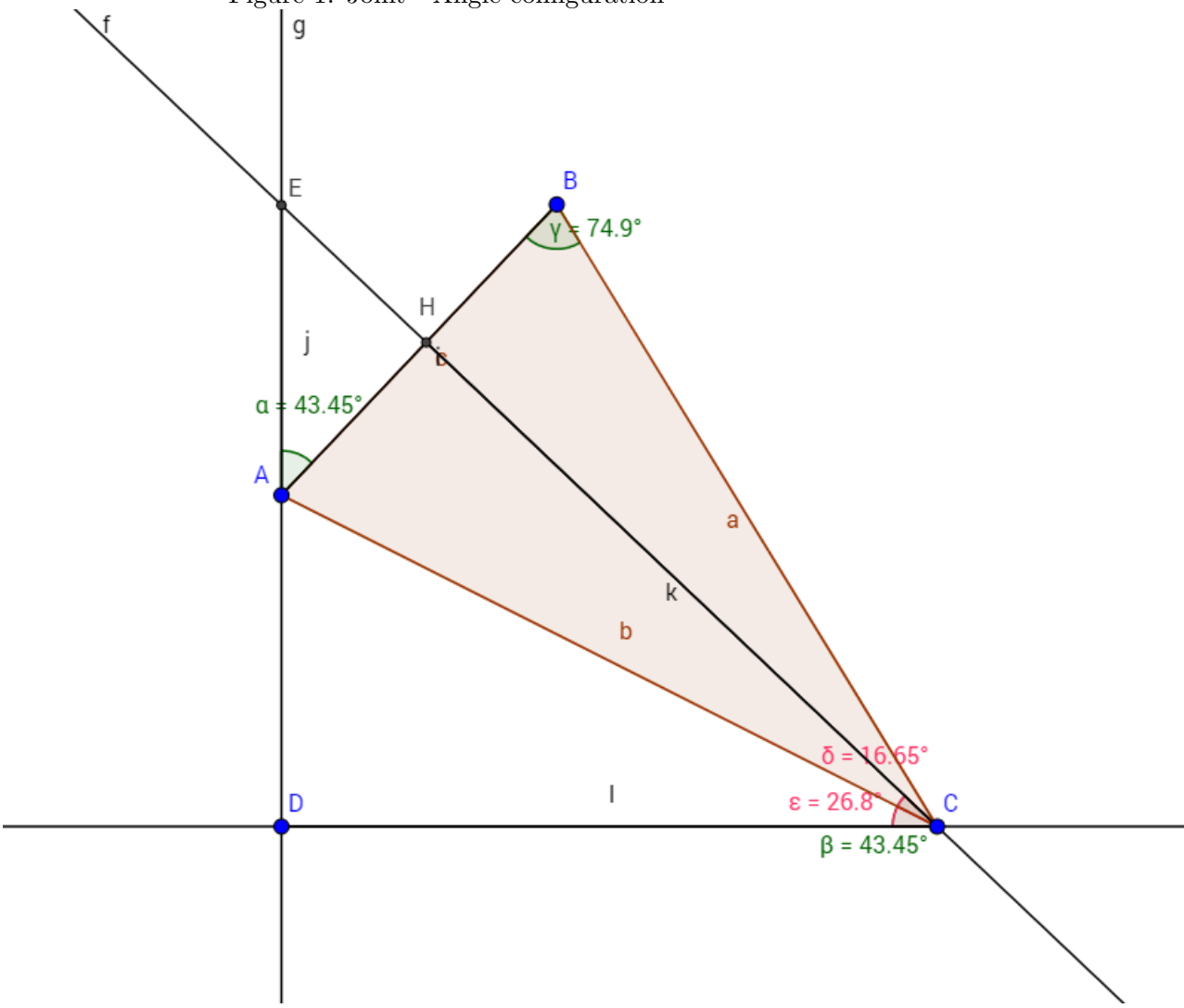
$$\alpha = theta_3 = \delta + \epsilon \tag{5}$$

$$\epsilon = \arctan(HB, HC) = \arctan(AB - \overbrace{\cos(\gamma)BC}^{HA}, \sin(\gamma)BC) \tag{6}$$

$$\delta = \arctan(AD, DC) \tag{7}$$

We conclude by calculating $theta_1$ which is the easiest of all, since $theta_3 = \arctan(\text{wrist\_center}_y, \text{wrist\_center}_x)$.

Figure 1: Joint - Angle configuration



2 Project Implementation

Rubric Point 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.**

A — The implementation of the python code is a straightforward application of the above formulas. The code does the following:

- Sets up all transformation matrices.
- Calculates a correction matrix for the misalignment of EE frame and URDF file frame
- Calculates the t_FK transform which is the final transform between frame 0 an EE frame.
- Uses all formulas discussed above, in the inverse_kinematics function and calculates angles theta_1 to theta_6
- All the above are done symbolically and once when the program begins.
- Once a pose is received, function *calculateFromPose* applies the values to the closed form formulas and returns the requests angles
- As a last step, forward kinematics are applied to the calculated angles and the point predicted is compared with the ground truth (pose data) and an error is calculated per pose and stored in an array. At the end of a batch of poses, the average error is calculated and reported.
- The code performs 8 or 9 out of 10 tries and when it misses, it is because the arm throws the cylinder of the self (something that probably has more to do with path planning than calculation of the inverse kinematics)