

Rubric Points

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

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.

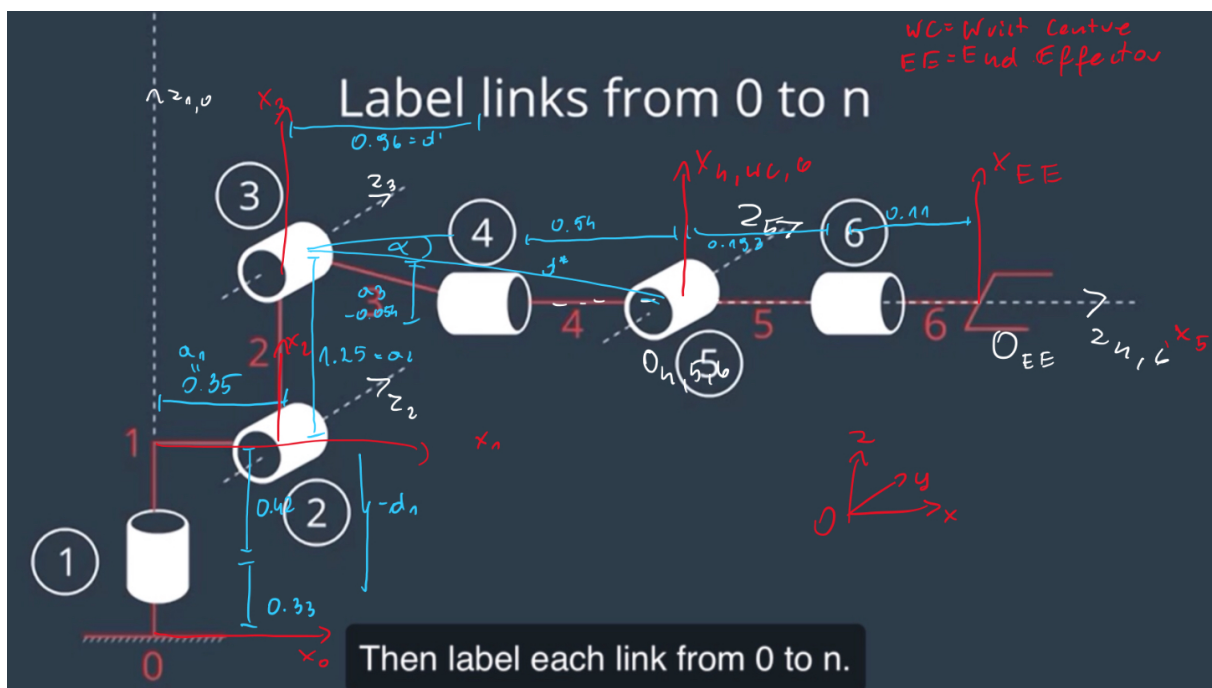
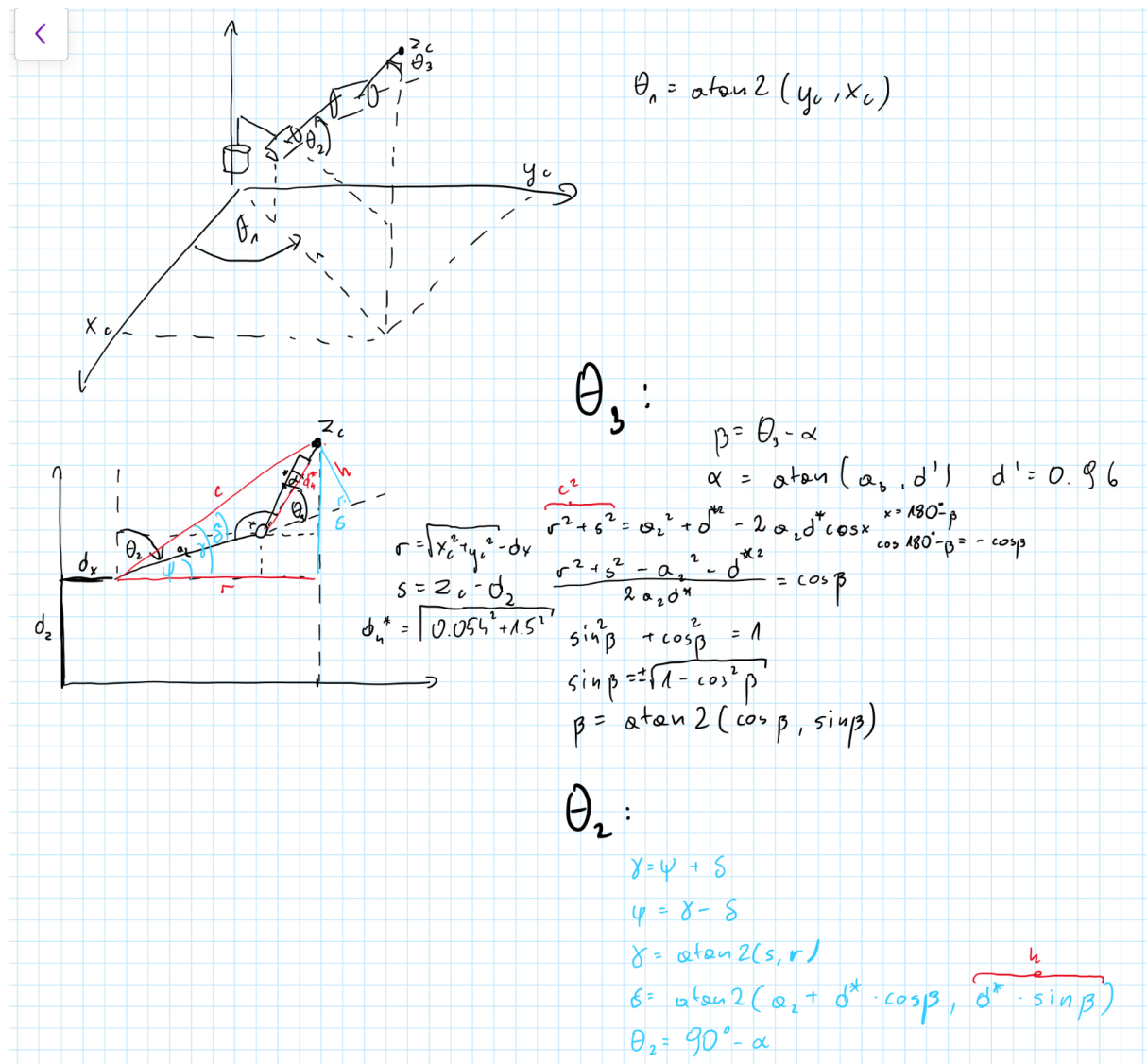


Figure 1: Link and origin frames' assignments

Links	$\alpha(i-1)$	$a(i-1)$	$d(i-1)$	$\theta(i)$
T_1^0	0	0	0.75	q_1
T_2^1	$-\pi/2$	0.35	0	$-\pi/2 + q_2$
T_3^2	0	1.25	0	q_3
T_4^3	$-\pi/2$	-0.054	1.5	q_4
T_5^4	$\pi/2$	0	0	q_5
T_6^5	$-\pi/2$	0	0	q_6
T_E^6	0	0	0.303	0

$$\begin{aligned}
T_1^0 &= \begin{bmatrix} \cos(q_1) & -\sin(q_1) & 0 & 0 \\ \sin(q_1) & \cos(q_1) & 0 & 0 \\ 0 & 0 & 1 & 0.75 \\ 0 & 0 & 0 & 1 \end{bmatrix} & T_2^1 &= \begin{bmatrix} \sin(q_2) & \cos(q_2) & 0 & 0.35 \\ 0 & 0 & 1 & 0 \\ \cos(q_2) & -\sin(q_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
T_3^2 &= \begin{bmatrix} \cos(q_3) & -\sin(q_3) & 0 & 1.25 \\ \sin(q_3) & \cos(q_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & T_4^3 &= \begin{bmatrix} \cos(q_4) & -\sin(q_4) & 0 & -0.054 \\ 0 & 0 & 1 & 1.5 \\ -\sin(q_4) & -\cos(q_4) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
T_5^4 &= \begin{bmatrix} \cos(q_5) & -\sin(q_5) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin(q_5) & \cos(q_5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & T_6^5 &= \begin{bmatrix} \cos(q_6) & -\sin(q_6) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin(q_6) & -\cos(q_6) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
T_{EE}^6 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.303 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

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

Figure 2: Solution for θ_{1-3}

θ_{4-6} is computed by solving $R_{EE}^3 = (R_3^0)^{-1} \cdot R_{EE}^0 = (R_3^0)^T \cdot R_{EE}^0$. Since

$$R_{EE}^3 = \begin{bmatrix} -\sin(q_4) \sin(q_6) + \cos(q_4) \cos(q_5) \cos(q_6) & -\sin(q_4) \cos(q_6) - \sin(q_6) \cos(q_4) \cos(q_5) & -\sin(q_5) \cos(q_4) \\ \sin(q_5) \cos(q_6) & -\sin(q_5) \sin(q_6) & \cos(q_5) \\ -\sin(q_4) \cos(q_5) \cos(q_6) - \sin(q_6) \cos(q_4) & \sin(q_4) \sin(q_6) \cos(q_5) - \cos(q_4) \cos(q_6) & \sin(q_4) \sin(q_5) \end{bmatrix}$$

and $(R_3^0)^T \cdot R_{EE}^0$ can be computed from input and θ_{1-3} we have:

```
1 cosq5 = R3_EE[1, 2]
2 for t5sgn in [1, -1]:
3     sinq5 = t5sgn * sqrt(1 - cosq5 ** 2)
```

```
4  theta5 = atan2(sinq5, cosq5)
5
6  sinq4 = t5sgn * R3_EE[2, 2]
7  cosq4 = -t5sgn * R3_EE[0, 2]
8  theta4 = atan2(sinq4, cosq4)
9
10 sinq6 = -t5sgn * R3_EE[1, 1]
11 cosq6 = t5sgn * R3_EE[1, 0]
12 theta6 = atan2(sinq6, cosq6)
```

Mutliple solutions

Both $\sin \beta$ from figure 2 and $\sin \theta_5$ can have more than one solution. Additionally each joint can be rotated by $\pm 360^\circ$, as long as it meets the joint specifications. In my program I consider all possible solutions and choose the most cost-effective configuration with regard to transtion time between joint poses. The algorithm is explained in detail in the next section.

Project Implementation

I've implemented the solution in `inverse_kinematics.py` in the same directory. For dealing with multiple solutions (regarding the two possible solutions for θ_3 , θ_5 or rotation by $\pm 360^\circ$) I compute them all (`SolveIK`). Then I discard the solutions that are impossible because of joint constraints specified in the urdf file (`SolveIKLimits`). Later I compute the cost of each solution by taking the maximum time that joint would travel from a previous position to a new position using joint velocities specified in the urdf model. I select the solution with the smallest cost.

Without the cost computation, if any solution is selected, the arm would often just rotate the last few joints in place to an alternative configuration that yields the same EE poision. I've found that using cost and computing maximum instead of a sum in the cost computation improves the solution greatly, since the joints can operate in parallel. Even though $-\sin(\beta)$ would not be very useful in the given world configuration, I've found that the cost model would be able to efficiently use it from time to time.

This is not perfect, as the robot would still perform some unnecessary rotations in place, perhaps a model involving more configurations than just the current configuration or a heuristic ranking the solutions would perform better.

My code always assumes there is a solution, I've not worked the cases where the EE pos would be out of range or the computations would lead to a division by zero.

Sidenotes

I've used a quaternion to a rotation matrix implementation I've found online to be able to work on the solution on a computer without ROS installed.

I've found Jupyter notebook's interactive widgets useful for testing forward kinematics (see "computations.ipynb" Forward Kinematics Playground).

Forward Kinematics Playground

```
In [4]: T0_1,T1_2,T2_3,T3_4,T4_5,T5_6,T6_EE = ik.DHMatrices()
def e(q11, q22, q33, q44, q55, q66, joints):
    from kuka_arm.scripts.inverse_kinematics import q1,q2,q3,q4,q5,q6
    mat = reduce(lambda acc, M: acc * M, [T1_2,T2_3,T3_4,T4_5,T5_6,T6_EE][::(joints-1)], T0_1)
    return pprint(mat.evalf(subs={q1: q11, q2: q22, q3: q33, q4: q44, q5: q55, q6: q66}, prec = 5))

spec = (-3.14,3.14,0.01)
widgets.interact(e, q11=spec, q22=spec, q33=spec, q44=spec, q55=spec, q66=spec, joints=(2,7))
```

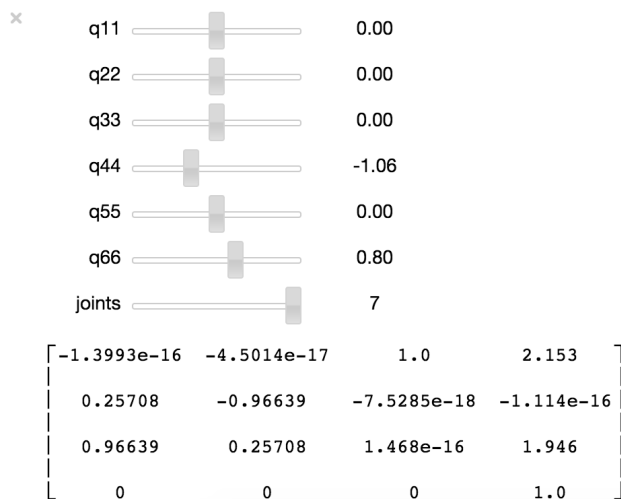


Figure 3: playground

The error between desired and computed EE position is small, less than 0.00001 on average. See the last section of [computations.ipynb](#) for code.