

Lab 1: Forward Kinematics and Planar Inverse Kinematics

Lab Report

Methods

3. Forward Kinematics

In the first part of Lab1, we are expected to implement the forward kinematics of the Panda robot arm. Our goal is to work out functions of different joint angles and lead to the final pose of the end effector as well as the positions of each joint, all relative to the base world frame we defined. Once implemented, we are supposed to obtain position of any joint and describe the status of the end effector in a based world frame with different robot configurations given to us.

3.2 Formulation

Before implementing the frame convention in code, we need to formulate our plans to create intermediate frame on various links of the robot. We start off by choosing the Denavit-Hartenberg Convention strategy and setting up link frames, shown in Figure 3.2.1. In each frame, we followed the steps of ‘define z-axis (rotation axis)’- ‘create x-axis by observing z_i and z_{i-1} ’- ‘finalize y-axis using Right Hand Rule’.

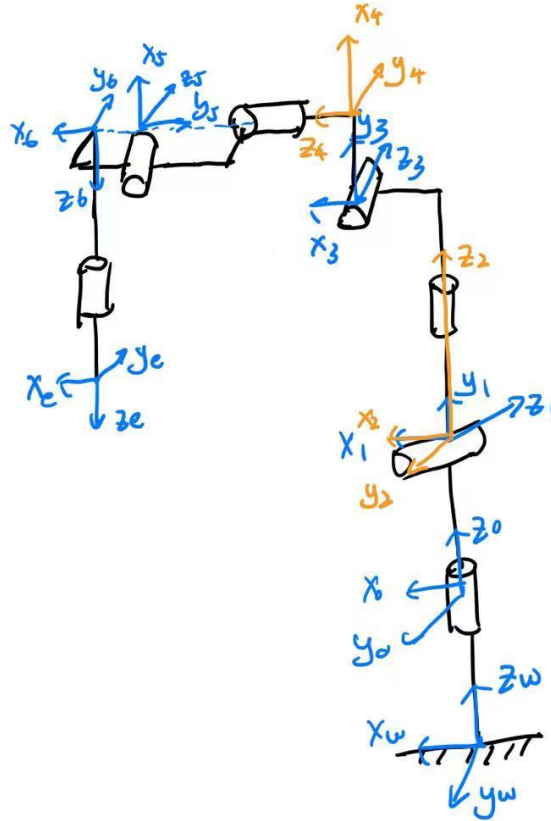


Figure 3.2.1 Coordinate Frames

To compute the transformation between two consecutive pair of frames, we need to work out transformation matrix, A_n for each pair. The matrix required four different parameters between each transformation, $a_i, \alpha_i, d_i, \theta_i$ and one of them is the unknown input, q_n . In this case, θ_i is the unknown parameter. Figure 3.2.2 shows the table of such four parameters in each transformation.

	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0.192	θ_1
2	0	$-\pi/2$	0	$-\theta_2$
3	0.082	$\pi/2$	0.316	θ_3
4	0.083	$\pi/2$	0	$\theta_4 + \pi$
5	0	$-\pi/2$	0.384	θ_5
6	0.088	$\pi/2$	0	θ_6
7	0	0	0.21	$\theta_7 - \frac{\pi}{4}$

Figure 3.2.2 Transformation parameters

With the data in the table above, we could put them into the general form of DH convention matrix, which is:

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.2.3 shows the final transformation matrices between two consecutive pairs of coordinate frames in code after plugging in the DH convention matrix:

```
AW0 = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0.141],[0,0,0,1]])
A01 = np.array([[math.cos(q[0]), 0, math.sin(q[0]), 0], [math.sin(q[0]), 0, -math.cos(q[0]), 0], [0, 1, 0, 0.192], [0, 0, 0, 1]])
A12 = np.array([[math.cos(-q[1]), 0, -math.sin(-q[1]), 0], [math.sin(-q[1]), 0, math.cos(-q[1]), 0], [0, -1, 0, 0], [0, 0, 0, 1]])
A23 = np.array([[math.cos(q[2]), 0, math.sin(q[2]), 0.082*math.cos(q[2])], [math.sin(q[2]), 0, -math.cos(q[2]), 0.082*math.sin(q[2])], [0, 1, 0, 0.316], [0, 0, 0, 1]])
A34 = np.array([[math.cos(q[3]+pi), 0, math.sin(q[3]+pi), 0.083*math.cos(q[3]+pi)], [math.sin(q[3]+pi), 0, -math.cos(q[3]+pi), 0.083*math.sin(q[3]+pi)], [0, 1, 0, 0], [0, 0, 0, 1]])
A45 = np.array([[math.cos(q[4]), 0, -math.sin(q[4]), 0], [math.sin(q[4]), 0, math.cos(q[4]), 0], [0, -1, 0, 0.384], [0, 0, 0, 1]])
A56 = np.array([[math.cos(q[5]-pi), 0, math.sin(q[5]-pi), 0.088*math.cos(q[5]-pi)], [math.sin(q[5]-pi), 0, -math.cos(q[5]-pi), 0.088*math.sin(q[5]-pi)], [0, 1, 0, 0], [0, 0, 0, 1]])
A67 = np.array([[math.cos(q[6]-pi/4), -math.sin(q[6]-pi/4), 0, 0], [math.sin(q[6]-pi/4), math.cos(q[6]-pi/4), 0, 0], [0, 0, 1, 0.21], [0, 0, 0, 1]])
```

Figure 3.2.3 Transformation matrices of each consecutive frames

With the matrices above, we could easily compute transformation between any link and the base frame with the equation:

$$T_n^0 = A_1(q_0) \cdot A_2(q_1) \cdot A_3(q_2) \cdot \dots \cdot A_n(q_{n-1})$$

To compute the full pose of the end effector as well as the position of each joint, we could make use of the transformation matrices we just obtained. For example, we set $o_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ as the origin of the base frame of my system, then we could get position of joint 1,

$$o_1 = A_1(q_0) \cdot o_0 = \begin{bmatrix} \cos q_0 & -\sin q_0 & 0 & 0 \\ \sin q_0 & \cos q_0 & 0 & 0 \\ 0 & 0 & 1 & 0.141 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.141 \\ 1 \end{bmatrix}$$

With such methodology, we could then solve the positions of the rest joints. Some general equals and steps are also included in Figure 3.2.4.

```

T0 = np.matmul(Aw0, A01)
T1 = np.matmul(T0, A12)
T2 = np.matmul(T1, A23)
T3 = np.matmul(T2, A34)
T4 = np.matmul(T3, A45)
T5 = np.matmul(T4, A56)
T6 = np.matmul(T5, A67)
Tee = T6

P1 = np.array([0,0,0.141,1])
P2 = np.matmul(T0, np.array([0,0,0,1]))
P3 = np.matmul(T1, np.array([0,0,0.195,1]))
P4 = np.matmul(T2, np.array([0,0,0,1]))
P5 = np.matmul(T3, np.array([0,0,0.125,1]))
P6 = np.matmul(T4, np.array([0,0,-0.015,1]))
P7 = np.matmul(T5, np.array([0,0,0.851,1]))
jointPositions = np.array([[P1[0], P1[1], P1[2]],\
[P2[0], P2[1], P2[2]],\
[P3[0], P3[1], P3[2]],\
[P4[0], P4[1], P4[2]],\
[P5[0], P5[1], P5[2]],\
[P6[0], P6[1], P6[2]],\
[P7[0], P7[1], P7[2]]])

```

Figure 3.2.4 Joint positions

By multiplying the DH matrices using post multiplication with world to zero frame homogeneous transformation matrix using pre multiplication, we can get the homogeneous transformation matrix from each intermediate frame to world frame respectively. Then, we can get each joint position by multiplying the homogeneous transformation matrix between their related intermediate frame and the world frame with their transition vector.

3.4 Testing

After implementing the above calculation into calculateFK.py, the visualize.py script was executed to determine whether the algorithm is correct or not. The result shows that the joint indicators all stay in their expected position, which means the algorithm works.

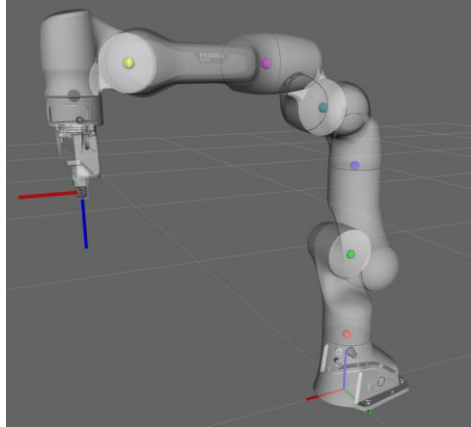
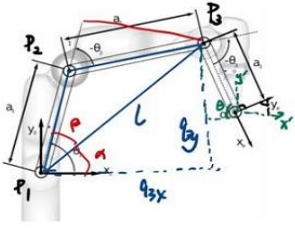


Figure 3.4 Visualization

4. Planar Inverse Kinematics

In the second part of Lab1, we will need to conduct an inverse kinematics method for a simplified abstract RRR robot arm similar in the prelab. Before implementing such method into code, we conducted some numerical transformation with some given information, T_x, T_y , which are the location of the end effector, as well as the angle θ , which is the total angle of rotation of the frame relative to the base frame.

Figure 4.2.1 shows general steps of converting the given data into our targets, q . First, we will have to add an auxiliary line from $p1$ to $p3$ to form a triangle shape, $p1p2p3$. Within such triangle, we will utilize the Law of cosine to



law of cosines: $c^2 = a^2 + b^2 - 2ab \cos \gamma$

$$l_2^2 = a_1^2 + a_2^2 - 2a_1 a_2 \cos(\pi + q_2)$$

$$q_2^x^2 + q_2^y^2 = a_1^2 + a_2^2 - 2a_1 a_2 \cos(\pi + q_2)$$

$$q_2^x^2 + q_2^y^2 = a_1^2 + a_2^2 + 2a_1 a_2 \cos q_2$$

$$\cos q_2 = \frac{q_2^x^2 + q_2^y^2 - a_1^2 - a_2^2}{2a_1 a_2}$$

$$q_2 = \cos^{-1} \left(\frac{q_2^x^2 + q_2^y^2 - a_1^2 - a_2^2}{2a_1 a_2} \right)$$

$\therefore \theta$ is currently negative in diagram
between world frame and end effector frame

$$\theta = q_1 - q_2 - q_3$$

$$\therefore q_3 = q_1 - q_2 - \theta$$

Given t_x, t_y, θ

$$\begin{aligned} q_2^x &= t_x - a_1^2 \cdot \cos \theta \\ q_2^y &= t_y - a_1^2 \cdot \sin \theta \end{aligned} \quad \left\{ \begin{array}{l} \text{from diagram} \end{array} \right.$$

$$\begin{aligned} q_1 &= q + \beta \\ &= \arctan\left(\frac{q_2^y}{q_2^x}\right) + \arctan\left(\frac{a_2 \cdot \sin(-q_2)}{a_1 + a_2 \cos(q_2)}\right) \\ &= \arctan\left(\frac{q_2^y}{q_2^x}\right) - \arctan\left(\frac{a_2 \sin q_2}{a_1 + a_2 \cos q_2}\right) \end{aligned}$$

Figure 4.2.1 Numerical transformation of IK

First, we will have to add an auxiliary line from point 1 to point 3 to form a triangle shape, $p_1p_2p_3$. Within such triangle, we will utilize the Law of cosine to rewrite q_2 with some parameters that we already knew. It is clear in the figure that we still need the location information of p_3 to fully reorganize the expression of q_2 , thus on the left-hand side, we derived the x and y location of point 3 as a function of the location of the end effector, T_x, T_y . In this case, we concluded a general expression for q_2 . Then we turned back to q_1 . Notice that by adding the line between point 1 and point 2, we divided the angle of q_1 into two parts, α and β . Since α is the included angle between L and q_3^3 , both of which are known, we could easily get $\alpha = \arctan\left(\frac{q_3^y}{q_3^x}\right)$. By using the same method of creating right triangle, in which β becomes the included angle, we can derive the function for β as well. Then $q_1 = \alpha + \beta$.

Since the resultant rotation angle of q_1, q_2 and q_3 are given as θ , and due to the positive/negative characteristic of $\theta_1, \theta_2, \theta_3$, we concluded that the function for q_3 :

$$q_3 = q_1 - q_2 - \theta$$

With θ given and q_1, q_2 derived earlier.

4.3 Implementation

After deriving the IK solution using the approach explained above, the equations were implemented in PlanarIK.py as `rrr_abstract_ik` function for automating finding each q value. The function takes in the expected coordinate of the end effector and the angle between the world frame and the end effector frame, and output each joint variable q . Below is the implementation in code:

```
t_x = target['o'][0]
t_y = target['o'][1]
t_theta = target['theta']

q3x = t_x - a3 * np.cos(t_theta)
q3y = t_y - a3 * np.sin(t_theta)

q2_a = -np.arccos((q3x**2 + q3y**2 - a1**2 - a2**2)/(2*a1*a2))
q2_b = np.arccos((q3y**2 + q3x**2 - a1**2 - a2**2)/((2*a1*a2)))
q1_a = np.arctan2(q3y, q3x) - np.arctan2((a2*np.sin(q2_a)), (a1+a2*np.cos(q2_a)))
q1_b = np.arctan2(q3y, q3x) - np.arctan2((a2*np.sin(q2_b)), (a1+a2*np.cos(q2_b)))
q3_a = t_theta - q1_a - q2_a
q3_b = t_theta - q1_b - q2_b
```

Figure 4.3 IK Solution in code

4.4 Testing

The accuracy of the function implemented in PlanarIK.py can be verified using the FK solution we developed in previous section. Since in previous section we already tested that the FK solution can work properly, we can use the output of the IK function as input of the FK function. By observing whether the indicators overlap with each joint or not, we can identify whether the IK solution works or not. The picture below is the visualized result. We can see our IK solution can effectively calculate the joint configuration.

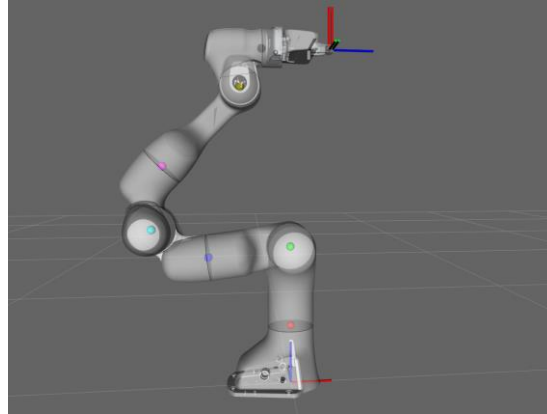


Figure 4.4 IK Verification

Analysis

5.3.1 Gravity

The gravity is disabled in the simulator by default. With the gravity disabled, the result of the FK visualization shows in the picture below:

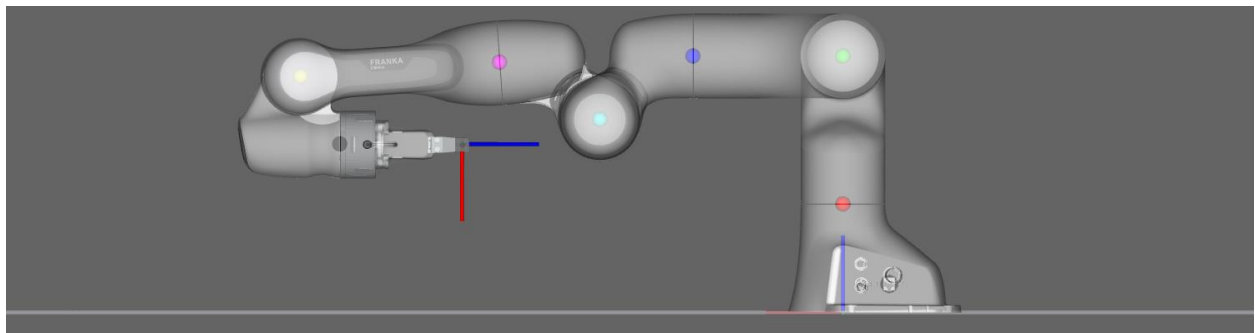


Figure 5.3.1.1 FK Visualization (Gravity Disabled)

The result of the IK visualization shows in the picture below:

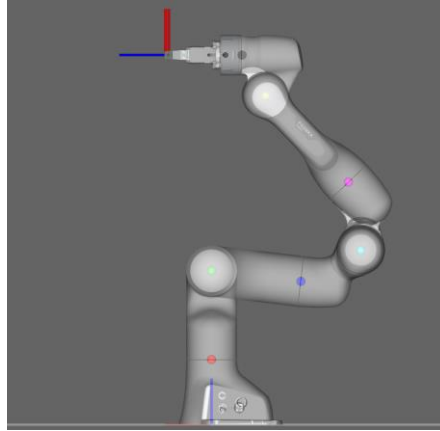


Figure 5.3.1.2 IK Visualization (Gravity Disabled)

If we enable gravity, the result of the FK visualization shows in the picture below:

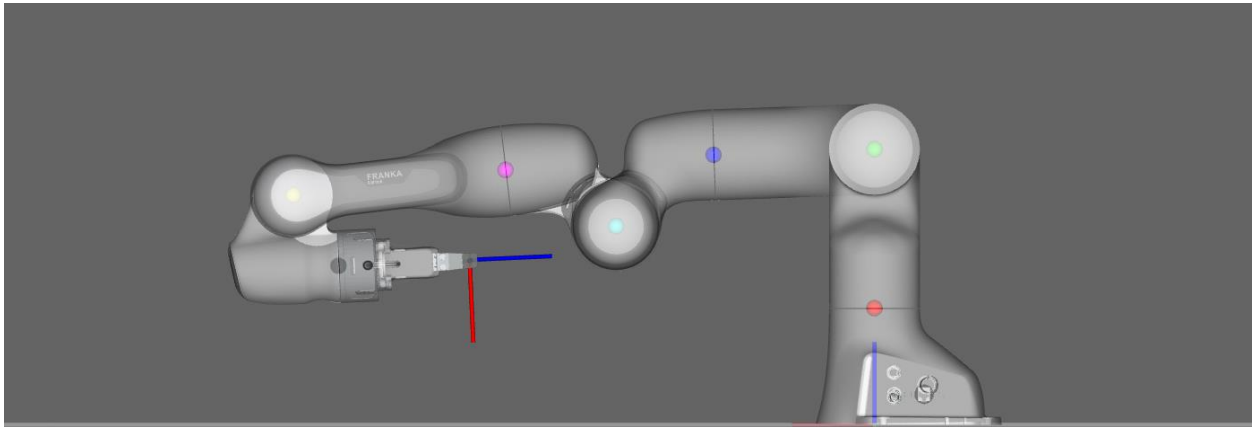


Figure 5.3.1.3 FK Visualization (Gravity Enabled)

The result of the IK visualization shows in the picture below:

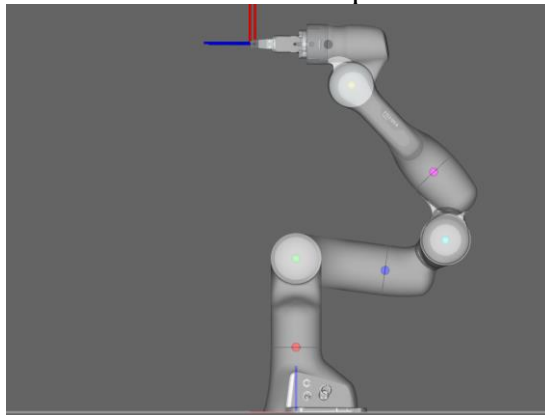


Figure 5.3.1.4 IK Visualization (Gravity Enabled)

After enabling gravity, we noticed that the arm became lower in the vertical axis, and the robot arm always shook after reaching the desired position. This might be due to gravity affecting the robot arm, making it hard to stabilize and creating uncertainty in FK and IK solutions.

5.3.2 Reachable Workspace

Since the sample points in the plot were generated using random.uniform for each element in the input array, the first assumption was that the points were evenly generated and the second assumption was the number of sampling is large enough, which was 1500 points in this case. The joints were already confined with the limitation dictionary provided within the code base, but the robot might not be able to locate its end effector at any point within the plot since the arm itself occupy some space. The physical volume of the robot may prevent the end effector reach some points in the plot. Another possibility of points that cannot reach might come from observation by eye. Even though the plot was divided into seven based on each joint, the overlapping of points still make some positions hard to distinguish. Therefore, some points in the plot might still be impossible to reach.

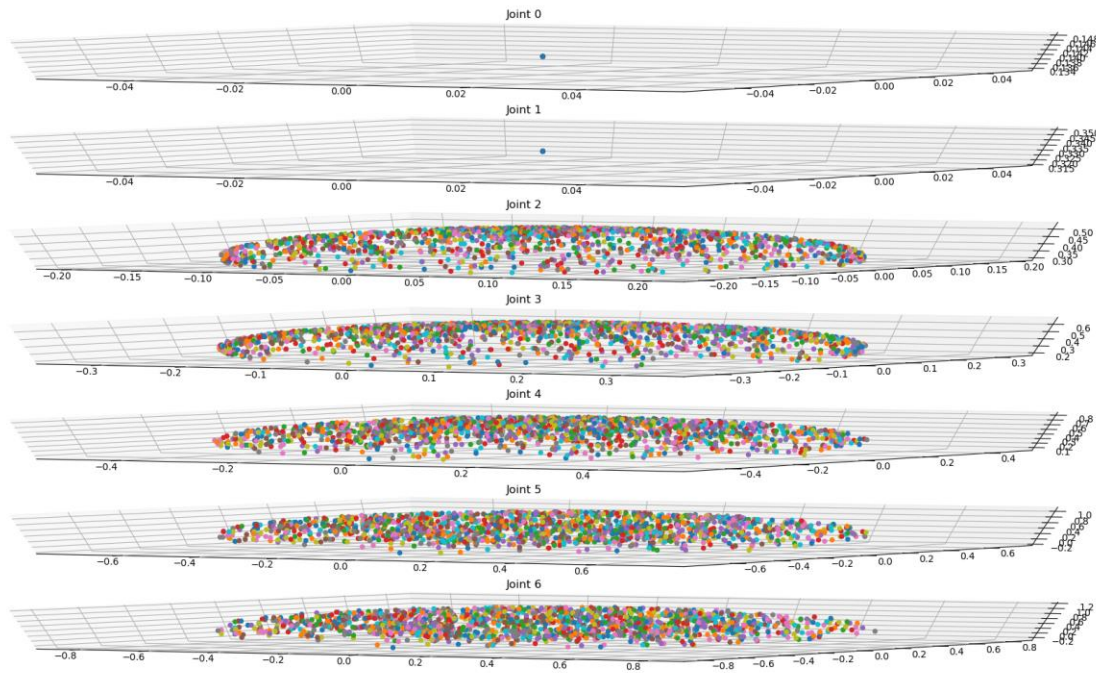


Figure 5.3.2

5.3.3 Extending Inverse Kinematics to 3D

In the simplified model of the robot arm in this lab, when implementing the IK method, we found it takes time to figure out the geometry between the angles, joints, and frames and the sometimes the resultant expression could be complicated. If we implement IK to a real 3D model just like Panda arm, we must consider whether it could provide some solid solutions. In Lab1 for instance, we could predict IK method will produce two sets of solutions, but when it turns into 3D, it was hard to make such predictions that how many potential answers the system might have, or there might not be any valid solution. Another problem is that if implementing the IK into 3D model, the workload of numerical transformation and computation could be huge, and it might be hard to conduct any testing process to verify the result.