KINEMATIC CHARACTERIZATION OF THE LYNX

MEAM 520 LAB-01

Dinesh Jagai
Department of Computer Science
University of Pennsylvania
Philadelphia, PA
dinesh97@seas.upenn.edu

Thomas Wang
Department of Computer Science
University of Pennsylvania
Philadelphia, PA
ttwang@seas.upenn.edu

Monday June 22nd 2020

1 Concepts

1.1 Forward Kinematics

1.2) A Drawing of the joint angles with the relevant dimensions is shown below:
Note that the center of the coordinate frame of joint 4 is the same as the center of the coordinate frame of joint 3. I drew it like that to avoid any confusion (this is reflected in the DH table and taken account when calculating the joint centers). Also, the circled numbers in green 1, .., 6 highlights the links 1, .., 6 as shown in the DH table.

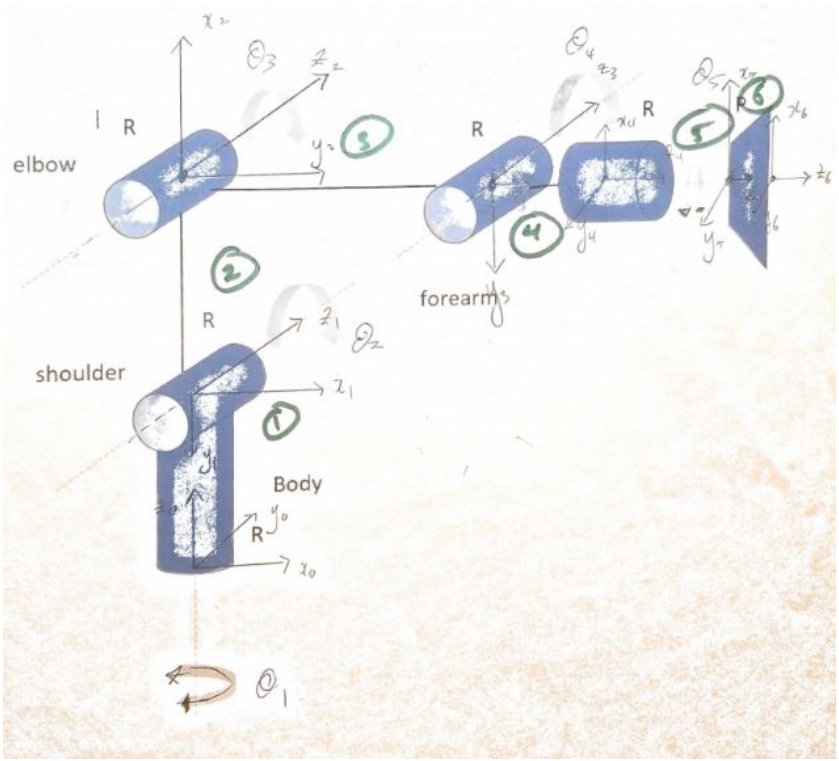


Figure 1.1.2: Figure with all the joint axes for the lynx labeled

1.3) Table showing the DH parameters based on our axes from figure 1.1.2

Link	a_i	α_i	d_i	θ_i
1	0	-90	d_1	θ_1
2	a_2	0	0	$\theta_2 - 90$
3	a_3	0	0	$\theta_3 + 90$
4	0	-90	0	$\theta_4 - 90$
5	0	0	d_5	θ_5
6	0	0	l_g	0

1.4) Denote T_i^j to be the transformation matrices for transforming between the coordinate frame i to the coordinate frame in j

$$T_1^0 = \begin{bmatrix} cos(\theta_1) & 0 & -sin(\theta_1) & 0 \\ sin(\theta_1) & 0 & -cos(\theta_1) & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^1 = \begin{bmatrix} sin(\theta_2) & cos(\theta_2) & 0 & a_2 sin(\theta_2) \\ -cos(\theta_2) & sin(\theta_2) & 0 & -a_2 cos(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^2 = \begin{bmatrix} -sin(\theta_3) & -cos(\theta_3) & 0 & -a_3 sin(\theta_3) \\ cos(\theta_3) & -sin(\theta_3) & 0 & a_3 cos(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^3 = \begin{bmatrix} sin(\theta_4) & 0 & cos(\theta_4) & 0 \\ -cos(\theta_4) & 0 & sin(\theta_4) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5^4 = \begin{bmatrix} cos(\theta_5) & -sin(\theta_5) & 0 & 0 \\ sin(\theta_5) & cos(\theta_5) & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_g \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.5)

$$T_6^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5$$

1.2 Inverse Kinematics

2.1) Now

$$T_5^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4$$

Using an online calculator, T_5^0 is given below:

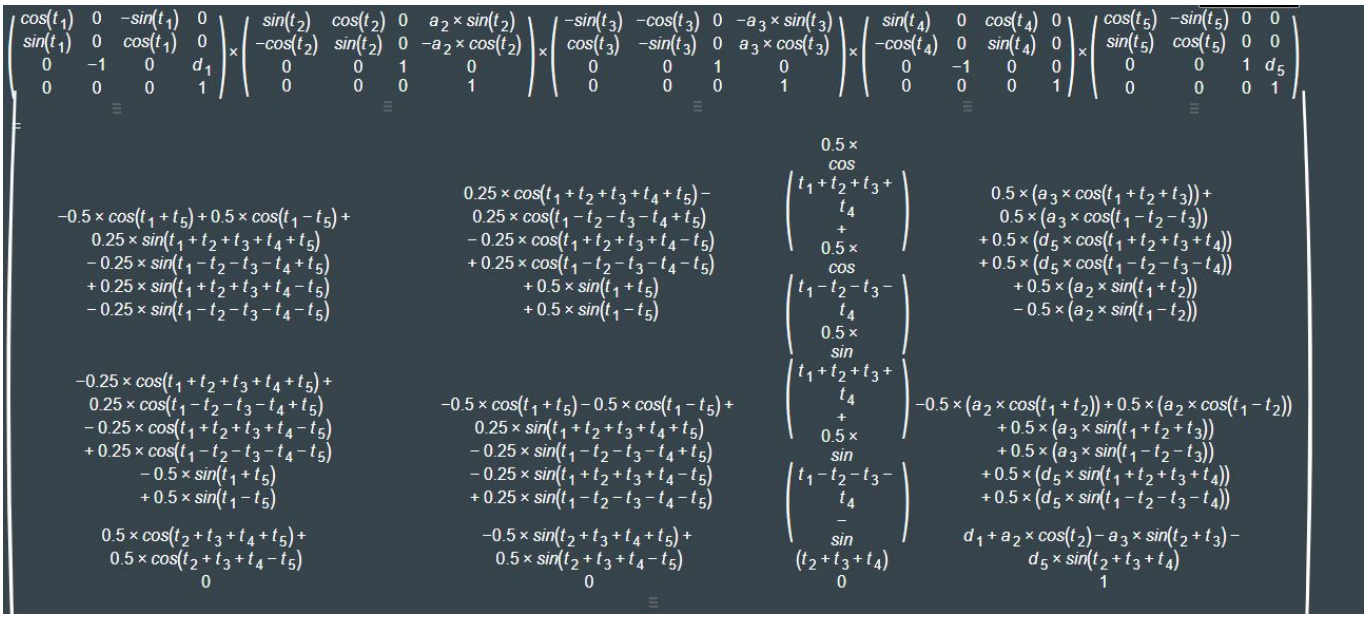


Figure 1.2.1: Figure Showing T_5^0

Note t_i is used here to represent Θ_i From the above matrix we shall derived equations for Θ_i in terms of x, y, z
The equations are shown in the pdf inserted below.
Note that the values for $t_i, i \in [2..5]$ depend on the values in our T_5^0 matrix

Inverse Kinematics

From the homogeneous transformation, T_5^0, we have:

> $N^1B\ T_6^0$ is merely $d_5 + l_g$ in the last col. since it's a translation of l_g.

$$2x = a_2\sin(t_1+t_2) - a_2\sin(t_1-t_2) + a_3\cos(t_1+t_2+t_3) + a_3\cos(t_1-t_2-t_3) + d_5\cos(t_1+t_2+t_3+t_4) + d_5\cos(t_1-t_2-t_3-t_4) \quad -(1)$$

$$2y = -a_2\cos(t_1+t_2) + a_2\cos(t_1-t_2) + a_3\sin(t_1+t_2+t_3) + a_3\sin(t_1-t_2-t_3) + d_5\sin(t_1+t_2+t_3+t_4) - d_5\sin(t_1-t_2-t_3-t_4) \quad -(2)$$

$$z = d_1 + a_2\cos(t_2) - a_3\sin(t_2+t_3) - d_5\sin(t_2+t_3+t_4) \quad -(3)$$

$$y = x\tan(t_1) \quad -(4) \quad \{\text{from Geometry}\}$$

from eq (1) & eq (2), we have:

$$2x = a_2\,2\cos(2t_1/2)\sin(2t_2/2) + a_3\,2\left(\cos(2t/2)\cos\left(\frac{2(t_2+t_3)}{2}\right)\right) + d_5\left(2\cos\left(\frac{2t_1}{2}\right)\cos\left(\frac{2(t_2+t_3+t_4)}{2}\right)\right)$$

$$2y = a_2\,2\sin\left(\frac{2t_1}{2}\right)\sin(2t_2/2) + a_3\sin(2t_1/2)\cos\left(\frac{2(t_2+t_3)}{2}\right) + 2d_5\left(\sin\left(\frac{2t_1}{2}\right)\cos\left(\frac{2(t_2+t_3+t_4)}{2}\right)\right)$$

$$\Rightarrow x = a_2\cos(t_1)\sin(t_2) + a_3\cos(t_1)\cos(t_2+t_3) + d_5\cos(t_1)\cos(t_2+t_3+t_4) \quad -(5)$$

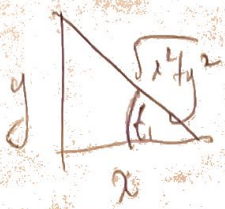
$$\Rightarrow y = a_2\sin(t_1)\sin(t_2) + a_3\sin(t_1)\cos(t_2+t_3) + d_5\sin(t_1)\cos(t_2+t_3+t_4) \quad -(6)$$

eq (6) / eq (5) $\Rightarrow \tan t_1 = y/x =$ eq (4). i.e. $\theta_1 = t_1 = \text{atan2}(y/x)$

If $x = y = 0$ then t_1 will have no value since any value can work.

Now, consider $x\cos(t_1) + y\sin(t_1) = a_2\sin(t_2) + a_3\cos(t_2+t_3) + d_5\cos(t_2+t_3+t_4)$ $-(7)$

Also, $z - d_1 + d_5\sin(t_2+t_3+t_4) = a_2\cos(t_2) - a_3\sin(t_2+t_3)$ $-(8)$



using $\cos(t_i)$ & $\sin(t_i) = c_i$ & s_i respectively, we have:

$$x c_1 + y s_1 = a_2 s_2 + a_3 c_{23} + d_5 c_{234}$$

$$z - d_1 + d_5 s_{234} = a_2 c_2 - a_3 s_{23}$$

$$\Rightarrow \quad x c_1 + y s_1 - d_5 c_{234} = a_2 s_2 + a_3 c_{23}$$

$$z - d_1 + d_5 s_{234} = a_2 c_2 - a_3 s_{23}$$

$$\Rightarrow \quad \left.\begin{aligned} x c_1 + y s_1 &= a_2 s_2 + a_3 c_{23} + d_5 c_{234} \\ z - d_1 &= a_2 c_2 - a_3 s_{23} - d_5 s_{234} \end{aligned}\right\} \text{planar RRR}$$

$$\frac{x^2}{\sqrt{x^2+y^2}} + \frac{y^2}{\sqrt{x^2+y^2}} = a_2 s_2 + a_3 c_{23} + d_5 c_{234} \quad -(7)$$

$$z - d_1 = a_2 c_2 - a_3 s_{23} - d_5 s_{234} \quad -(8)$$

$$\Rightarrow \quad \sqrt{x^2+y^2} = a_2 s_2 + a_3 c_{23} + d_5 c_{234} \quad -(7)$$

$$z - d_1 = a_2 c_2 - a_3 s_{23} - d_5 s_{234} \quad -(8)$$

$$\left.\begin{aligned} \sqrt{x^2+y^2} - d_5 c_{234} &= a_2 s_2 + a_3 c_{23} \quad -(7) \\ z - d_1 + d_5 s_{234} &= a_2 c_2 - a_3 s_{23} \quad -(8) \end{aligned}\right\} \text{planar RR}$$

If $(t_2 - t_3) = \pi/2 \Rightarrow$ we have

$$\left.\begin{aligned} \sqrt{x^2+y^2} - d_5 c_{234} &= a_2 s_2 + a_3 s_{23} \\ z - d_1 + d_5 s_{234} &= a_2 c_2 - a_3 c_{23} \end{aligned}\right\} \text{planar RR with } a_x = z - d_1 + d_5 s_{234}$$

$$a_y = \sqrt{x^2+y^2} - d_5 c_{234}$$

$$a_1 = a_2 \;\&\; a_2 = a_3$$

$$t_3 = \pi/2 - \cos^{-1}\left(\frac{\left(z - d_1 + d_5 S_{234}\right)^2 + \left(\sqrt{x^2+y^2} - d_5 C_{234}\right)^2 a_2^2 - a_3^2}{2 a_2 a_3}\right)$$

$$t_2 = a\tan2\left(\frac{\sqrt{x^2+y^2} - d_5 C_{234}}{z - d_1 + d_5 S_{234}}\right) - a\tan2\left(\frac{a_3 \cos t_3}{a_2 - a_3 \cos t_3}\right)$$

N.B S_{234} & C_{234} can be obtain from T_5^0 as follows, $\left(\frac{T_5^0[2,0]}{S_5} = C_{234}\right)$

& $S_{234} = \left(\frac{T_5^0[2,2]}{\quad}\right)$

Furthermore, $\tan(t_5) = \frac{-T_5^0[2,1]}{T_5^0[2,0]}$ $\Rightarrow t_5 = a\tan2\left[\frac{T_5^0[2,1]}{T_5^0[2,0]}\right]$

Also, $t_4 = \sin^{-1}(S_{234}) - t_2 - t_3$

2.2) Calculate the IK for the T_5^0 and see if any of the joint angles exceed the maximum limit

2.3)

2.4) I would try to take the solutions that cover the maximum possible range of positions such the the joint limits aren't exceeded.
I'll also pick that ones that are closed to the initial configuration

2 Coding Assignment

2.1 Forward Kinematics

Inserted in appendix i
For this part all, the locations of each joint were essentially found by post multiplying the transformation matrix by the starting/zero orientation `np.array([[0],[0],[0],[1]])`
For example, the location of joint2, P_2 was found by multiplying $T_0^2 \cdot np.array([[0],[0],[0],[1]])$
This makes sense for joints $0, 3 \ \& \ 5$ because the location of the origin of the coordinate frame of the joint is the same as the the center of its respective joint. However, an exception to this rule was point P_4, this was because the center of this the coordinate frame of this joint was actually joint 3 as explained in 1.1.2. To account for this, we multiplied the transformation matrix taking the origin to that to of the coordinate frame of joint 3 with respective to the base frame.

2.2 Inverse Kinematics

Inserted in appendix ii

2.3 Testing

3 Stimulation

1. When testing the forward kinematics using values of $q = [0, 0, 0, 0, 0, 0]$ and $q = [\pi/3, \pi/2, 0, 0, 0, 0]$ the error between our simulated and calculated joint position values grew as the joint number increased, with the last joint being 25-29mm larger than our calculated position for the same q.

   For $q = [0, 0, 0, 0, 0, 0]$, the first three joint positions have the same results. However, for the other joints, there are errors of about 20-30mm. It also seems like there are very small variations in the simulation that results in difference of 0.001 between the simulation and the calculation, especially for the second column. Similarly for $q = [\pi/3, \pi/2, 0, 0, 0, 0]$, the joint positions are the same (or only a little bit different) for the first few joints. Then, for the last few, the difference goes up to about 30mm. The simulation also has values that are only different by around 0.001mm.

2. First, I chose $q = [0, 0, -\pi/2, 0, 0, 0]$, where the robot is fully extended upwards. I chose this because we learned in class that this position is singular for the robot, and it's usually a position that you want to avoid. The joint positions are the same for the first two columns, while some are slightly different by 0.001mm. The last column has values that are closer to the calculations than the previous configurations. This shows that positions that are straight forward is easier for the robot to become and has less uncertainties.

   Then, I chose $q = [\pi/3, \pi/3, \pi/3, \pi/3, \pi/3, \pi/3]$ because I wanted to utilize every joint and put all in motion. The result is similar, with some joint positions the same and others being about 20mm different. This shows that utilizing more joints doesn't necessarily cause the uncertainties to go up, and the calculations are still quite similar to the simulation.

3. While we didn't get the IK function to fully work, we can predict that the results would have impreciseness similar to FK. It might have more impreciseness in the joint positions because the robot would care more about the end position than joint movements.

3.1 Questions To Think About

1. Between the FK calculation and the simulation, the difference lies in specific joint locations. There are always some joint positions that are exactly the same, but most of the variables are still different, some by 0.001mm and some by 20mm. The bigger differences are usually in the later joints, and the first and second joints are usually the same. This illustrates that the more joints that are connected, the more uncertainties in the calculations. One would expect the variables to be exactly the same, but in reality there's always going to be places where it's not precise.

2. The major factor affecting accuracy is the number of joints involved. The first few joints have much better accuracy than later ones, where the value of difference of joint positions goes from 0.001mm to 30mm. One would expect joint positions of the calculation to be the same as the simulation, since it's forward kinematics and we know each of the joint angles. However, in reality, we see that it's impossible to have perfectly precise robots, and we can only try to minimize the impreciseness. The more complex the robot is, the more likely it would have impreciseness. To combat this issue, we can have more robots that are either less complex (with less joints) or robots that can have simpler motions. The angles of each joint also makes a difference because it matters how dramatic a robot is. Using prismatic joint could potentially be more precise because it's easier to manufacture and mass produce accurate prismatic joints. Additional information about the durability and preciseness of each joint would be needed.

4 Appendix

Listing 1: Calculate_FK.py

```
import numpy as np

class Main():

    def __init__(self):
        """
        This is the dimension of the Lynx Robot stated as global variable

        """
        # Lynx ADL5 constants in mm
        self.d1 = 76.2                  # Distance between joint 0 and joint 1
        self.a2 = 146.05                # Distance between joint 1 and joint 2
        self.a3 = 187.325               # Distance between joint 2 and joint 3
        self.d4 = 34                    # Distance between joint 3 and joint 4
        self.d5 = 76.2                  # Distance between joint 3 and joint 5
        self.lg = 28.575                # Distance between joint 5 and end effector (gripper
            length)

        # Joint limits
        self.lowerLim = np.array([-1.4, -1.2, -1.8, -1.9, -2.0, -15]).reshape((1, 6)) # Lower joint
            limits in radians (grip in mm (negative closes more firmly))
        self.upperLim = np.array([1.4, 1.4, 1.7, 1.7, 1.5, 30]).reshape((1, 6))  # Upper joint
            limits in radians (grip in mm)

    def forward(self, q):
        """
        INPUT:
        q - 1x6 vector of joint inputs [q0,q1,q2,q3,q4,lg]

        OUTPUTS:
        jointPositions - 6 x 3 matrix, where each row represents one
                 joint along the robot. Each row contains the [x,y,z]
                 coordinates of the respective joint's center (mm). For
                 consistency, the first joint should be located at
                 [0,0,0].
        T0e      - a 4 x 4 homogeneous transformation matrix,
                 representing the end effector frame expressed in the
                 base (0) frame
        """
        # Your code starts from here
        T_0_1 = np.array([[np.cos(q[0]), 0, -1*np.sin(q[0]), 0],
                        [np.sin(q[0]), 0, np.cos(q[0]), 0],
                        [0, -1, 0, self.d1],
                        [0, 0, 0, 1]])

        T_1_2 = np.array([[np.sin(q[1]), np.cos(q[1]), 0, self.a2*np.sin(q[1])],
                        [-1*np.cos(q[1]), np.sin(q[1]), 0, -1*self.a2*np.cos(q[1])],
                        [0, 0, 1, 0],
                        [0, 0, 0, 1]])

        T_2_3 = np.array([[-1*np.sin(q[2]), -1*np.cos(q[2]), 0, -1*self.a3*np.sin(q[2])],
                        [np.cos(q[2]), -1*np.sin(q[2]), 0, self.a3*np.cos(q[2])],
                        [0, 0, 1, 0],
                        [0, 0, 0, 1]])

        T_3_4 = np.array([[np.sin(q[3]), 0, np.cos(q[3]), 0],
                        [-1*np.cos(q[3]), 0, np.sin(q[3]), 0],
                        [0, -1, 0, 0],
                        [0, 0, 0, 1]])

        T_4_5 = np.array([[np.cos(q[4]), -1*np.sin(q[4]), 0, 0],
```

```
                [np.sin(q[4]), np.cos(q[4]), 0, 0],
                [0, 0, 1, self.d5],
                [0, 0, 0, 1]])

        T_5_6 = np.array([[1, 0, 0, 0],
                [0, 1, 0, 0],
                [0, 0, 1, self.lg],
                [0, 0, 0, 1]])

        P = np.array([[0],[0],[0],[1]])
        P_1 = T_0_1@P
        P_2 = T_0_1@T_1_2@P
        P_3 = T_0_1@T_1_2@T_2_3@P
        P_4 =
            T_0_1@T_1_2@T_2_3@T_3_4@(np.array([[self.d4*np.sin(q[3])],[0],[self.d4*np.cos(q[3])],[1]]))
        P_5 = T_0_1@T_1_2@T_2_3@T_3_4@T_4_5@P
        print(P)
        print(P_1)
        print(P_2)
        print(P_3)
        print(P_4)
        print(P_5)

        jointPositions = np.vstack((np.delete(P.T,3),
                                    np.delete(P_1.T,3),
                                    np.delete(P_2.T,3),
                                    np.delete(P_3.T,3),
                                    np.delete(P_4.T,3),
                                    np.delete(P_5.T,3)))
        T0e = T_0_1@T_1_2@T_2_3@T_3_4@T_4_5@T_5_6
        # Your code ends here

        return jointPositions, T0e
```

Listing 2: Calculate_IK.py

```
import numpy as np
import math

class Main():

   def __init__(self):
      """
      This is the dimension of the Lynx Robot stated as global variable

      """
      # Lynx ADL5 constants in mm
      self.d1 = 76.2                  # Distance between joint 0 and joint 1
      self.a2 = 146.05                # Distance between joint 1 and joint 2
      self.a3 = 187.325               # Distance between joint 2 and joint 3
      self.d4 = 34                    # Distance between joint 3 and joint 4
      self.d5 = 76.2                  # Distance between joint 3 and joint 5
      self.lg = 28.575                # Distance between joint 5 and end effector (gripper length)

      # Joint limits
      self.lowerLim = np.array([-1.4, -1.2, -1.8, -1.9, -2.0, -15]).reshape((1, 6)) # Lower joint
          limits in radians (grip in mm (negative closes more firmly))
      self.upperLim = np.array([1.4, 1.4, 1.7, 1.7, 1.5, 30]).reshape((1, 6))  # Upper joint limits
          in radians (grip in mm)

   def test(self, T0e, q, isPos):
      """
      Added a test function

      """
      T0e_q_intial = np.array([0, 0, 1, 292.1, 0, -1 ,0 ,0 ,1, 0, 0, 222.25, 0, 0, 0,
          1]).reshape((4, 4))
      T0e_q_1 = np.array([0, 0.7071, 0.7071, 206.5459, 0, -0.7071 ,0.7071 ,206.5459 ,1, 0, 0,
          222.25, 0, 0, 0, 1]).reshape((4, 4))
      T0e_q_2 = np.array([0, 0.7071, 0.7071, 413.0918, 0, -0.7071 ,0.7071 ,413.0918 ,1, 0, 0,
          444.5, 0, 0, 0, 1]).reshape((4, 4))
      T0e_q_3 = np.array([0.75, 0.3417, 0.5663, 126.483, 0.433, -0.9009,-0.0299 ,73.025 ,0.5,
          0.2676, -0.8236, -30.716, 0, 0, 0, 1]).reshape((4, 4))

      if ((T0e == T0e_q_intial).all()):
         try:
            assert (len(q) == 5)
            assert (isPos == 1)
            assert (q[0] == 0)
            assert (q[1] == 0)
            assert (q[2] == 0)
            assert (q[3] == 0)
            assert (q[4] == 0)
         except AssertionError as e:
            raise( AssertionError( "----Error with %s. %s----" % (T0e_q_intial,e)))
      elif (((T0e == T0e_q_1).all())):
         try:
            assert (len(q) == 5)
            assert (isPos == 1)
            assert (q[0] == 0.7854)
            assert (q[1] == 0)
            assert (q[2] == 0)
            assert (q[3] == 0)
            assert (q[4] == 0)
         except AssertionError as e:
            raise( AssertionError( "----Error with %s. %s----" % (T0e_q_1,e)))
      elif (((T0e == T0e_q_2).all())):
         try:
            assert (len(q) == 0)
```

```
            assert (isPos == 0)
         except AssertionError as e:
            raise( AssertionError( "----Error with %s. %s----" % (T0e_q_2,e)))
      elif (((T0e == T0e_q_3).all())):
         try:
            assert (len(q) == 5)
            assert (isPos == 1)
            assert (q[0] == 0.5236)
            assert (q[1] == 0)
            assert (q[2] == 1.0472)
            assert (q[3] == 0)
            assert (q[4] == 0)
         except AssertionError as e:
            raise( AssertionError( "----Error with %s. %s----" % (T0e_q_3,e)))
      else :
         print("Erroneous T0e")


def inverse(self, T0e):


   """
   INPUT:
   T - 4 x 4 homogeneous transformation matrix, representing
    the end effector frame expressed in the base (0) frame
    (position in mm)

   OUTPUT:
   q - a n x 5 vector of joint inputs [q1,q2,q3,q4,q5] (rad)
    which are required for the Lynx robot to reach the given
    transformation matrix T. Each row represents a single
    solution to the IK problem. If the transform is
    infeasible, q should be all zeros.
   isPos - a boolean set to true if the provided
       transformation T is achievable by the Lynx robot as given,
       ignoring joint limits
   """

   isPos = 1
   q = np.zeros((5, 1))
   # print(q)

   # # Your code ends here

   return q, isPos
```