University of California, Riverside

EE/ME144/EE283A
Foundations of Robotics

Fall 2021

Lab 4 Report

Nov 5, 2021

Name	SID	Section	Group Number
Simraj Singh	862155747	Thursday	4
Shaheriar Malik	862154387	Thursday	4

1. Problem Statement

In Lab 4, we were tasked with writing a python script to control the ReactorX 150 robot arm using forward and inverse kinematics. Specifically, we computed the position of the end effector using POE with the given joint angles for forward kinematics and using the analytical approach (basic trigonometry to find theta values given the position of the arm) for inverse kinematics.

2. Design Idea

The main design behind programming the robot arm was taking the link lengths and joint angles into consideration when doing the calculations mentioned above. For forward kinematics, following the steps discussed in class, we first made a 4x4 'home' matrix with the position and orientation of the end effector joint 4. Next, we assigned the three omega values for x, y, z as well as the positions for the three joints, q1, q2, and q3. To get the linear velocity, we needed to get the cross product of negative omega and the positions (-w x q). To get the 6x1 screw vector we just concatenate the angular and linear velocities. Next we turn those 6x1 vectors into skew symmetric matrices which helped us calculate the exponential coefficients and finally calculating the final transformation matrix using left multiplication by the home matrix. The positions of the end effector are given by the rightmost row of the transformation matrix.

For inverse kinematics, we had to find values for theta1, theta2, theta3, alpha, beta1, beta2, and gamma given the dimensions of the ReactorX 150 arm (as shown in Figures 4.2 and 4.3). From Figure 4.1, we can see the dimensions used in order to calculate the aforementioned values. To calculate theta1, we computed the arctan(y,x). For alpha, we again used arctan but this time with values of 0.050 and 0.15 (corresponding to the lengths of link 3 and 4 in meters). For gamma, we computed the arctan of z-0.1039 (the height minus length of link 1 and 2) and sqrt(x^2 + y^2) (hypotenuse). To find beta1, we used the law of cosines with interior angles from Figure 4.3 and the length of links 3, 4, and hypotenuse of the associated triangle. After finding beta1, we are able to find theta2 by doing pi/2 - alpha - beta1 - gamma. To find beta 2, we again used the law of cosines with angle and link length values from the same triangle as before. Lastly, to compute theta3, we did pi - alpha - pi/2 - (pi - beta2) (angles of a triangle must add up to 180). Thus, given the x, y, and z positions of the end effector, we were able to successfully return the joint angles needed to move the robot arm.

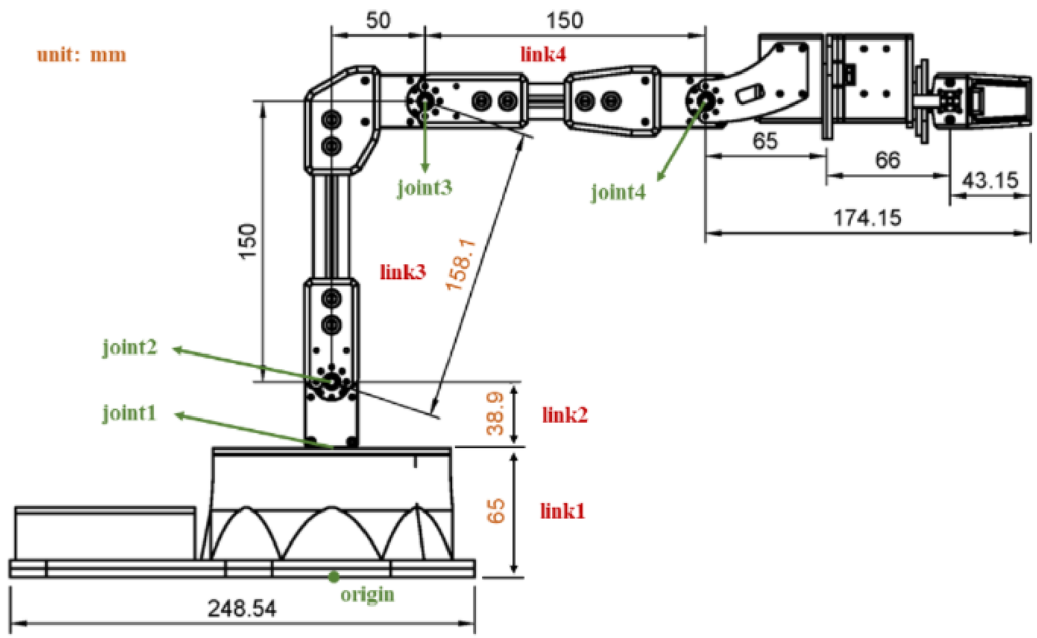

Figure 4.1: Dimensions of the ReactorX 150 manipulator as given in the lab manual

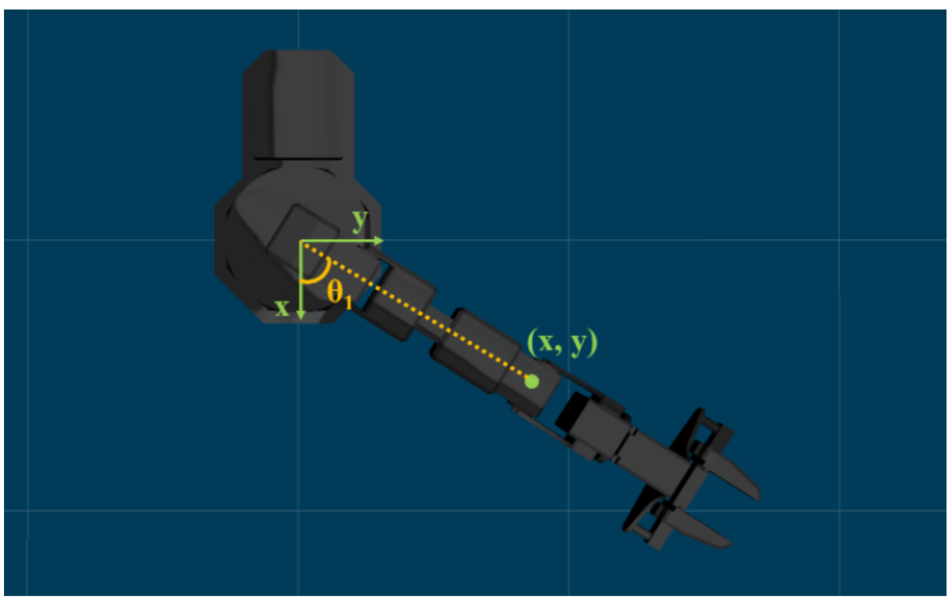

Figure 4.2: Annotated figure used to understand the relationship between the values we need to calculate and the values we already have from the lab manual

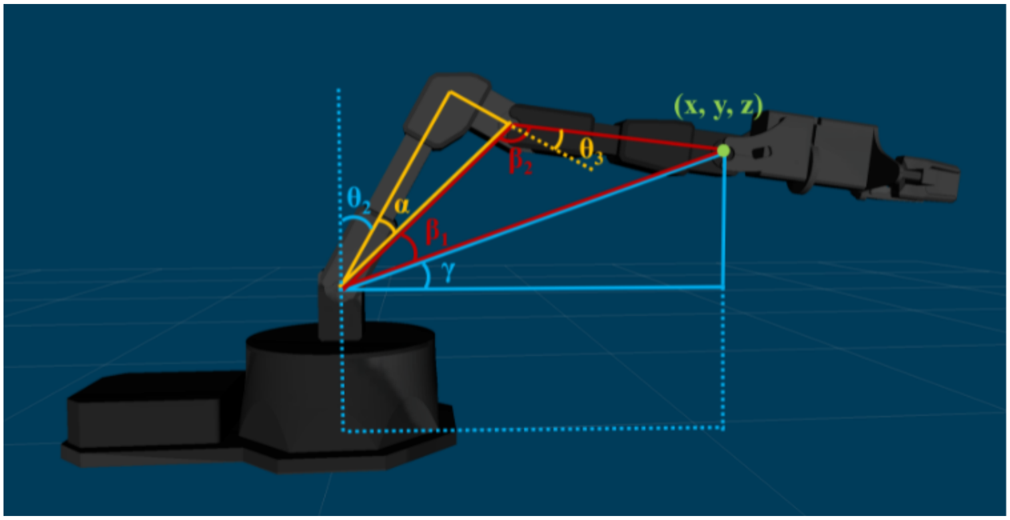

Figure 4.3: Annotated figure used to understand the relationship between the values we need to calculate and the values we already have from the lab manual

```python
def forward_kinematics(joints):
    # input: joint angles [joint1, joint2, joint3]
    # output: the position of end effector [x, y, z]
    # add your code here to complete the computation

    link1z = 0.065
    link2z = 0.039
    link3x = 0.050
    link3z = 0.150
    link4x = 0.150
    joint1 = joints[0]
    joint2 = joints[1]
    joint3 = joints[2]

    M = [[1,0,0,link3x+link4x],
        [0,1,0,0],
        [0,0,1,link1z+link2z+link3z],
        [0,0,0,1]]

    w1 = np.array([0,0,1])
    w2 = np.array([0,1,0])
    w3 = np.array([0,-1,0])

    q1 = np.array([0,0,link1z])
    q2 = np.array([0,0,link1z+link2z])
    q3 = np.array([link3x,0,link3z+link1z+link2z])

    v1 = np.cross(-w1, q1)
    v2 = np.cross(-w2, q2)
    v3 = np.cross(-w3, q3)

    s1 = np.concatenate([w1, v1])
    s2 = np.concatenate([w2, v2])
    s3 = np.concatenate([w3, v3])

    s1 = mr.VecTose3(s1)
    s2 = mr.VecTose3(s2)
    s3 = mr.VecTose3(s3)

    e1 = mr.MatrixExp6(s1*joint1)
    e2 = mr.MatrixExp6(s2*joint2)
    e3 = mr.MatrixExp6(s3*joint3)

    T = np.matrix(e1)*np.matrix(e2)*np.matrix(e3)*np.matrix(M)

    x = T.item(0,3)
    y = T.item(1,3)
    z = T.item(2,3)

    return [x, y, z]
```

Figure 4.4: Code for forward kinematics implementation

```
def inverse_kinematics(position):
    # input: the position of end effector [x, y, z]
    # output: joint angles [joint1, joint2, joint3]
    # add your code here to complete the computation

    link1z = 0.065
    link2z = 0.039
    link3x = 0.050
    link3z = 0.150
    link4x = 0.150
    x = position[0]
    y = position[1]
    z = position[2]

    #Finding theta 1
    theta1 = atan2(y,x)
    alpha = atan2(0.050,0.15)
    sqX = sqrt(x**2 + y**2)

    #Finding theta 2
    gamma = atan2(z-0.1039,sqX)
    hypo = sqrt(sqX**2 + (z-0.1039)**2)
    beta1 = acos(((0.1581**2) + hypo**2 - (0.15**2))/(2 * 0.1581 * hypo))
    theta2 = (pi/2) - alpha - beta1 - gamma

    #Finding theta 3
    beta2 = acos(((0.1581**2) + (0.15**2) - hypo**2)/(2 * 0.1581 * 0.15))
    betaTemp = pi - beta2
    theta3 = pi - alpha - pi/2 - betaTemp

    joint1 = theta1
    joint2 = theta2
    joint3 = theta3

    return [joint1, joint2, joint3]
```

Figure 4.5: Code for inverse kinematics implementation

3. Results

Ultimately, we were able to successfully calculate where the arm is going to be given the thetas for forward kinematics and given the position/calculating the thetas for inverse kinematics. By using test case values from the autograder, we were able to ensure our calculations were correct in comparison to the true values which were calculated using the methods discussed in lecture. The most difficult part of this lab was making sure we did the trigonometry correctly for inverse kinematics as we did all of the calculations by hand. In lab 5, we make sure these functions perform as expected using RViz and compare the values these functions return with the actual values the engine returns.

4. Appendix

How to run the code:

- Give the scripts executable permission:
    - `chmod +x forward_kinematics.py`
    - `chmod +x inverse_kinematics.py`
- First add print statements at the end of these scripts that have test parameters for both functions, then run the files using:
    - `./forward_kinematics.py`
    - `./inverse_kinematics.py`