ECE 470: Lab 4

Name: **Ashank Behara**
NetId: **abehara2**
TA: **Weihang (Eric) Liang**
Section: Thursday 3:00 PM
Submitted: **December 3, 2020**

Shared information/content from my previous report

Objective

The objective of this lab was to compute the inverse kinematics of a UR3 6-axis robot and then computing the forward kinematics of the robot using the Product of Exponentials method to validate correct movements. Given screw axes of all joints, end-effector configuration at a zero-configuration for the entire robot, and relationships between joint angles and links, we were tasked with moving the robot to a passed configuration given different link lengths for the robot.

Method

Computing the inverse kinematics to this problem required a lot of careful computation and clear definition of the alpha, beta, and gamma constraints when trying to solve the inverse kinematics. Here are the symbolic solutions that I came up with for each of the rotational positions of each joint of the robot. The code is simple enough to be interpreted mathematically. I tried to translate the definitions and the code itself from the original file a little bit to fit the way math is interpreted by a human and hope that these definitions suffice.

```
d_cen = sqrt(xcen^2 + ycen^2) # distance to {x,y,z}cen point
a = arctan2(ycen, xcen)
b = arcsin((l02 + l06 - l04)/d_cen)

θ1 = a - b
θ5 = PI/2 + θ1 - yaw # subtract yaw (input) for correct orientation

#################################################################################

d_end = np.sqrt(x3end^2 + y3end^2 + z3end^2) # distance to {x,y,z}3end point
g = law_cos(l03, l05, d_end) # law of cosines
b = law_cos(d_end, l03, l05) # law of cosines
a = np.arcsin(z3end/d_end)

θ2 = -(a + b)
θ3 = PI - g #180 - t3 = gamma
θ4 = -(PI - (a + b + g))
θ5 = - PI/2
```

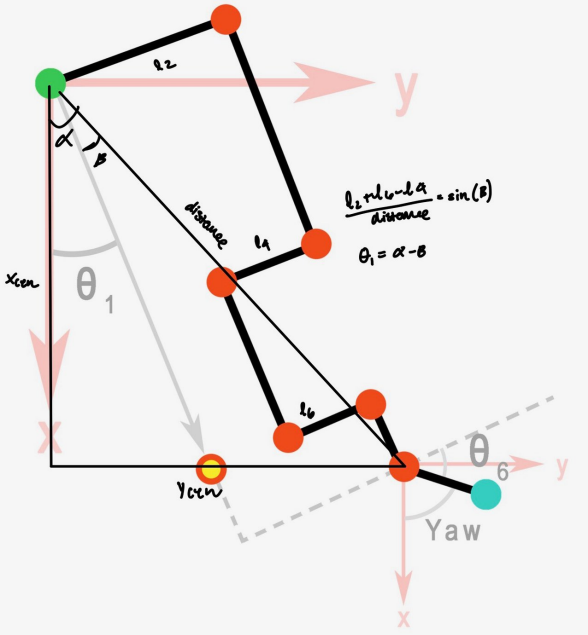


Figure 1: Top View

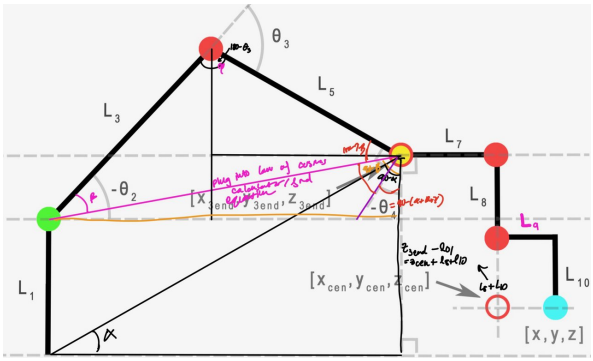


Figure 2: Side View

Testing and Validation

There are two sets of test position and orientation values that were provided to validate the calculation of the inverse kinematics.

Test 1 (see appendix)

Inputs: $[0.2, 0.40, 0.05][45]$
Measured position: $[0.201, 0.403, 0.049]$
Error: 0.0033

Test 2 (see appendix)

Inputs: $[0.15, -0.1, 0.25][-45]$
Measured position: $[0.153, -0.101, 0.249]$
Error: 0.00332

Sources of Error

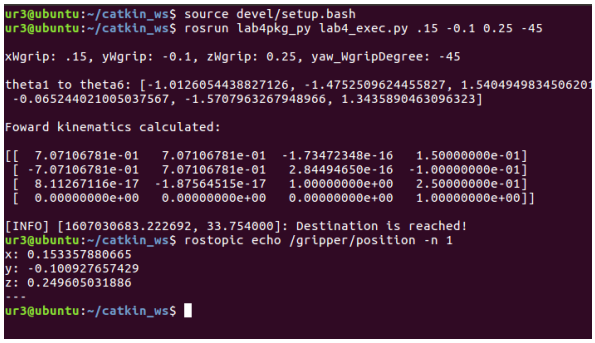
The scalar values for both test cases were extremely low and are more or less negligible. As all computations were completed by a computer, calculation error as a possible source is unlikely unless certain functions truncated values. The main source of error can be attributed to the measuring of all the link lengths. As these robots were manufactured with certain tolerances, and a human measured these values, it is likely that there is some small discrepancy between true and measured positions with is represented by the also small error.

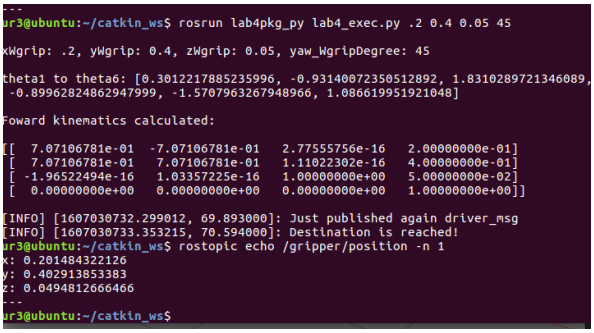
Conclusion

This lab was a great learning experience. At the very beginning of the lab I struggled with the concept of inverse kinematics, but with a lot of help from online lectures and resources, I was able to complete the assignment. This lab was very hard for me and at many points I thought I was not going to be able to complete it, but I am glad I was able to figure out a correct solution. The biggest mistake I made throughout this entire process was not reading the instruction manual clearly before hand. I did not realize that I was to copy my solution for Lab 3 into this lab, and instead tried to compute an entirely different end effector matrix and screw axes with arbitrarily made up angles that I tried to measure myself from the diagram in the lab, but after many many hours of messing with values (and of course re-reading the manual), the solution worked itself out.

Appendix

Test Cases from Demo





Code

```python
#!/usr/bin/env python
import numpy as np
from scipy.linalg import expm
from lab4_header import *

def get_skew_symmetric(w,v):
    skew = np.zeros((4,4))
    skew[0] = [0,-w[2],w[1],v[0]]
    skew[1] = [w[2],0,-w[0],v[1]]
    skew[2] = [-w[1],w[0],0,v[2]]
    skew[3] = [0,0,0,0]
    return skew

def Get_MS():
    # ==================== Your code starts here ====================#
    # Fill in the correct values for S1~6, as well as the M matrix
    M = np.zeros((4,4))
    M[0] = [0, -1, 0, .390]
    M[1] = [0, 0, -1, .401]
    M[2] = [1, 0, 0, .2155]
    M[3] = [0, 0, 0, 1]
    w1 = np.array([0,0,1])
    w2 = np.array([0,1,0])
    w3 = np.array([0,1,0])
    w4 = np.array([0,1,0])
    w5 = np.array([1,0,0])
```

```
    w6 = np.array([0,1,0])

    q1 = np.array([-.150,.150,.010])
    q2 = np.array([-.150,.270,.162])
    q3 = np.array([.094,.270,.162])
    q4 = np.array([.307, .177, .162])
    q5 = np.array([.307,.260,.162])
    q6 = np.array([.390,.260,.162])

    v1 = -np.cross(w1, q1)
    v2 = -np.cross(w2, q2)
    v3 = -np.cross(w3, q3)
    v4 = -np.cross(w4, q4)
    v5 = -np.cross(w5, q5)
    v6 = -np.cross(w6, q6)

    s1 = get_skew_symmetric(w1,v1)
    s2 = get_skew_symmetric(w2,v2)
    s3 = get_skew_symmetric(w3,v3)
    s4 = get_skew_symmetric(w4,v4)
    s5 = get_skew_symmetric(w5,v5)
    s6 = get_skew_symmetric(w6,v6)

    S = [s1, s2, s3, s4, s5, s6]

    # ================================================================#
    return M, S


"""
Function that calculates encoder numbers for each motor
"""
def lab_fk(theta1, theta2, theta3, theta4, theta5, theta6):

    # Initialize the return_value
    return_value = [None, None, None, None, None, None]

    # =========== Implement joint angle to encoder expressions here ===========
    print("Foward kinematics calculated:\n")

    # ================== Your code starts here ====================#
    theta = np.array([theta1,theta2,theta3,theta4,theta5,theta6])
    T = np.eye(4)

    M, S = Get_MS()

    T = np.dot(T,expm(S[0]*theta1))
    T = np.dot(T,expm(S[1]*theta2))
    T = np.dot(T,expm(S[2]*theta3))
    T = np.dot(T,expm(S[3]*theta4))
    T = np.dot(T,expm(S[4]*theta5))
    T = np.dot(T,expm(S[5]*theta6))
    T = np.dot(T,M)

    # ================================================================#
```

```
    print(str(T) + "\n")

    return_value[0] = theta1 + PI
    return_value[1] = theta2
    return_value[2] = theta3
    return_value[3] = theta4 - (0.5*PI)
    return_value[4] = theta5
    return_value[5] = theta6

    return return_value


"""
Function that calculates an elbow up Inverse Kinematic solution for the UR3
"""
def law_cos(a,b,c):
    return np.arccos((a**2 + b**2 - c**2) / (2*a*b))

def lab_invk(xWgrip, yWgrip, zWgrip, yaw_WgripDegree):

    # theta1 to theta6
    thetas = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

    l01 = 0.152
    l02 = 0.120
    l03 = 0.244
    l04 = 0.093
    l05 = 0.213
    l06 = 0.083
    l07 = 0.083
    l08 = 0.082
    l09 = 0.0535
    l10 = 0.059   # thickness of aluminum plate is around 0.01
    end_offset = 0.027
    yaw_WgripDegree = PI*yaw_WgripDegree/180

    xgrip = xWgrip + 0.15
    ygrip = yWgrip - 0.15
    zgrip = zWgrip - 0.01
    xcen = xgrip - l09*np.cos(yaw_WgripDegree)
    ycen = ygrip - l09*np.sin(yaw_WgripDegree)
    zcen = zgrip

    d_cen = np.sqrt(xcen**2 + ycen**2)
    a = np.arctan2(ycen, xcen)
    b = np.arcsin((l02 + l06 - l04)/d_cen)

    thetas[0] = a - b
    thetas[5] = PI/2 + thetas[0] - yaw_WgripDegree

    x3end = xcen - l07*np.cos(thetas[0]) + (l06 + end_offset)*np.sin(thetas[0])
    y3end = ycen - l07*np.sin(thetas[0]) - (l06 + end_offset)*np.cos(thetas[0])
```

```
    z3end = zcen + l10 + l08 - l01 #0

    d_end = np.sqrt(x3end**2 + y3end**2 + z3end**2)
    g = law_cos(l03, l05, d_end)
    b = law_cos(d_end, l03, l05)
    a = np.arcsin(z3end/d_end)

    thetas[1]= -(a + b)
    thetas[2]= PI - g
    thetas[3]= -(PI - (a + b + g))
    thetas[4]= - PI/2

    print("theta1 to theta6: " + str(thetas) + "\n")

    return lab_fk(float(thetas[0]), float(thetas[1]), float(thetas[2]), \
        float(thetas[3]), float(thetas[4]), float(thetas[5]))
```