ECE 470: Lab 3

Name: **Ashank Behara**
NetId: **abehara2**
TA: **Weihang (Eric) Liang**
Section: Thursday 3:00 PM
Submitted: **November 4, 2020**

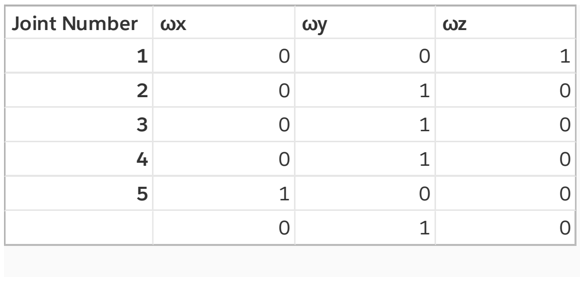
Objective

The objective of this lab was to compute the forward kinematics of a UR3 6-axis robot using the Product of Exponentials method. Given screw axes of all joints and end-effector configuration at a zero-configuration for the entire robot, we were tasked with moving the robot to a new configuration given different joint angles for each screw.

Method

Computing the forward kinematics of a multi-axis is fairly simple when the entire initial configuration is completely defined. There are two main parts to computing this: the end effector configuration and the screw axes.

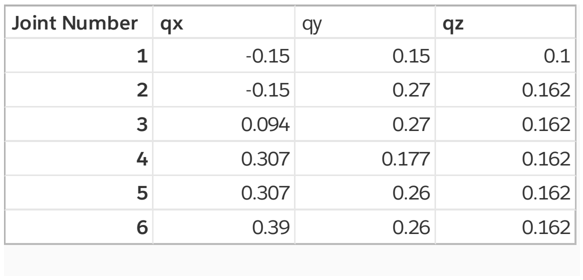
To compute the screw axes, we must identify every point of rotation, determine in what axis the joint is rotating, and lastly the direction of it. The vectors computed from can be denoted as an ω vector. The following figure is a table of them all.



Joint Number	ωx	ωy	ωz
1	0	0	1
2	0	1	0
3	0	1	0
4	0	1	0
5	1	0	0
	0	1	0

Figure 1: ω vectors

Once we compute the omega, vectors, we must find a point that lies on that rotation axis. We can represent these points as a vector that we denote as a *q* vector. The following figure is a table of them all with points in millimeters. An issue that arose when demoing my lab was instead of dividing each number by *float(1000.0),* I divided by *int(1000)* which led to all the *q* values to becoming 0 and resulted in a complete difference of my transformation matrix.



Joint Number	qx	qy	qz
1	-0.15	0.15	0.1
2	-0.15	0.27	0.162
3	0.094	0.27	0.162
4	0.307	0.177	0.162
5	0.307	0.26	0.162
6	0.39	0.26	0.162

Figure 2: *q* vectors

The next step in this process is to take the cross product of our negative omega vectors and position vectors. The following figure has the results of all of these computations.

Joint Number	vx	vy	vz
1	0.15	0.15	0
2	-0.162	0	-0.15
3	-0.162	0	0.094
4	-0.162	0	0.307
5	0	0.162	-0.26
6	-0.162	1	0.39

Figure 3: *v* vectors

Using the ω and v vectors, we can simply append them together to get 6 new vectors denoted by S. The vectors in their raw form are not fully ready to be used to calculate the forward kinematics of the robot, so we must find the corresponding skew symmetric matrix for each one. The following figures have each of the skew symmetric matrices.

0	-1	0	0.15
1	0	0	0.15
0	0	0	0
0	0	0	0

Figure 4.1: *[s1]*

0	0	1	-0.162
0	0	0	0
-1	0	0	-0.15
0	0	0	0

Figure 4.2: *[s2]*

0	0	1	-0.162
0	0	0	0
-1	0	0	0.094
0	0	0	0

Figure 4.3: *[s3]*

0	0	1	-0.162
0	0	0	0
-1	0	0	0.307
0	0	0	0

Figure 4.4: *[s4]*

0	0	0	0
0	0	-1	0.162
0	1	0	-0.26
0	0	0	0

Figure 4.5: *[s5]*

0	0	1	-0.162
0	0	0	0
-1	0	0	0.39
0	0	0	0

Figure 4.6: *[s6]*

Now that all the skew symmetric matrices of the screw axes have been calculation, the next big piece of finding the forward kinematics is to find the end effector transformation matrix. To compute this, we must find the rotation matrix from the robot's space frame to the end effector's frame. We are given both the frames, so computing this is trivial. The second main part of this is to find the position of the end effector with respect to the origin of the space frame which we can easily find using the diagram provided. Combining these two matrices gives a matrix M that is defined in the figure below.

0	-1	0	0.39
0	0	-1	0.401
1	0	0	0.2155
0	0	0	1

Figure 5: *M*

We finally have all of the building blocks to compute our forward kinematics calculation. The equation specifically to calculate the transformation matrix for the UR3 robot is as follows where the theta values are arbitrary inputs provided.

$$T_{06} = e^{[s_1](\theta_1)} \cdot e^{[s_2](\theta_2)} \cdot e^{[s_3](\theta_3)} \cdot e^{[s_4](\theta_4)} \cdot e^{[s_5](\theta_5)} \cdot e^{[s_6](\theta_6)} \cdot M$$

I tried to use python's `sympy` library to represent the individual $e^{[s](\theta)}$, but python's `scipy expm()` function was not able to parse the symbolic variable as a float. Therefore, I had to settle with representing the skew symmetric matrices individually and I hope that is sufficient information. I directly plugged in the values as shown above to compute the final transformation matrix. The diagram I used to calculate all of the previous points is displayed below.

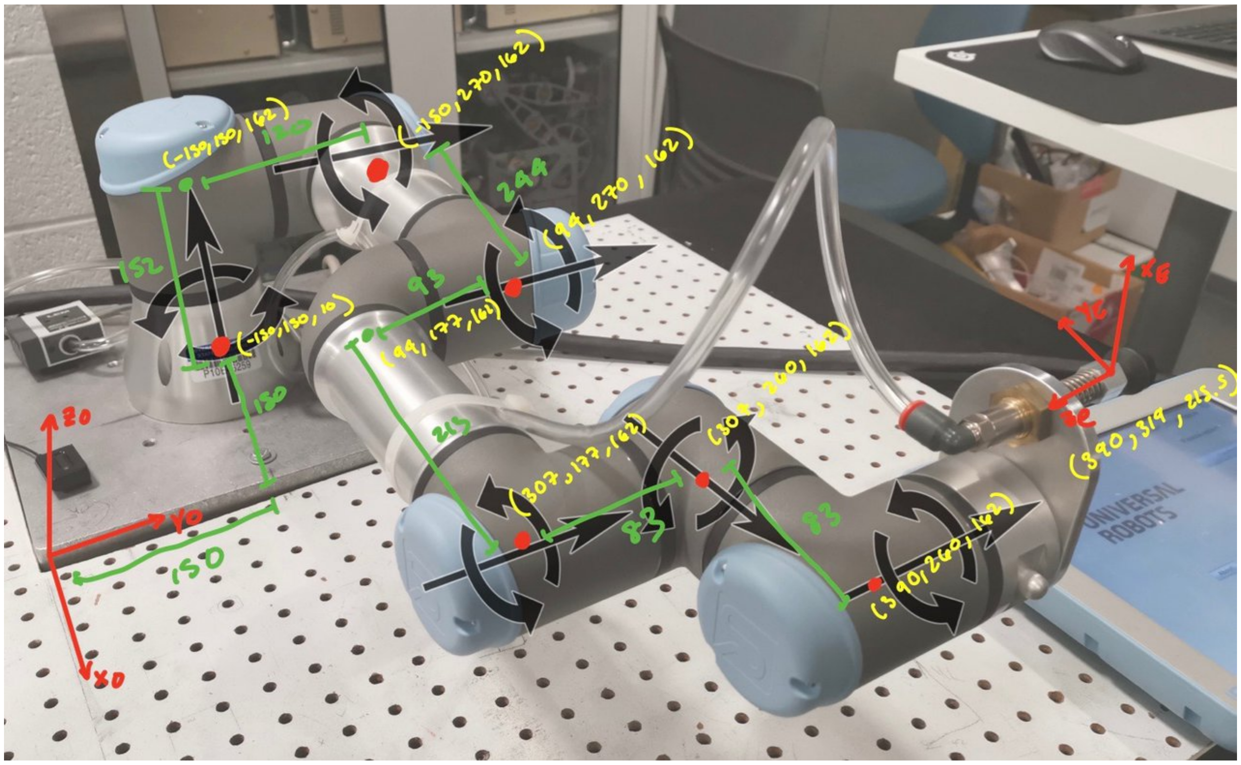


Figure 6: Hand calculations and screw axes

Testing and Validation

There are two sets of test theta values that were provided to validate the calculation of the forward kinematics and the final transformation matrix.

Test 1 (see appendix)

Thetas: $[-105, -105, 100, -50, -110, -30]$
Calculated Transformation Matrix:

0.7975	-0.289	0.5296	-0.1266
-0.168	0.7367	9.655	-0.1698
-0.5795	-0.6113	0.539	0.3772
0	0	0	1

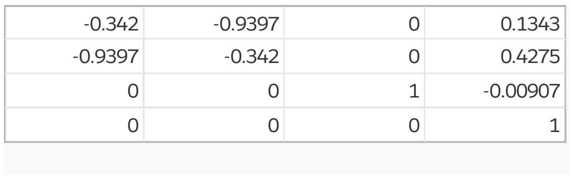
Figure 7.1: Transformation Matrix for Test 1

Calculated d: $[-0.126650, -0.169756, 0.377241]$
Measured r: $[-0.124749, -0.171936, 0.378785]$
Error: 0.003279

Test 2 (see appendix)

Thetas: $[20, -45, 105, -60, -90, 0]$

Calculated Transformation Matrix:



-0.342	-0.9397	0	0.1343
-0.9397	-0.342	0	0.4275
0	0	1	-0.00907
0	0	0	1

Figure 7.2: Transformation Matrix for Test 2

Calculated d: $[0.134280, 0.427462, 0.009071]$
Measured r: $[0.135557, 0.430415, 0.008565]$
Error: 0.003256

Sources of Error

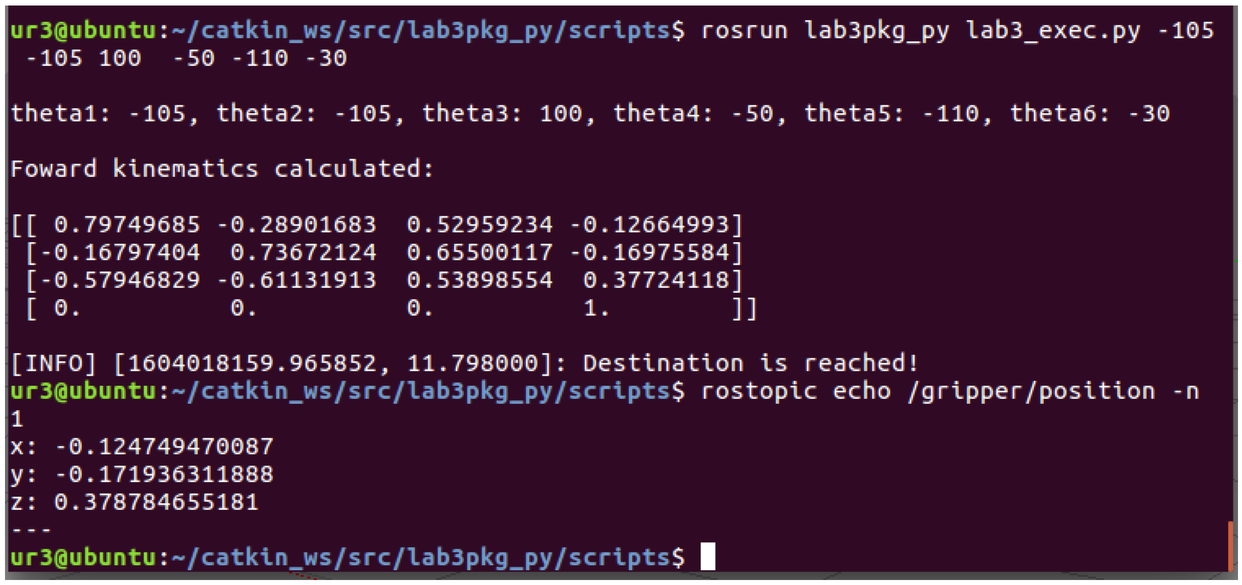
The scalar values for both test cases were extremely low. As all computations were completed by a computer, calculation error as a possible source is unlikely unless certain functions truncated values. The main source of error can be attributed to the computation of 6 values. As these robots were manufactured 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 forward kinematics, but after catching up with the material in the class the problem became very straightforward. It was just a matter of splitting things up and making sure calculations were correct. I ran into a little issue with lossy conversions just before my demo started which led to some mishaps, but after working through the issues I was able to have a successful demo.

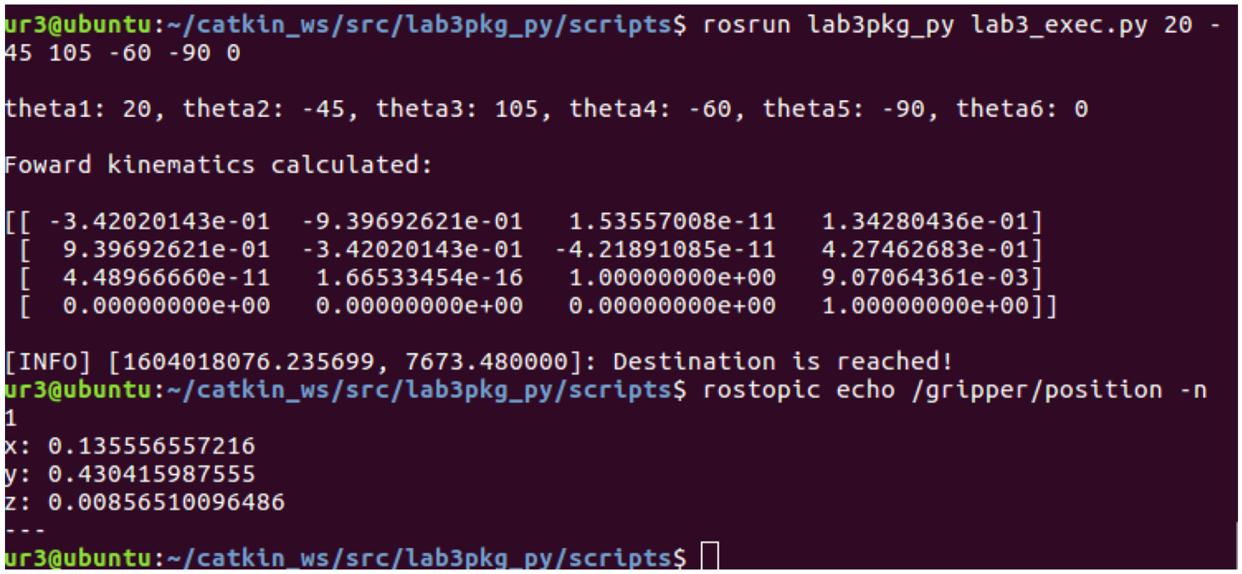
Appendix

Test 1



```
ur3@ubuntu:~/catkin_ws/src/lab3pkg_py/scripts$ rosrun lab3pkg_py lab3_exec.py -105
 -105 100  -50 -110 -30

theta1: -105, theta2: -105, theta3: 100, theta4: -50, theta5: -110, theta6: -30

Foward kinematics calculated:

[[ 0.79749685 -0.28901683  0.52959234 -0.12664993]
 [-0.16797404  0.73672124  0.65500117 -0.16975584]
 [-0.57946829 -0.61131913  0.53898554  0.37724118]
 [ 0.          0.          0.          1.        ]]

[INFO] [1604018159.965852, 11.798000]: Destination is reached!
ur3@ubuntu:~/catkin_ws/src/lab3pkg_py/scripts$ rostopic echo /gripper/position -n
1
x: -0.124749470087
y: -0.171936311888
z: 0.378784655181
---
ur3@ubuntu:~/catkin_ws/src/lab3pkg_py/scripts$ 
```

Test 2



```
ur3@ubuntu:~/catkin_ws/src/lab3pkg_py/scripts$ rosrun lab3pkg_py lab3_exec.py 20 -
45 105 -60 -90 0

theta1: 20, theta2: -45, theta3: 105, theta4: -60, theta5: -90, theta6: 0

Foward kinematics calculated:

[[ -3.42020143e-01  -9.39692621e-01   1.53557008e-11   1.34280436e-01]
 [  9.39692621e-01  -3.42020143e-01  -4.21891085e-11   4.27462683e-01]
 [  4.48966660e-11   1.66533454e-16   1.00000000e+00   9.07064361e-03]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]

[INFO] [1604018076.235699, 7673.480000]: Destination is reached!
ur3@ubuntu:~/catkin_ws/src/lab3pkg_py/scripts$ rostopic echo /gripper/position -n
1
x: 0.135556557216
y: 0.430415987555
z: 0.00856510096486
---
ur3@ubuntu:~/catkin_ws/src/lab3pkg_py/scripts$
```

Code

```
#!/usr/bin/env python
import numpy as np
from scipy.linalg import expm
import sympy as sym
class FK:
    def __init__(self):
        pass
    """
    Use 'expm' for matrix exponential.
    Angles are in radian, distance are in meters.
    """
    def get_skew_symmetric(self,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(self):
        # =================== 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 = self.get_skew_symmetric(w1,v1)
        s2 = self.get_skew_symmetric(w2,v2)
        s3 = self.get_skew_symmetric(w3,v3)
        s4 = self.get_skew_symmetric(w4,v4)
        s5 = self.get_skew_symmetric(w5,v5)
        s6 = self.get_skew_symmetric(w6,v6)
        S = [s1, s2, s3, s4, s5, s6]
        # ==============================================================#
        return M, S

    """
    Function that calculates encoder numbers for each motor
    """
    def error(self,r,d):
        return np.sqrt(np.square(r[0] - d[0]) + np.square(r[1] - d[1]) + np.square(r[2
    def lab_fk(self, 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 = self.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 + np.pi
        return_value[1] = theta2
        return_value[2] = theta3
        return_value[3] = theta4 - (0.5*np.pi)
        return_value[4] = theta5
        return_value[5] = theta6
        return return_value
def main():
    fk1 = FK()
    fk1.lab_fk( -105, -105, 100, -50, -110, -30)
    r1 = [-0.124749, -0.171936, 0.378785]
    d1 = [-0.126650, -0.169756, 0.377241]
    r2 = [0.135557, 0.430415, 0.008565]
    d2 = [0.134280, 0.427462, 0.009071]

    e1 = fk1.error(r1, d1)
    e2 = fk1.error(r2, d2)
    print("E1: ", e1)
    print("E2: ", e2)

if __name__ == '__main__':
    main()
```