Lab 4 Report: Forward Kinematics for UR3e Robot

Shenghua Ye, Xuan Tang, Jie Wang
Student ID: 3200112420, 3200115277 3200112404
Instructor: Yang Liangjing

April 8, 2024

1 Introduction

The forward kinematics of a robot is essential to determine the position and orientation of its end-effector. With appropriate joint angles setting, we can let the robot to fit in the correct position that are stable.The lab aims to bridge the theoretical aspects of robot kinematics with practical application using ROS and Python.

2 Objectives

The objectives of this lab are:

- To solve the exponential forward kinematic equations for the UR3e robot.
- To write a Python function that moves the UR3e to a configuration specified by the user.
- To compare the exponential forward kinematic estimation with the actual robot movement.

3 Theoretical Background

Mathematical Foundation of Forward Kinematics

The forward kinematics of a robot can be mathematically represented by a series of transformations from the base frame to the end-effector frame. These transformations are a function of the robot's joint parameters. For a robotic arm with n joints, the position and orientation of the end-effector, $\mathbf{T}_{\text{base}}^{\text{effector}}$, can be computed as:

$$\mathbf{T}_{\text{base}}^{\text{effector}} = \prod_{i=1}^{n} \mathbf{T}_{i-1}^{i}(\theta_i)$$

where $\mathbf{T}_{i-1}^{i}(\theta_i)$ represents the transformation from the frame of joint $i-1$ to the frame of joint i, and θ_i is the angle of the ith joint. These transformations typically involve rotations and translations, capturing the spatial relationships between adjacent links of the robot.

This formula highlights the cumulative effect of each joint's movement on the final position and orientation of the end-effector. By adjusting the joint angles θ_i, the robot can achieve a wide range of positions and orientations, enabling it to perform various tasks.

4 Procedure

The procedure is divided into three tasks, aligning with the lab's objectives to ensure a thorough understanding and application of Forward Kinematics on the UR3e robot. Each task progressively builds upon the previous, from theoretical groundwork to practical implementation and analysis.

1. **Theoretical Solution:** Initially, we deduce the forward kinematics using the exponential map formula, similar to the approach taken in Assignment 2. This stage involves calculating the theoretical solution, leveraging the foundational kinematics equations to derive the transformation matrices.

2. **Physical Implementation:** Subsequently, we transition to the empirical phase by encoding the theoretical findings into a Python script. A pivotal component of our code is the utility function ***skew(s)***, designed to compute the skew symmetric matrix of the screw axis vector S. The script employs a loop to sequentially multiply the transformation matrices, thus simulating the robot's movement. The implementation is straightforward yet powerful, effectively bridging theory with practice.

In alignment with understanding the robot's configuration, below is a tabular representation of the Screw Axes (S1 to S6) for the UR3e robot in its zero position configuration, complemented by a visual depiction of the physical setup:

tableScrew Axes in Zero Configuration

Joint	$S_i = \omega$
S1	(90.00)
S2	(0.00)
S3	(0.00)
S4	(−90.00)
S5	(0.00)
S6	(0.00)

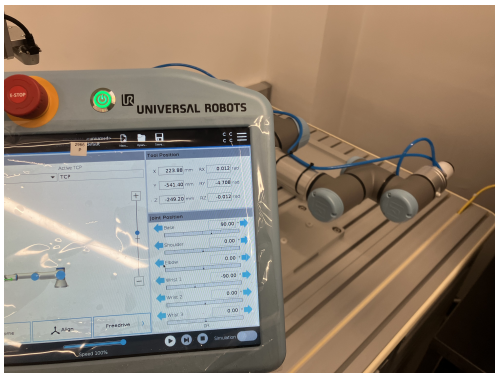

Figure 1: Physical Zero Configuration Space of UR3e

This helps us better predict and check the position and orientation of the robot's end-effector with Forward Kinematics.

5 Results

Here is the Python terminal result, showing our pass the test cases and computed the Forward Kinematics correctly.

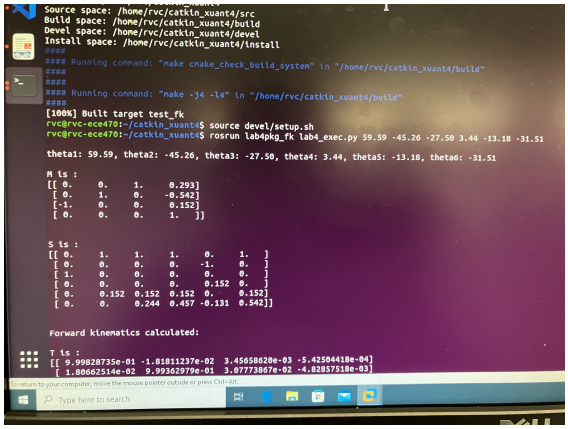

Figure 2: Correct M,S matrix

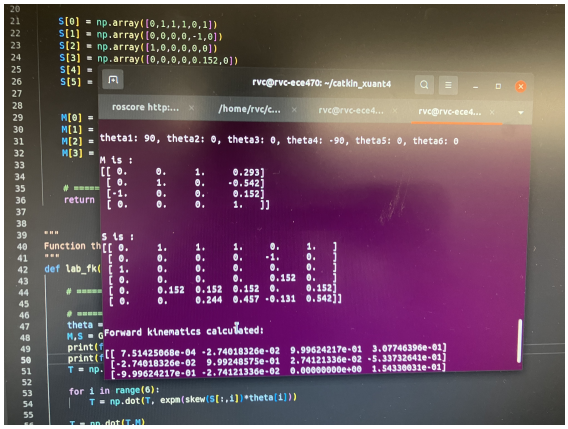

Figure 3: Although there are some bugs, we passed the test cases finally

6 Discussion

In our development, we encountered two major problems while computing the M and S matrices:

1. We misunderstood the input type of the function *lab_fk*, considering it as radians. However, we later discovered that the main program had already transformed the theta into degree measure.

2. We misunderstood the zero configuration of UR3e, which should be 'compensated' rather than added again. It should be:

$$\theta = np.array([\theta_1 - 0.5 * \pi, \theta_2, \theta_3, \theta_4 + 0.5 * \pi, \theta_5, \theta_6])$$

   instead of:

$$\theta = np.array([\theta_1 + 0.5 * \pi, \theta_2, \theta_3, \theta_4 - 0.5 * \pi, \theta_5, \theta_6])$$

3. We calculate the function using a helper function skew() which takes S input into the helper matrix for further calculation.

These misunderstandings led to initial inaccuracies in our kinematic calculations, which were corrected upon realizing the errors.

7 Conclusion

All in all, this lab implement the knowledge from Course and Assignment 2 into the ROS python code. It enhances our understanding on robot movement and control. Further more, it underscore the importance of precise mathematical modeling and attention to detail in configuring robotic systems, lessons that are invaluable for our future endeavors in robotics and engineering.

8 References

1. ECE470 Lab4 Manual: Forward Kinematics.

2. Lynch, Kevin M., and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. Preprint version.

3. Universal Robots A/S. "User Manual - UR3e e-Series - SW 5.10 - English international." Last modified on Mar 16, 2021. Available online: https://www.universal-robots.com/download/manuals-e-seriesur20ur30/user/ur3e/510/user-manual-ur3e-e-series-sw-510-english-international-en/

4. Clearpath Robotics. "URE ROS Setup." UR Setup Tutorial 0.0.0 documentation. Available online: https://www.clearpathrobotics.com/ure-ros-setup-universal-robots-ros-driver/

5. Universal Robots. "UR3e: Ultra-lightweight, compact cobot." Available online: https://www.universal-robots.com/products/ur3-robot/

9 Appendix: Code Implementation

```
def Get_MS():
        # Fill in the correct values for S1~6, as well as the M matrix
        M = np.eye(4)
        S = np.zeros((6,6))

        S[0] = np.array([0,1,1,1,0,1])
        S[1] = np.array([0,0,0,0,-1,0])
        S[2] = np.array([1,0,0,0,0,0])
        S[3] = np.array([0,0,0,0,0.152,0])
        S[4] = np.array([0,0.152,0.152,0.152,0,0.152])
        S[5] = np.array([0,0,0.244,0.457,-0.131,0.542])


        M[0] = np.array([0,0,1,0.293])
        M[1] = np.array([0,1,0,-0.542])
        M[2] = np.array([-1,0,0,0.152])
        M[3] = np.array([0,0,0,1])
        return M, S


"""
Function that calculates encoder numbers for each motor
"""
def lab_fk(theta1, theta2, theta3, theta4, theta5, theta6):
        theta = np.array([theta1 - 0.5 * PI, theta2, theta3,
 theta4 + 0.5 * PI, theta5, theta6])
        M,S = Get_MS()
        print(f"M is :\n{M}\n\n")
        print(f"S is :\n{S}\n\n")
        T = np.eye(4)
        for i in range(6):
                T = np.dot(T, expm(skew(S[:,i])*theta[i]))
        T = np.dot(T,M)
        print("Forward kinematics calculated:\n")
        print(f"T is :\n{T}\n\n")
        return T

def skew(s):
        return np.array([
                [0, -s[2], s[1], s[3]],
                [s[2], 0, -s[0], s[4]],
                [-s[1], s[0], 0, s[5]],
                [0,0,0,0]
        ])
```