ECE 470: Lab 5
Inverse Kinematics

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

April 29, 2024

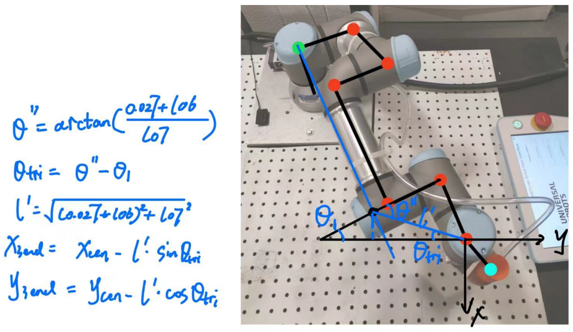
Contents

1 Introduction

In this lab, we are required to calculate and implement the inverse kinematics on the UR3e robot arm. Considering the complexity of computing the setting of robot arm, the lab simplified the condition, fixing some part of the joint with constant angle. All we need to do is to do senior-high school level calculation and implement the process in Python. After that, we need to verify the correctness with Forward Kinematics code in Lab4.

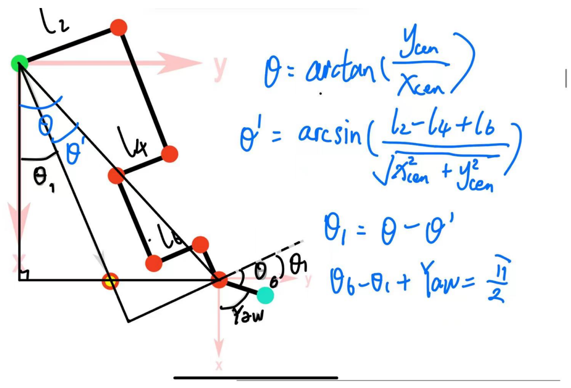
2 written derivation of the inverse kinematics solution

Below image is the hand-derivation of IK:



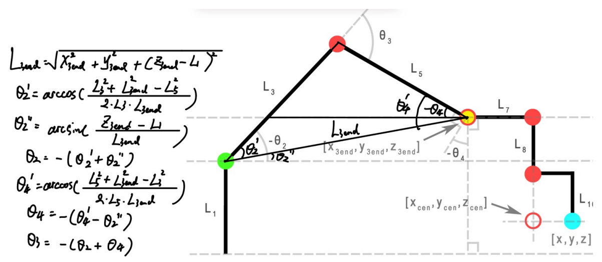
$\theta'' = \arctan\left(\frac{0.027 + L_6}{L_7}\right)$
$\theta_{tri} = \theta'' - \theta_1$
$L' = \sqrt{(0.027+L_6)^2 + L_7^2}$
$x_{3end} = x_{cen} - L' \cdot \sin\theta_{tri}$
$y_{3end} = y_{cen} - L' \cdot \cos\theta_{tri}$
θ_1
θ''
θ_{tri}
y
x

Figure 1: Physical Layout of the UR3e, we calculate the angle based on the given limb length



L_2
L_4
L_6
y
x
θ
θ'
θ_1
θ_6
θ_1
Yaw
$\theta = \arctan\left(\frac{y_{cen}}{x_{cen}}\right)$
$\theta' = \arcsin\left(\frac{L_2 - L_4 + L_6}{\sqrt{x_{cen}^2 + y_{cen}^2}}\right)$
$\theta_1 = \theta - \theta'$
$\theta_6 - \theta_1 + Yaw = \frac{\pi}{2}$

Figure 2: Top View Stick Pictorial of UR3e and Calculation



$L_{3end} = \sqrt{x_{3end}^2 + y_{3end}^2 + (z_{3end} - L_1)^2}$
$\theta_2' = \arccos\left(\frac{L_3^2 + L_{3end}^2 - L_5^2}{2 \cdot L_3 \cdot L_{3end}}\right)$
$\theta_2'' = \arcsin\left(\frac{z_{3end} - L_1}{L_{3end}}\right)$
$\theta_2 = -(\theta_2' + \theta_2'')$
$\theta_4' = \arccos\left(\frac{L_5^2 + L_{3end}^2 - L_3^2}{2 \cdot L_5 \cdot L_{3end}}\right)$
$\theta_4 = -(\theta_4' - \theta_2'')$
$\theta_3 = -(\theta_2 + \theta_4)$
θ_3
L_5
L_3
θ_4'
$-\theta_4$
L_7
L_{3end}
$-\theta_2$
θ_2'
θ_2''
$[x_{3end}, y_{3end}, z_{3end}]$
$-\theta_4$
L_8
L_1
$[x_{cen}, y_{cen}, z_{cen}]$
L_{10}
[x,y,z]

Figure 3: Side View Stick Pictorial of UR3e and Calculation

3 Source of Errors

As we learned in **CS450: Numerical Analysis**, several error can affect the accuracy and reliability of the joint angle calculations.

3.1 Numerical Approximations

The inverse kinematics solutions rely on numerical methods that approximate continuous mathematical operations. Finite-precision arithmetic in digital computers can introduce small discrepancies in calculations, particularly in trigonometric functions and matrix operations critical for robotic transformations.

3.2 Measurement and Calibration Errors

Physical discrepancies in the robot's structure, such as variations in link lengths and joint calibration, can lead to errors in the kinematics model. Misalignments and measurement inaccuracies directly impact the calculated positions of the robot's end-effector.

3.3 Assumptions in Kinematic Model

The kinematic model simplifies the robot by assuming rigid links and perfect joints. In reality, mechanical play, joint elasticity, and non-linear friction characteristics can deviate the actual movement from the computed paths, particularly under different load conditions.

3.4 Algorithmic Limitations

The algorithms used for solving inverse kinematics may not account for all possible configurations of the robot. For example, configurations near or at kinematic singularities can result in solutions that are not feasible or not unique, thus impacting the performance of the robot in executing precise movements.

3.5 Precision of Input Parameters

Finally, the precision of input parameters such as the desired position and orientation of the end-effector can significantly influence the output. Inaccuracies in specifying these inputs might be magnified through the kinematic chain, resulting in larger positional errors at the end-effector.

These factors must be carefully considered and managed to improve the accuracy of the inverse kinematics solutions implemented in robotic systems.

4 Conclusion

This lab exercise successfully derived and implemented inverse kinematics for the UR3e robot arm, integrating theoretical calculations with practical Python coding. The accuracy of our results was confirmed through comparison with forward kinematics, highlighting the effectiveness of our approach.

The project underscored the importance of addressing potential sources of error such as numerical inaccuracies and structural deviations, which are crucial for understanding both the limitations and capabilities of robotic kinematics.

Overall, this lab not only helped our understanding of robotics mechanics and control but also demonstrated the critical nature of precision in robotic engineering. The insights gained here will be valuable for future robotics and automation projects, emphasizing the practical application of theoretical knowledge.

5 Appendix: Code Implementation

Here, we merged the lab 4 and lab 5 code together into 'lab5_test.py' to fasten the verification. The code passed all test cases and got 80/80 grade in the lab demo.

```
import numpy as np
import math
from scipy.linalg import expm
import sys
import time

PI = np.pi

"""
You may write some helper functions as you need
Use 'expm' for matrix exponential
Angles are in radian, distance are in meters.
"""

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, theta2, theta3, theta4, 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]
        ])


"""
Function that calculates an elbow up Inverse Kinematic solution for the UR3
"""
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 = 0.104
        l07 = 0.085
        l08 = 0.092
        l09 = 0
        l10 = 0.07   # thickness of aluminum plate is around 0.01

        yaw_WgripDegree = yaw_WgripDegree * PI / 180

        xgrip = -yWgrip
        ygrip = xWgrip
        zgrip = zWgrip
        print("xyzgrip", xgrip, ygrip, zgrip)

        xcen = xgrip - np.cos(yaw_WgripDegree) * l09
        ycen = ygrip - np.sin(yaw_WgripDegree) * l09
        zcen = zgrip

        # theta1
        thetas[0] = math.atan2(ycen , xcen) - math.asin((0.027 + l06) / math.sqrt(xcen *
# Default value Need to Change
        # print(xcen, ycen, zcen, l02 - l04 + l06)
        # print(0.027 + l06)
    # l02 - l04 + l06 = 0.027 + l06 = 0.131

        # theta6
        thetas[5] = PI - (PI / 2 - thetas[0]) - yaw_WgripDegree      # Default value Need

        l_long = math.sqrt((l06 + 0.027) ** 2 + l07 ** 2)
        theta_tri = math.atan2(l07, (l06 + 0.027))

        x3end = xcen - l_long * math.sin(theta_tri - thetas[0])
        y3end = ycen - l_long * math.cos(theta_tri - thetas[0])
        z3end = zcen + l10 + l08
        temp = math.sqrt((z3end - l01) ** 2 + x3end ** 2 + y3end ** 2)
        print("xyz3end", x3end, y3end, z3end)

        thetas[1]= - (math.atan2((z3end - l01) , math.sqrt(x3end ** 2 + y3end ** 2)) + m
# Default value Need to Change
        thetas[2]=  PI - math.acos((l03 ** 2 + l05 ** 2 - temp ** 2) / (2 * l03 * l05))
# Default value Need to Change
        thetas[3]= - thetas[1] - thetas[2]  # Default value Need to Change
        thetas[4]= - PI / 2      # Default value Need to Change


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

        return thetas


def main():
    if len(sys.argv) != 5:
        print("\n")
        print("rosrun lab5pkg_ik lab5_exec.py xWgrip yWgrip zWgrip yaw_WgripDegree \n")
    else:
        print("\nxWgrip: " + sys.argv[1] + ", yWgrip: " + sys.argv[2] + \
              ", zWgrip: " + sys.argv[3] + ", yaw_WgripDegree: " + sys.argv[4] + "\n")

        # Assuming lab_invk and lab_fk are defined elsewhere and correctly imported
        new_dest = lab_invk(float(sys.argv[1]), float(sys.argv[2]), float(sys.argv[3]),

        T = lab_fk(new_dest[0], new_dest[1], new_dest[2], new_dest[3], new_dest[4], new_
        print("check T[0][3], T[1][3], T[2][3] with your input destination")

        print(f"T is \n\n{T}")
        print("\n\n")
        end_effct = T[0:3,3]
        print(f"end_effct is \n\n{end_effct}")
        print("\n\n")

# Ensure the following functions are defined and imported correctly:
# lab_invk() and lab_fk()

if __name__ == "__main__":
        try:
                main()
        except:
                print("error")
```