EE183DA

Assignment: Lab Report 1

Name: Zixuan Fan

UID: 904190493

Introduction

The purpose of this lab is to analyze the forward and inverse kinematics of a real-world kinematic linkage that has 4 or more joints. For this assignment, I choose to consider the right human shoulder and elbow as a simplified kinematic linkage. As illustrated in **Figure 1**, the right human shoulder can be considered as a combination of 3 (1 DOF) joints, j_1, j_2 and j_3, and the elbow as a single (1 DOF) joint j_4. For the purpose of this assignment, we regard the elbow to wrist as a rigid object, which does not consist of any joints in between. The should to elbow is labeled as link l_1 and the elbow to hand is labeled as link l_2.

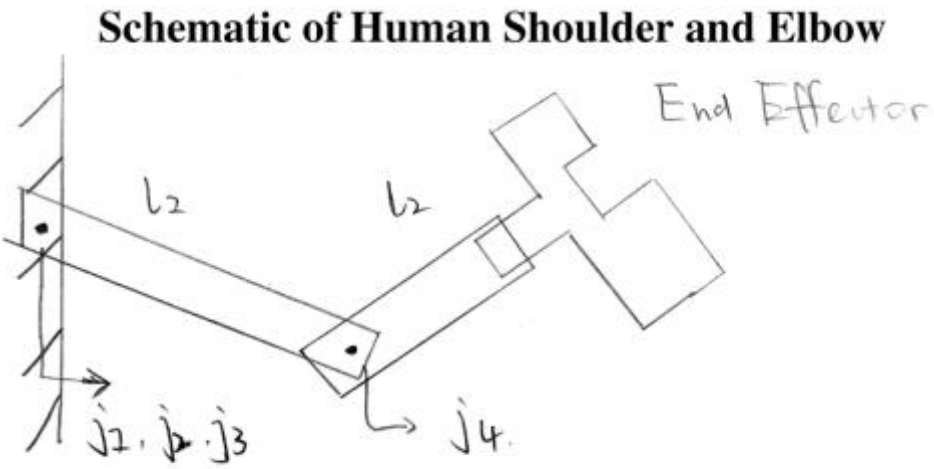

Figure 1: Schematic of human shoulder and elbow

In the operational space, the functionality of this robot is to perform a straight punch from the current position of the fist to the position of interest, a linear motion that may involve one or multiple joints to achieve the purpose. The end effector requires 6DOF state variables to describe its state. 3DOF rotational variable **R = {R_x, R_y, R_z}**, 3 DOF translational variables **d = {d_x, d_y, d_z}**. Together they make up the 3 by 4 transformation matrix **T** that we use to describe the state of the end effector in its operational space **b** to be:

$$\mathbf{T_b} = \begin{bmatrix} \boldsymbol{R_b} & \boldsymbol{d_b} \end{bmatrix} = \begin{bmatrix} x_x & y_x & z_x & d_x \\ x_y & y_y & z_y & d_y \\ x_z & y_z & z_z & d_z \end{bmatrix}$$

To make the position of the end effector in operational space homogeneous to the base frame **o,** we define the 4 by 4 homogeneous transformation matrix of the end effector to be:

$$\begin{bmatrix} \boldsymbol{p^o} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R_b^o} & \boldsymbol{d_b^o} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{p^b} \\ 1 \end{bmatrix}$$

Method

To compute the forward kinematics of the robot, we first calculate the modified *Denavit-Hartenberg parameters* of the linkage system. The geometry setup for the calculation of D-H parameters can be found in **Figure 2**.

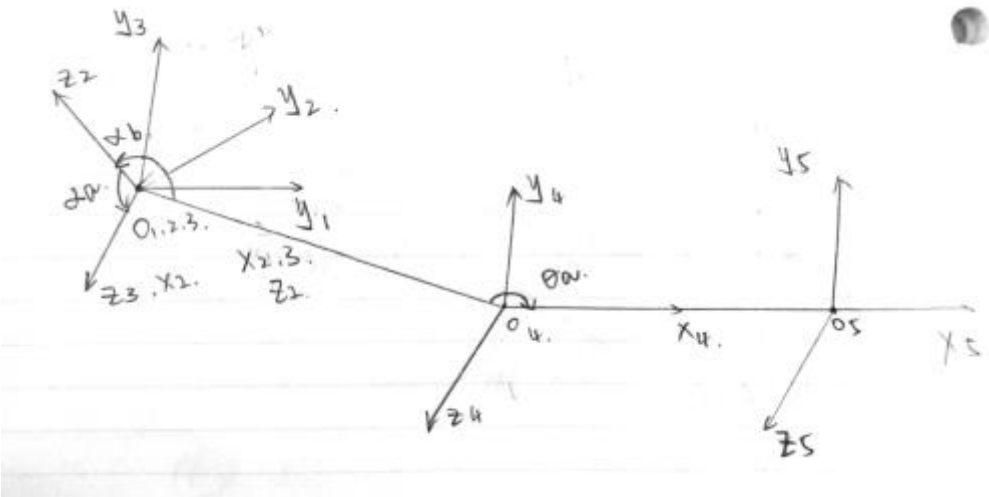

Figure 2: Geometry configurations of modified D-H parameters

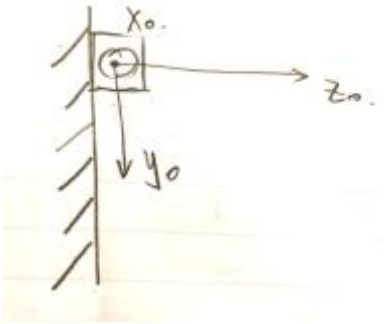

Figure 3: Configuration of the linkage base frame

We measure the values of the D-H parameters as the follows:

Parameters	i = 0	i = 1	i = 2	i = 3	i = 4	i = 5
d	0	0	0	0	0.2m	0.25m
θ	θ_x	90 °	0	θ_a	0	NaN
r	0	0	0	0.2m	0.25m	NaN
α	θ_z	90 °	90 °	0	0	NaN

Let i=5 to be our end effector. We define the base frame i=0 as shown in **Figure 3**. X_o, Y_o and Z_o represent the axes for pitch, yaw and roll motions of the shoulder joints respectively. The transformation matrices for each change of coordinates in i related to i-1 are shown as the follows:

$$\mathbf{T}_{\mathbf{5}}^{\mathbf{4}} = \begin{bmatrix} 1 & 0 & 0 & 0.25 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_{\mathbf{4}}^{\mathbf{3}} = \begin{bmatrix} cos\theta_a & -sin\theta_a & 0 & 0.2 \\ sin\theta_a & cos\theta_a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_{\mathbf{3}}^{\mathbf{2}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos90^\circ & -sin90^\circ & 0 \\ 0 & sin90^\circ & cos90^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_{\mathbf{2}}^{\mathbf{1}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos90^\circ & -sin90^\circ & 0 \\ 0 & sin90^\circ & cos90^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos90^\circ & -sin90^\circ & 0 & 0 \\ sin90^\circ & cos90^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{T}_{\boldsymbol{o}}^{\mathbf{1}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta_z & -sin\theta_x & 0 \\ 0 & sin\theta_x & cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta_z & 0 & sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta_y & 0 & cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta_z & -sin\theta_z & 0 & 0 \\ sin\theta_z & cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We then use the defined D-H parameters to generate the forward kinematics in *Python*, the script of which is provided in **Appendix I**.

$$\boldsymbol{T}_{\boldsymbol{n}}^{\boldsymbol{o}} = \boldsymbol{T}_{\mathbf{1}}^{\boldsymbol{o}} \boldsymbol{T}_{\mathbf{2}}^{\mathbf{1}} \boldsymbol{T}_{\mathbf{3}}^{\mathbf{2}} \ldots \boldsymbol{T}_{\boldsymbol{n}}^{\boldsymbol{n-1}}$$

$$\tilde{\boldsymbol{p}}^{\boldsymbol{o}} = \boldsymbol{T}_{\boldsymbol{n}}^{\boldsymbol{o}} \tilde{\boldsymbol{p}}^{\boldsymbol{n}}$$

The transformation matrix is a function of θ_x, θ_y, θ_z and θ_a. Since in our code, we have to assign a value to the variable θ_a, let $\theta_x = 0$ deg, $\theta_y = 0$ deg, $\theta_z = 0$ deg, $\theta_a = 45$ deg and when $\boldsymbol{p^5}$ ***= {0, 0, 0, 0}.*** The resulting transformation matrix is:

$$T_o^5 = \begin{bmatrix} 1.23259516e-32 & -8.65956056e-17 & 1.00000000e+00 & 1.22464680e-17 \\ -7.07106781e-01 & -7.07106781e-01 & -6.12323400e-17 & -1.76776695e-01 \\ 7.07106781e-01 & -7.07106781e-01 & -6.12323400e-17 & 3.76776695e-01 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\tilde{\mathbf{p}^o} = \mathbf{T}_5^0 \tilde{\mathbf{p}^5} = \mathbf{T}_5^0 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.22464680e-17 \\ -1.76776695e-01 \\ 3.76776695e-01 \\ 1 \end{bmatrix}$$

Since *Python* has an efficient data structure, *dictionary*, which uses hash table to store all data, it is feasible for use to simulate all results of the robotic linkage and then use the dictionary to look up the inverse kinematic parameters. The Python script to simulate all linkage motions are provided in **Appendix I.**

Results

We mentioned in the **Introduction** that the motion of performing a straight punch as the task of this kinematic linkage. In order to achieve the punch, we define the motion to be a combination of translations and rotations of the end effector in the operational space. The identical motion in the base frame can be found using the forward kinematics. Therefore, we define the motion to be moving from the default position $\boldsymbol{p^o = \{0.0, -0.18, 0.38, 1.0\}}$ to $\boldsymbol{p^o = \{0.0, 0.04, 0.41, 1.0\}}$. Using the coding from the previous implementation of inverse kinematics, we found that the configurations for the endpoint of the desired motion is $\boldsymbol{\theta_s = \{\theta x, \theta y, \theta z, \theta a\} = \{-30, 0, 40, -45\}}$ from position $\boldsymbol{\theta_d = \{\theta x, \theta y, \theta z, \theta a\} = \{0, 0, 0, 0\}}$.

With the configurations generated by inverse kinematics of the system, we plot the end effector trajectory in 3d.

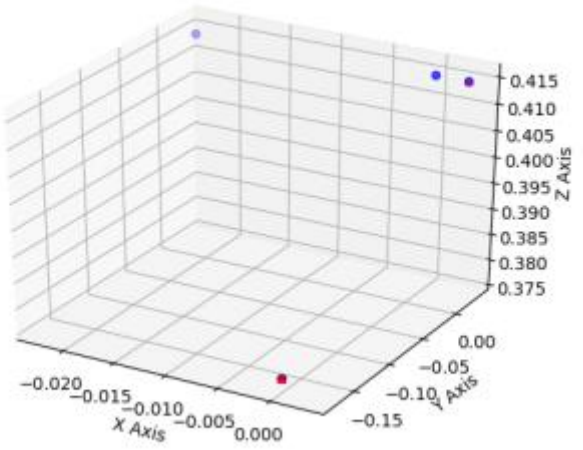

Figure 4: Trajectory of the end effector in operational space

From **Figure 4**, we observe that the single linear motion from two red points was composed of 4 rotational movements from the joints. The end effector travels more than necessary to achieve the end state. It is not efficient in terms of energy and time consumption. One way of improvement would be adding more joints to the linkage and change it into a redundant system. In this case, the end effectors will have more mapping from inverse kinematic configurations.

Conclusion

This lab was still challenging for me because I had to figure out a lot of detail configurations of the forward and inverse kinematics. The whole lab took me about 6 hours to finish, where debugging the code and reviewing the D-H parameters took most of the time.

Appendix I

Python script to generate forward and inverse kinematics

```
#!/usr/bin/python/

import numpy as np
from math import *
from pprint import pprint
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt


identityMatrix_44 = np.matrix([
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1]])

max_thetaX = 90
min_thetaX = -90
max_thetaY = 90
min_thetaY = -90
max_thetaZ = 90
min_thetaZ = -90
max_thetaA = 45
min_thetaA = -45
default_thetaA = 45

interval_level1 = 10
interval_level2 = 4
interval_level3 = 2
interval_level4 = 1

# class of transformation matrix
# forward kinematic class
# inverse kinematic class
# class of a transformation matrix

# the end effector class
class EndEffector(object):
    def __init__(self):
        self.px = 0
        self.py = 0
        self.pz = 0
        self.pos = np.matrix([[self.px], [self.py], [self.pz], [1]])

    def setPos(self, x, y, z):
        self.px = x
        self.py = y
        self.pz = z
        self.pos = np.matrix([[self.px], [self.py], [self.pz], [1]])


# the Forward Kinematics class
class FK(object):
    def __init__(self):
        self.transMatrix = identityMatrix_44
        self.endEffector = EndEffector()

    def setEndEffector(self, x, y, z):
        self.endEffector.setPos(x, y, z)

    def addTransMatrix(self, type, a, p):
        a = radians(a)

        if type == "rx":
            added = np.matrix([
```

```
            [1, 0, 0, p[0]],
            [0, cos(a), -sin(a), p[1]],
            [0, sin(a), cos(a), p[2]],
            [0, 0, 0, 1]
        ])

    elif type == "ry":
        added = np.matrix([
            [cos(a), 0, sin(a), p[0]],
            [0, 1, 0, p[1]],
            [-sin(a), 0, cos(a), p[2]],
            [0, 0, 0, 1]
        ])

    elif type == "rz":
        added = np.matrix([
            [cos(a), -sin(a), 0, p[0]],
            [sin(a), cos(a), 0, p[1]],
            [0, 0, 1, p[2]],
            [0, 0, 0, 1]
        ])

    else:
        added = identityMatrix_44

    self.transMatrix = added * self.transMatrix
    #print self.transMatrix

def calculateFK(self):
    return self.transMatrix * self.endEffector.pos

""" set default position of the robot, in its operational space,
let thetaX = 0, thetaY = 0, thetaZ = 0, thetaA = 45, calculate its coordinates in the
base frame.
"""
human_arm = FK()
human_arm.setEndEffector(0, 0, 0)
human_arm.addTransMatrix("rx", 0, [0.25, 0, 0])
human_arm.addTransMatrix("rz", default_thetaA, [0.2, 0, 0])
human_arm.addTransMatrix("rx", 90, [0, 0, 0])
human_arm.addTransMatrix("rz", 90, [0, 0, 0])
human_arm.addTransMatrix("rx", 90, [0, 0, 0])

# print human_arm.calculateFK()

lookUpTable = dict()

""" function for simulate every position of the robotic linkage
we use Python dictionary to store all possible configurations of the joints according to
every point in the operational space
"""

def simulateIK():

    for tx in np.arange(min_thetaX, max_thetaX + interval_level1, interval_level1):
        for ty in np.arange(min_thetaY, max_thetaY + interval_level1, interval_level1):
            for tz in np.arange(min_thetaZ, max_thetaZ + interval_level1, interval_level1):
                for ta in np.arange(min_thetaA, max_thetaA + interval_level1, interval_level1):
                    robot = FK()
                    robot.setEndEffector(0, 0, 0)
                    robot.addTransMatrix("rx", 0, [0.25, 0, 0])
                    robot.addTransMatrix("rz", default_thetaA, [0.2, 0, 0])
                    robot.addTransMatrix("rx", 90, [0, 0, 0])
                    robot.addTransMatrix("rz", 90, [0, 0, 0])
```

```
                robot.addTransMatrix("rx", 90, [0, 0, 0])

                robot.addTransMatrix("rx", tx, [0, 0, 0])
                robot.addTransMatrix("ry", ty, [0, 0, 0])
                robot.addTransMatrix("rz", tz, [0, 0, 0])
                robot.addTransMatrix("rz", ta, [0, 0, 0])


                loopUpKey = robot.calculateFK().tolist()

                # convert all float to integers
                for i in range(0, 3):
                    loopUpKey[i][0] = round(loopUpKey[i][0], 2)

                lookUpTable[str(loopUpKey)] = [tx, ty, tz, ta]

    return lookUpTable

# [[0.0], [0.04], [0.41], [1.0]]': [-30, 0, 40, -45],
# lookUpKey = [[0.03], [-0.03], [0.41], [1.0]]
# posIK = simulateIK()
# pprint(posIK)

# forward kinematics trajectory
pos0 = human_arm.calculateFK().tolist()
print pos0

human_arm.addTransMatrix("rx", -30, [0, 0, 0])
pos1 = human_arm.calculateFK().tolist()
print pos1

human_arm.addTransMatrix("ry", 0, [0, 0, 0])
pos2 = human_arm.calculateFK().tolist()
print pos2

human_arm.addTransMatrix("rz", 40, [0, 0, 0])
pos3 = human_arm.calculateFK().tolist()
print pos3

human_arm.addTransMatrix("rz", -45, [0, 0, 0])
pos4 = human_arm.calculateFK().tolist()
print pos4

# generate forward kinematics trajectory plots
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = [pos0[0], pos1[0], pos3[0], pos4[0]]
y = [pos0[1], pos1[1], pos3[1], pos4[1]]
z = [pos0[2], pos1[2], pos3[2], pos4[2]]
xs = [pos0[0], pos4[0]]
ys = [pos0[1], pos4[1]]
zs = [pos0[2], pos4[2]]

ax.scatter(x, y, z, c="b", marker="o")
ax.scatter(xs, ys, zs, c="r", marker="^")

ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_zlabel('Z Axis')

fig.savefig('data.png')
```