Lab 4

November 24, 2020

[5]:
```python
"""
MATLAB version AUTHOR:
    Dr. Cynthia Sung (crsung@seas.upenn.edu)
    Modified by Gedaliah Knizhnik (knizhnik@seas.upenn.edu) 08/28/19
Python version transformed AUTHOR:
    Zichen Lao (lao0910@seas.upenn.edu) 06/05/20
"""

import numpy as np

class calculateFK():

    def __init__(self):
        """
        This is the dimension of the Lynx Robot stated as global variable

        """
        # Lynx Dimensions in mm
        self.L1 = 76.2    # distance between joint 0 and joint 1
        self.L2 = 146.05  # distance between joint 1 and joint 2
        self.L3 = 187.325 # distance between joint 2 and joint 3
        self.L4 = 34      # distance between joint 3 and joint 4
        self.L5 = 34      # distance between joint 4 and center of gripper

        # Joint limits
        self.lowerLim = np.array([-1.4, -1.2, -1.8, -1.9, -2.0, -15]).
 ↪reshape((1, 6))    # Lower joint limits in radians (grip in mm (negative␣
 ↪closes more firmly))
        self.upperLim = np.array([1.4, 1.4, 1.7, 1.7, 1.5, 30]).reshape((1, 6))␣
 ↪          # Upper joint limits in radians (grip in mm)

    def forward(self, q):
        """
        INPUT:
        q - 1x6 vector of joint inputs [q0,q1,q2,q3,q4,lg]

        OUTPUTS:
```

```
        jointPositions - 6 x 3 matrix, where each row represents one
                    joint along the robot. Each row contains the [x,y,z]
                    coordinates of the respective joint's center (mm). For
                    consistency, the first joint should be located at
                    [0,0,0].
        T0e         - a 4 x 4 homogeneous transformation matrix,
                    representing the end effector frame expressed in the
                    base (0) frame
        """
        # Your code starts from here
        # Frame 1 w.r.t Frame 0
        T1 = np.array([[np.cos(q[0]), -np.sin(q[0])*np.cos(-np.pi/2), np.
↪sin(q[0])*np.sin(-np.pi/2), 0],
                       [np.sin(q[0]), np.cos(q[0])*np.cos(-np.pi/2), -np.
↪cos(q[0])*np.sin(-np.pi/2), 0],
                       [0, np.sin(-np.pi/2), np.cos(-np.pi/2), self.L1],
                       [0, 0, 0, 1]])

        # Frame 2 w.r.t Frame 1
        T2 = np.array([[np.cos(q[1]-(np.pi/2)), -np.sin(q[1]-(np.pi/2)), 0,␣
↪self.L2*np.cos(q[1]-(np.pi/2))],
                       [np.sin(q[1]-(np.pi/2)), np.cos(q[1]-(np.pi/2)), 0, self.
↪L2*np.sin(q[1]-(np.pi/2))],
                       [0, 0, 1, 0],
                       [0, 0, 0, 1]])

        # Frame 3 w.r.t Frame 2
        T3 = np.array([[np.cos(q[2]+(np.pi/2)), -np.sin(q[2]+(np.pi/2)), 0,␣
↪self.L3*np.cos(q[2]+(np.pi/2))],
                       [np.sin(q[2]+(np.pi/2)), np.cos(q[2]+(np.pi/2)), 0, self.
↪L3*np.sin(q[2]+(np.pi/2))],
                       [0, 0, 1, 0],
                       [0, 0, 0, 1]])

        # Frame 4 w.r.t Frame 3
        T4 = np.array([[np.cos(q[3]-(np.pi/2)), -np.sin(q[3]-(np.pi/2))*np.
↪cos(-np.pi/2), np.sin(q[3]-(np.pi/2))*np.sin(-np.pi/2), 0],
                       [np.sin(q[3]-(np.pi/2)), np.cos(q[3]-(np.pi/2))*np.
↪cos(-np.pi/2), -np.cos(q[3]-(np.pi/2))*np.sin(-np.pi/2), 0],
                       [0, np.sin(-np.pi/2), np.cos(-np.pi/2), 0],
                       [0, 0, 0, 1]])
        # Frame 5 w.r.t Frame 4
        T5 = np.array([[np.cos(q[4]), -np.sin(q[4]), 0, 0],
                       [np.sin(q[4]), np.cos(q[4]), 0, 0],
                       [0, 0, 1, self.L4 + self.L5],
                       [0, 0, 0, 1]])
```

```
        x = np.empty((6, 4)).reshape((6, 4))
        zeroPos = np.array([0, 0, 0, 1]).reshape((1, 4))
        zeroPos_trans = np.transpose(zeroPos)

        # Position of First Joint (Base Revolute)
        x[0, :] = zeroPos

        # Position of Second Joint (Shoulder Revolute)
        x[1, :] = np.transpose(T1.dot(zeroPos_trans))

        # Position of Third Joint (Elbow Revolute)
        x[2, :] = np.transpose((T1.dot(T2)).dot(zeroPos_trans))

        # Position of Fourth Joint (1st Wrist)
        x[3, :] = np.transpose(((T1.dot(T2)).dot(T3)).dot(zeroPos_trans))

        # Position of Fifth Joint (2nd Wrist)
        x[4, :] = np.transpose((((T1.dot(T2)).dot(T3)).dot(T4)).dot(np.
 ↪array([0, 0, self.L4, 1]).reshape((4, 1))))

        # Position of Gripper (Base of the Gripper)
        x[5, :] = np.transpose(((((T1.dot(T2)).dot(T3)).dot(T4)).dot(T5)).
 ↪dot(zeroPos_trans))
        # Outputs the 6x3 of the locations of each joint in the Base Frame
        jointPositions = x[0:6,0:3]

        T0e = ((((T1.dot(T2)).dot(T3)).dot(T4)).dot(T5))
        # Your code ends here

        return jointPositions, T0e
```

[1431]:
```
import numpy as np


def FK_velocity (q, dq, joint):
    """
    :param q: 1 x 6 vector corresponding to the robot's current configuration
    :param dq: 1 x 6 vector corresponding to the robot's current joint␣
 ↪velocities
    :param joint: an integer in [0,6] corresponding to which joint you are␣
 ↪tracking
    :return:
    v     - The resulting linear velocity in the world frame
    omega - The resulting angular velocity in the world frame
    """
#     d1 = 76.2                      # Distance between joint 0 and joint 1
```

```
#     a2 = 146.05                    # Distance between joint 1 and joint 2
#     a3 = 187.325                   # Distance between joint 2 and joint 3
#     d4 = 34                        # Distance between joint 3 and joint 4
#     d5 = 68                        # Distance between joint 3 and joint 5
#     lg = 0                         # Distance between joint 5 and end␣
↪effector (gripper length)

    v = np.array([0, 0, 0])
    omega = np.array([0, 0, 0])

    if(joint == 0 or joint >= 6): return([0.0, 0.0, 0.0],
                                   [0.0, 0.0, 0.0])

#FK_Velocity
    Jac=calcJacobian(q,joint)
#     print(Jac.shape)
    FKvel=np.matmul(Jac,dq).T #[:joint+1]
    #print(FKvel)
    v=FKvel[0:3]
    omega=FKvel[3:6]
    return v, omega
```

[9]:
```
def calcJacobian(q, joint):

    Jacv=np.zeros((3,6)) #zeroes after col: joint+1
    Jacw=np.zeros((3,6)) #zeroes after col: joint+1

    #For loop with the size of 3 x joint
    R=calculateFK()

    z=[] # np.array(3,joint)
    z.append(np.array([0,0,1]))
    z.append(np.array([-np.sin(q[0]),np.cos(q[0]),0]))
    z.append(np.array([-np.sin(q[0]),np.cos(q[0]),0]))
    z.append(np.array([-np.sin(q[0]),np.cos(q[0]),0]))

    Jp,T0e=R.forward(q)
    z.append([T0e[0,2],T0e[1,2],T0e[2,2]])
    z.append([T0e[0,2],T0e[1,2],T0e[2,2]])

    for i in range (1,joint+1):
        Jo=Jp[5]-Jp[i-1]
        Jr=np.cross(z[i-1],Jo)
        Jacv[0,i-1]=Jr[0]
        Jacv[1,i-1]=Jr[1]
        Jacv[2,i-1]=Jr[2]
```

```
    z = np.array(z)
    z[joint+1:len(z)] *= 0
    Jacw=np.transpose(np.array(z))
    return (np.append(Jacv, Jacw, axis=0))
```

[3]:
```
# import numpy as np
# def calcJacobian(q, joint):
#     """
#     Calculate the Jacobian of a particular joint of the  robot in a given␣
↪configuration
#     :param q: 1 x 6 vector of joint inputs [q1,q2,q3,q4,q5,q6]
#     :param joint: scalar in [0,6] representing which joint we care about
#     :return: J - 6 x (joint-1) matrix representing the Jacobian
#     """

#     if joint <= 1:
#         return np.array([])
#     FK = calculateFK()
#     jointPositions, T0 = FK.forward(q)

#     Jw1 = np.zeros((3, joint-1))

#     for i in range(joint-1):
#         Jw1[:, i] = T0[range(3), 2, i]

#     Jv2 = np.zeros((3, joint-1))
#     for i in range(joint-1):
#         Jv2[:, i] = np.cross(T0[range(3), 2, i], jointPositions[joint-1, :].T␣
↪- T0[range(3), 3, i])
#     J = np.vstack((Jv2, Jw1))
#     return J
```

[7]:
```
#INVERSE KINEMATIC VELOCITY
def IK_velocity (q, v, omega, joint):
    Jac=calcJacobian(q,joint)
    vel=np.append(v,omega)
    vind=np.argwhere(np.isnan(vel))
    filteredJac=np.delete(Jac,vind.T,0)
    nv=np.delete(vel,vind.T,0).T
    dq=np.matmul(np.linalg.pinv(filteredJac),nv)
    return dq
```

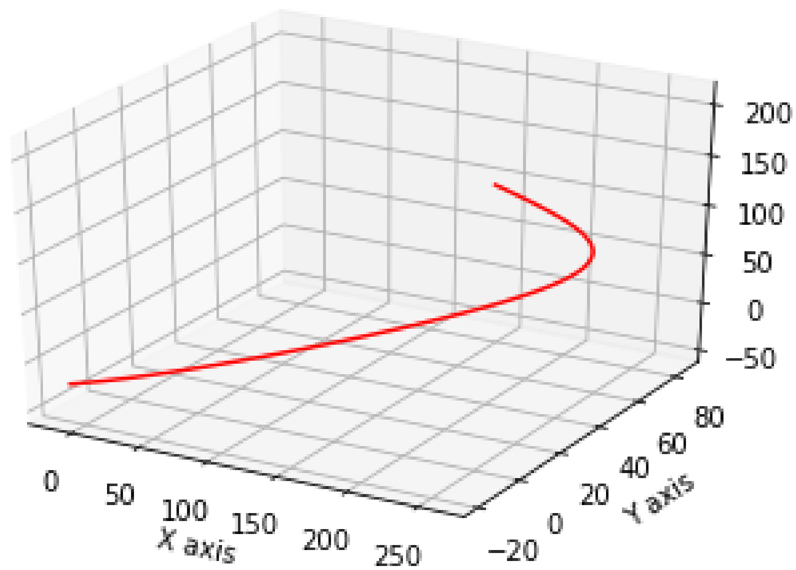
[11]:
```
import re, seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap
```

```
import pandas as pd
import seaborn as sns
```

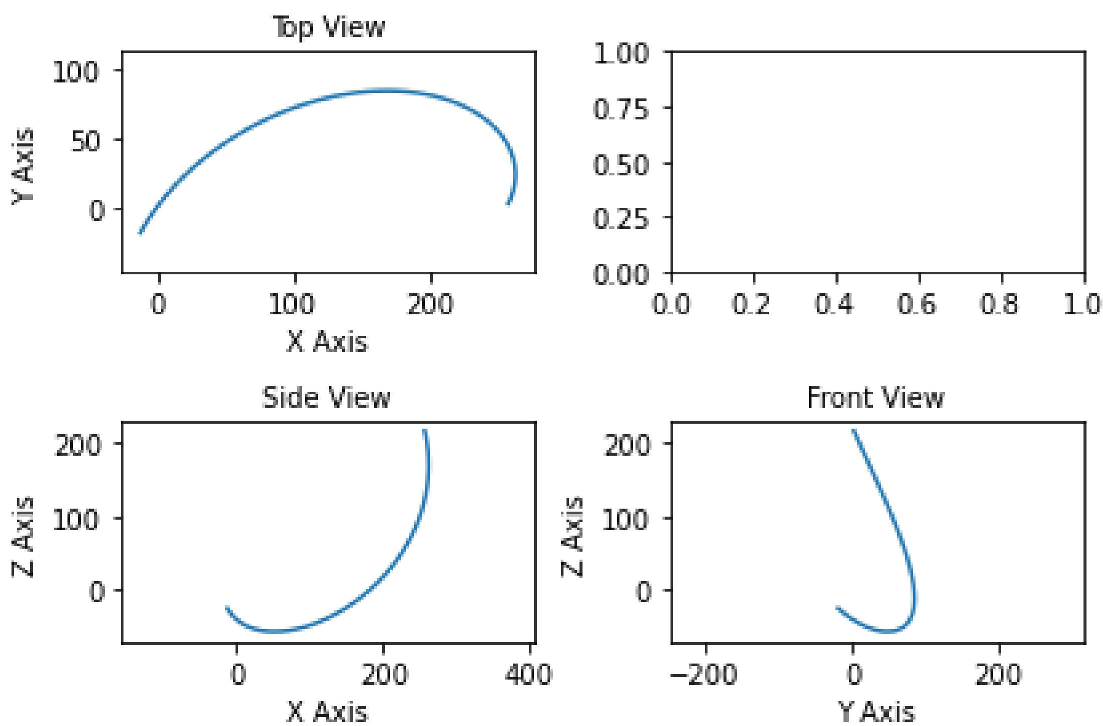
[1778]:

```
#PART 3 EVALUATION: CONSTANT DQ TRAJECTORY
# Data for a three-dimensional line
zlinet = []
xlinet =[]
ylinet = []
#for loop through different time steps
q=np.zeros((6))
x = 0.01
dq=np.array([x]*6)
R = calculateFK()
jointNum = 5
for t in range(0,100):
    q=q+dq
    Jp,_=R.forward(q)
    xlinet.append(Jp[jointNum][0])
    ylinet.append(Jp[jointNum][1])
    zlinet.append(Jp[jointNum][2])
ax = plt.axes(projection='3d')
ax.plot3D(xlinet, ylinet, zlinet, 'red')
plt.xlabel("X axis")
plt.ylabel("Y axis")

fig, axs = plt.subplots(2, 2)

axs[0, 0].plot(xlinet, ylinet)
axs[0, 0].set_title('Top View', fontsize=10)
axs[0, 0].axis('equal')
axs[0, 0].set_xlabel("X Axis")
axs[0, 0].set_ylabel("Y Axis")


axs[1, 0].plot(xlinet, zlinet)
axs[1, 0].set_title('Side View', fontsize=10)
axs[1, 0].axis('equal')
axs[1, 0].set_xlabel("X Axis")
axs[1, 0].set_ylabel("Z Axis")

axs[1, 1].plot(ylinet, zlinet)
axs[1, 1].set_title('Front View', fontsize=10)
axs[1, 1].axis('equal')
axs[1, 1].set_xlabel("Y Axis")
axs[1, 1].set_ylabel("Z Axis")

fig.tight_layout()
```

```
plt.show()
```



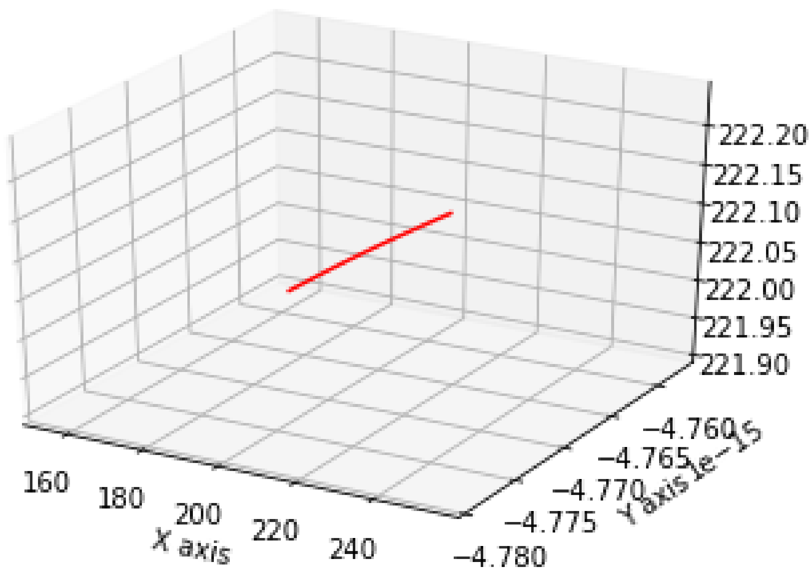
200
150
100
50
0
−50
0
50
100
150
200
250
X axis
−20
0
20
40
60
80
Y axis

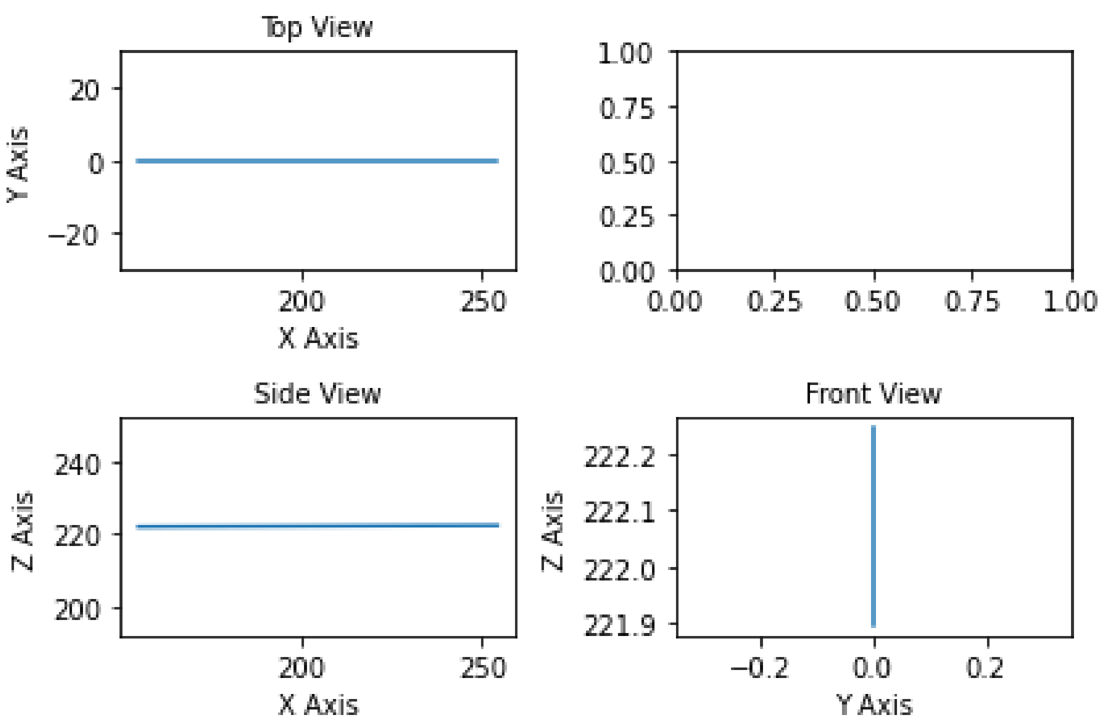


Top View
Y Axis
X Axis
100
50
0
0
100
200
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0
Side View
Z Axis
X Axis
200
100
0
0
200
400
Front View
Z Axis
Y Axis
200
100
0
−200
0
200

[1781]:
```
#PART 3 EVALUATION: LINEAR TRAJECTORY
# Data for a three-dimensional line
zlinel = []
xlinel =[]
ylinel = []
v=np.array([-1,0,0])
omega=np.array([0]*3)

#for loop through different time steps
q=np.zeros((6))
x = 0.01
R = calculateFK()
jointNum = 5
qdot=IK_velocity (q, v, omega, jointNum)
for t in range(0,100):
    q=q+qdot
    qdot=IK_velocity (q, v, omega, jointNum)
    Jp,_=R.forward(q)
    xlinel.append(Jp[jointNum][0])
    ylinel.append(Jp[jointNum][1])
    zlinel.append(Jp[jointNum][2])

ax = plt.axes(projection='3d')
ax.plot3D(xlinel, ylinel, zlinel, 'red')
plt.xlabel("X axis")
plt.ylabel("Y axis")

fig, axs = plt.subplots(2, 2)

axs[0, 0].plot(xlinel, ylinel)
axs[0, 0].set_title('Top View', fontsize=10)
axs[0, 0].axis('equal')
axs[0, 0].set_xlabel("X Axis")
axs[0, 0].set_ylabel("Y Axis")


axs[1, 0].plot(xlinel, zlinel)
axs[1, 0].set_title('Side View', fontsize=10)
axs[1, 0].axis('equal')
axs[1, 0].set_xlabel("X Axis")
axs[1, 0].set_ylabel("Z Axis")

axs[1, 1].plot(ylinel, zlinel)
axs[1, 1].set_title('Front View', fontsize=10)
axs[1, 1].axis('equal')
axs[1, 1].set_xlabel("Y Axis")
axs[1, 1].set_ylabel("Z Axis")
```

```
fig.tight_layout()

plt.show()
```

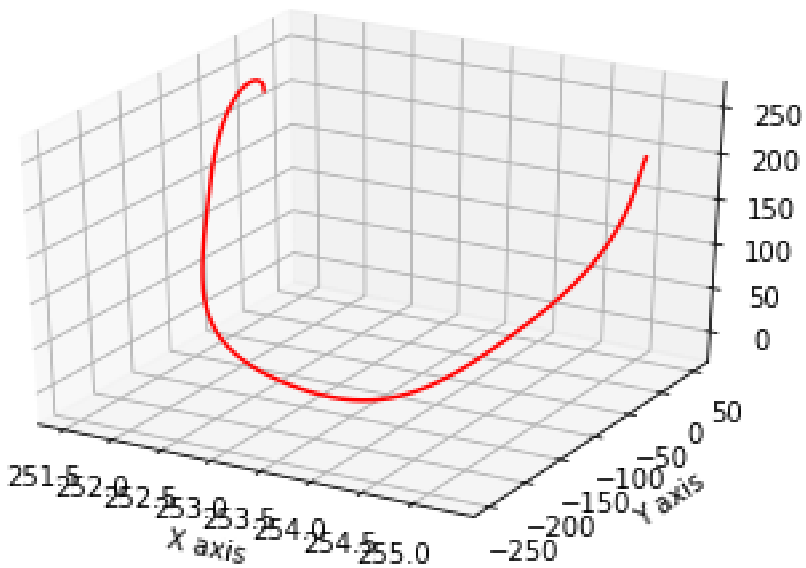




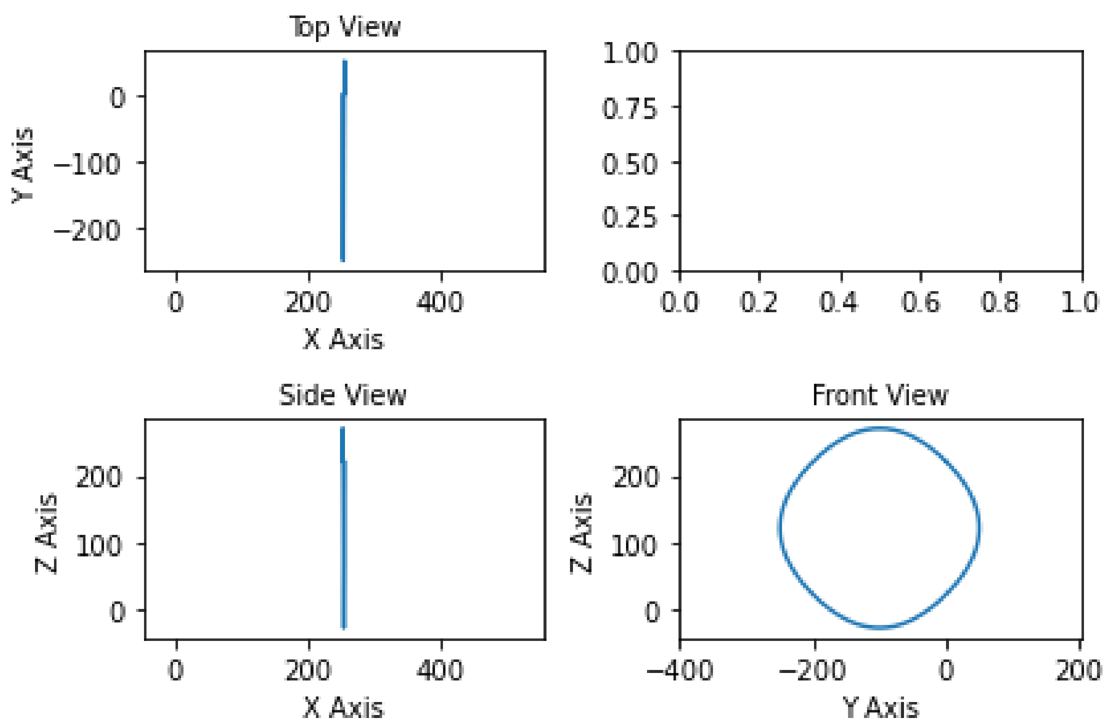
[12]:
```
#PART 3 EVALUATION: CIRCLE TRAJECTORY
# Data for a three-dimensional line
zline = []
xline =[]
yline = []
mf=2
v=np.array([0,1*mf,-1*mf])
omega=np.array([np.NaN,np.NaN,np.NaN])

#for loop through different time steps
q=np.array([0,0,0,0,0,0])
x = 0.01
R = calculateFK()
jointNum = 5
#y=np.sqrt(25-(x**2))
#qdot=IK_velocity (q, v, omega, jointNum)
IV=100
for t in range(0,IV):
    qdot=IK_velocity (q, v, omega, jointNum)
    q=q+qdot
    #v=v+np.array([0,0.01,-0.01])
    v1=(-1-1)/IV*mf
    v2=(-1+1)/IV*mf
    v=v+np.array([0,v1,v2])
    #print(np.round(v[1]**2,6))
    #omega=np.array([np.NaN,0,np.NaN])
    #v,omega=FK_velocity (q, qdot,jointNum)
    Jp,_=R.forward(q)
    xline.append(Jp[5][0])
    yline.append(Jp[5][1])
    zline.append(Jp[5][2])
    #print(Jp[jointNum])
#v=np.array([0,-1,-1])
for t in range(0,IV):
    qdot=IK_velocity (q, v, omega, jointNum)
    q=q+qdot
    v1=(-1+1)/IV*mf
    v2=(1+1)/IV*mf
    v=v+np.array([0,v1,v2])
    Jp,_=R.forward(q)
    xline.append(Jp[5][0])
    yline.append(Jp[5][1])
    zline.append(Jp[5][2])
#v=np.array([0,-1,1])
for t in range(0,IV):
    qdot=IK_velocity (q, v, omega, jointNum)
```

```
    q=q+qdot
    v1=(1+1)/IV*mf
    v2=(1-1)/IV*mf
    v=v+np.array([0,v1,v2])
    Jp,_=R.forward(q)
    xline.append(Jp[5][0])
    yline.append(Jp[5][1])
    zline.append(Jp[5][2])
#v=np.array([0,1,1])
for t in range(0,IV):
    qdot=IK_velocity (q, v, omega, jointNum)
    q=q+qdot
    v1=(1-1)/IV*mf
    v2=(-1-1)/IV*mf
    v=v+np.array([0,v1,v2])
    Jp,_=R.forward(q)
    xline.append(Jp[5][0])
    yline.append(Jp[5][1])
    zline.append(Jp[5][2])

ax = plt.axes(projection='3d')
ax.plot3D(xline, yline, zline, 'red')
plt.xlabel("X axis")
plt.ylabel("Y axis")

fig, axs = plt.subplots(2, 2)

axs[0, 0].plot(xline, yline)
axs[0, 0].set_title('Top View', fontsize=10)
axs[0, 0].axis('equal')
axs[0, 0].set_xlabel("X Axis")
axs[0, 0].set_ylabel("Y Axis")


axs[1, 0].plot(xline, zline)
axs[1, 0].set_title('Side View', fontsize=10)
axs[1, 0].axis('equal')
axs[1, 0].set_xlabel("X Axis")
axs[1, 0].set_ylabel("Z Axis")

axs[1, 1].plot(yline, zline)
axs[1, 1].set_title('Front View', fontsize=10)
axs[1, 1].axis('equal')
axs[1, 1].set_xlabel("Y Axis")
axs[1, 1].set_ylabel("Z Axis")

fig.tight_layout()
```

```
plt.show()
```



251.5
252.0
252.5
253.0
253.5
254.0
254.5
255.0
X axis
50
0
−50
−100
−150
−200
−250
Y axis
250
200
150
100
50
0



Top View
Y Axis
X Axis
Side View
Z Axis
X Axis
Front View
Z Axis
Y Axis

[1756]:
```
# axes instance
fig = plt.figure(figsize=(6,6))
ax = Axes3D(fig)

# get colormap from seaborn
cmap = ListedColormap(sns.color_palette("husl", 256).as_hex())

# plot
sc = ax.scatter(xline, yline, zline, s=40, c=xline, marker='o', cmap=cmap,␣
 ↪alpha=1)
ax.set_xlabel('X ')
ax.set_ylabel('Y ')
ax.set_zlabel('Z ')

# legend
plt.legend(*sc.legend_elements(), bbox_to_anchor=(1.05, 1), loc=2)

# save
plt.savefig("scatter_hue", bbox_inches='tight')
```