ENPM673

Perception for Autonomous Robots

Project-1 Report

Venkata Sai Sricharan Kasturi

UID:119444788

Problem 1: In the given video, a red ball is thrown against a wall. Assuming that the trajectory of the ball follows the equation of a parabola:

1. Detect and plot the pixel coordinates of the center point of the ball in the video. (Hint: Read the video using OpenCV's inbuilt function. For each frame, filter the red channel)

2. Use Standard Least Squares to fit a curve to the extracted coordinates. For the estimated parabola you must,

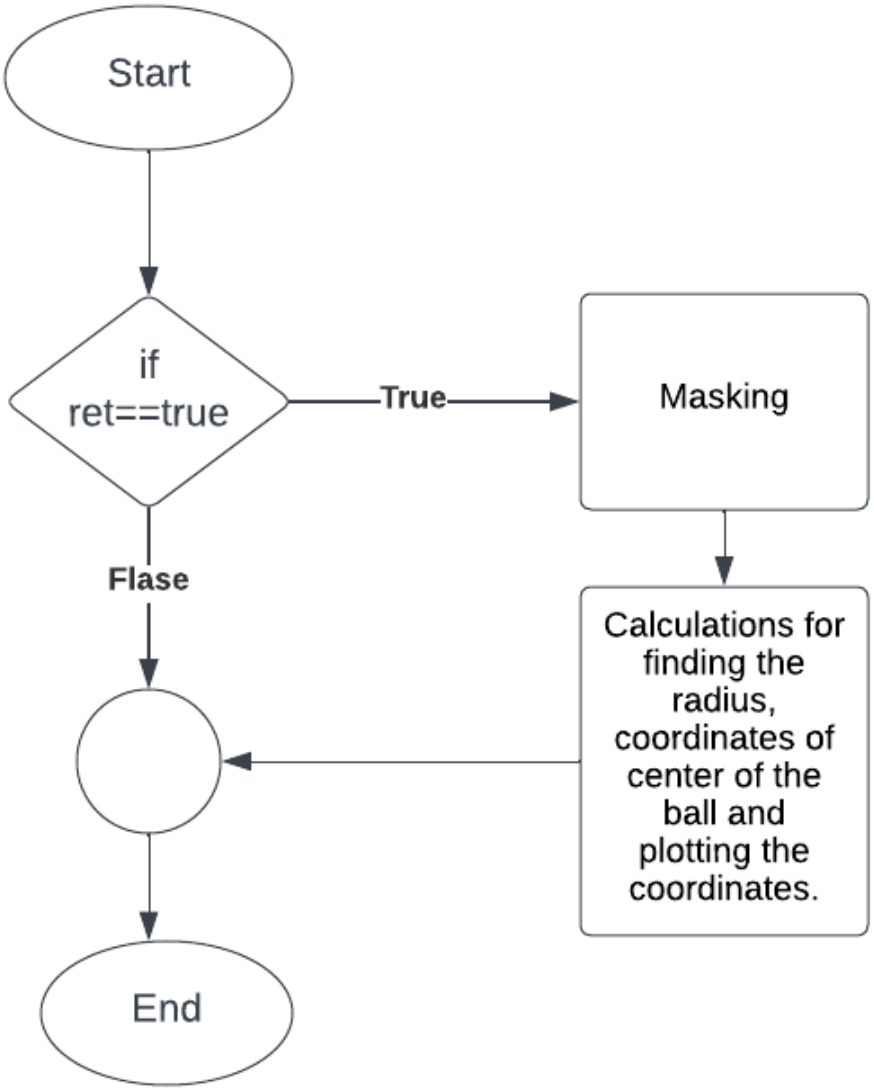
a. Print the equation of the curve.

b. Plot the data with your best fit curve.

3. Assuming that the origin of the video is at the top-left of the frame as shown below, compute the x-coordinate of the ball's landing spot in pixels, if the y-coordinate of the landing spot is defined as 300 pixels greater than its first detected location.

Problem 1:

1)



Pipeline of 1.1

Problems encountered:

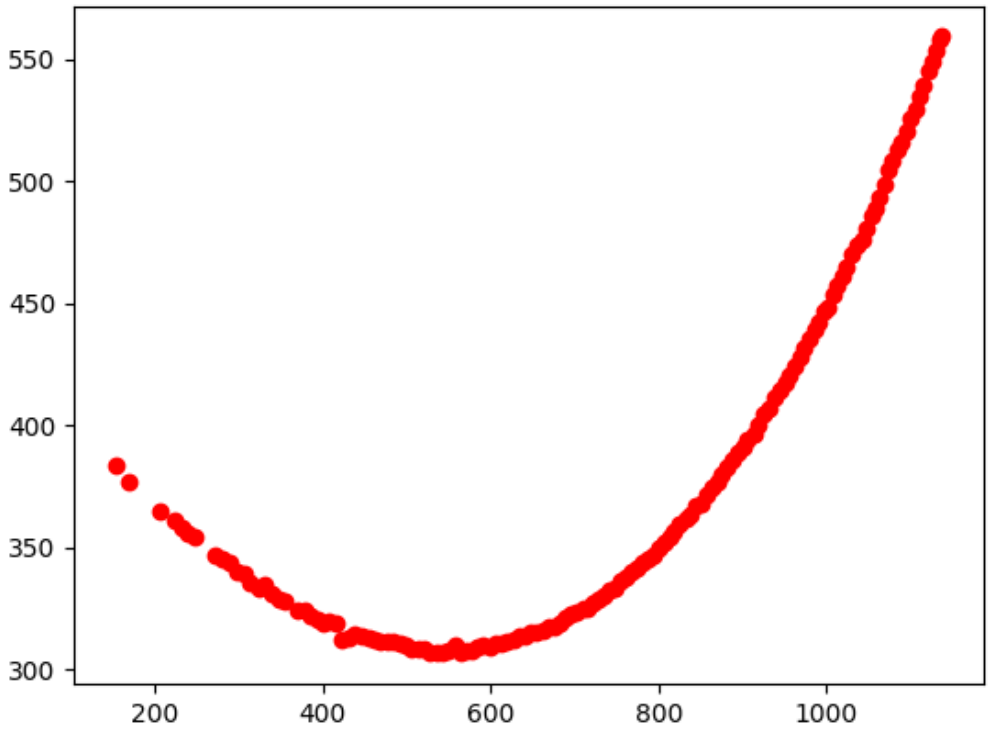
1) Importing the video file: Initially I faced issues importing the video file due to the path issues. Resolved them creating the proper path.
2) Filtering: Initially I converted the colors from BGR to RGB. The RGB values are not completely matching with the ball color. Then resolved this issue by converting to HSV.

Code:

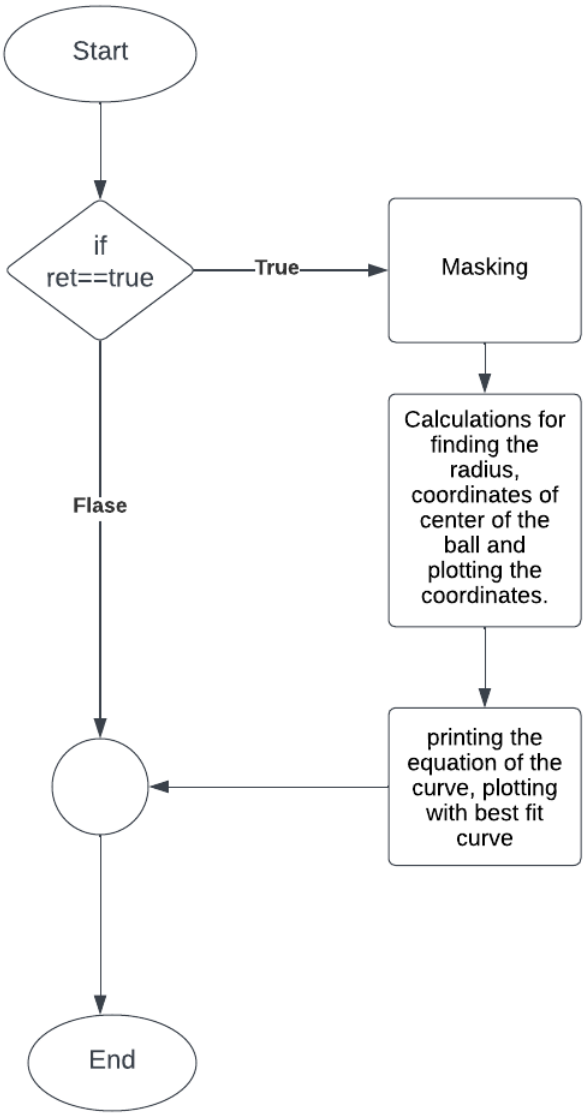
```
import cv2 as cv,os
import numpy as np
import matplotlib.pyplot as plt,math
from mpl_toolkits.mplot3d import Axes3D
import csv as csv
import random


# # Creating path
CURRENT_DIR = os.path.dirname(__file__)
path=os.path.join(CURRENT_DIR,'ball.mov')
# Loading the video
vid = cv.VideoCapture(path)

def ball_detection(xy,mask):
    count=0
    minx=0
    miny=0
    for x in range(mask.shape[0]):
        for y in range(mask.shape[1]):
            if mask[x][y]==255:
                if x<minx or minx==0:
                    minx=x
                if y<miny or miny==0:
                    miny=y
                count+=1
    #radius
    # print(count)
    # print(xy)
    if count!=0 and count>50:
        r=float(math.sqrt(count/3.14))
        centre_x= minx+r
        centre_y=miny+r
        xy=np.append(xy,[[centre_x,centre_y]],0)
    return xy
```

```
xy=np.empty((0,2))
while(vid.isOpened()):
    ret,frame=vid.read()
    if ret==True:
        hsv=cv.cvtColor(frame,cv.COLOR_BGR2HSV)

        red_1=np.array([7,5,10])
        red_2=np.array([30,255,255])
        mask=cv.inRange(frame,red_1,red_2)
        mask=cv.rectangle(mask,(650,0),(mask.shape[1],300),(0,0,0),-1)
        mask=cv.rectangle(mask,(0,465),(400,1000),(0,0,0),-1)
        mask=cv.rectangle(mask,(500,500),(1000,800),(0,0,0),-1)

        cv.imshow('frame',frame)

        key =cv.waitKey(1)
        if key ==ord('q'):
            break
        xy=ball_detection(xy,mask)
    else:
        break


#plot
x=np.linspace(0,1400,10000)
y=np.linspace(0,700,10000)
vid.release()
cv.destroyAllWindows()
plt.plot(xy[:,1],xy[:,0],'ro')

plt.show()
```

Output:



2)



Pipeline of 1.2

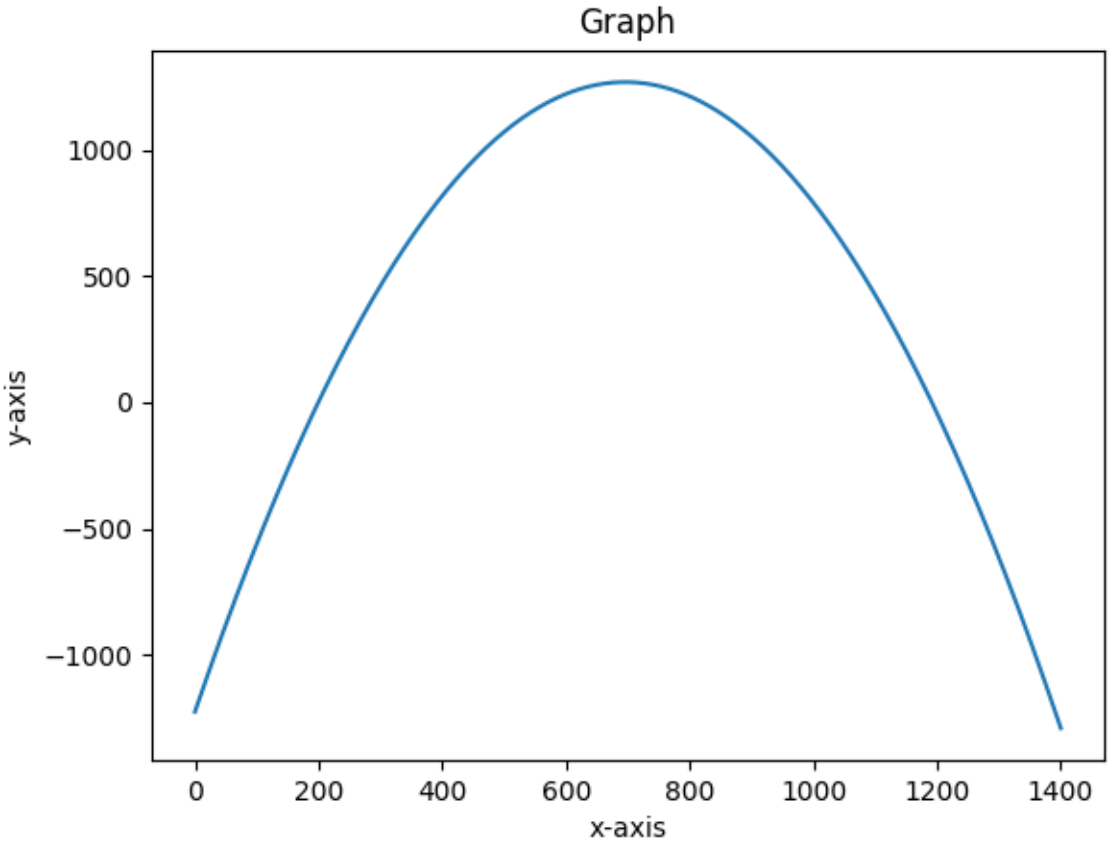
Problems encountered:

1) Converting the data to matrix form. Resolved by using the vstack() and transposing.
2) Mathematical calculations: Initially confused to find the inverse of matrix without using inbuilt functions later I figured it out.
3) Plotting graph

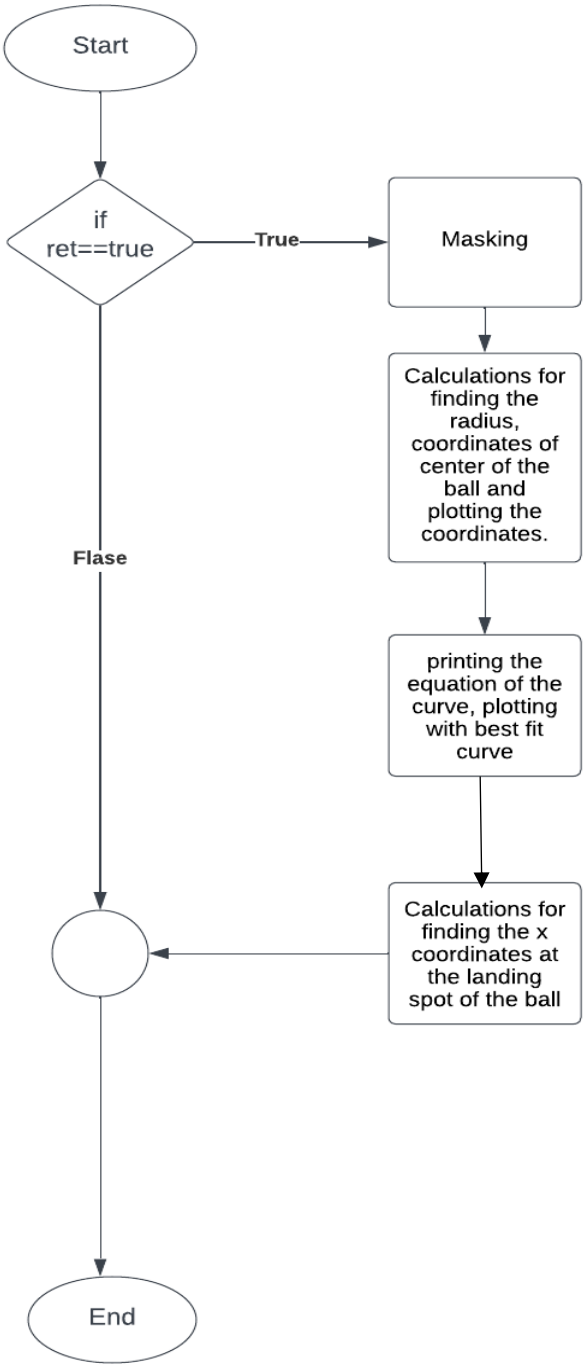
Code:

```
n=np.shape([x])
x_1=xy[:,0]**2
x_2=xy[:,0]*1
a1=np.array([x_1])
a2=np.array([x_2])
a3=np.ones_like(x_1)

A=np.vstack((a1,a2,a3))
A_T=A.T
y=xy[:,1]
# Y=np.array([y1 for y1 in y]).T
Y=np.vstack(y)
Y_T=Y.T
# inverse
A_T_inv=np.linalg.inv(np.dot(A,A_T))
P=np.dot(A_T,A_T_inv)
B=np.dot(Y_T,P)
# graph
print(B)
a=B[0:,0]
b=B[0:,1]
c=B[0:,2]

y = a*x**2 + b*x + c
plt.plot(x, y)

# Add labels and title
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title("Graph")

# Display the graph
plt.show()
```

Output:


Graph
1000
500
0
−500
−1000
y-axis
0
200
400
600
800
1000
1200
1400
x-axis

3)



Pipeline of 1.3

Problems encountered:

No problems encountered for this problem.

Code:

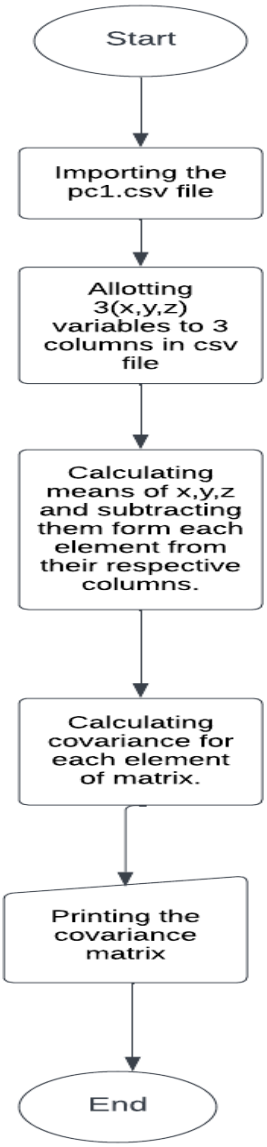
```
y_new=xy[0][1]+300
# new equation
c_new=c-y_new
discriminant = b**2 - 4*a*c_new

# check if the roots are real or complex
if discriminant < 0:
    print("The roots are complex.")
else:
    # calculate the roots using the quadratic formula
    root1 = (-b + math.sqrt(discriminant)) / (2*a)
    root2 = (-b - math.sqrt(discriminant)) / (2*a)
    #nprint("The roots are:", root1, "and", root2)
    x_new=np.maximum(root1,root2)
    print("The x-coordinate of the ball’s landing spot in pixels is:",x_new)
```

Output:

```
PS C:\Users\chara> & C:/Users/chara/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/chara/Downloads/problem_1.py
[[-5.15693853e-03  7.17341646e+00 -1.22596602e+03]]
The x-coordinate of the ball’s landing spot in pixels is: [1093.44285852]
PS C:\Users\chara> []
```

Problem 2)

1) **A)**



Pipeline of 2.1.1

Problems encountered:

1) Importing the pc1 file: Initially I used “pandas” to import and read the csv file. This resulted in deleting the zeroth row from the file. Later used “csv” to import and the file.

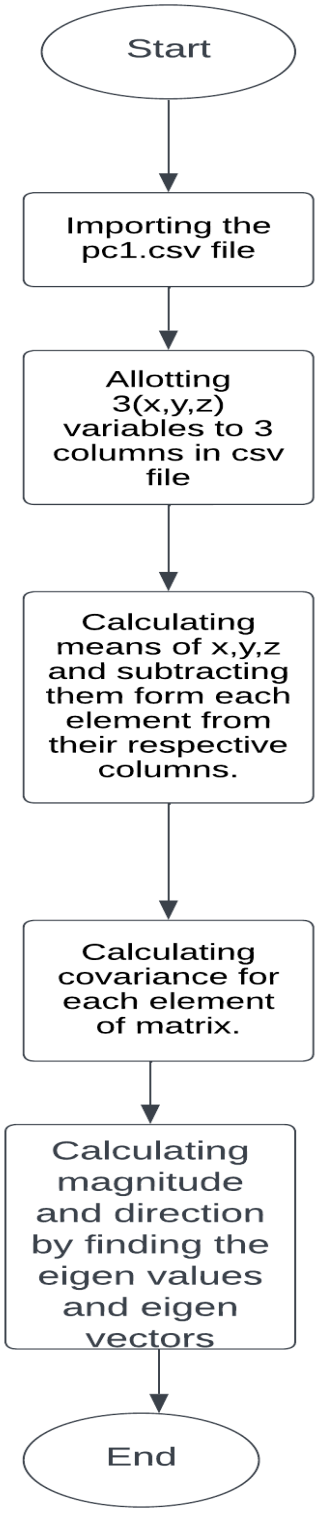
Code:

```
path="C:/Users/chara/Downloads/pc1.csv"          # Change path here
pc1 = np.genfromtxt(path,delimiter=',')

ck=np.array(pc1)

x=np.array(ck)
X=x[:,0]
y=np.array(ck)
Y=y[:,1]
z=np.array(ck)
Z=z[:,2]

X_mean=np.mean(X)
Y_mean=np.mean(Y)
Z_mean=np.mean(Z)
# Subtract the means from each variable
X_diff = X - X_mean
Y_diff = Y - Y_mean
Z_diff = Y - Z_mean

# Calculate the covariance matrix
cov_matrix = np.zeros((3, 3))
cov_matrix[0, 0] = np.sum(X_diff * X_diff) / (len(X) - 1)
cov_matrix[0, 1] = np.sum(X_diff * Y_diff) / (len(X) - 1)
cov_matrix[0, 2] = np.sum(X_diff * Z_diff) / (len(X) - 1)
cov_matrix[1, 0] = cov_matrix[0, 1]
cov_matrix[1, 1] = np.sum(Y_diff * Y_diff) / (len(Y) - 1)
cov_matrix[1, 2] = np.sum(Y_diff * Z_diff) / (len(Y) - 1)
cov_matrix[2, 0] = cov_matrix[0, 2]
cov_matrix[2, 1] = cov_matrix[1, 2]
cov_matrix[2, 2] = np.sum(Z_diff * Z_diff) / (len(Z) - 1)

print("The covarianvce matrix is:",cov_matrix)
```

Output:

```
The covarianvce matrix is: [[33.7500586  -0.82513692 -0.82513692]
 [-0.82513692 35.19218154 35.19218154]
 [-0.82513692 35.19218154 46.89061184]]
```

B)



Pipeline of 2.1.2

Problems encountered:

1) Faced problems while calculating the magnitude and direction.

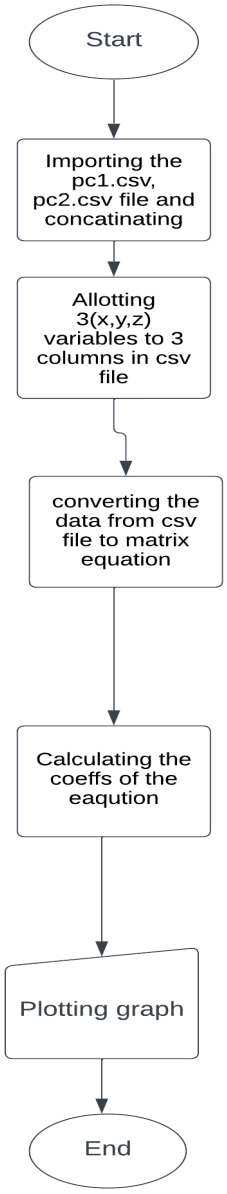
Code:

```
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
j=np.argmin(eigenvalues)
normal=eigenvectors[:,j]
magnitude=np.sqrt(normal.dot(normal))
print("The direction of the normal surface is", normal)
print("The magnitude of the normal is", magnitude)
```

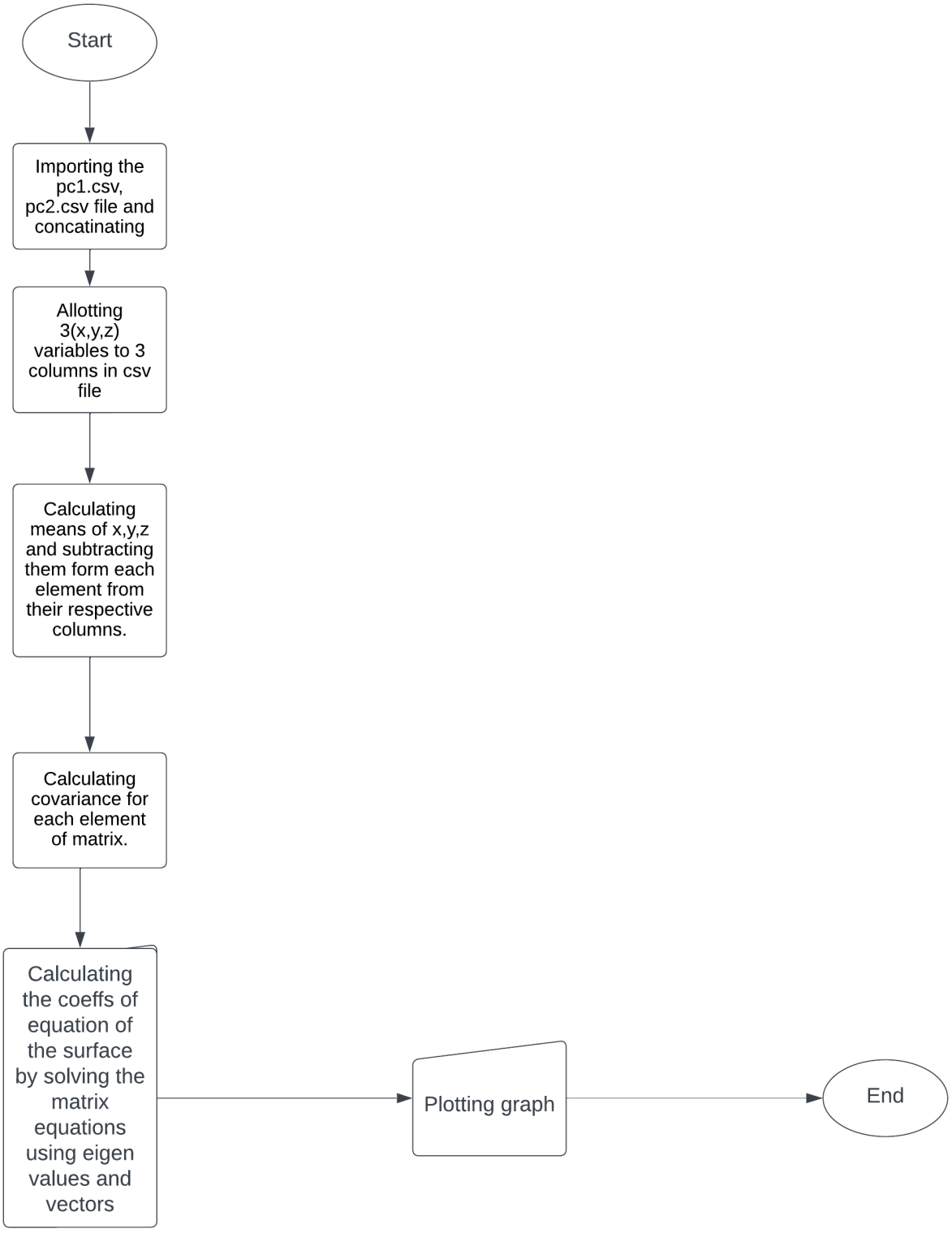
Output:

```
The direction of the normal surface is [-0.00338403 -0.76290695  0.64649944]
The magnitude of the normal is 1.0
```

2) A)



Pipeline of 2.2.1(Standard Least Square)



Pipe line of 2.2.1(Total Least Square)

Problems encountered:

1) **Faced problems while plotting the graph for Total least square method.**

Code:

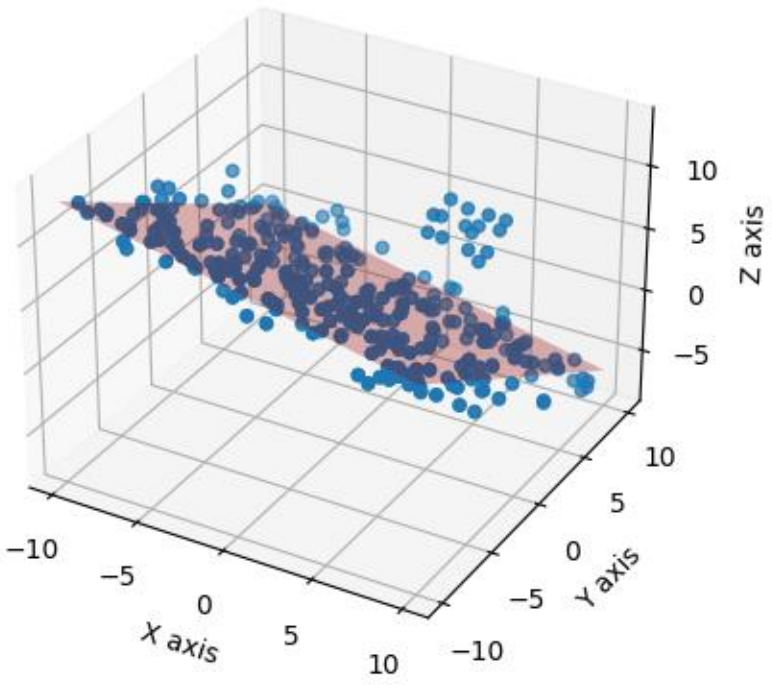
```
# ************************************************************STANDARD LEAST
SQUARE************************************************************
import numpy as np
import matplotlib.pyplot as plt
import csv as csv
from mpl_toolkits.mplot3d import Axes3D

path1="C:/Users/chara/Downloads/pc1.csv"          # Change path here
pc1 = np.genfromtxt(path1,delimiter=',')
path2="C:/Users/chara/Downloads/pc2.csv"          # Change path here
pc2 = np.genfromtxt(path2,delimiter=',')
data=np.concatenate([pc1,pc2])
print(data)

x1=data[:,0]
x2=data[:,1]
# print(x1)
# print(x2)
x3=np.ones_like(x1)
E=np.vstack((x1,x2,x3)).T
E_T=E.T
print(E_T)
G=np.vstack(data[:,2])
print(G)
# equation=E_Tx+Fy+c-G_T=0
E_T_inv=np.linalg.inv(np.dot(E_T,E))
R=np.dot(E_T_inv,E_T)
F=np.dot(R,G)
print(F)

i=F[0]
j=F[1]
k=F[2]

a=data[:,0]
b=data[:,1]
c=data[:,2]
```

```
x = np.linspace(min(a),max(a), 10)
y = np.linspace(min(b), max(b), 10)
x, y = np.meshgrid(x, y)
z = (i*x + j*y + k)
print(z)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Create a scatter plot
ax.plot_surface(x, y, z,color='r',alpha=0.3)
ax.scatter(a,b,c)


# Set the axis labels
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

# Show the plot
plt.show()

# **********************************************************************TOTAL
LEAST SQUARE**********************************************************************
x1_mean=np.mean(x1)
x2_mean=np.mean(x2)
x3=G
x3_mean=np.mean(x3)
# print("x3:",x3)
x1_difference=x1-x1_mean
x2_difference=x2-x2_mean
x3_difference=x3-x3_mean
# Calculate the covariance matrix
covar_matrix = np.zeros((3, 3))
covar_matrix[0, 0] = np.sum(x1_difference * x1_difference) / (len(x1) - 1)
covar_matrix[0, 1] = np.sum(x1_difference * x2_difference) / (len(x1) - 1)
covar_matrix[0, 2] = np.sum(x1_difference * x3_difference) / (len(x1) - 1)
covar_matrix[1, 0] = covar_matrix[0, 1]
covar_matrix[1, 1] = np.sum(x2_difference * x2_difference) / (len(x2) - 1)
covar_matrix[1, 2] = np.sum(x2_difference * x3_difference) / (len(x2) - 1)
covar_matrix[2, 0] = covar_matrix[0, 2]
covar_matrix[2, 1] = covar_matrix[1, 2]
covar_matrix[2, 2] = np.sum(x3_difference * x3_difference) / (len(x3) - 1)
# print("The covarianvce matrix is:",covar_matrix)
eigenvalues, eigenvectors = np.linalg.eig(covar_matrix)
```

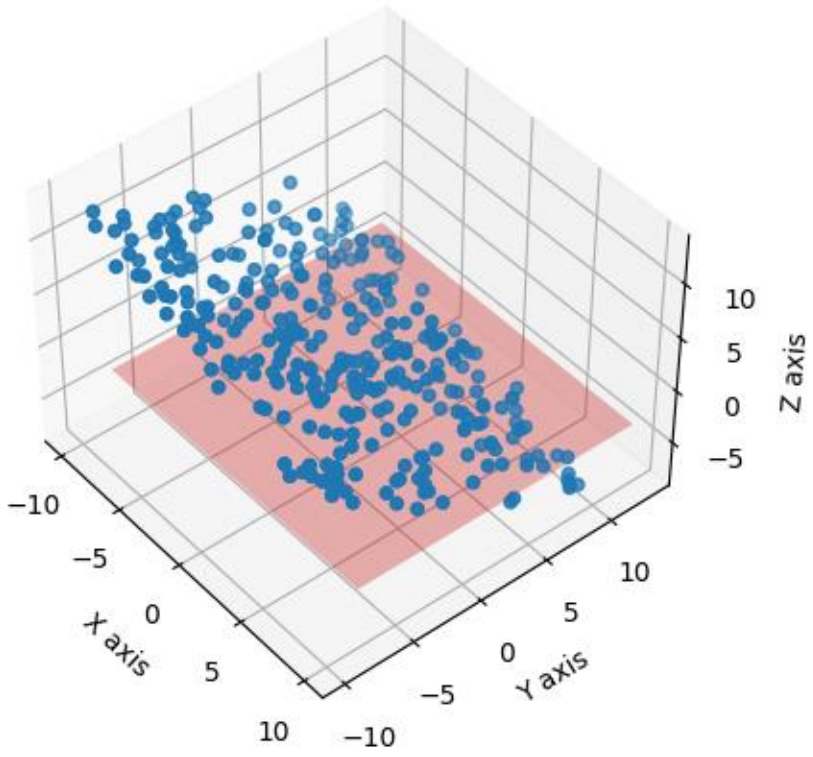
```
e=np.argmin(eigenvalues)
vectors=eigenvectors[:,e]
# print(vectors)
q=vectors[0]
w=vectors[1]
e=vectors[2]
r=np.column_stack((q,w,e))
mean=np.array([x1_mean,x2_mean,x3_mean]).reshape((-1,1))
print(mean)
print(r)
t=np.dot(r,mean)
print(t)
# equation=Qx+Wy+Ez+t=0
Q=q/e
W=w/e
E=e
T=t/e
v= np.linspace(min(a),max(a), 10)
k = np.linspace(min(b), max(b), 10)
v,k = np.meshgrid(v,k)
p= (-Q*v - W*k - T)
print(z)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Create a scatter plot
ax.plot_surface(v, k, p,color='r',alpha=0.3)
ax.scatter(a,b,c)


# Set the axis labels
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

# Show the plot
plt.show()
```

Output:

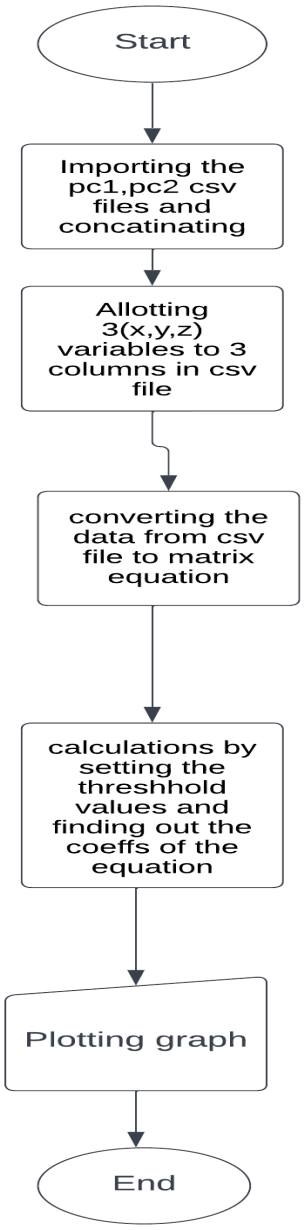
Standard Least Square:



Total least square method:



2)



Pipeline 2.2.2

Problems encountered:

1) Assigning large number of variables resulted in the syntax errors.
2) Initially scattered plots were not displayed due to the improper slicing of csv files.
3) Tried different threshold values and number of iterations to get the result.

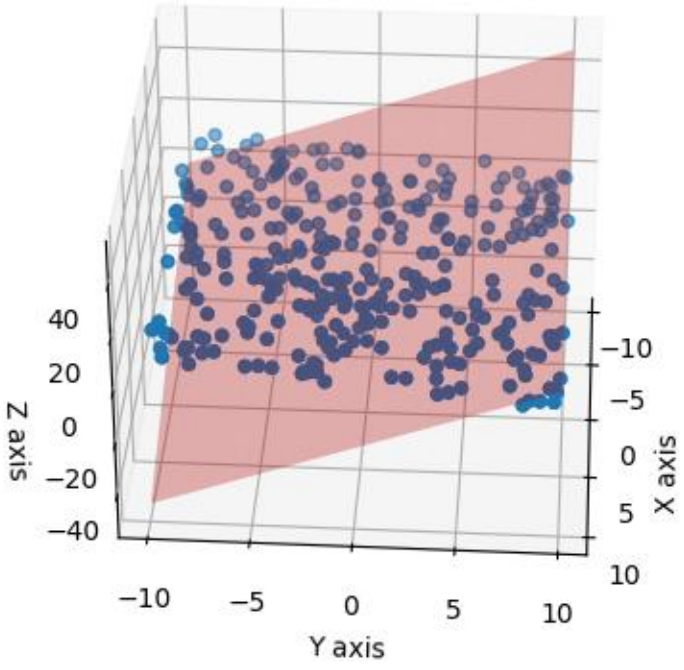
Code:

```
min_points=3
Treshhold=0.20
no_iterations=300
md=np.empty((0,3))
for i in range (no_iterations):
    random_indices = data[(np.random.choice(len(data)),3), :]
    # print(random_indices)
    Mean=np.mean(random_indices)
    aa=random_indices[:,0]
    bb=random_indices[:,1]
    cc=random_indices[:,2]
    aa_diff=aa-Mean
    bb_diff=aa-Mean
    cc_diff=cc-Mean
    covari_matrix=np.zeros((3,3))
    covari_matrix = np.zeros((3, 3))
    covari_matrix[0, 0] = np.sum(aa_diff*aa_diff) / (3 - 1)
    covari_matrix[0, 1] = np.sum(aa_diff*bb_diff) / (3 - 1)
    covari_matrix[0, 2] = np.sum(aa_diff * cc_diff) / (3 - 1)
    covari_matrix[1, 0] = covari_matrix[0, 1]
    covari_matrix[1, 1] = np.sum(bb_diff * bb_diff) / (3 - 1)
    covari_matrix[1, 2] = np.sum(bb_diff* cc_diff) / (3 - 1)
    covari_matrix[2, 0] = covari_matrix[0, 2]
    covari_matrix[2, 1] = covari_matrix[1, 2]
    covari_matrix[2, 2] = np.sum(cc_diff * cc_diff) / (3 - 1)
    eigenvalues, eigenvectors = np.linalg.eig(covari_matrix)
    ss=np.argmin(eigenvalues)
    f=eigenvectors[:,ss].reshape(-1,1)
    u=f.T.dot(mean.reshape(-1,1))
    l=np.append(f,u,0)

    distance=np.empty((0,1))
    for point in data:
```

```
        c,k,s=point
        dist=(l[0]*c+l[1]*k+l[2]*s+l[3])/np.sqrt(l[0]**2+l[1]**2+l[2]**2)
        distance=np.append(distance,dist)
    # print(distance)
    innerpoints=np.array([data[i,:] for i in range(len(data)) if distance[i] <
Treshhold])
    if len(innerpoints)>len(md):
        md=innerpoints.copy()
print(md)
aa=md[:,0]
bb=md[:,1]
cc=md[:,2]
aa_diff=aa-Mean
bb_diff=aa-Mean
cc_diff=cc-Mean
covari_matrix=np.zeros((3,3))
covari_matrix = np.zeros((3, 3))
covari_matrix[0, 0] = np.sum(aa_diff*aa_diff) / (3 - 1)
covari_matrix[0, 1] = np.sum(aa_diff*bb_diff) / (3 - 1)
covari_matrix[0, 2] = np.sum(aa_diff * cc_diff) / (3 - 1)
covari_matrix[1, 0] = covari_matrix[0, 1]
covari_matrix[1, 1] = np.sum(bb_diff * bb_diff) / (3 - 1)
covari_matrix[1, 2] = np.sum(bb_diff* cc_diff) / (3 - 1)
covari_matrix[2, 0] = covari_matrix[0, 2]
covari_matrix[2, 1] = covari_matrix[1, 2]
covari_matrix[2, 2] = np.sum(cc_diff * cc_diff) / (3 - 1)
eigenvalues, eigenvectors = np.linalg.eig(covari_matrix)
ss=np.argmin(eigenvalues)
f=eigenvectors[:,ss].reshape(-1,1)
u=f.T.dot(mean.reshape(-1,1))
l=np.append(f,u,0)

print("L",l)
q=l[0]
w=l[1]
t=l[2]
e=l[3]
Q=q/e
W=w/e
# E=e
T=t/e
v= np.linspace(min(a),max(a), 10)
k = np.linspace(min(b), max(b), 10)
v,k = np.meshgrid(v,k)
E= (-Q*v - W*k - T)
```

```
print(z)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Create a scatter plot
ax.scatter(data[:,0],data[:,1],data[:,2])
ax.plot_surface(v, k, E,color='r',alpha=0.3)



# Set the axis labels
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

# Show the plot
plt.show()
```

Output:



Description by comparing all three methods:

In standard least square method, we use the vertical distances to the points, minimizing the error by squaring the distances. In total least square method, we take the perpendicular distances to the points from the reference plane and square them, it is better suitable when the noise is high and there are dependent and independent variables. Where as in ransac method we consider some threshold and find the optimum plane passing through, it is better suitable when the noise is low. Theoretically, total least square method provides the best result. coming to the outlier rejection, through this problem standard least square method given the best output.