Harshal Shirsath, 119247419, shirsath@umd.edu

Perception ENPM673, PROJECT 1

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)

Project 1 > Project 1.1 > Project 1.2.py > ...

```python
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
from numpy import matrix

#1.1-------------------------------------------------------------------------------------------------------------------------
cap = cv2.VideoCapture('F:/Perception(673)/Projects/Project 1/ball.mp4')
x_array=[]
y_array=[]

while(1):
    ret, frame = cap.read()
    if ret == True:
    # It converts the BGR color space of image to HSV color space
            red = np.uint8([[[0,0,255 ]]])
            hsv_red = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

    # Threshold of red in HSV space
            lower_red = np.array([0, 50, 50])
            upper_red = np.array([2, 255, 255])

    # preparing the mask to overlay
            mask = cv2.inRange(hsv_red, lower_red, upper_red)
        # red = np.uint8([[[255,0,255]]])

    # Display the resulting frame
            cv2.imshow("red",frame)
           #print('vi')

    # Press Q on keyboard to  exit
            if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
    else:
            break


    # The black region in the mask has the value of 0,
    # so when multiplied with original image removes all non-red regions
    result = cv2.bitwise_and(frame, frame, mask = mask)

    cv2.imshow('frame', frame)
    cv2.imshow('mask', mask)
```

Project 1 > Project 1.1 > Project 1.2.py > ...

```
68
69
70  # 1.2-----------------------------------------------------------------------------------------------------------------
71
72  # Finding the Least Square Line Fitting
73
74  # B=(X.T*X)**-1(X.T*Y)
75  #Converting lists to arrays
76  x_array=np.array(x_array)
77  y_array= np.array(y_array)
78  A = np.column_stack([x_array**2, x_array, np.ones(len(x_array))]) #X matrix
79  # print('A:',np.shape(A))
80
81  #Finding the transpose of the matrix A
82  A_t= np.transpose(A)                                              #X.T= A transpose matrix
83  # print('A_t:',A_t)                                               #Printing X.T= A transpose matrix
84  # print(np.size(A))
85
86  #Dot product of X.T*X
87  B_A=np.dot(A_t,A)                                                 #X.T*X
88  # print(np.shape(B_A), B_A)
89  # B_inv= np.linalg.inv(B_A)
90  B_a= np.dot(A_t,y_array)                                          #X.T*Y
91  B= np.dot(B_A,B_a)                                                #(X.T*X)**-1(X.T*Y)
92  # print('B:',np.shape(B))
93  # print('y_array:',np.shape(y_array))
94
95  # print('B:',B)
96  # Extracting the coefficient of B[] matrix
97  a= B[0]
98  b= B[1]
99  c= B[2]
100 # print('B_x', B_x)
101 # print('B_y', B_y)
102 # print('B_z', B_z)
103
104 #Define the range of the x coordinates inorder to get the y coordinates
105 x_range=np.linspace(min(x_array),max(x_array), 100)
106 # print('x_range:', x_range)
107 x_range=np.array(x_range)                              # Converting list to array
108 # print('x_range', x_range)
109
```

```
Project 1 > Project 1.1 > Project 1.2.py > ...
44
45
46      non_zero= np.where(mask>0) #2d array)
47  #     print(non_zero)
48  #     pixelpoints = cv.findNonZero(mask)
49
50  # Extracting the non-zero values from the array especially rows non_zero[]
51      y=non_zero[0]
52      # print("x=",y)
53      x=non_zero[1]
54  #     print("y=",x)
55
56  # Calculating the mean of the function
57      if(len(x)>0 and len(y)>0):
58          x_mean=np.mean(x)
59          y_mean= np.mean(y)
60          # print('x_mean:', x_mean)
61          # print('y_mean', y_mean)
62          #removing all the nan values from the arrays of mean
63          if not (math.isnan(x_mean) and math.isnan(y_mean) ):
64                  x_array.append(x_mean)
65                  y_array.append(y_mean)
66  # print(x_array)
67  # print(y_array)
68
69
```

```
Project 1 > Project 1.1 > Project 1.2.py > ...
104 #Define the range of the x coordinates inorder to get the y coordinates
105 x_range=np.linspace(min(x_array),max(x_array), 100)
106 # print('x_range:', x_range)
107 x_range=np.array(x_range)                           # Converting list to array
108 # print('x_range', x_range)
109
110
111 # Matrix B and it's elements
112 coefficient= np.linalg.solve(B_A,B_a)
113 a,b,c = coefficient                                 # Coefficients of the matrix B[]
114 y_range = a*x_range**2 + (b*x_range) + c
115 print(f"y = {a}*x**2 + {b}*x + {c}", y_range)       #print the parabolic equations
116
117 plt.figure()
118 plt.gca().invert_yaxis()
119 plt.plot(x_array, y_array , 'bo', "blue" )          #Plotting the x_array & y_array which is the center of mean co-ordinates of x and y
120 # plt.gca().invert_yaxis()
121 plt.plot(x_range, y_range,'-', "red" )              #Least Square Fitting curve fir plot
122 plt.show()
123
124 # 1.3-------------------------------------------------------------------------------------------------------------------------------------------------
125 # X-coordinate of the ball's landing spot in pixels
126 landing_y = 300 + y_array[0]                                #Landing y-Coordinates pixels
127 landing_x = (-b + np.sqrt(b**2 - 4*a*(c - landing_y)))/(2*a)    #Landing x-Coordinates pixels
128 print(f"Landing x-coordinate: {landing_x} pixels")
129
130 cv2.destroyAllWindows()
131 cap.release()
```

```
y = 0.0005521669333184457*x**2 + -0.5594812506305554*x + 462.96310366852157 [461.73920245 455.37959988 449.16728153 443.10224742 437.18449754
 431.41403189 425.79085048 420.31495329 414.98634033 409.80501161
 404.77096712 399.88420686 395.14473083 390.55253903 386.10763147
 381.81000813 377.65966903 373.65661415 369.80084351 366.0923571
 362.53115493 359.11723698 355.85060326 352.73125378 349.75918853
 346.93440751 344.25691072 341.72669816 339.34376983 337.10812573
 335.01976587 333.07869024 331.28489884 329.63839166 328.13916873
 326.78723002 325.58257554 324.5252053  323.61511928 322.8523175
 322.23679995 321.76856663 321.44761754 321.27395269 321.24757206
 321.36847567 321.63666351 322.05213558 322.61489188 323.32493241
 324.18225717 325.18686617 326.33875939 327.63793685 329.08439854
 330.67814446 332.41917461 334.30748899 336.34308761 338.52597045
 340.85613753 343.33358884 345.95832438 348.73034415 351.64964815
 354.71623638 357.93010885 361.29126555 364.79970647 368.45543163
 372.25844102 376.20873465 380.3063125  384.55117458 388.9433209
 393.48275145 398.16946623 403.00346524 407.98474848 413.11331595
 418.38916766 423.81230359 429.38272376 435.10042816 440.96541679
 446.97768965 453.13724674 459.44408807 465.89821362 472.49962341
 479.24831743 486.14429568 493.18755816 500.37810487 507.71593582
 515.20105099 522.8334504  530.61313404 538.5401019  546.61435401]
Landing x-coordinate: {1493.032362753291}
```

Pipeline 1.1:

1. Importing the video in the code by providing the path of the video
2. Read the frames from the video one by one and convert them to HSV for better working on object detection.
3. Provide a range of HSV values for the detection of a particular-colored object (red colored object thresholding)
4. Once the thresholding is done properly then we can mask the object.
5. After thresholding collect all the non-zero pixels from the threshold frames.
6. Consider one frame and extract the pixels making sure the pixels are collected in minimum to maximum range in the x direction and likewise for the y direction.
7. Add these pixels in separate arrays and find the mean of x and y directions separately.
8. The calculated mean from the previous step will provide the center point of the ball in each frame.
9. Plot the center point of every frame.

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.

Pipeline 1.2:

1. The ball's center points are already extracted from the previous task. These values are to be used to find the parabolic fitting for the plotted curve.
2. Using the numpy library we create the matrix which consists the y=ax^2 + bx +c
3. Find the y matrix.
4. In a particular range of x array and y array (from the previous task) take a few samples and implement them in the least square formula for the coefficients a, b &c.
5. Least Square Fitting method: $B=(X^T.X)^{-1}x(X^T.Y)$
6. Compute the coefficients in the formula to find the y range.
7. This gives the corresponding x and y coordinates to plot the parabola.
8. Plot the parabola using matplotlib and superimpose it with the previously scattered plot.

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.

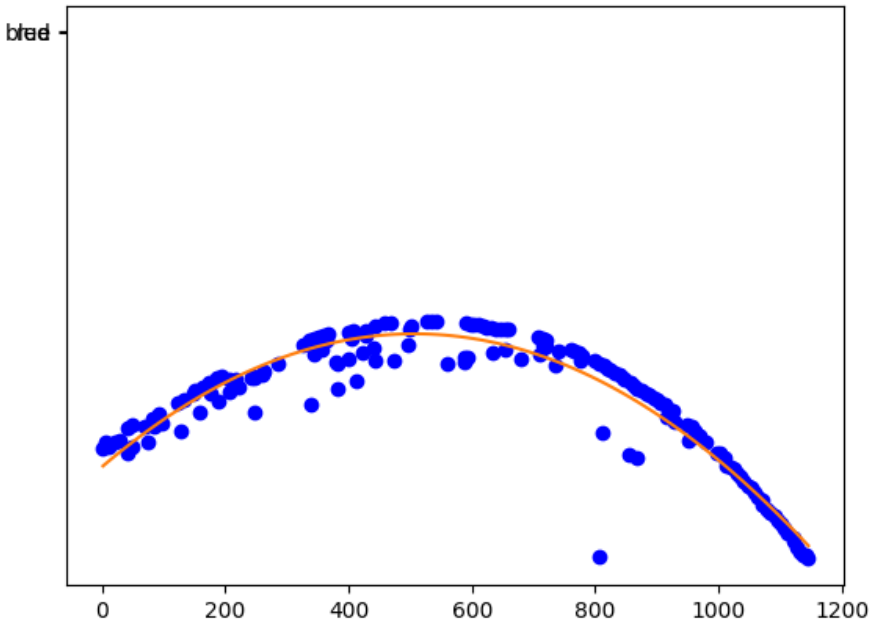
Pipeline 1.3:

1. To figure out the y-coordinate landing spot of the ball extract the first coordinate of the ball in the frame where the ball first appeared.
2. Add the above step with 300 pixels to get the landing pixels of the ball in the frame.
3. Use of the square root method will be helpful to find the x-coordinate of the landing ball.
4. Input the coefficients from the previous problem to commute in the square root formula.

Problems faced:

1. While providing the video's absolute path backslash was pasted as copied from the path of the video giving a problem reading the video. (F:\Perception(673)\Projects\Project 1\Project 1.1\ball.mp4- wrong path)
2. Thresholding trial and error detection for the fine thresholding. Tried to find a function to find the precise thresholding filter to overlay the mask value. (For both : the upper and lower range of the ball)
3. The mean x_array and y_array is computed with nan values which gave an error while plotting the graph as a null array graph plot. Removing the nan values in order to get sorted arrays without any nan values.
4. Confusion between the row and column of the frame since the origin is on the top left part of the frame while extracting the pixel elements from the 2D array.
5. B=(X.T*X)**-1*(X.T*Y) matrix multiplication was unable to give a 3x1 matrix for the coefficients where the problem was in the ndarrays in X.T*Y. As this was a list that was to be converted to arrays. (ND array error occurred multiple times).
6. Extracting the minimum values and maximum values of the x_array which was giving an error with the 'list not callable'.

RESULTS:



Problem 2:

Given are two csv files, pc1.csv and pc2.csv, which contain noisy LIDAR point cloud data in the form

of (x, y, z) coordinates of the ground plane.

1. Using pc1.csv:

a. Compute the covariance matrix.

b. Assuming that the ground plane is flat, use the covariance matrix to compute the magnitude and direction of the surface normal.

Pipeline 2.1(a)

1. In this function extract the data according to the columns.
2. The data extracted in the form of array is used to calculate the mean of each column.
3. Using the functions of numpy covariance(np.cov) we can find the covariance matrix.

Pipeline 2.1(b)

1. The center coordinates from the previous problems are used to find the eigen values and eigen vectors.
2. Using the covariance from the previous problem to compute it in the eigen values and eigen vectors using Numpy linalg(covariance matrix).
3. This step provides the eigenvalues and eigenvectors respectively.
4. Sort the data in ascending order and look for minimum eigenvalue. The corresponding eigenvector which is normal to it represents the direction of the plot.
5. The eigenvector with the largest eigenvalue corresponds to the direction of maximum variation in the point cloud, which is also the direction of the surface normal.
6. The magnitude of the surface normal is calculated by taking the square root of the largest eigenvalue.

2. In this question, you will be required to implement various estimation algorithms such as Standard Least Squares, Total Least Squares and RANSAC.

a. Using pc1.csv and pc2, fit a surface to the data using the standard least square method and the total least square method. Plot the results (the surface) for each method and explain your interpretation of the results.

Pipeline for Standard Least Square

1. Read the data points from the pc1.csv and pc_2.csv file and import them to the script.
2. Then calculate the covariance using the numpy.cov() with all the data points provided.
3. The above process resulted in a covariance matrix that is used to get eigenvalues and eigenvectors.
4. Then find the eigenvectors which are normal. We select the minimum eigenvalue and the corresponding eigenvector is normal of the plane.(Using function np.argmin(eigenvalues)).
5. Once the points are extracted the mean of all the columns with axis 0 is selected as a centroid of all the data points.
6. Before printing compute the magnitude and direction for the normal of the plane.
7. The best fit data is pointed while returning the normal of the plane.

Pipeline for Total Least Square

1. Read the data points and import them into the script.
2. Add an additional coordinate value of 1 to each data point.
3. Use linalg(augmented_matrix), as this will return the Singular Value Decomposition.(A,B,S- np.linalg(augmented_matrix))
4. The final column of the matrix S is the normal of the plane which points the best fit data.

Pipeline for RANSAC method:

1. Read the data from the files and import it in the scripts.
2. Set the inlier threshold, set the maximum number of iterations and set the minimum number of points required to for the plane.
3. Now for each iteration:
4. Randomly select any point from the dataset and form a subset.
5. Using these subsets obtain the parameters of the curve.
6. In addition to it compute the distance between each data point and the curve.
7. To figure out the inliers in the data set compare the distances to the thresholds.
8. The parameters are re-estimated for the curve when the inliers are greater than the predefined inliers.

Problems Faced:

1. Converting list to matrix
2. Algorithm understanding and some functions like np.linalg

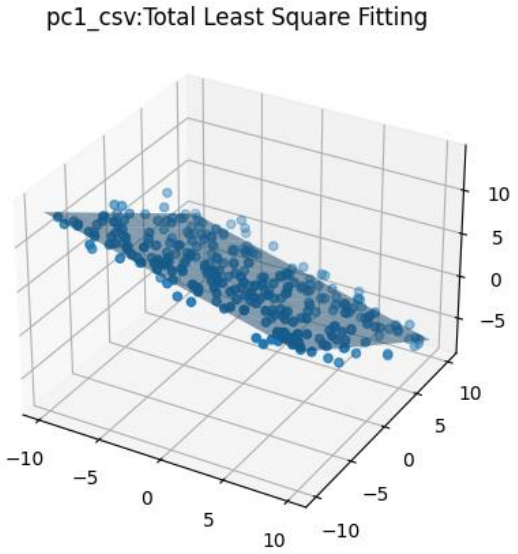
3. Problems with matrix shape and their particular column extraction.

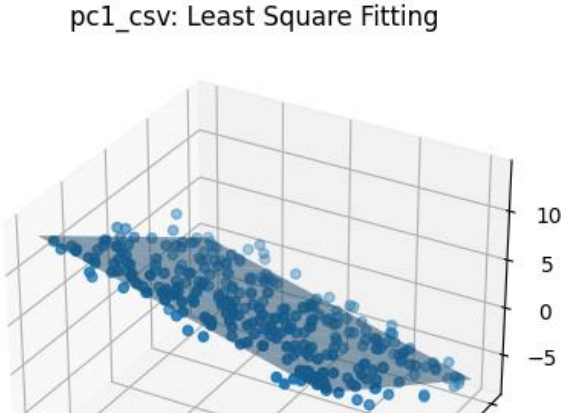
```
magnitude: 1.0
direction: 1.2355054021855683
point: [-3.9605536  -3.76911903  7.42769327]
```

```
point: [ 4.81882991  0.58425351 -0.01766476]
d: -2.8847897640314715
unit_normal: [0.24147224 0.54860699 0.80045083]
x_min, x_max: -9.906090476149059 9.976940131357331
y_min, y_max: -9.945935722129946 9.95924502657347
```

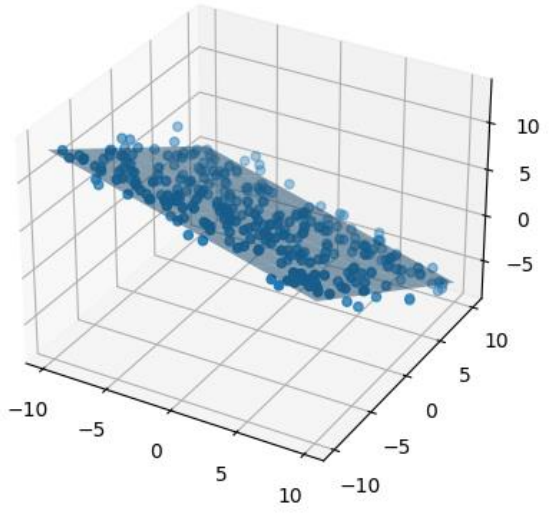
b. Additionally, fit a surface to the data using RANSAC. You will need to write RANSAC code from scratch. Briefly explain all the steps of your solution, and the parameters used. Plot the output surface on the same graph as the data. Discuss which graph fitting method would be a better choice of outlier rejection.

RESULTS:

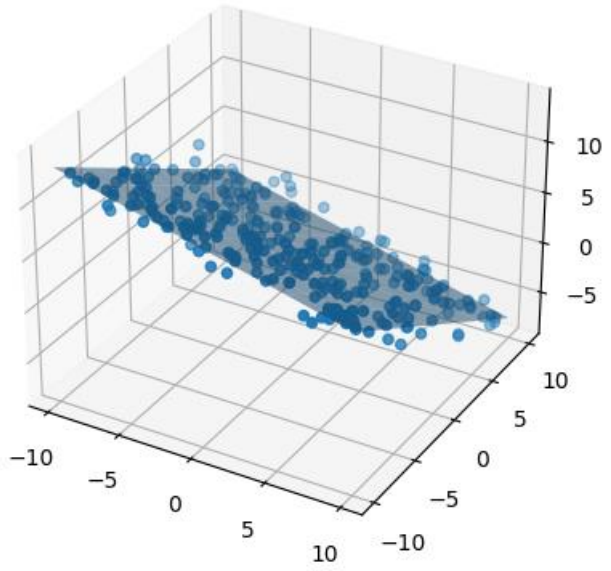




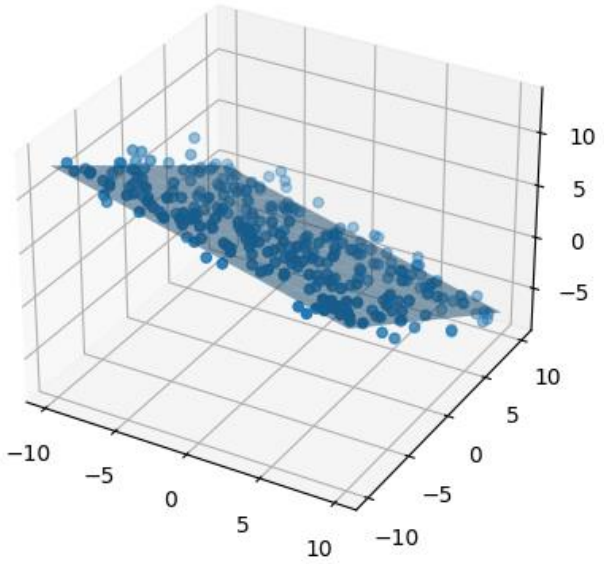
pc1_csv:RANSAC fitting



pc2_csv:Least Square Fitting



pc2_csv: RANSAC fitting



pc2_csv:Total Least Square Fitting