Project 1
ENPM 673

Aashrita Chemakura
UID# 119398135
achemaku@umd.edu

Problem 1.

1.1)

Pipeline:

Read the video using cv2.VideoCapture() function.
Convert the video into HSV using cv2.cvtColor() function and create a mask to extract the red color pixels from the ball.
Defining the threshold values to extract red color pixels using cv2.inRange() function.
Computing the centroid of the red ball using cv2.moments() function and saving the coordinates.
Plotting the trajectory of the ball using matplotlib pyplot's plot() function.

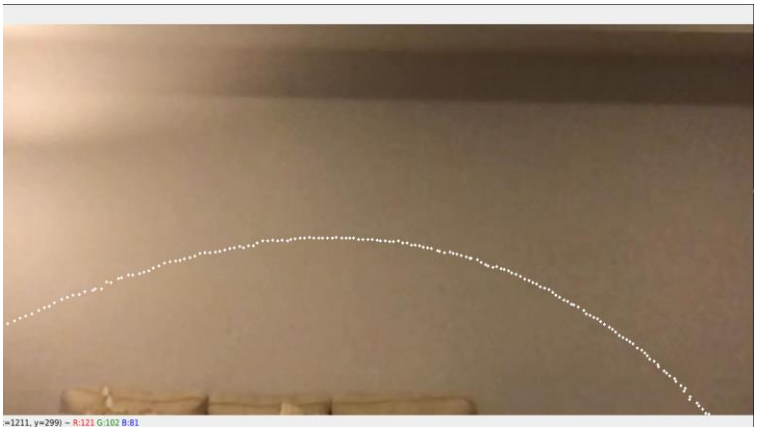

Figure 1- Trajectory of the ball path

Problems Encountered:

The problem faced was that the red color was sensitive to lighting conditions of the video so it failed to extract few of the red color pixels of the ball. The solution to this was to toggle the HSV threshold values of the red color pixel to obtain the perfect masking of the ball.

Result: Below is the plot obtained for the trajectory of the ball.

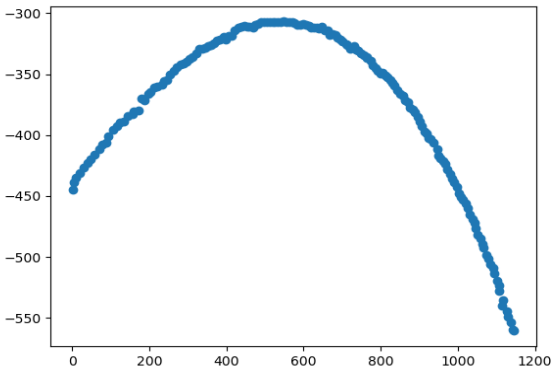

Figure 2- Plot of the trajectory of the ball

1.2)

Pipeline:

Using the general equation of parabola $y = ax^2 + bx + c$.
This equation can be written as y = B X where,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ y_n \end{bmatrix},\ B = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, X = \begin{bmatrix} {x^2}_1 & x_1 & 1 \\ {x^2}_2 & x_2 & 1 \\ . & . & . \\ {x^2}_n & x_n & 1 \end{bmatrix}$$

The Solution for this equation is given by

$$B = (X^T X)^{-1}(X^T Y)$$

The least square method is used and solved by np.linalg() function to determine the values of a, b, c which give the best fit for pixel coordinates.
Plot the obtained pixel coordinates and the curve on the graph using matplotlib.plot() function.

Problems Encountered:

The curve obtained was initially inverted due to the image coordinates and pixel coordinates not having the same origins this problem was solved by multiplying a negative (-) sign to the Y-axis.

Result:

The plot obtained for the equation of the curve and the trajectory of the ball is as follows.

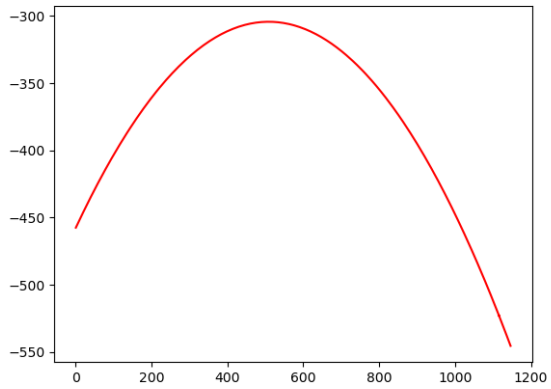

Figure 3- Plot of the equation of the parabolic curve

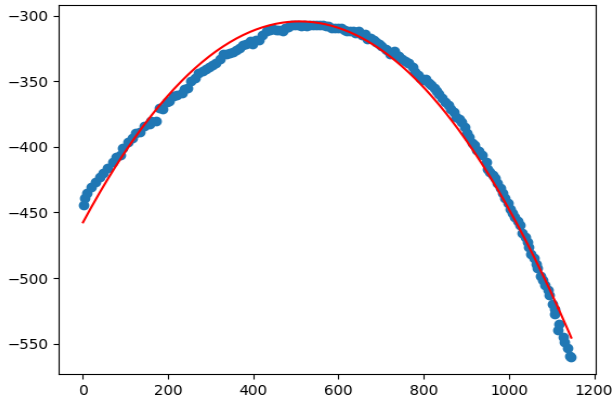

Figure 4- Plot of the parabolic curve and the trajectory of the ball

1.3)

Pipeline:

The general equation of parabola is $y = ax^2 + bx + c$.
In the question it is stated that the y coordinate of the ball is 300 pixels greater than its first detected location so the equation according to the new coordinates is taken as

$$y_n - y_1 = 300 \quad \rightarrow \quad y_n = y_1 + 300$$

The obtained new equation is - $y_n = a{x_n}^2 + bx_n + c$

Which can be written as $a{x_n}^2 + bx_n + c - 300 - y_1 = 0$

The actual solution for the equation can be given by:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

But, when the equation with negative sign is taken, the value of the x-coordinate becomes negative. Hence, the solution of the equation will be given by:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Print the equation of the curve and the solution of the equation.
There were x-coordinate values of the landing spot of the ball obtained for this curve out of which one of them was negative. Since, the x-coordinate value of the ball cannot be negative, the highest value was chosen.

Problems Encountered:
There were no problems encountered in this question.

Result:
The obtained equation of the curve is and the X-coordinate value for the landing spot of the ball is:

```
Equation of curve is [0.00059325] x**2+ [-0.6035489] x+ [-286.65453952]
The x- coordinate of the landing spot of the ball is- [1370.0501177]
```

Problem 2.

2.1) (a)

Pipeline:
Load the data from the *pc1.psv* file using numpy's *loadtxt* function.
The covariance matrix is calculated as:

$$\begin{bmatrix} Var(A) & Cov(A,B) & Cov(A,C) \\ Cov(A,B) & Var(B) & Cov(B,C) \\ Cov(A,C) & Cov(B,C) & Var(C) \end{bmatrix}$$

To obtain the covariance matrix, the x, y and z coordinates from the data must be extracted and stored in variables.
The mean for each coordinate is calculated and subtracted from the respective data coordinate and squared.

$$A = (A_1 - A_1\_\text{mean})^2$$

The variance for this is calculated by:

$$Var = \frac{Sum(A)}{Length\ of\ the\ data\ coordinate}$$

To compute the covariance of the coordinates, the mean of each data coordinate is to subtracted from the respective data coordinate.

$$A = (A_1 - A_1\_\text{mean})$$

The covariance is computed by taking the dot product of the two data coordinates and divided by the length of the data coordinate.

$$Cov(A,B) = \frac{A\,.\,B}{Length\ of\ the\ data\ coordinate}$$

The covariance matrix is obtained by using np.matrix() function.

Problems Encountered:
There were no problems encountered for this question.

Result:
The covariance matrix obtained is:

```
The covariance matrix is:
[[ 33.6375584   -0.82238647 -11.3563684 ]
 [ -0.82238647  35.07487427 -23.15827057]
 [-11.3563684  -23.15827057  20.5588948 ]]
```

2.1) (b)

Pipeline:
The ground plane is assumed to be flat to calculate the surface normal from the obtained covariance matrix.
In order to find the surface normal, the eigen vectors of the covariance matrix are to be calculated. This can be done using nunpy's linear algebraic eig function.
The eigen vector with the smallest eigen value is calculated using numpy's argmin function.
This represents the direction of the surface normal.
The magnitude of the surface normal is obtained by calculating the square root of sum of squares of eigenvectors.

Problems Encountered:
There were no problems encountered for this question.

Result:
The surface normal obtained in a three-dimensional vector space is:

```
The surface normal is:
[[0.28616428]
[0.53971234]
[0.79172003]]
```

The magnitude of surface normal is:

```
The magnitude of the surface normal is:
 [[1.]]
```

2.2) (a)
(i) <u>Least Squares:</u>

Pipeline:
Importing the library Axes 3D from mpl_toolkits.mplot3D.
Load the data files 1 and 2 using np.loadtxt() function.
The x, y and z coordinates of the data are extracted and stores in the form of a numpy array.
The equation of the plane is $ax + by + cz = d$
This equation can be written as:

$$z = -\frac{a}{c}x - \frac{b}{c}y + \frac{d}{c}$$

Which is equivalent to-

$$z = a'x + b'y + c'$$

This equation in matrix form-

$$B = Ax$$

Where,

$$B = \begin{bmatrix} z_1 \\ z_2 \\ . \\ z_n \end{bmatrix}, \quad A = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, X = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ . & . & . \\ x_n & y_n & 1 \end{bmatrix}$$

The solution of this equation is obtained as-

$$B = (X^T X)^{-1}(X^T Y)$$

The equation of the plane is obtained by using np.linalg() function.
The plot is obtained by using axis.scatter3d() function.

Problems Encountered:

There were no problems encountered in this question.

Result:

```
Equation of the plane for data set 1 using is : -0.35395482408123213 *x + -0.6685514549381812 *y + 3.
2025536281246327

Equation of the plane for data set 2 is : -0.2518840428649011 *x + -0.6717366909532275 *y + 3.6602566
93232872
```

The obtained 3D plot of the plane is :

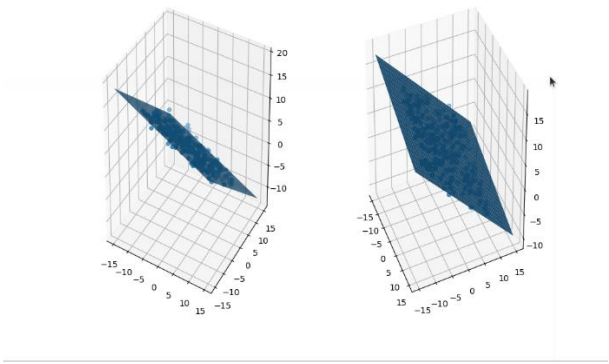

Figure 5

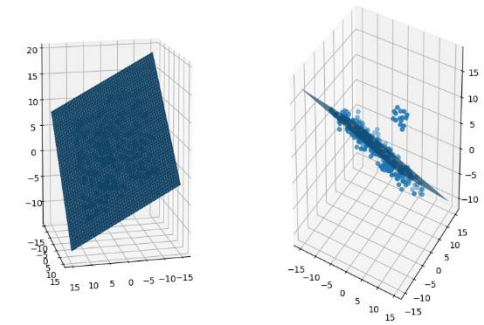

Figure 6

Interpretation:

The 3D plot of the plane obtained from this methos is accurate but has many outliers and can be computed in a better way by using other methods or normalizing the data.

(ii) Total Least Squares-

Pipleline:

Importing the library Axes 3D from mpl_toolkits.mplot3D.
Load the data files 1 and 2 using np.loadtxt() function.
The x, y and z coordinates of the data are extracted and stored in the form of a numpy array.
Compute SVD for both the data sets and obtain the values of eigen values and eigen vectors.
The eigen values and eigen vectors are obtained for the following equations using np.linalg.eig() function.

$$B = (X^T X) \quad \text{and} \quad B = (X X^T)$$

Once, the eigen values and vectors are computed, obtain sigma and diagonalized matrix for the equation. Compute the normal for the equation and the solution is obtained.
The 3D plot is obtained by using axis.scatter3D() function.

Problems Encountered:
Initially, the plane of the plot wasn't obtained and the number of outliers were more it took some time to understand the 3D plotting and after going through a lot of websites, obtained the proper solution to the equation.

Result:
The 3D plot obtained is as follows:

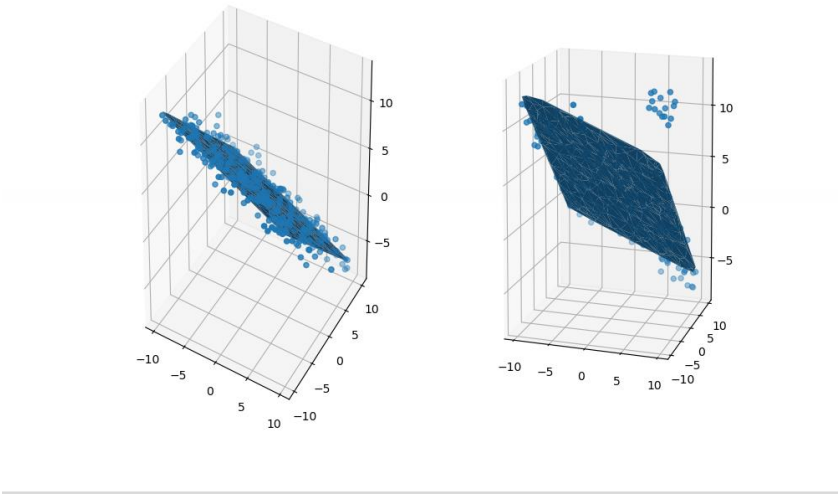

Figure 7

The equations of the planes:

```
Equation 1:  0.28616427612095185 *x+ 0.5397123383073396 *y+ 0.7917200256094294 *z+ 2.5344641945425836
 =0
Equation 2:  -0.22107409283980364 *x+ -0.5873941957270393 *y+ -0.7785205869476044 *z+ -2.849444521836
679 =0
```

Intertrepation:
The plot obtained from this methos is a bit more accurate than least square with lesser number of outliers.

2.2) (b)

Pipeline:
Importing the library Axes 3D from mpl_toolkits.mplot3D.
Load the data files 1 and 2 using np.loadtxt() function.
The x, y and z coordinates of the data are extracted and stored in the form of a numpy array.
The data is normalized by the formula:

$$x1 = x - xmean$$

Computing the SVD function for each value of the coordinate and compute the eigen values, eigen vectors and normal for the equations.
Choose any 3 random points from the data file arbitrarily using np.random.choice() function.
Find the equation of the plane passing through them.

Calculate the distance of each point in the data set and obtain a threshold value such that any point lying within the threshold value is considered as an inlier.
Obtain the value of total number of inliers.
Reiterate the process a few more times to find the equation of the plane with the maximum number of inliers.
The obtained plane is considered as the best fit plane.
The 3D plot is obtained by using axis.scatter3D() function.

Problems Encountered:
Initially, due to the use of np.random.choice() function, there were random points selected out of which many of them were outliers and the plan obtained was not accurate. After multiple trial and errors and iterations, the solution implemented was to normalize the data so that the chosen point lies within a specified range this helped in obtaining the perfect plane.

Result:

```
Equation of the Plane for pc1:  -0.21115136056550637 *x+ -0.5318518887184912 *y+ -0.8200906482809744
*z+ -2.623172393730889 =0

Inliers count found in pc1:  300
Equation of the Plane for pc2:  0.2773895889982252 *x+ 0.524715592647561 *y+ 0.8048158564217747 *z+ 2
.979924077418619 =0

Inliers count found in pc2:  315
```

The plot 3D obtained is as follows:

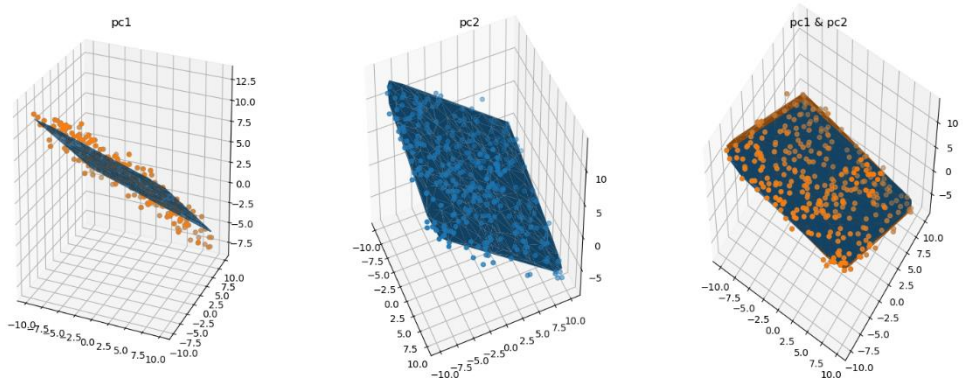

Figure 8

Interpretation:
The 3D plot of RANSAC is more accurate than the plot obtained from total least square and least square with less number of outliers.