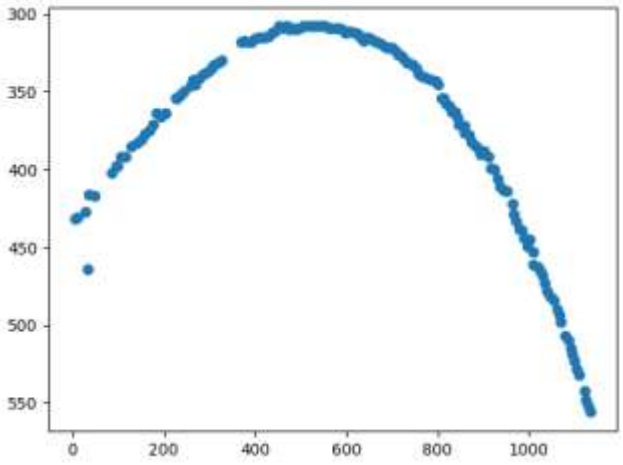
ENPM 673: Project 1

Aditi Bhoir (119197257)

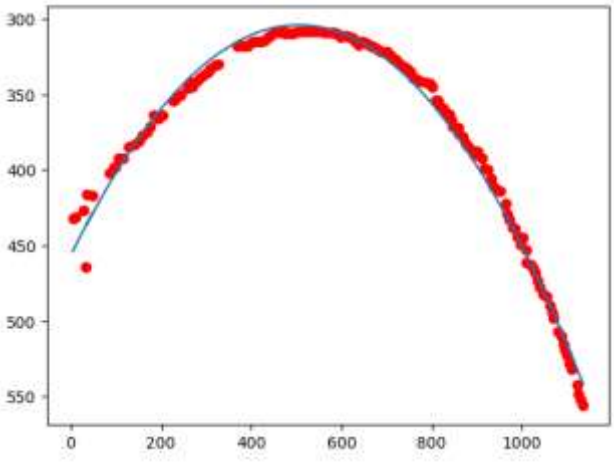
PROBLEM 1:

Steps Followed:

1. Imported the necessary libraries, video is read using OpenCV's VideoCapture function in each frame of the video.
2. The color space is converted from BGR to HSV. The lower and upper ranges of the red pixel in hsv are defined.
3. A mask is created using inRange function, this will set pixels falling in the range as 1 and others as 0.
4. A kernel is defined and mask is subjected to opening and closing operations to remove noise.
5. All the non-zero pixels are extracted from the mask, min and max x and y are calculated, mean is taken to find the center of the ball in each frame.
6. After all the frames have been processed the center of ball in each frame are plotted.
7. Used least square method to fit the trajectory of the ball into a parabola. The equation of parabola is y = ax*2 + bx + c. Wrote the equations in the form of Ax = B, where A is (n,3) matrix having x**2, x and 1 in column. Solve to get the coefficients a, b, and c.
8. Plot the fitted parabola using scatter plot.
9. Print the equation of the parabola.
10. To find y coordinate of the landing spot, substitute the x coordinate in the parabola equation.



Trajectory of the center of the ball



Fitted parabola

Problems Encountered:

The pixels were not visible in the start because I had put random values in the lower and upper range for the cv.inRange function. Researching more about how to choose hsv values helped to solve this problem. The centers I was getting earlier were not correct, therefore was giving very uneven plot with a lot of noise. I have applied morphological operations to remove the noise. As soon as the data was noise free, the curve fitting issue was solved.

Another issue was to make the input data in the correct format. If the data frame does not have the correct column or row positions, there were few indexing erros.

PROBLEM 2:

2.1

Steps Followed:

1. Import the necessary libraries.
2. Calculated the mean, covariance of (x, y), (y, z), (x, z), (x, x) , (y, y), (z, z) using following formula.

$$\operatorname{corr}(\mathbf{X}) = \begin{bmatrix} 1 & \frac{\mathrm{E}[(X_1-\mu_1)(X_2-\mu_2)]}{\sigma(X_1)\sigma(X_2)} & \cdots & \frac{\mathrm{E}[(X_1-\mu_1)(X_n-\mu_n)]}{\sigma(X_1)\sigma(X_n)} \\ \frac{\mathrm{E}[(X_2-\mu_2)(X_1-\mu_1)]}{\sigma(X_2)\sigma(X_1)} & 1 & \cdots & \frac{\mathrm{E}[(X_2-\mu_2)(X_n-\mu_n)]}{\sigma(X_2)\sigma(X_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\mathrm{E}[(X_n-\mu_n)(X_1-\mu_1)]}{\sigma(X_n)\sigma(X_1)} & \frac{\mathrm{E}[(X_n-\mu_n)(X_2-\mu_2)]}{\sigma(X_n)\sigma(X_2)} & \cdots & 1 \end{bmatrix}.$$

3. The surface normal to the plane is calculated by using eigen values and eigen vectors of the covariance matrix.
4. The surface normal is given by the smallest eigen value and corresponding eigen vector because variance is minimum in this direction.

2.2

a.

For all the three methods, the csv files are read using pd.read_csv function and data is extracted from the file and put into arrays for X, Y and Z coordinates. The equation of the plane is given by

```
ax + by + cz + d = 0
```

divide everything by c we get,

`(a/c)x + (b/c)y + z + (d/c) = 0`, which can be written as

```
mx + ny + z + o = 0
```

<u>Least Square Method:</u>

The plane equation can be written in the form.

```
Bx = Y
```

Where

$$A = \begin{matrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ xn & yn & 1 \end{matrix}, \quad x = \begin{matrix} m \\ n \\ o \end{matrix}, \quad B = \begin{matrix} z1 \\ z2 \\ z3 \end{matrix}$$

The solution for this equation is given by:

```
m, n, o = inv(B^T @ B) B^T @ Y
```

After we find the coefficients we plot the points and fitted plane.

Total Least Square Method

1. Importing libraries and extracting data from the given csv files.
2. First find the centroid i.e mean of x, y, and z coordinates. Shift all the data towards origin by subtracting the centroid from original coordinates.
3. Create a matrix of new data sets x, y, and z as columns of this matrix.
4. Calculate

```
X = A^T @ A
```

5. Calculate the eigen values and eigen vectors. Find the smallest eigenvalue and corresponding eigen vector.
6. The coefficients of the plane equation (a, b, c) are given by this eigen vector.
7. Calculate the distance d by using formula

```
d = a*np.mean(X_data1) + b*np.mean(Y_data1) + c*np.mean(Z_data1)
```

8. Plot the data points and fitted plane.

As it is difficult to visualize that which method between the least square and total least square gives the best solutions, I calculated the squared error for both the methods after the selection of optimal plane.

It is calculated by

```
squared_error = np.sum((a * X_data1 + b * Y_data1 + c * Z_data1 - d) ** 2)
```

Interpretation of the results:
Error for total least square is much lesser than that for least square, which we already know because total least square is more robust than least square as all x, y and z are considered independent.
One more observation is that, as dataset pc2 has more outliers as compared to the pc1, the error for least square methods is more for pc2 than that of pc1.

2.2.b

RANSAC Method:

Select the parameters such as num of iterations, threshold distance, num of inliers.

Parameters:

Num of iterations- I have not calculated the num of iterations using the formula given in the ppt. I have randomly taken value as 1000 which is best suited in most of the cases.

Threshold Distance – this is the distance which need to be compared with the distance from plane. If the distance is less than the threshold distance, the point is considered as inlier point.

Num of inliers – I have kept num of inliers as -1, because in worst case scenario there will be at least 0 points which are not inliers i.e. none of the points satisfy the if condition.

1. Importing libraries and extracting data from the given csv files.
2. Define the parameters such as num of iterations, threshold distance, best no of inliers.

3. In each iteration randomly selecting three points from the data set and finding the coefficient of the equation of the plane.
4. Now, find the distance of all the points from this plane using formula,

```
distance_from_plane = np.abs( m*X_data1 + n *Y_data1 - Z_data1 + o)/np.sqrt( m**2
+     n**2 + 1)
```

5. If this distance from the plane for that particular point is less than the threshold distance, this point will be considered as an inlier otherwise it is rejected.
6. If the number of inliers is greater than the previous number of inliers, we update the best no of inliers.
7. Select the coefficients giving the maximum number of inliers during the iterations. This is our optimal solution.
8. Plotting the fitted plane.

Implement these methods for both the datasets.

Interpretation of the results:
As the three points are randomly selected in RANSAC method there is no fix error for this method. But still, as the num of outliers increased from pc1 to pc2, the error increases.

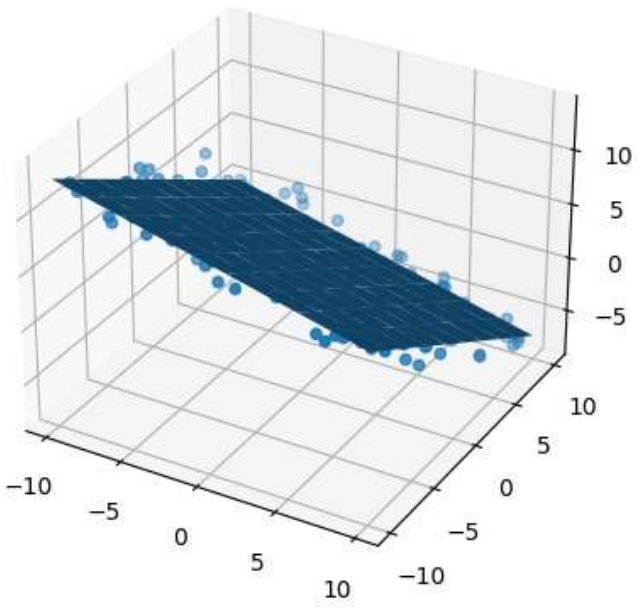
RANSAC should be a better choice for outlier rejection because it is very robust. Also, it is designed to work for a large amount of data. RANSAC iteratively fits models to subsets of the data, which reduces the impact of outliers on the results. During each iteration, RANSAC identifies inliers and uses them to fit the model. This process is repeated until enough inliers have been identified to produce a good model.

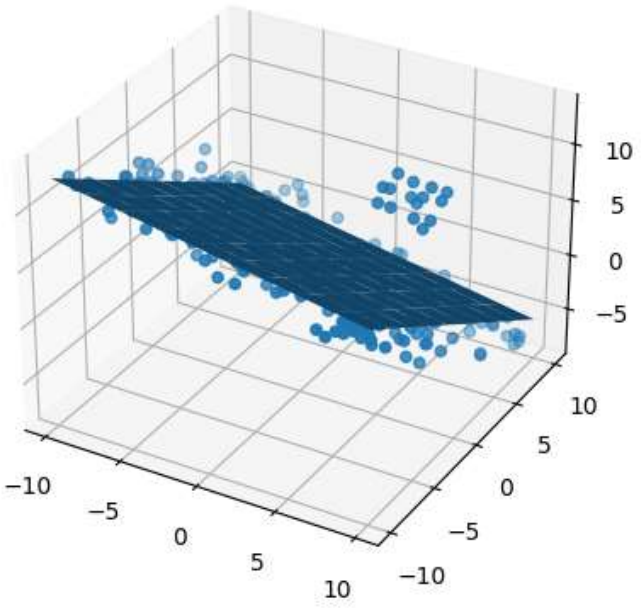
But in our case, I am getting Total Least Square to be the better method because its giving me least squared error. It is possible because the implementation of these methods depends on the type of the data set. RANSAC is computationally expensive and depends on the parameters of the we have chosen. In our case, pc1 does not have many outliers, so in that case using total least square would be more beneficial as its giving better solution and its less expensive.

For the pc2, there is only one cluster outside the other inliers, the total least square also in this case is giving high amount of error. But the error I am getting for RANSAC is highly fluctuating and depends on the series of parameters.

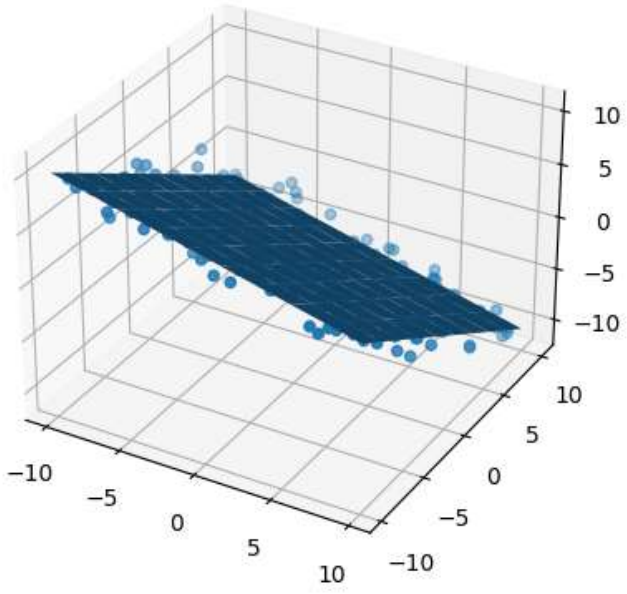
In conclusion, for our datasets, total least square seems to be better method but computing parameters for RANSAC can make significant difference.

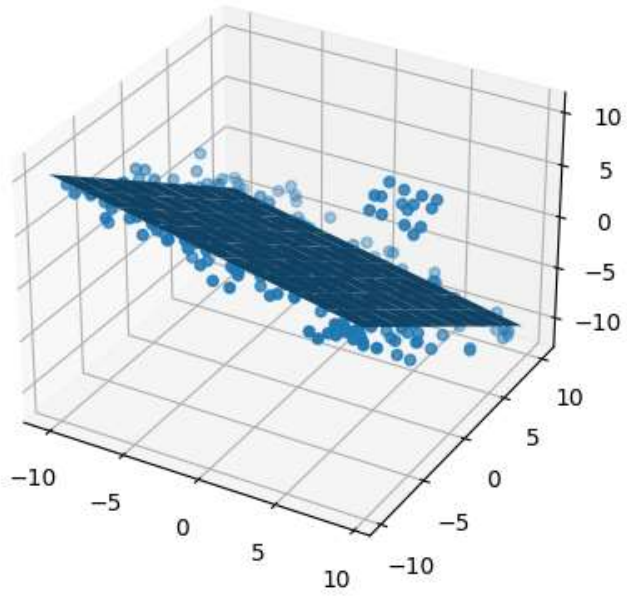
Output results:



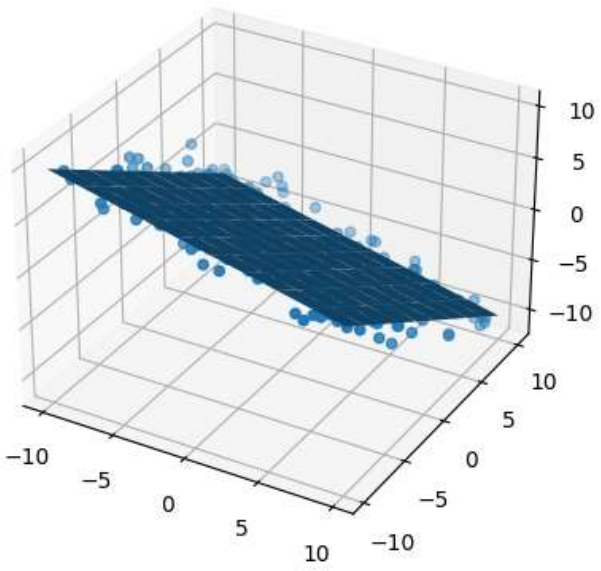


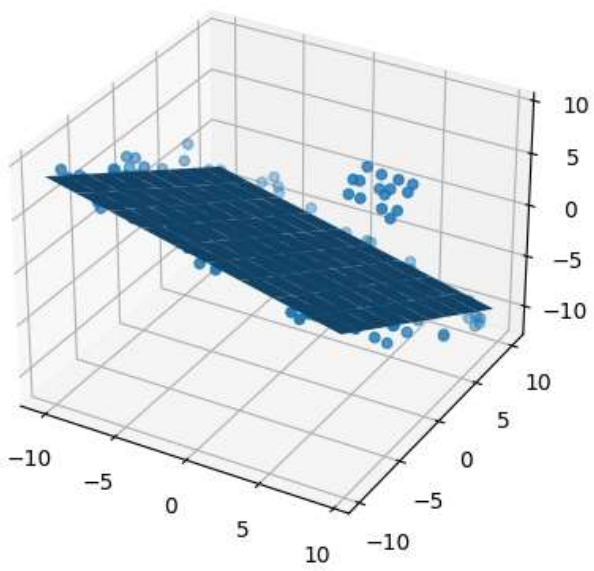
PC1 Least Square Method PC2





PC1 Total Least Square PC2





PC1 RANSAC PC2

Problems Encountered:

I did face any issue in 2.1 except for some silly mistakes.

In 2.2 and 2.3, biggest issue was to understand the three techniques. Least square method was easy to understand and implement too. RANSAC method was difficult to understand. In these methods, non-invertible matrix, out of bound index, plane was not visible are some of the issues I faced. In RANSAC method the parameters were chosen randomly earlier. The code requires several parameters to work correctly, if these parameters are not set correctly the results can be inaccurate. Most if these issues were solved by looping correctly, choosing correct values for parameters etc.

Another issue was to make the input data in the correct format. As this involves, a lot of matrix operations, there were problems when the shape of matrices to be multiplied were not correct.