ENPM 673 Project 1:

Nicholas Novak
Section 0101

Problem 1:

1. Detect and plot the pixel coordinates of the center point of the ball in the video. [10]

   To solve this problem, I used the code provided in appendix 1, Problem 1.1. This way, I masked each frame to remove any pixels outside of the red HSV range. Additionally, some pixels of the hand throwing the ball existed within the mask. Attempting to remove these pixels with HSV or RBG boundaries also removed too much of the ball to reliably track. As such, I overlayed a black mask over the part of the video where the hand is to prevent any pixels from showing through to the mask. The resulting plot was generated in matplotlib based on the balls position in the frame:

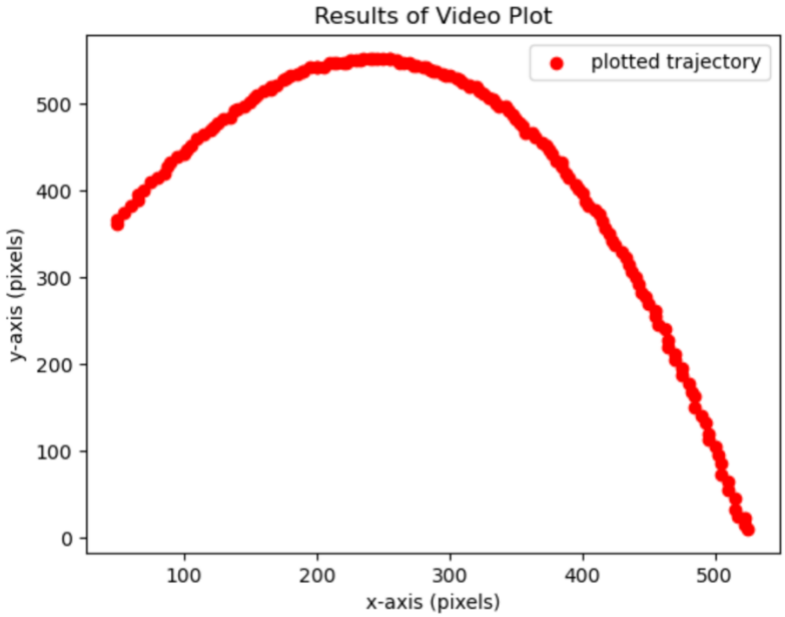

Figure 1: Trajectory of the center of the red ball, in pixels

2. Use Standard Least Squares to fit a curve to the extracted coordinates. For the estimated parabola you must,
   Print the equation of the curve. [5]
   Plot the data with your best fit curve. [5]

To solve this problem, I implemented the 2D least squares method as described in class. This implementation can be seen in appendix 1, Problem 1.2. An interesting problem with this method was ensuring the plot and equation were correct. In plotting the graph, I discovered that the hand pixels were still coming through the mask, which is why I implemented the second black mask for the hand region of the image. This gave a better resulting curve. For the equation, I backed out the parabola using 3 points generated from the results of the least squares method as this way I could ensure the parabola best fit the points that the least squares method generated. Further, to find the final location, I used the following equation: $x_{final} = \sqrt{\frac{y_{final}}{a} - y_{vertex}} + x_{vertex}$ [1], where a is found as the first variable of the parabola equation and the vertex was found as the maximum plotted values. My graph result and equation are shown below:

NOTE: My equation is expressed in terms of the bottom left of the image as an origin. You can see it plotted over the other two lines in Figure 4 below:

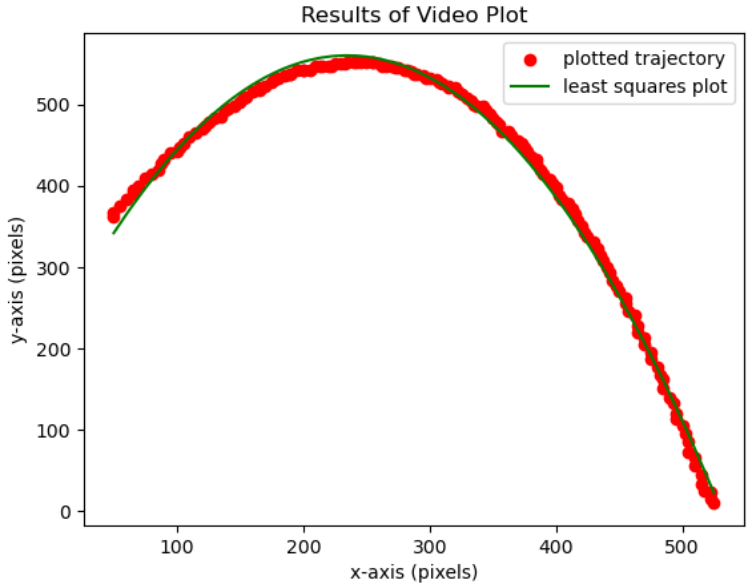

Figure 2: Least squares fit to the tracked red ball trajectory.

```
Equation of parabola:
 y= -0.03239709807147432 x^2 +  3.0272793427363527 x +
41.313675250687325
```

Figure 3: Equation of the curve fit where the bottom left of the image is the origin

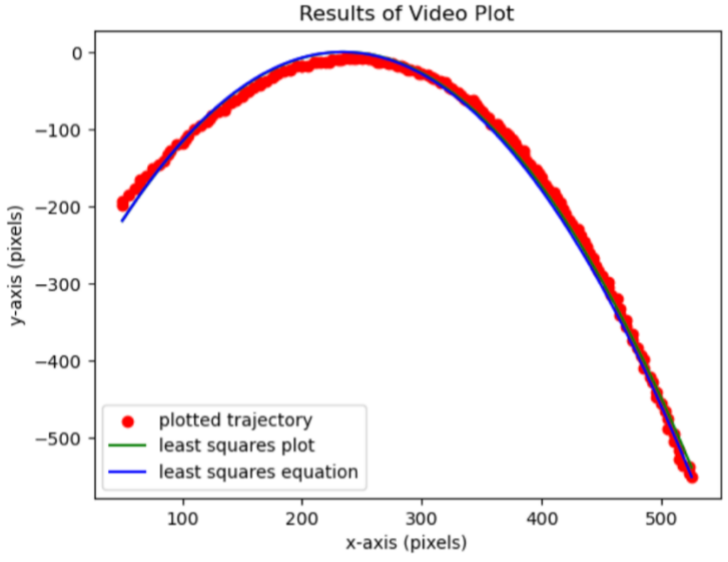

Figure 4: Equation fit to the plot and the final ground landing position of the ball in x-pixels

```
x-value at y =  1137.5  pixels (ground):
650.6381621919668
```

Figure 5: x location of the ball when it hits the ground at 1137.5 pixels down from the top left corner

Problem 2:

1. Using pc1.csv:
   a. Compute the covariance matrix. [15]
   b. Assuming that the ground plane is flat, use the covariance matrix to compute the magnitude and direction of the surface normal. [15]

To solve this problem, I wrote the program seen in appendix 1, Problem2.1.a. This allowed me to find the covariance matrix and eigen values of said matrix. I used the covariance matrix from lecture, $\Sigma = \begin{bmatrix} S_x & COV(y,x) \\ COV(x,y) & S_y \end{bmatrix}$ Where Σ is the covariance matrix for 2D data, and extrapolated it into 3 dimensions based on the format of the above matrix. This gave me the resulting covariance matrix:

```
Covariance Matrix:
 [[ 33.7500586   -0.82513692 -11.39434956]
 [ -0.82513692  35.19218154 -23.23572298]
 [-11.39434956 -23.23572298  20.62765365]]
EigVals and Vects of pc1.csv:
 [ 0.66950978 34.65757844 54.24280557]

 [[ 0.28616428  0.90682723 -0.30947435]
 [ 0.53971234 -0.41941949 -0.72993005]
 [ 0.79172003 -0.04185278  0.60944872]]
```

Figure 6: Resulting covariance matrix and Eigenvalues and vectors of pc1.csv

Next, I took the covariance matrix and used numpy's linear algebra to get the eigenvalues and eigenvectors (Figure 6). I was able to take the minimum of these as the surface normal. This gave me the resulting direction and magnitude of the surface normal seen in pc1.csv:

```
Normal direction (x,y,z):  [ 0.28616428  0.90682723
-0.30947435]
Normal magnitude:  0.6695097797767815
```

Figure 7: Resulting direction and magnitude of pc1.csv's surface normal

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. [20]
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. [20]

To solve part a, I first referenced the lecture to find the least squares fitting equations for use in this application. However, the lecture only covered a 2d application and this was in 3 dimensions. As such, a difficulty for me in this question was figuring out how best to apply the 2d methods to a 3d problem. I used a great deal of time (close to 2 days of work) looking up methods and other resources online to find the best approach to this problem. I believe I found a good approach for standard least squares, but I did not find an effective method in total least squares and my results were less than desirable.

To plot the standard least squares method, I used the code found in appendix 1, Problem 2.a.1. This gave me rather good looking results for both pc1 and pc2, as seen below. The plane equations were -0.353955 x + -0.668551 y + 3.202554 = z and -0.251884 x + -0.671737 y + 3.660257 = z, respectively.

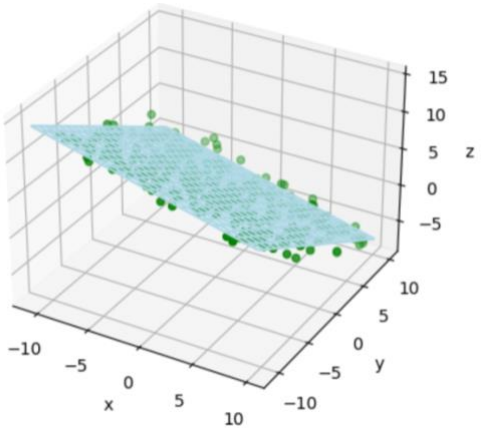

Figure 1: Standard least squares applied to pc1.csv

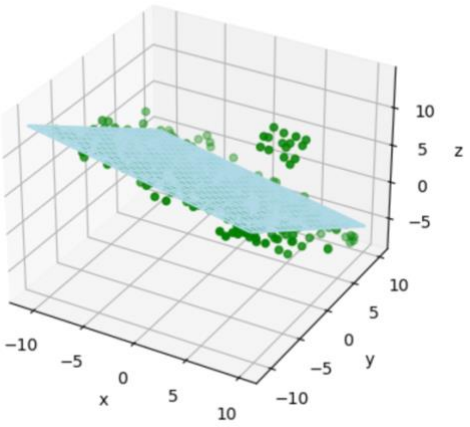

Figure 9: Standard least squares applied to pc2.csv

In my attempt to use total least squares, I did produce some results, but they were clearly not correlated to the lidar data. My method was as follows. First, I got the U matrix by obtaining the values of the sum of each of the axis values minus the mean of that axis' values. I then multiplied this matrix by its transpose. I found the eigen values and vectors of this matrix and obtained the minimum value and corresponding vector. Then, I Used the equation $Z[r,c] = ((minVect[0] * X[r,c] + minVect[1] * Y[r,c] + minVect[2]) - minVect[0] * x\_mean - minVect[1] * y\_mean)/z\_mean$ to calculate each z-value. Finally, I plotted the results and obtained a plane corresponding to them. Despite following both lecture and online resources, it appears either my method was incorrect or this method was not the proper use case for this data.

To generate these results, the code in Appendix 1, Problem2.a.2 was used.

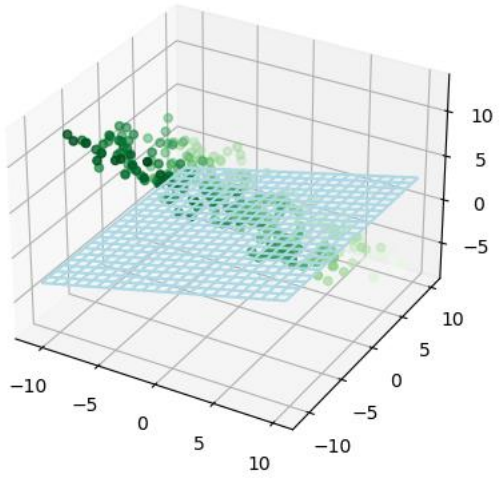

Figure 10: Total Least Squares applied to pc1.csv

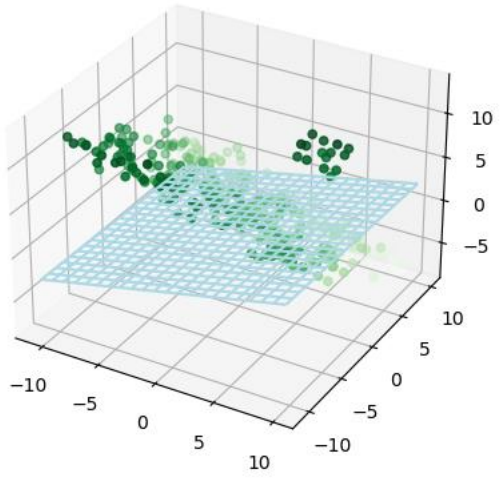

Figure 11: Total least squares applied to pc2.csv

Discussion:
Based on the results, it seems that standard least squares better fits the provided lidar data. This could be due to coding errors but may also be due to the fact that total least squares heavily penalizes outliers and both files, especially pc2.csv, have a plethora of outliers

Part b:

To develop a ransac algorithm, I followed the steps laid out in lecture and also searched for some online resources. My approach was as follows. First, I would iterate a number of times determined by my p ,e, and s values (set at 0.95, 0.3, and 3). This came out to be between 6 and 20 times. The maximum iterations for this set was 445510. Next, I would obtain 3 random points from the given csv dataset. Then, I took 2 vectors formed by these 3 points and the point shared by the 2 vectors. I found the normal to these vectors by crossing them and found the d parameter for the plane equation by dotting the normal with the origin point shared by the 2 initial vectors. Finally, I would calculate the distance to each point for the given iteration by using $distance = (a * x1 + b * y1 + c * z1 - d)/sqrt(a ** 2 + b ** 2 + c ** 2)$ for each x, y, and z. I made sure to exclude the generated points from this calculation as they would give a distance of 0. Then, I found the number of values within an acceptable distance threshold away from the plane, defined through trial and error as 1.85. Finally, I would check the number of points within the threshold to the last iteration's amount. If the current iteration had more, I would cache that value over the previous and repeat the iteration. This way, I ended up with equation parameters and inliers for plotting at the most optimal result. I then plotted these results. Unfortunately, my algorithm did not fit the data very well. My results were as shown below:

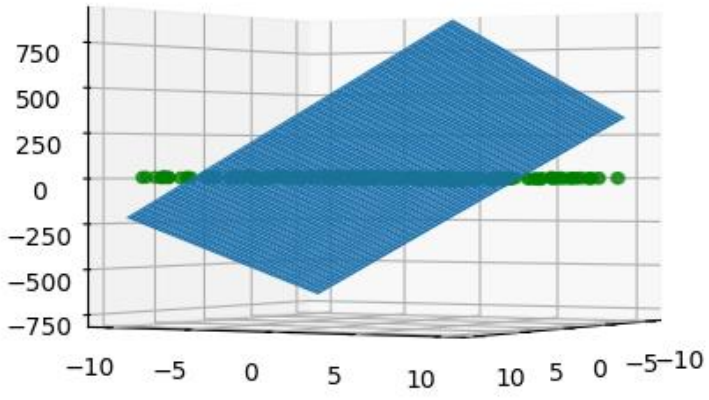

Figure 12: Ransac applied to pc1.csv. Note the very high z axis

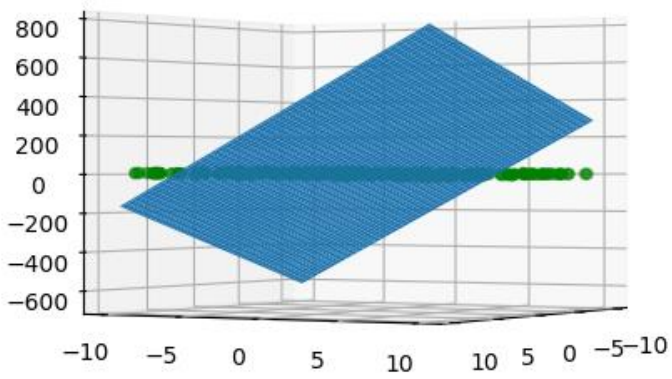

Figure 13: Ransac applied to pc2.csv. Note the different and high z axis. The normal outliers seen in pc2.csv are not visible as a result.

Discussion:

After tuning parameters for hours on my ransac algorithm and attempting to fit my total least squares to the data, I have concluded that standard least squares is the most acceptable approach for this data. This is because my approach both fit the data best and was least expensive to apply. Beyond any inaccuracies in my code, the standard least squares ran the fastest of all algorithms. For a robot that is operating with limited hardware, every byte of data and second of runtime counts. As such, in the given lidar data's environment, I would select standard least squares for its low computational intensity and ability to fit to the surface planes relatively well.

In a more general sense, I expect that outlier rejection would be best handled by a regular ransac algorithm. This is because it would be able to completely disregard any major outliers, where least squares methods usually fall short. The only downside is that ransac can be more computationally intensive and would require a larger dataset to ensure there are enough inliers for proper fitting.

Appendix 1: Code

Problem 1:

Part 1,2,3: See file 673Project1_1.py

Problem2:

Part 1, Part 2.a;total least squares: See file 673Project1_2.py

Part 2.a: See file 673Project1_2.py and 673Project1_3.py

Part 2.b: See file 673Project1_4.py