Project 1

ENPM673

Perception for Autonomous Robots

UID: 119061770

Name: Rohan Maan

Problem 1:

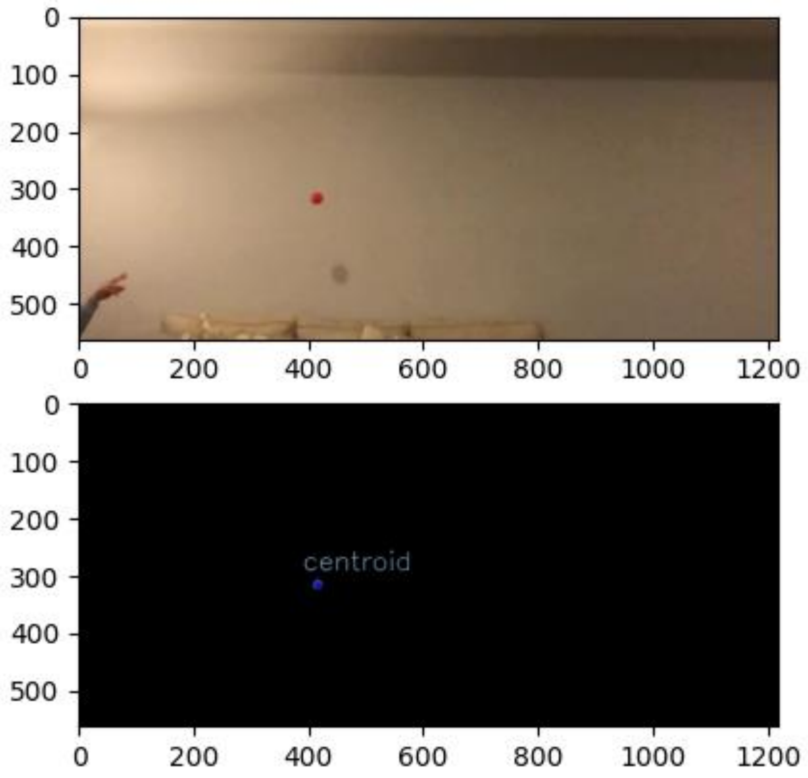
In the given video, a red ball is thrown against a wall. Assuming that the trajectory of the ball follows the equation of a parabola:

1. **Detect and plot the pixel coordinates of the center point of the ball in the video.**

   Ans. This program named Problem_1.py starts by importing libraries, most of which are standard libraries. The code begins by writing a function for calculating the centroid (named – *calculate_centroid*), can be found in defined_functions_1.py file. This function takes input of image. I am passing a filtered image, where only the ball pixels are non-zero values and rest are all 0 (hence black color). The function takes average of x and y coordinates of non-zero values (indicates ball region) and this average indicates the central position of the ball in each frame.

   Extracting Red ball:

   1. Load the image using cv2.imread (for sample)
   2. To read all frames from the video, we can use cv2.VideoCapture() function. In my code, I store all the frames in a list called *videoframes*.
   3. On each frame, we are supposed to iterate to find a red ball, which means in between a certain threshold, only the red color of the ball should be highlighted and the rest will be marked black.
   4. I filtered the red image by setting lower and higher threshold as follows: [0, 0, 130] and [65, 65, 255] indicating lower and upper values for RGB each.
   5. After this filter or mask is obtained, I used cv2.bitwise_and function so that the rest of the image value becomes black and ball can be identified and separated from the background.
   6. Post this, an additional threshold on gray image helps to be sure on the filtered image having just the ball in focus.
   7. A few additional cv functions allow us to mark a circle on the identified spot of the ball and this is also what we used to plot the final coordinates and later used for plotting curve across the following points.

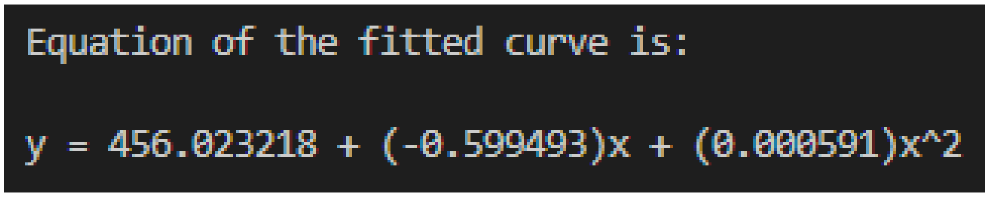


Problems faced:- Tuning the value to obtain pixels for the red ball was tough, it took a while. I had to try different combinations of RGB threshold values to achieve the accuracy of ball. At first, I was unable to detect images in few frames, but eventually after tuning the threshold values, it worked for each frame. Additionally, after filtering the ball, I performed another filter on gray scale image so to remove any components of hand, which was closest match to the ball color and hence it caused maximum issues while tuning.

2. Standard Least Square was used to fit the curve for extracted coordinates. For this we have to first find the coefficients of equation of parabola using the x and y coordinate pair for various location of ball. The equation comes out to be:

   Problems faced: In the beginning I had issues with getting inverse, it somehow worked differently in Google Colab and then in python. So, I had to rewrite complete function, and it worked.

a.



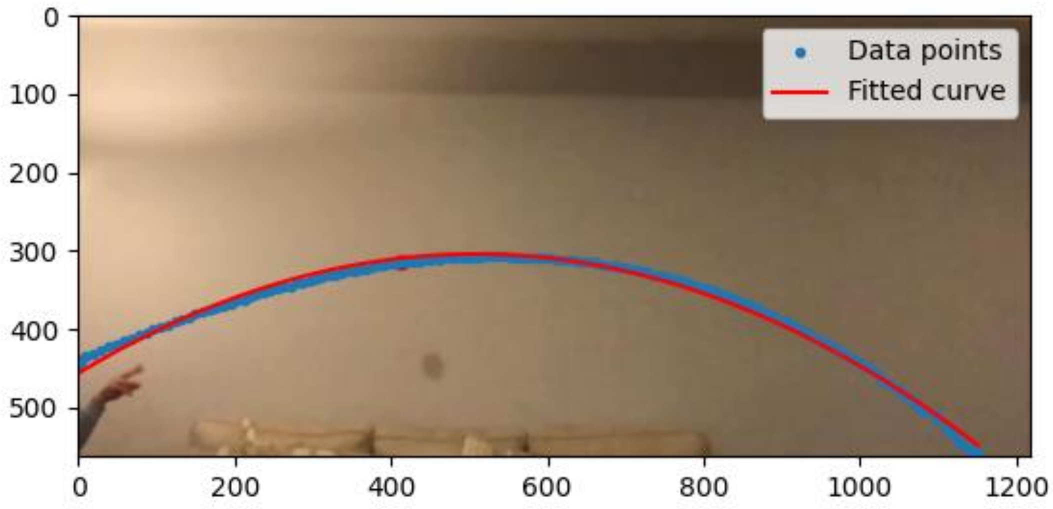
```
Equation of the fitted curve is:

y = 456.023218 + (-0.599493)x + (0.000591)x^2
```

b.

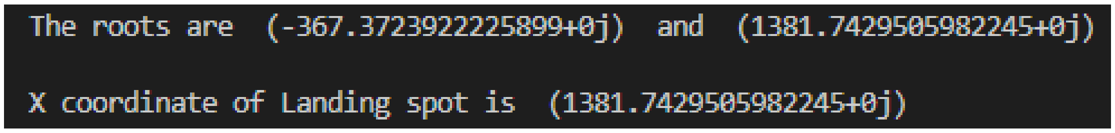
Here, we have plotted x and y points for centroid of ball in each frame. On top of this the curve was plotted, for this we first need to use the coordinates and find the coefficients of the parabolic equation and later this is used to plot the curve making use of the available x values in our frame.

Problems: I faced a lot of problem here mainly into finding the perfect threshold for the ball and particularly because the values are variable for each red, green and blue channel. Components of image other than the ball also have components of red and hence just filtering red was not enough.



3.

Part 3 involves considering coordinate frame where the top left corner is the origin, and this is why I set the coordinate to that beforehand. The condition requires us to add 300 pixel to the entry y intercept. As per our equation, when x = 0, y = 456 and hence total becomes 456 + 300 = 756. At Y=756, we have to calculate the value of x. As we have a quadratic equation, we can find the value of X and that comes out to be -367 and 1381, according to our coordinate, we know that we are looking for the positive value and hence the X will be 1381 at Y = 756. The output from the code is as follows:



```
The roots are  (-367.3723922225899+0j)  and  (1381.7429505982245+0j)

X coordinate of Landing spot is  (1381.7429505982245+0j)
```

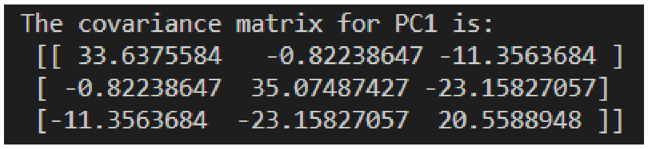
Problems faced: When we plot just the curve, the origin is present on left bottom and hence y values go from 0 to 500, making the curve upside down. However, when we use cv2 image, the origin is at the top left and hence the graph plots as required.

Problem 2:

I have assigned Problem_2.py to solve question 2 and this has main components present in it, however the functions can be found in defined_functions_2.py, each to solve curve for Standard Least Square, Total Least Square and RANSAC and to find covariance matrix. Then these functions were called with values from PC1 and PC2 data lidar data points.

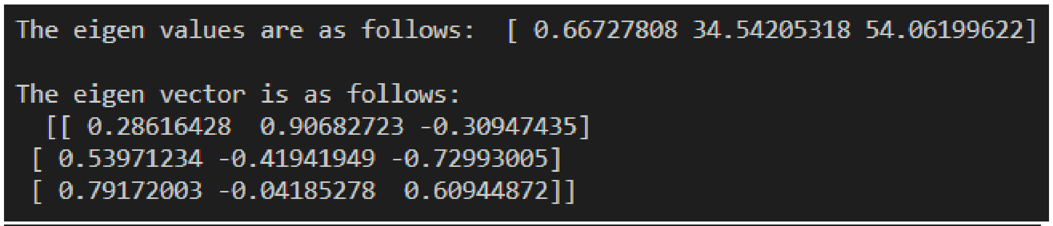
a. The code begins by importing some standard libraries including numpy, pandas and matplotlib.
b. The first step is reading the csv file and extracting data from it. This was done using pandas as they have very convenient functions available to perform reading data from csv.
c. I have stored the values obtained from PC1 into an array named dataFrame1 and PC2 coordinates into array named dataFrame2.
d. As asked in first question, to computer covariance matrix, I have written a function named find_covariance(dataFrame), which calculates covariance from scratch.
e. **Problem faced**: Didn’t face any problem as such, I just wrote the method to find covariance step by step in the function.
f. For part b, we can calculate the magnitude and direction of the surface normal by finding eigen_values and eigen_vector of the covariance

matrix. The eigen vector corresponding to minimum eigen value denotes the direction of normal and the magnitude is denoted by root of minimum eigen value

g. 

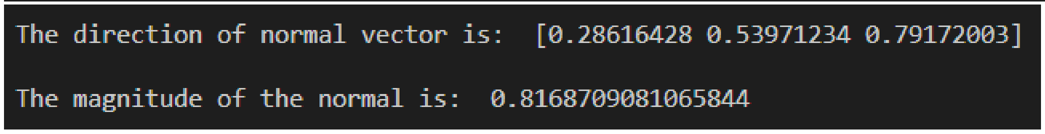
```
The covariance matrix for PC1 is:
 [[ 33.6375584   -0.82238647 -11.3563684 ]
 [ -0.82238647  35.07487427 -23.15827057]
 [-11.3563684  -23.15827057  20.5588948 ]]
```

h. **Problems faced**: Didn't face any problem as we were allowed to use inbuilt functions here.

i. 

```
The eigen values are as follows:  [ 0.66727808 34.54205318 54.06199622]

The eigen vector is as follows:
  [[ 0.28616428  0.90682723 -0.30947435]
 [ 0.53971234 -0.41941949 -0.72993005]
 [ 0.79172003 -0.04185278  0.60944872]]
```

j. 

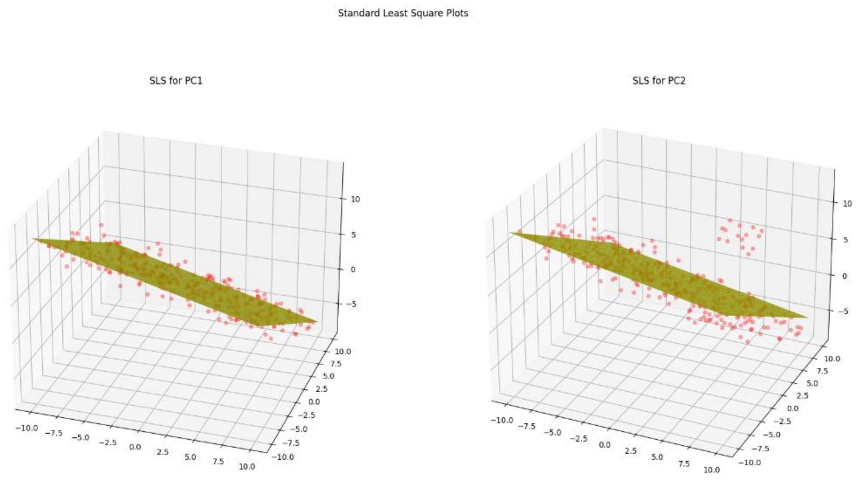
```
The direction of normal vector is:  [0.28616428 0.53971234 0.79172003]

The magnitude of the normal is:  0.8168709081065844
```

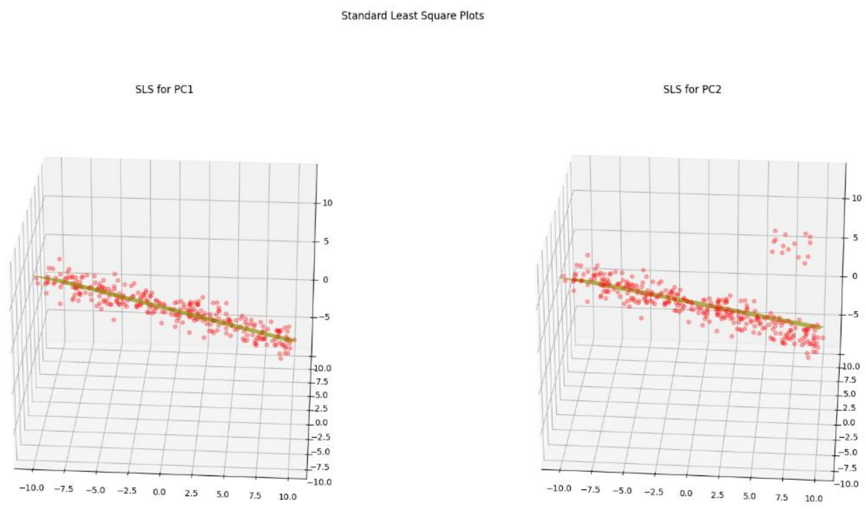
k. In part 2, I have written three separate functions for Least Square, Total Least Square and RANSAC.

l. Standard Least Square function works in the same way, as indicated in question 1, by calculating the coefficients of a parabola, except for that it will now apply to 3D instead of 2D points.

m. Problems faced: I faced problems while writing the algorithm, the steps going from theory to practical, brings in a lot of scope of errors, including how data is interpreted and hence I had to fix that.

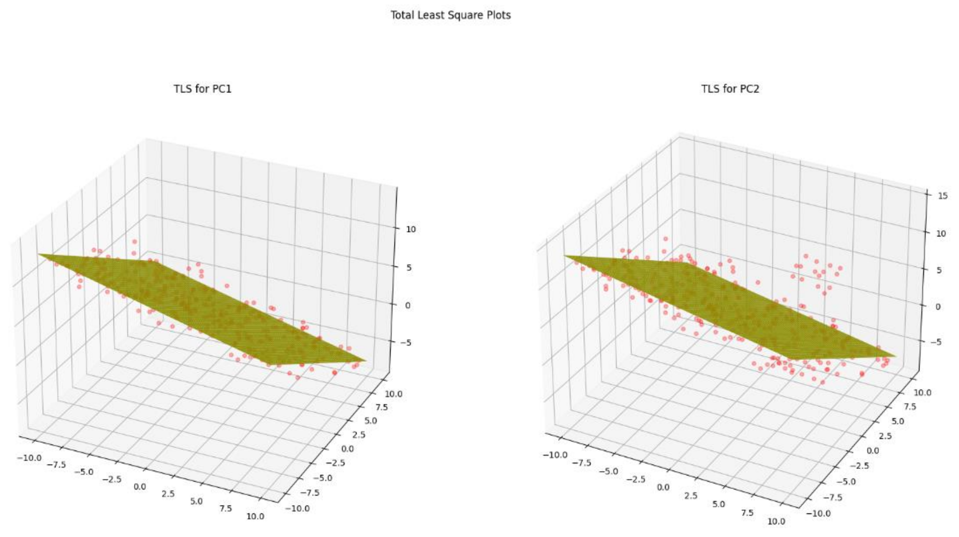


n.

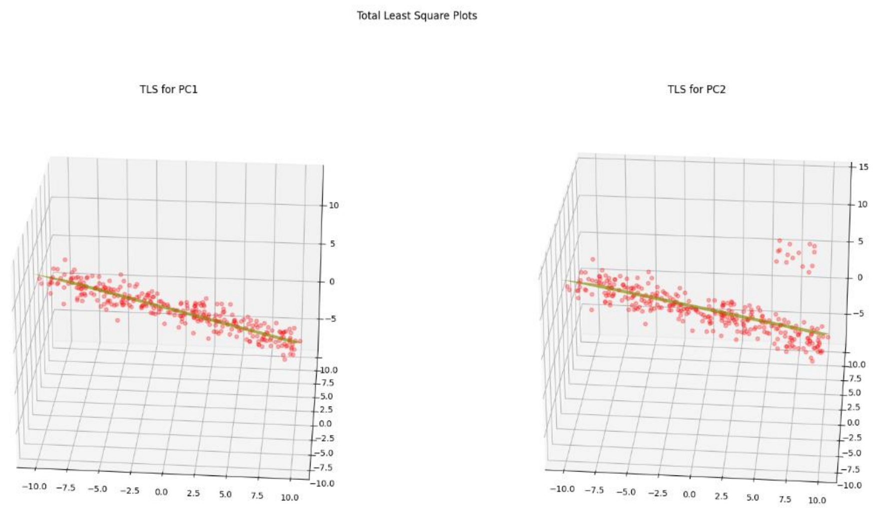


o.

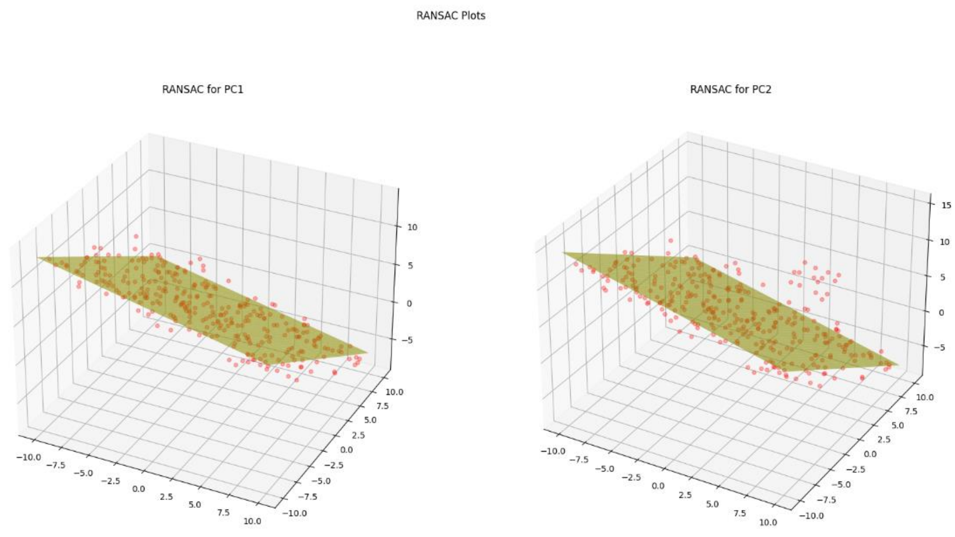
p. Total Least Square function finds distance in both the x and y axis(diagonal) and tries to minimize the distance. This can be done by finding out the eigen values and eigen vectors, then extracting the one with least eigen value. We will utilize these eigenvector values to compute the coefficient and hence obtain the curve of the parabola.



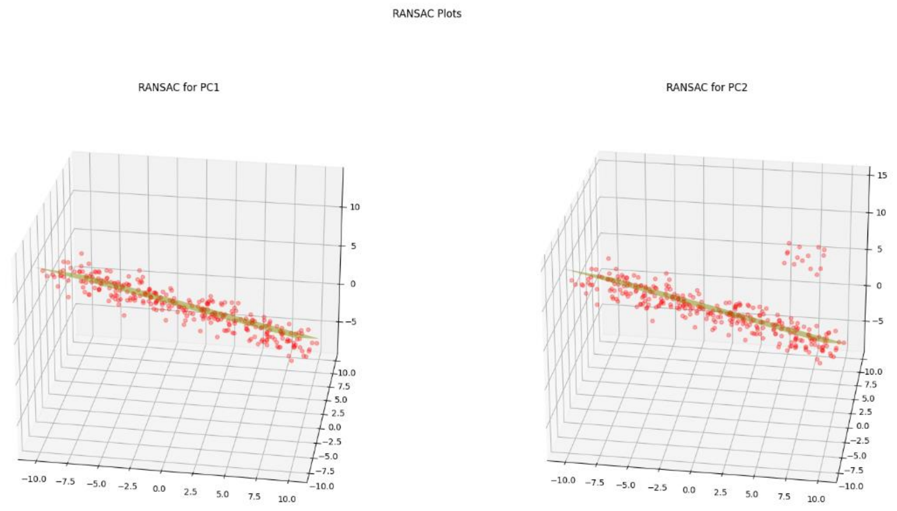
q.



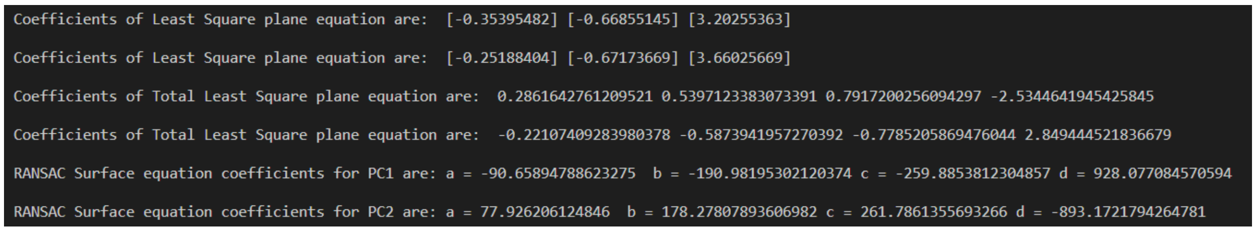
r.
s. **Interpretation: Both standard least square and total square are not robust to outliers and can see the graph bending towards the outlier points. Total Least square took more time to execute because of calculation in both axis instead of just x axis in standard least square. Overall, TLS is more robust than standard least square method.**
t. Coming to RANSAC, we pick up any 3 random points and try to find inliers within a certain threshold (0.2 in our case) and this is used to find the best model which indicates the point with maximum number of inliers.
u. **Problems encountered: It took a while to figure out the number of iterations to run for RANSAC and decided on 1000, which gave quite robust results with the given data. Additionally, I faced problem while deciding the threshold for distance and finally used 0.1.**



v.



w.



```
Coefficients of Least Square plane equation are:  [-0.35395482] [-0.66855145] [3.20255363]

Coefficients of Least Square plane equation are:  [-0.25188404] [-0.67173669] [3.66025669]

Coefficients of Total Least Square plane equation are:  0.2861642761209521 0.5397123383073391 0.7917200256094297 -2.5344641945425845

Coefficients of Total Least Square plane equation are:  -0.22107409283980378 -0.5873941957270392 -0.7785205869476044 2.849444521836679

RANSAC Surface equation coefficients for PC1 are: a = -90.65894788623275  b = -190.98195302120374 c = -259.8853812304857 d = 928.077084570594

RANSAC Surface equation coefficients for PC2 are: a = 77.926206124846  b = 178.27807893606982 c = 261.7861355693266 d = -893.1721794264781
```

The above mentioned are the coefficients of each plane plotted for PC1 and PC2 respectively.

Interpretation: My conclusion from the model fitting tells me that RANSAC is much more robust to outliers and Standard Least Square fitting is the most sensitive one, with total least square fitting existing somewhere in between.