ENPM673 Project 1

Darshit Miteshkumar Desai; (Dir ID: darshit); (UID: 118551722)

February 22, 2023

Contents

List of Figures

0.1 Problem 1

0.1.1 Q1: Detect and plot the pixel corrdinates of the ball

Pipeline

This problem is solved using hue saturation value masking where the frame of the video is threshold-ed such that it filters only the red hue out of the frames.

The hsv values are tuned iteratively to achieve minimum amount of background noise in the frames. This is done to achieve a better fitting parabolic curve.

For plotting the center of the ball we will use the mask frames, the following approach is followed:

- The masked frame is basically a large 2D numpy matrix with 0 and 1 (or 255) as the pixel intensity. We use the function *numpy.nonzero*() to find the (x, y) coordinates of the bright intensity pixels
- The second step is to find the centroid of that bright pixel ed blob we got from the earlier step. We used the function *numpy.mean*() to calculate the centroid of the indexes which we collected earlier.
- Problem encountered: When plotting this centroid indices we found that since we considered the image frame as a matrix the top most point of the ball in the video is the least most y-dimension. Since in a matrix when we go towards the top of the rows the row indices decrease.

Result of plotting the ball centers

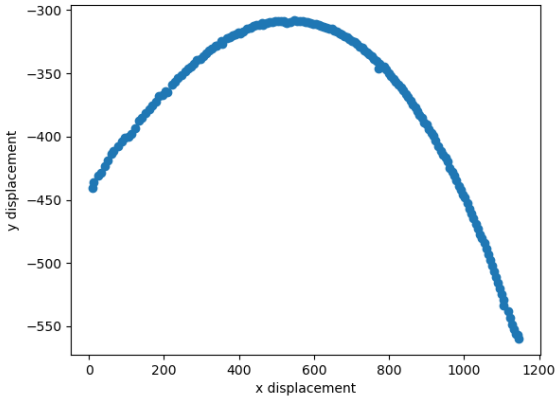

Figure 1: Plot of the center points of the curve, scattered data points (in blue)

0.1.2 Q2: Use standard least squares to fit the curve

a. Print the equation of the curve

Pipeline The first step would be to remove the empty frames which are appended into the list earlier while finding the bright intensity pixels earlier. The second step would be to setup the parabola equation and than do the least squares minimization using that equation.

Let the equation of the parabola be given as below:

$$y = a + bx + cx^2 \tag{1}$$

We have to find the coefficients a, b and c from the given set of points gathered from the earlier approach. To find the coefficients we have the data set $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$ from the list we accumulated earlier.

The minimization function for a function $f(x)$ can be return as below:

$$\Pi = \sum_{i=1}^{n} [y_i - f(x_i)]^2 = \sum_{i=1}^{n} [y_i - (a + bx_i + cx_i^2)]^2 = min. \tag{2}$$

We know that to minimize a function we have to differentiate the function and equate it to zero, In other terms it can be noted that a, b, and c are unknown coefficients while all x_i and y_i are given. To obtain the least square error, the unknown coefficients a, b, and c must be equal to zero for the differentiation.

$$\frac{\partial \Pi}{\partial a} = 2\sum_{i=1}^{n}[y_i - (a + bx_i + cx_i^2)] = 0 \tag{3}$$

$$\frac{\partial \Pi}{\partial b} = 2\sum_{i=1}^{n} x_i[y_i - (a + bx_i + cx_i^2)] = 0 \tag{4}$$

$$\frac{\partial \Pi}{\partial c} = 2\sum_{i=1}^{n} x_i^2[y_i - (a + bx_i + cx_i^2)] = 0 \tag{5}$$

Expanding the equations given above,

$$\sum_{i=1}^{n} y_i = a\sum_{i=1}^{n} 1 + b\sum_{i=1}^{n} x_i + c\sum_{i=1}^{n} x_i^2 \tag{6}$$

$$\sum_{i=1}^{n} x_i y_i = a\sum_{i=1}^{n} x_i + b\sum_{i=1}^{n} x_i^2 + c\sum_{i=1}^{n} x_i^3 \tag{7}$$

$$\sum_{i=1}^{n} x_i^2 y_i = a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i^3 + c\sum_{i=1}^{n} x_i^4 \tag{8}$$

From the equations (6), (7) and (8) we can solve it as a system of equations $Ax = b$, where A, x and b are given as,

$$A = \begin{bmatrix} \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{bmatrix} \tag{9}$$

$$x = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{10}$$

$$b = \begin{bmatrix} y_i \\ x_i y_i \\ x_i^2 y_i \end{bmatrix} \tag{11}$$

The solution of which is calculated using the python function *numpy.linalg.solve*(A, b). This gives $1x3$ numpy vector whose columns are basically a, b and c coefficients of the parabola equation.

The resulting equation is given by:

$$y = -458.603 + 0.6069x - 0.000599x^2 \tag{12}$$

b. Plotting the data with the best fit curve

Pipeline The best fit curve is plotted by defining a function in the code which returns the value of y when given the values of x. A linear space of x is defined using the *numpy.linspace*(). After which the matplotlib function $matplotlib.plot(x, f(x))$ is called to plot the datapoints.

The plot result is shown as below:

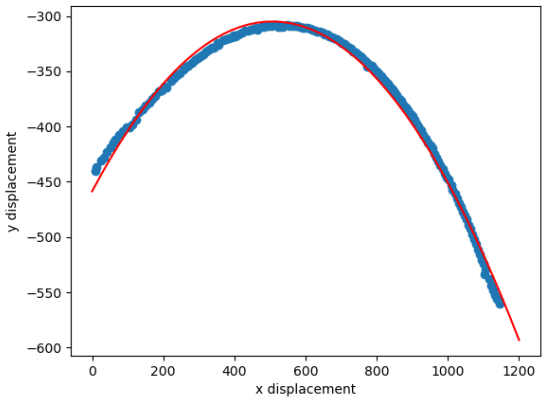

Figure 2: Plot of the fitted curve (in red) with the scattered data points (in blue)

0.1.3 Q3: Compute the landing location of the ball

Pipeline

The landing location of the ball is calculated from the quadratic equation formula $Ax^2 + Bx + C = 0$. To get the parabola equation in the quadratic form we need the value of y which is given as 300 more than the first appearance of the ball in the frame of the video.

This can be easily figured out from the list of y variables we have by indexing the $y[0]$ element and than adding 300 to that element.

$$y[0] + 300 = a + bx + cx^2 \Rightarrow cx^2 + bx + a - y[0] - 300 = 0 \tag{13}$$

The above equation is solved in the code using the quadratic formula for finding out the roots.

It is observed that one of the roots is negative since the image's x-axis can only have positive values that root is ignored.

The resulting landing location is given as $(1358.99, -740.5)$

0.1.4 Problems encountered in the Q1 combination of all parts:

As this problem set was taken as a one single problem and one logical error cascaded into the other part of code not working I have listed the problems faced here:

- Background noise at some of the filter values, where the image of the hand was still in the frames since the hand had also a slight tinge of red. This was corrected by sharpening the red filter values as a compromise to ball's image in some of the frames
- Since we were looking for positive values in the mask for finding the center coordinates of the ball, when the ball wasn't there the coordinates were appended with "nan" values. It was later figured out when the code threw errors during fitting of the parabola.
- Unable to read the frames when the code is run. This was corrected by adding an if statement like the TA showed in class to check whether the frame was being returned or not.

0.2 Problem 2: LIDAR point cloud fitting

0.2.1 Q2.1 Operations on the point cloud 1

2.1.a Compute the covariance matrix

Pipeline The covariance matrix for point cloud 1 is calculated using the below set of equations. The same are coded in a combined python script containing parts 2.1.a and 2.1.b

$$\begin{bmatrix} var(x) & cov(x,y) & cov(x,z) \\ cov(x,y) & var(y) & xov(y,z) \\ cov(x,z) & cov(y,z) & var(z) \end{bmatrix} \tag{14}$$

Where:

$$var(x) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n} \tag{15}$$

$$var(y) = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n} \tag{16}$$

$$var(z) = \frac{\sum_{i=1}^{n}(z_i - \bar{z})^2}{n} \tag{17}$$

$$cov(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n} \tag{18}$$

$$cov(y,z) = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(z_i - \bar{z})}{n} \tag{19}$$

$$cov(x,z) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(z_i - \bar{z})}{n} \tag{20}$$

Where $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$, $\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$ and $\bar{z} = \frac{\sum_{i=1}^{n} z_i}{n}$

2.1.b Compute the surface normal's magnitude and direction from the covariance matrix

Pipeline To compute the surface normal's direction and magnitude the eigen values of the covariance matrix is found using the numpy function $val, vec = numpy.linalg.eig(cov)$ which returns two vectors one containing the eigen values and the second one containing the eigen vectors.

The direction of the surface normal is given by the corresponding eigen vector which has the smallest eigen value.

The magnitude of the surface normal vector is calculated from the norm of the surface normal vector sliced earlier. For this the numpy function $numpy.linalg.norm(eigvec)$ is used.

Results of Q2.1

The covariance matrix, magnitude of the surface normal and the direction of the surface normal is given as below:

$$Direction : \begin{bmatrix} 0.28616428, & 0.90682723, & -0.30947435 \end{bmatrix} \tag{21}$$

$$Magnitude : 0.99 \tag{22}$$

$$CovarianceMatrix = \begin{bmatrix} 33.6375584 & -0.82238647 & -11.3563684 \\ -0.82238647 & 35.07487427 & -23.15827057 \\ -11.3563684 & -23.15827057 & 20.5588948 \end{bmatrix} \tag{23}$$

Problems encountered in Q2.1

- After importing the csv data for calculating the covariance matrix we faced the array shape error due to incorrect shape during multiplication. The shape was corrected using reshape function while taking the mean of the numpy arrays.

0.2.2 Q2.2 Applying Standard least sqaure, Total least square and RANSAC on point cloud 1 and 2

2.2.a Applying standard and total least squares on the point clouds 1 and 2

Standard least squares pipeline The standard least squares method is used to fit the following plane equation:

$$ax + by + c = z \tag{24}$$

The equation given above can also be represented in matrix form as:

$$Ax = b \tag{25}$$

Where A, x and b can be substituted and written as:

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ & \dots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \dots \\ z_n \end{bmatrix} \tag{26}$$

Where a, b and c are the unknowns or coefficients of the desired plane we want to find, This is found by multiplying A^T on both sides of the equation (25) and than moving the $A^T A$ term to the RHS by applying the inverse,

$$(A^T A)b = A^T x \Rightarrow b = (A^T A)^{-1} A^T x \tag{27}$$

Results for Standard least squares The results for simple least squares fitting point cloud 1 looks as below: The results for simple least squares fitting point cloud 2 looks as below:

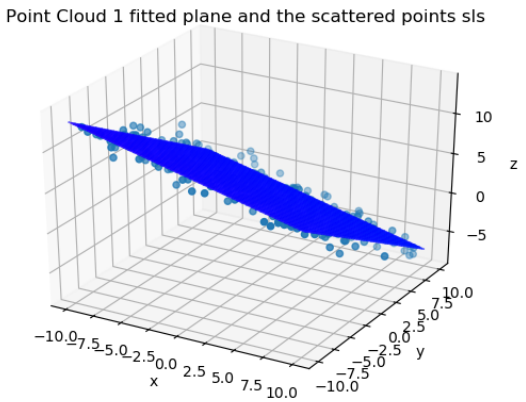

Figure 3: Simple least squares fitting of point cloud 1

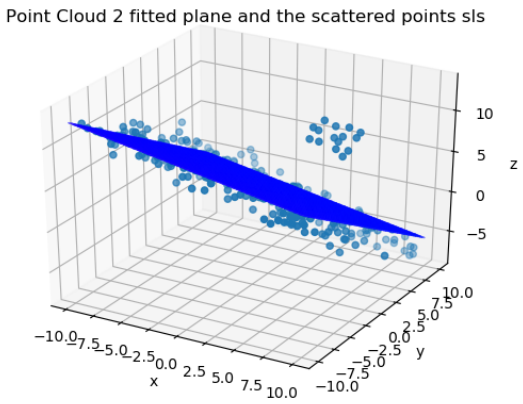

Figure 4: Simple least squares fitting of point cloud 2

Total least squares pipeline The total least squares method is used to fit the following plane equation:

$$ax + by + cz = d \tag{28}$$

Distance D between point (xi, yi) and plane $ax + by + cz + d = 0$;

$$D = \frac{|ax_i + by_i + cz_i - d|}{\sqrt{a^2 + b^2 + c^2}} \tag{29}$$

Find (a, b, c, d) such that it minimizes the sum of squared of perpendicular distances, for that $a^2 + b^2 + c^2 = 1$
The minimization function is given as

$$E = \sum_{i=1}^{n} (ax_i + by_i + cz_i - d)^2 \tag{30}$$

Differentiating the minimization function by d and equating it to 0,

$$\frac{\partial E}{\partial d} = \sum_{i=1}^{n} -2(ax_i + by_i + cz_i - d) = 0 \tag{31}$$

Finding the value of d by moving the terms on the RHS,

$$d = \frac{a}{n}\sum_{i=1}^{n} x + \frac{b}{n}\sum_{i=1}^{n} y + \frac{c}{n}\sum_{i=1}^{n} z \Rightarrow a\bar{x} + b\bar{y} + c\bar{z} \tag{32}$$

Back substituting value of d in equation (30),

$$E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}) + c(z_i - \bar{z}))^2 \tag{33}$$

The function E can be rewritten as below:

$$E = \left\| \begin{bmatrix} (x_1 - \bar{x}) & (y_1 - \bar{y}) & (z_1 - \bar{z}) \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ (x_n - \bar{x}) & (y_n - \bar{y}) & (z_n - \bar{z}) \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \right\|^2 = (UN)^T UN \tag{34}$$

Differentiating E with respect to N

$$\frac{dE}{dN} = 2(U^T U)N = 0 \tag{35}$$

Solution to $(U^T U)N = 0$, subject to $||N||^2 = 1$: eigenvector of $U^T U$ associated with the smallest eigenvalue (i.e., the least squares solution to the homogeneous linear system of equation $UN = 0$).

Basically the eigen vector with the smallest eigen value gives us a, b, c and than this values are back substituted in equation (32) to find the value of d.

Results for Total least squares The results for total least squares fitting point cloud 1 looks as below:

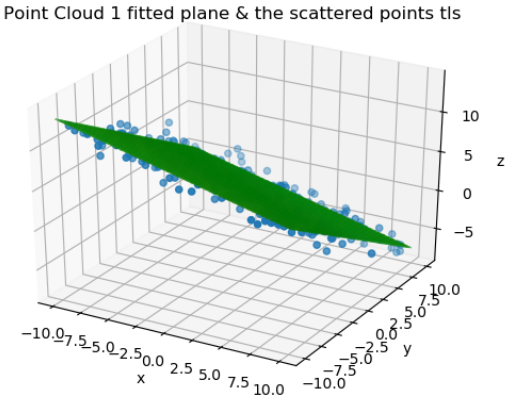

Figure 5: Total least squares fitting of point cloud 1

The results for total least squares fitting point cloud 2 looks as below:

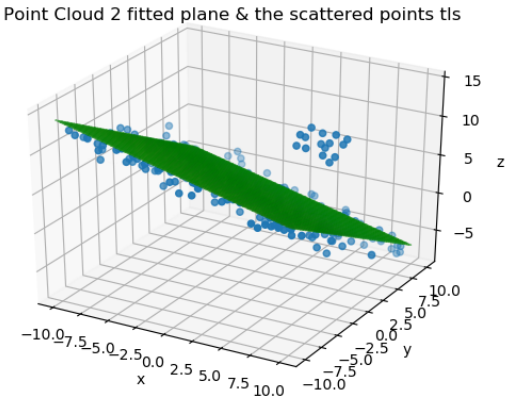

Figure 6: Total least squares fitting of point cloud 2

Interpretation of results obtained in Q2.2.a

- In the case of the datasets given in this problem both the standard least square and the total least square method seem to give the best fitting output. Although simple least square might fail for vertical lines and might be rotation invariant in this scenario it didn't matter much.

- It was also the notion that total least squares is a much better method than the standard least squares but in both the datasets the output was the same due to low outlier ratio. Also it was taught to us that total least square is computationally intensive although the eigs function of numpy ran pretty fast on my computer. Although it maybe the case when the same method is used on a edge processor of a robot the total least square and standard least square might slow down the functioning of a robot since they will be doing complex matrix operations in a limited memory environment.

2.2.b Applying RANSAC on the point clouds 1 and 2

RANSAC pipeline The RANSAC code works as follows:

- Step 1: Define the thresholds for the algorithm like distance and the amount of inliers, no. of iterations
- Step 2: Form a while loop, in this select 3 random points from the data set and form a plane out of it
- Step 3: Measure the distances of all points in the data set from the generated plane if they are within the threshold distance add it to the inliers list
- Step 4: Check whether the number of inliers are greater than the inlier threshold or not if they are add it to the final list. If in another iteration the number of inliers increase than the current amount of inliers reassign the target plane variable and the inliers
- Step 5: Return the best inliers and the best plane given by the function and use it to plot the 3d plane as well as the inliers. Notice in pointcloud 2 the points isolated separately do not come in the inliers list and hence RANSAC is able to fit and filter consistent points from the dataset of point cloud 2

There are basically 3 functions written in the python script for running RANSAC.

- ransac(): This is the function which houses the loop for running ransac algorithm. It returns the plane coefficients and the list of points which are inliers. This function contains the parameters of the ransac algorithm like threshold distance, inlier quantity threshold, number of iterations, quantity of points for forming the plane. From this function two more functions are called responsible for forming the plane and calculating the distance of points in the data set from the formed plane
- check_plane(): This function takes a list of 3 (x, y, z), points, let it be A, B and C as arguments and form a plane using the following math:

$$\vec{BA} = \vec{BO} - \vec{AO} \tag{36}$$

$$\vec{CA} = \vec{CO} - \vec{AO} \tag{37}$$

$$\vec{BA} X \vec{CA} = \vec{n} \tag{38}$$

  The coefficient a, b, c of the plane are the three elements of the cross product vector

$$a, b, c = [\vec{n}[0], \vec{n}[1], \vec{n}[2]] \tag{39}$$

  The value of d can be found out by:

$$d = \vec{n}.\vec{AO} \tag{40}$$

  Where $\vec{AO}, \vec{BO}, \vec{CO}$ are distances of points $A, BandC$ from the origin

  This function returns the coefficients of the plane $ax + by + cz + d = 0$.

- dist_tempplane_points(): This function checks the distance of points in the dataset from the generated plane. This function takes two arguments the list of points and the plane coefficients. It calculates the distance of the point from the plane using the below formula

$$distance = \frac{ax_i + by_i + cz_i + d}{\sqrt{a^2 + b^2 + c^2}} \tag{41}$$

  This function than return the value of the distance back to the ransac() function where it is compared continuously and if they lie inside the distance threshold defined they are added to the inliers list

Results of RANSAC The results for RANSAC fitting for point cloud 1 looks as below:

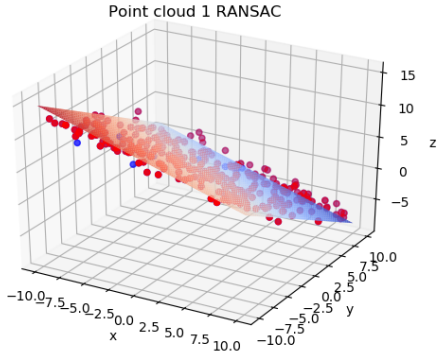

Figure 7: RANSAC fitting of point cloud 1

The results for RANSAC fitting for point cloud 2 looks as below:

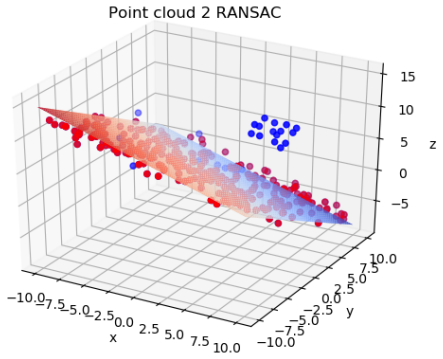

Figure 8: RANSAC fitting of point cloud 2

0.2.3 Problems encountered Q2.2:

- While plotting the surface plane the shape of the vectors x, y, z caused an error while using the function $numpy.plot\_surface()$, this was corrected by uniformly placing the x,y,z values using the $numpy.meshgrid()$ function. This problem was faced for both Q2.2.a and Q2.2.b
- While passing the numpy array sometimes the array gets appended maybe because numpy passes the array by reference instead of by value so modified the code to make a copy of the array before doing any modifications on it.
- Since the parameters of RANSAC had to be tuned I faced a hard time adjusting my algorithm's parameters and the thresholds. Ultimately I was able to arrive at a value of distance thresholds where for both datasets about 95% of the points were inliers.

0.2.4 Discussion on graph fitting methods

- From the results obtained from the datasets it can be observed that all of the methods were able to almost fit the plane with majority of datapoints.

- Usually standard least squares and total least squares are adversely affected from the outliers present in the dataset sometimes which result in a plane which is formed completely offset to the majority of the datapoints
- However in our case since the amount of outliers were significantly small as compared to the inliers $15-20$ for dataset 2 specifically. This observation is evident in dataset 1 where there are almost no outliers in the RANSAC method with a threshold distance of 1.8 units the inliers were 291/300
- For dataset 2 with threshold distance at 2.35 the inliers were 300/315. It is worth to note that above $2.35-5$ the number of inliers remain the same since the 15 outliers are displaced from the plane at an average distance of 5 units
- Overall it can be said that for Standard least squares and Total least squares a separate method to reject those outliers might be required (Not in this case since they are low in number). But in case of RANSAC the algorithm considers the outliers and the inliers and plots the best fitting plane with 95% accurracy
- In conclusion I would like to say that Ransac is the best graph fitting method for this scenario of datasets since it had pretty high inlier ratio around 300:315. This apart from the low memory taken by Ransac for computations might work well in real world environments where this algorithms might be deployed for real time filtering of LIDAR data points

0.3 References

1. Lecture Notes of ENPM673 Course