University of Maryland

College Park

PROJECT 1

ENPM673 - PERCEPTION FOR AUTONOMOUS ROBOTS

Red Ball Tracking using OpenCV
Estimation Algorithms using Python

Author:

Aaqib Barodawala (119348710)

Instructor and TAs:

Dr. Samer Charifa

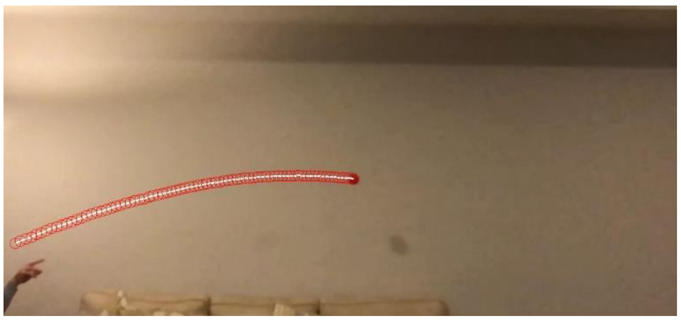
Arunava Basu, Madhu NC,

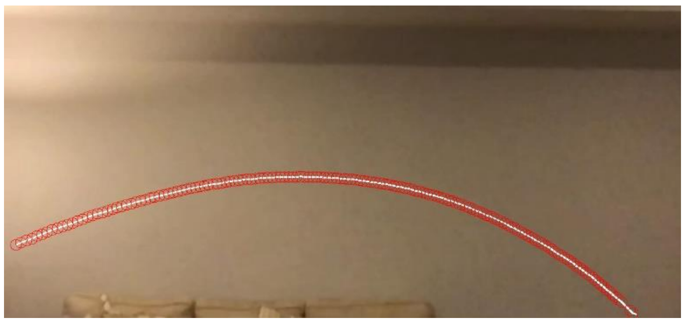
Ritvik Oruganti

PROBLEM 1.1:
Detect the ball and plot the pixel coordinates of its centre point in the video.

SOLUTION:

The solution begins with importing all the necessary libraries for OpenCV, math libraries (NumPy and SymPy) along with matplotlib library for basic plotting. The video is then read in using OpenCV's "read()" function to get each frame of the video. Once the frames are separated, each frame is then converted to HSV (Hue Saturation Value) colour space to help differentiate between red and various other colours. Using another OpenCV function, "inRange()" helps to provide masking using threshold values to separate the red pixels and convert them to white. Using a Median Blur and Erode function from OpenCV helps to remove the outliers present in each frame. Once the red pixels are converted to white which only represent the pixels of the ball, we use another OpenCV function "findNonZero()" to get all the pixel coordinates of white colour. After taking mean of all the detected coordinates in each frame, the resulting coordinates represent the centre point of the ball which is thrown into an array for plotting.





PROBLEMS FACED:

Masking each frame proved to be hassle for finding the red color of the ball. With continuous trial error for threshold values helped to separate the ball from the background and applying Median Blur and Erode functions helped to remove the outliers after masking.

PROBLEM 1.2:
Estimate the trajectory of the ball using the extracted coordinates by applying the Standard Least Square Method. Print the equation of estimated curve and plot the same with the actual trajectory.

SOLUTION:

There are two ways to implement this solution, using matrix operations and using normal equations to estimate the equation of the curve. Both will be discussed below.
Disclaimer – I have used second method to estimate the curve.

1. Matrix Operations:

The equation of second-degree parabola is given by:

$$y = ax^2 + bx + c$$

Since the centre point of the ball is already extracted from each frame, the points can then be substituted into the above equation to get n number of equations, where n represents the number of pixel coordinates found. This system of equations can then be written as a vector/matrix equation:

$$A\vec{x} = \vec{b}$$

Where A represents the $n\ x$ 3 coefficient matrix,

$\vec{x}$ represents the matrix of a, b, c; $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$

$\vec{b}$ represents $n\ x$ 1 matrix of all the values of y.

The coefficients of the estimated curve can then be calculated using Standard Least Square Method by using the given formula:

$$\vec{x} = (A^T A)^{-1}\left(A^T \vec{b}\right)$$

After finding the estimated coefficients, the equation of parabola can then be formulated and fitted with the actual trajectory of the ball.

2. Normal Equations:

The equation of second-degree parabola is given by:

$$y = ax^2 + bx + c$$

Since the centre point of the ball is already extracted from each frame, the points can then be tabulated and the corresponding unknowns in the below normal equations can be found. These are the x^2, x^3, x^4, xy and x^2y values with their sum.

The normal equations are then given by:

$$\sum y = an + b\sum x + c\sum x^2$$

$$\sum xy = a\sum x + b\sum x^2 + c\sum x^3$$

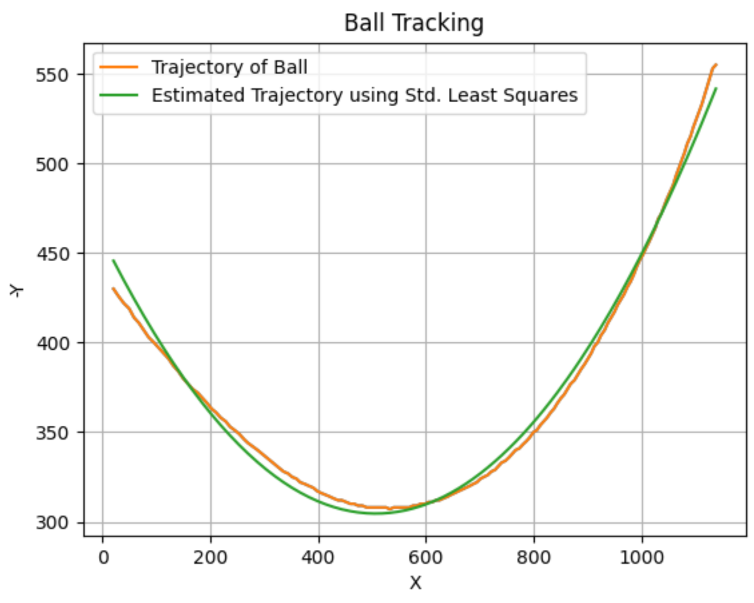
$$\sum x^2y = a\sum x^2 + b\sum x^3 + c\sum x^4$$

Substituting the values in these normal equations leads to three equations with three unknowns. Solving them gives the estimated coefficients and the estimated equation of parabola can then be formed.

Link: https://atozmath.com/example/CONM/LeastSquare.aspx?he=e&q=2

PROBLEMS FACED:

Solving for coefficients of the equation proved to be a hassle. Using SymPy library helped.

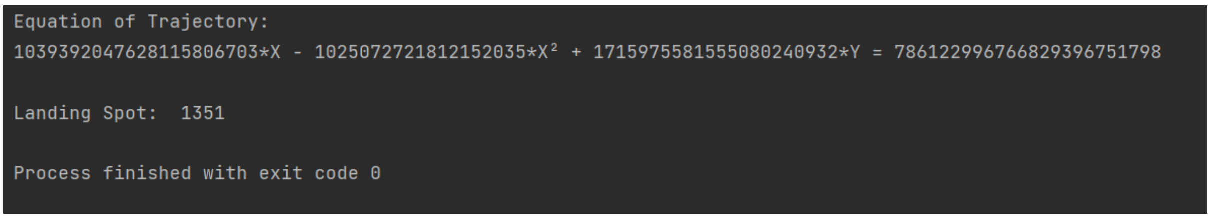


PROBLEM 1.3:

Find x-coordinate of the landing spot of the ball if the y-coordinate is given as 300 pixels greater than its first detected location.

SOLUTION:

After finding the estimated equation of the parabolic curve, the x-coordinate of the landing spot can be easily found by substituting the y-coordinate in the equation and solving for x. Sympy library was used to solve this problem.



```
Equation of Trajectory:
10393920476281158067O3*X - 1025072721812152035*X² + 171597558155508024O932*Y = 786122996766829396751798

Landing Spot:  1351

Process finished with exit code 0
```

PROBLEMS FACED:

I was getting negative values for x-coordinate of the landing spot. Further debugging helped me in noticing that there were two solutions to the equation. Using the second solution helped in getting the appropriate value of x-coordinate.

PROBLEM 2.1.a:
Using pc1.csv, compute the covariance matrix.

SOLUTION:

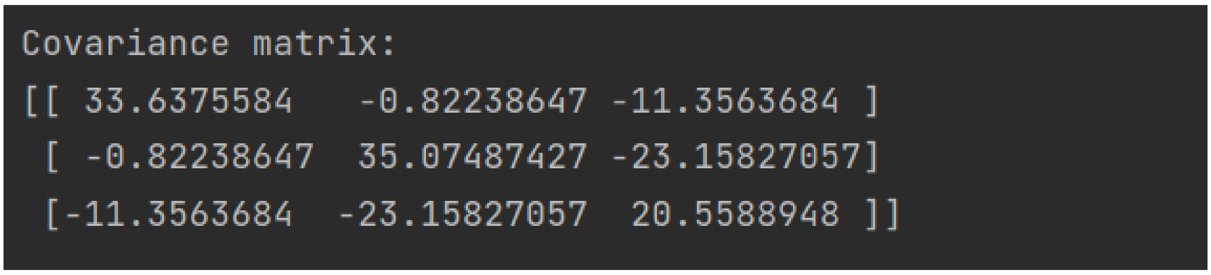
Using the Pandas library, the csv file was read in and the X, Y, Z points were then stored in a NumPy array. The "np.vstack()" was then used to make a new 2D NumPy array, combining all the X, Y, Z points respectively. The mean of all X, Y, Z points respectively is then computed using the "mean(axis=0)" function. Using "shape" function, total elements can be found out and then the Covariance matrix can be computed using the following formula:

$$Cov(\vec{X}) = \frac{1}{n}\left[(\vec{X} - \vec{U})(\vec{X} - \vec{U})^T\right]$$

Where $\vec{X}$ represents n x 3 data vector,

$\vec{U}$ represents 1 x 3 mean vector,

n represents the total number of X, Y, Z elements.



```
Covariance matrix:
[[ 33.6375584   -0.82238647 -11.3563684 ]
 [ -0.82238647  35.07487427 -23.15827057]
 [-11.3563684  -23.15827057  20.5588948 ]]
```

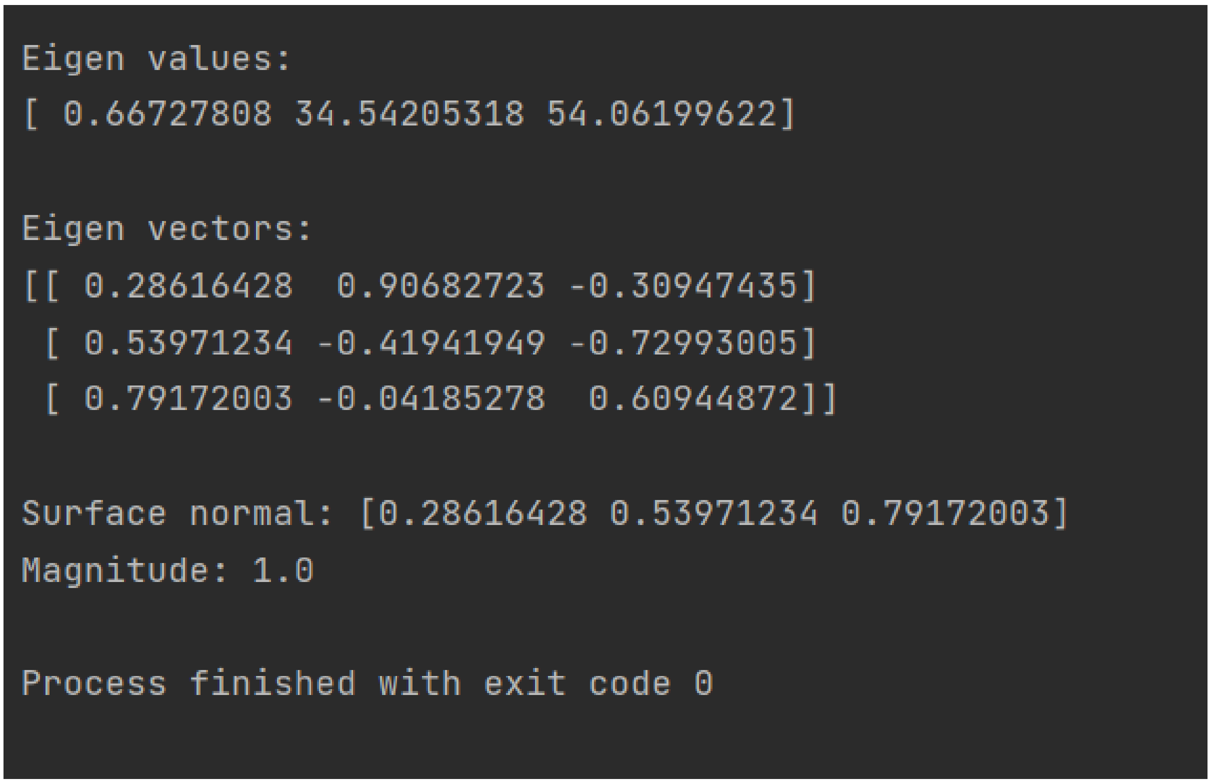
PROBLEMS FACED:

Reading the csv files proved to be a hassle at first. With further research, using Pandas library helped in reading the data as data frame and then converting the data into 2D array with NumPy helped in processing the data.

PROBLEM 2.1.b:
Using pc1.csv, use the covariance matrix to compute the magnitude and direction of the surface normal.

SOLUTION:

After finding the covariance matrix, the smallest eigenvalue and the corresponding eigen vector of the covariance matrix needed to be computed to find the surface normal and its magnitude. The NumPy function “linalg.eig()” can be used to calculate this. The surface normal is defined as the eigen vector with the smallest eigenvalue of the covariance matrix and its norm shall give the magnitude. “linalg.norm()” can be used to calculate the norm.



```
Eigen values:
[ 0.66727808 34.54205318 54.06199622]

Eigen vectors:
[[ 0.28616428  0.90682723 -0.30947435]
 [ 0.53971234 -0.41941949 -0.72993005]
 [ 0.79172003 -0.04185278  0.60944872]]

Surface normal: [0.28616428 0.53971234 0.79172003]
Magnitude: 1.0

Process finished with exit code 0
```

PROBLEMS FACED:

None.

PROBLEM 2.2.a:
Using pc1.csv and pc2.csv, fit a surface to the data using Standard Least Square and Total Least Square method. Plot the results and explain your interpretation.

SOLUTION:

STANDARED LEAST SQUARE

Using the Pandas library, the csv file was read in and the X, Y, Z points were then stored in a NumPy array. All the Z values are then stored in n x 1 matrix and X matrix is formulated by using the "np.vstack()" function.

$$X = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix}; \mathrm{Z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix}; \mathrm{B} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

The coefficients of the estimated surface plane can then be computed using the following equation:

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

For plotting, the following equation of the plane is then used to get all the Z-coordinate values:

$$z = ax + by + c$$

TOTAL LEAST SQUARE

Using the Pandas library, the csv file was read in and the X, Y, Z points were then stored in a NumPy array. The "np.vstack()" was then used to make a new 2D NumPy array, combining all the X, Y, Z points respectively. The mean of all X, Y, Z points respectively is then computed using the "mean(axis=0)" function. The u matrix is then computed as:

$$u = (\vec{X} - \vec{U})(\vec{X} - \vec{U})^T$$

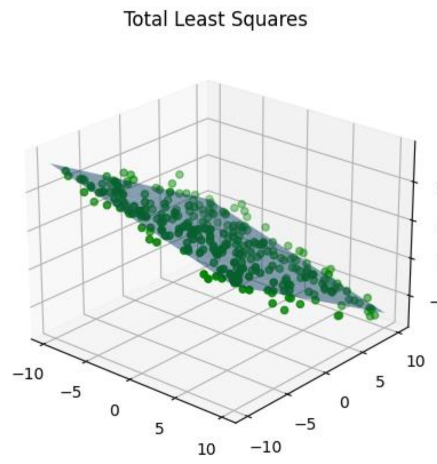
The coefficients of the estimated surface plane can then be computed by calculating the eigen vector with the smallest eigenvalue of the u matrix. To calculate d, the following equation is used:

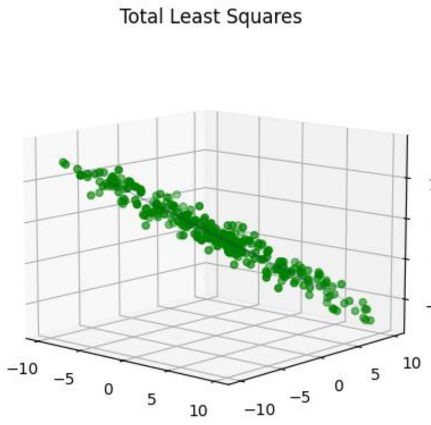
$$d = a\bar{x} + b\bar{y} + c\bar{z}$$

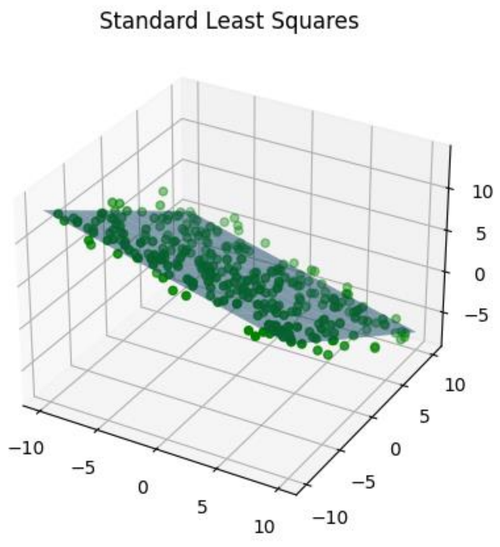
For plotting, the following equation of the plane is then used to get all the Z-coordinate values:

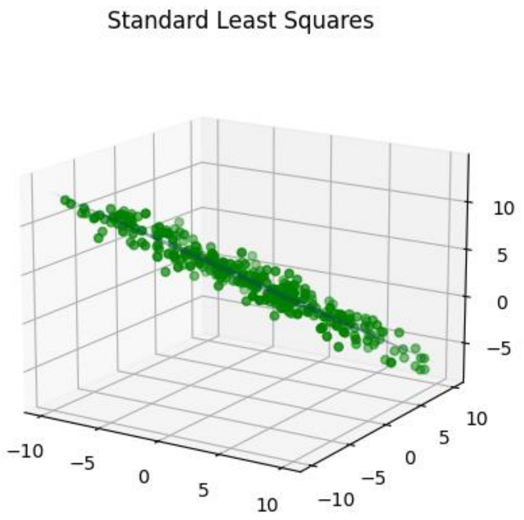
$$z = -\left(\frac{1}{c}\right)(ax + by - d)$$

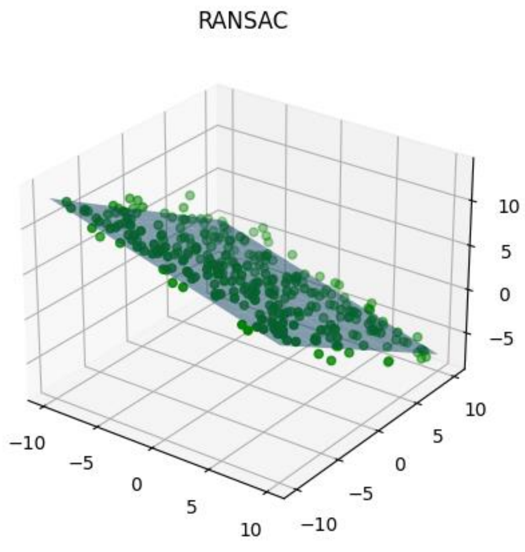
PC1:

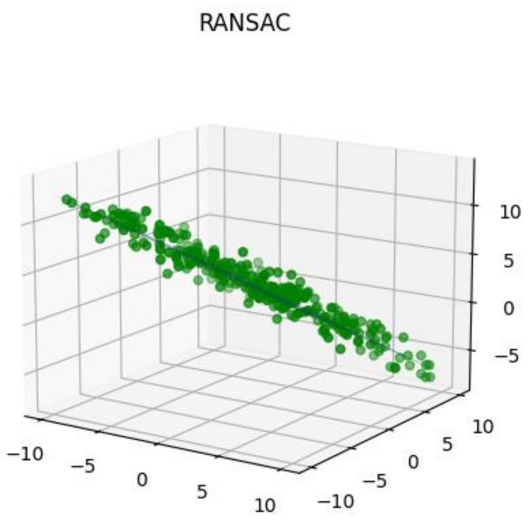

Total Least Squares
10
5
0
−5
−10
−5
0
5
10
−10
−5
0
5
10


Total Least Squares
10
5
0
−5
−10
−5
0
5
10
−10
−5
0
5
10


Standard Least Squares
10
5
0
−5
−10
−5
0
5
10
−10
−5
0
5
10

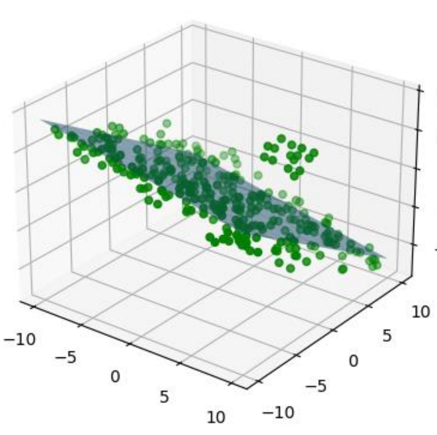

Standard Least Squares
10
5
0
−5
−10
−5
0
5
10
−10
−5
0
5
10


RANSAC
10
5
0
−5
−10
−5
0
5
10
−10
−5
0
5
10

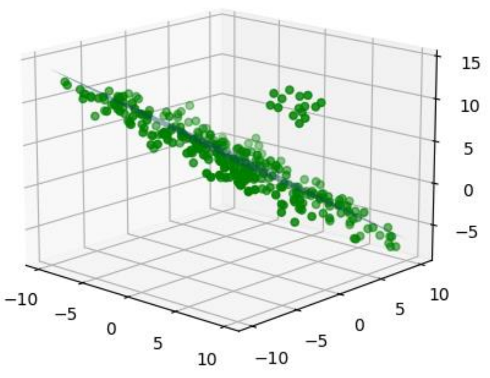

RANSAC
10
5
0
−5
−10
−5
0
5
10
−10
−5
0
5
10

PC2:

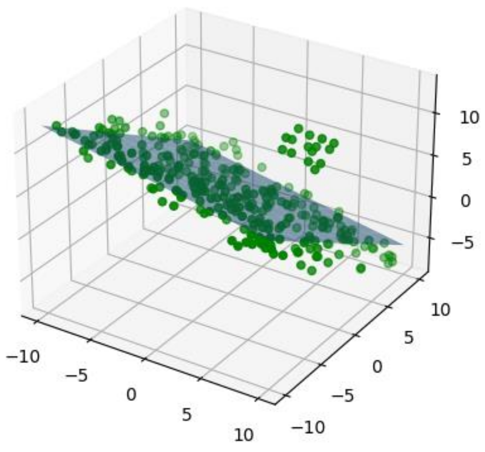
Total Least Squares



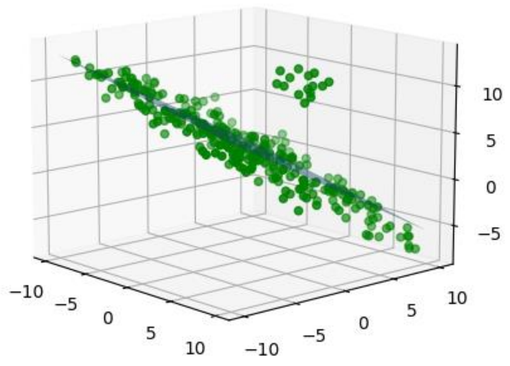
Total Least Squares



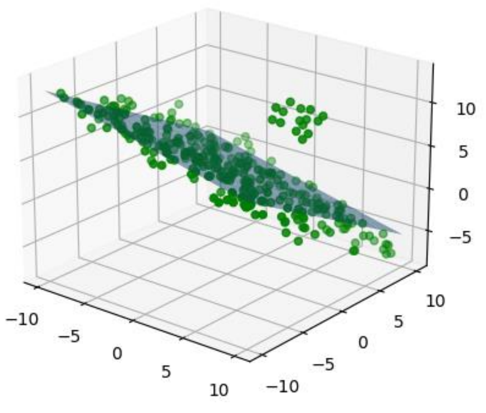
Standard Least Squares



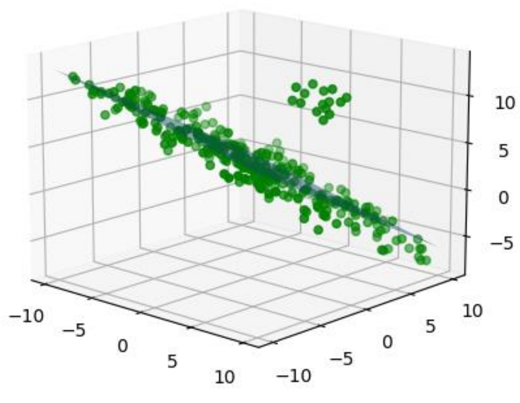
Standard Least Squares



RANSAC



RANSAC



PROBLEM 2.2.b:
Using pc1.csv and pc2.csv, fit a surface to the data using RANSAC method. Plot the results and explain your interpretation.

SOLUTION:

RANSAC algorithm takes in the following parameters:

k - The maximum number of iterations allowed in the algorithm

t - A threshold value for determining when a data point is within that threshold distance

d - The number of close data points(inliers) required to assert that a model fits well to data

Using the Pandas library, the csv file was read in and the X, Y, Z points were then stored in a NumPy array. The "np.vstack()" was then used to make a new 2D NumPy array, combining all the X, Y, Z points respectively. For each iteration, a sample plane is chosen where three random points are selected from the dataset to define that plane. For this sample plane, the above steps were again repeated to find the coefficients of its equation using the least square method. The following steps involve checking if that for any given point in the dataset, if it's not in the sample array and if the distance of that point from the sample plane is less than the user-defined threshold value, that point is appended in the list of inliers.

For optimizing the model, the number of inliers is again checked with the parameter d. If the number of inliers is greater than d, a better model is generated by adding the inlier datapoints to the sample datapoints. An error function then checks if the mean distance of all the datapoints from the estimated plane is less than previous generated mean distance. If the current error is less, that means a better model has been generated and that model is then returned to calculate the coefficients of equation of the estimated plane using Standard Least Squares.

For plotting, the following equation of the plane is then used to get all the Z-coordinate values:

$$z = ax + by + c$$

INTERPRETATIONS FROM OUTPUT GRAPHS:

The choice of best algorithm among TLS, SLS and RANSAC for surface fitting depends on the data at hand. For Standard Least Square, it is a commonly used method for this kind of problem. However, it does not consider the possibility of outliers. As a result, if outliers are present, it does not perform well.

For Total Least Square, it is similar to SLS, but it is more computationally expensive. However, it performs better than SLS as it takes into account both vertical and horizontal error distances.

Lastly, for Random Sample Consensus or RANSAC, the method is quite robust and less sensitive to outliers. Since its algorithm loops over multiple iterations. It optimizes the model constantly, taking into account the number of inliers present each time and finds the best model with the highest percentage of inliers.

In general, if the data contains outliers, RANSAC may be the most appropriate algorithm for surface fitting. However, if the data is relatively noise-free, least squares methods may be sufficient. Total least squares can be used when both the independent and dependent variables have measurement errors.

PROBLEMS FACED:

Getting the logic correct proved to be a hassle at first. With further research and study, after getting the logic correct, plotting the data became easier. Also faced similar issue as Problem 2.1.a. However, once the previous problem was rectified, solving this problem became easier.