# ENPM 673
# PERCEPTION FOR AUTONOMOUS ROBOTS

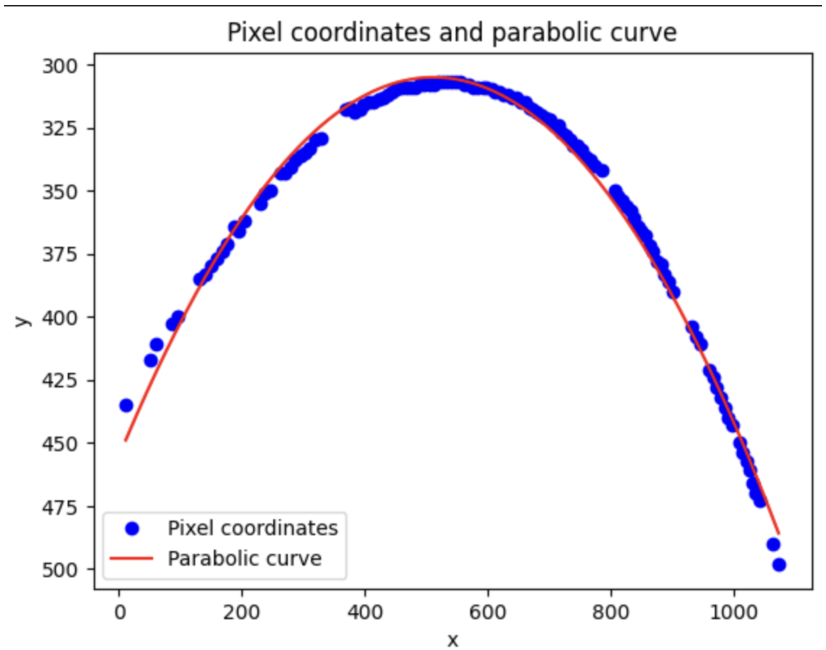
KIRAN AJITH
119112397

**Problem 1**

a. To detect and plot the pixel coordinates of the center of the ball, the following steps were taken:

- Load the video using the VideoCapture function of openCV
- Loop though each frame and convert the frame to HSV color space
- Create a mask for the red color range
- Perform erosion and dilation to remove noise from the mask
- Find the contours of the objects in the mask
- loop through the contours to find the red ball
- calculate the area of the contour, if the area is too small,ignore the contour
- find the center of the contour and draw a circle around the centre of the  contour
- Display the frame with the red ball contour and center

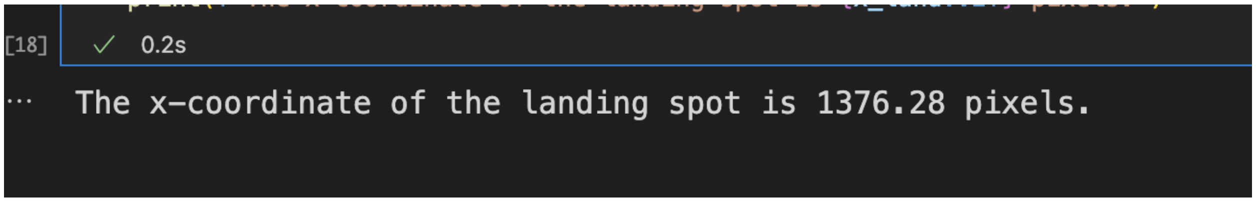
b. Two empty lists are appended with the coordinates of the center of the ball. They are then used to plot the trajectory of the ball.



c. Given that the origin of the frame is at the top left corner and y-coordinate of the landing spot is defined as 300 pixels greater than its first detected location.

Ie, y_land= y_0 + 300

X- coordinate of the landing spot is obtained by substituting the y_land in the trajectory equation



```
[18]  ✓ 0.2s

...  The x-coordinate of the landing spot is 1376.28 pixels.
```

Problems faced:
Creating a mask of the ball was tricky as in some of the frames, the hand in the frame was also getting added to it. The upper boundary of HSV color spaced had to be changed based on trial and error to get the proper mask of the ball.

**Note:**
When running the program as a .py file in vscode,it sometimes throws an error(-25 assertion failed). Hitting the ‘q’ key right before the video ends seems to work. As the 3 parts are compiled together in the same program, they get executed sequentially. (Example, after closing the Graph window of part 2, the coordinates of the landing spot is display, which is part 3)

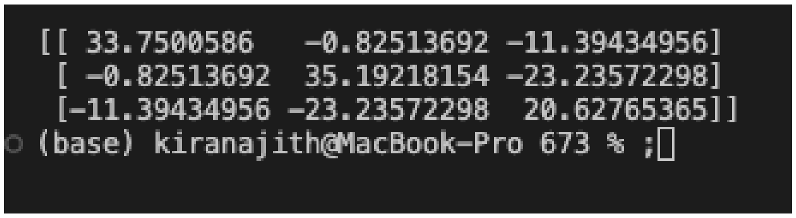
**Problem 2.1**

a. We start by reading the data from the ‘pc1.csv’ file. The covariance matrix can be obtained by two methods:

   (i). Using the buit-in np.cov() function of openCV
   (ii). Using performing operations on the data

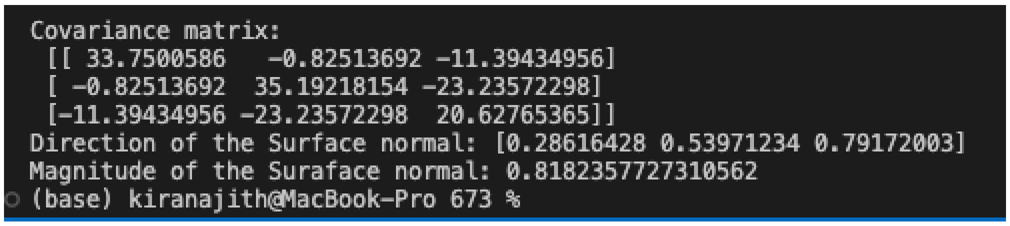
   In this project, the second method is adopted. Following are the operations done on the data to obtain the covariance matrix:
   - Calculate the mean of each column of data
   - Subtract the mean value from each data point in the column to obtain the centered data
   - Find the product of centered data with its transpose
   - The covariance matrix is obtained by dividing the product with the number of data points

Output:



```
 [[ 33.7500586   -0.82513692 -11.39434956]
  [ -0.82513692  35.19218154 -23.23572298]
  [-11.39434956 -23.23572298  20.62765365]]
○ (base) kiranajith@MacBook-Pro 673 % ;
```

b. Surface normal is the vector perpendicular to the ground and has a magnitude of one. It is assumed that the ground plane is flat. The covariance matrix can be used to compute the maghntiude and direction of the surface normal by following the steps:
   - FInd the eigen value and eigen vector of the covariance matrix
   - The eigen vector with the smallest eigen value is the surface normal
   - The magnitude of the surface normal is equal to the square root of the smallest eigenvalue.



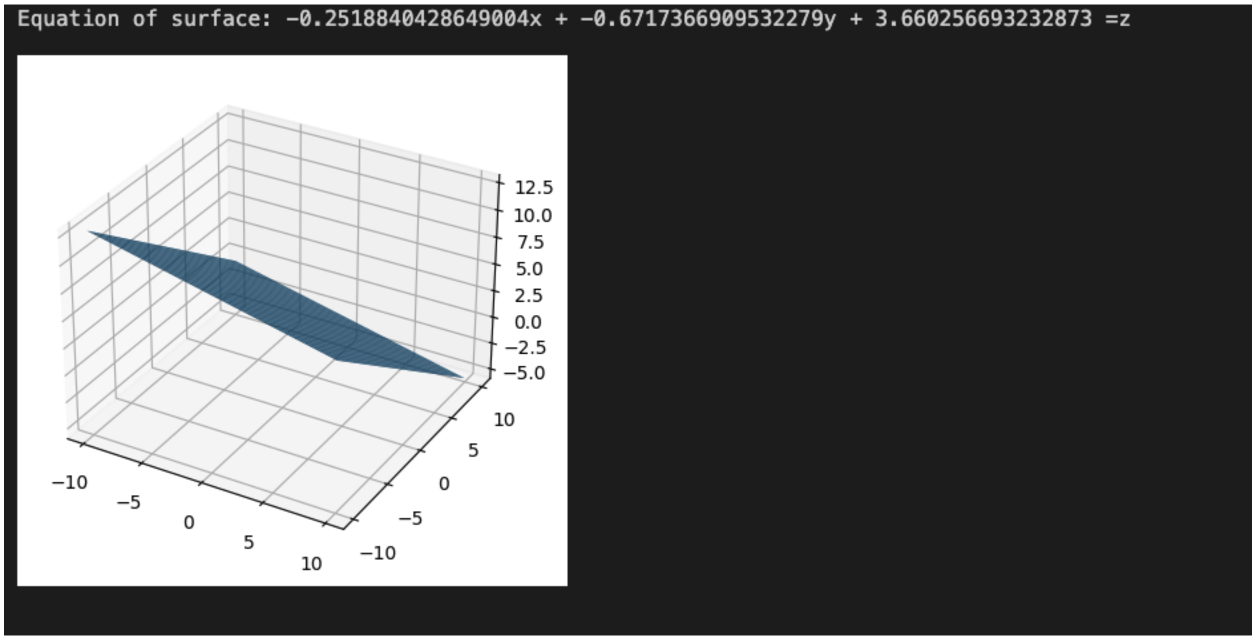
```
 Covariance matrix:
  [[ 33.7500586   -0.82513692 -11.39434956]
  [ -0.82513692  35.19218154 -23.23572298]
  [-11.39434956 -23.23572298  20.62765365]]
 Direction of the Surface normal: [0.28616428 0.53971234 0.79172003]
 Magnitude of the Suraface normal: 0.8182357727310562
○ (base) kiranajith@MacBook-Pro 673 %
```

Problems faced:

The data may contain outliers, which drastically alter the estimated covariance matrix, which is one potential issue that arises while generating the covariance matrix. Before constructing the covariance matrix, this issue is resolved  by using outlier identification techniques like clustering or statistical tests to recognise and exclude the outlier points from the data.

**Problem 2.2**

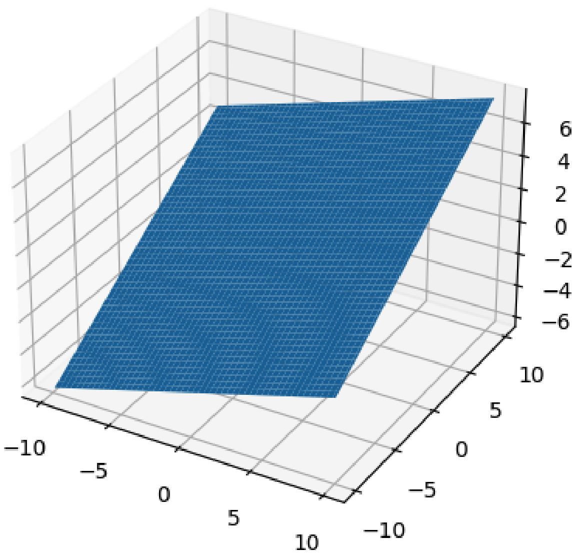
a. To fit the surface using the standard least square method, the steps are as follows:
    - Read the data from pc1.csv and pc2.csv
    - To fit the surface to the data, we need to solve the equation AX=B where A is a matrix of the form
      [ x1 y1 1],
      [x2  y2 1],
      ……
    - We solve for x using the normal equations
    - Coefficients of the surface equation is then obtained
    - Using the coefficients of the surface equation Z=Ax+By+C we form a meshgrid and plot the surface using matplotlib.



```
Equation of surface: -0.2518840428649004x + -0.6717366909532279y + 3.660256693232873 =z
```

To fit the surface using the Total least square method,the steps are as follows:
- Read the data as a numpy array
- Calculate the mean of each column of the array
- Subtract the mean with the data points
- Create a matrix with x,y,z, column of data and compute the SVD of the matrix

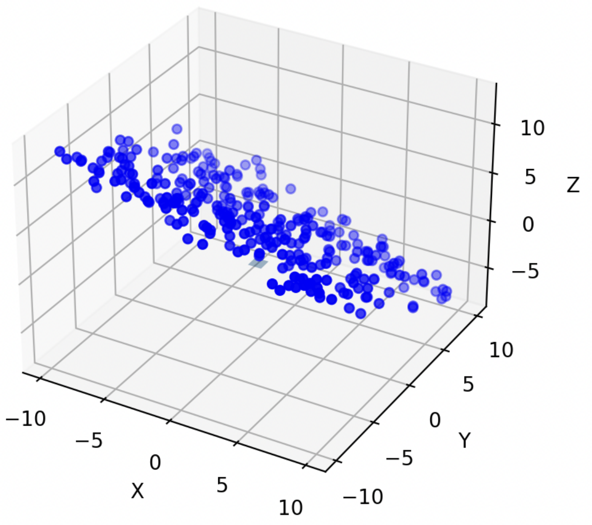
- Determine the lowest singular value of A and the singular vector that corresponds to it. If A has more rows than columns, this can be accomplished by choosing the last row of the V matrix provided by SVD, or if A has more columns than rows, by choosing the last column of the U matrix.
- The singular matrix is then made a 3*1 matrix
- Divide the remaining elements of the reshaped singular vector by the negative of the previous element to obtain the model parameters.
- The add the mean of the data to the model parameters to obtain the final model
- Compute the coordinates of the surface over a grid of points
- Plot the surface



b. Following are the steps to implement RANSAC from scratch and fit a surface to the data
- Load the data from ‘pc1.csv’ and ‘pc2.csv’ into a numpy array
- Define a function ransac() that takes five arguments for
  -data,
  n - the minimum number of data points required to fit the model,
  k - the maximum number of data points required to fit the model
  t -’a threshold value for determining when a data point fits a model and the number of close data points required to assert that a model fits well.
  d - how many close data points are requited to check if the model fits well.

- ransac() returns the model that best fit the data
- The model calculates the x,y,z coordinates of the surface using numpy’s meshgrid() function.
- A scatter plot of the data points are displayed using pyplot.scatter()

- The surface is then plotted using pyplot.plot_surface()



Problem Faced:
Initially obtained a suboptimal model due to high outliers in the data. This was reduced byincreasing the number of iterations, increasing the minimum number of points required to fit the model, and adjusting the distance threshold.