Homework1
ENPM673

Pratik Sunil Bhujbal
UID: 117555295
Email: pbhujbal@umd.edu

Problem 1:

Part1

FOV is given by, $\phi = 2\,x\,tan^{-1}(\frac{d}{2f})$

Where f is focal length

$$\phi = 2\,x\,tan^{-1}(\frac{14}{2\,x\,25}) = 31.28\,degrees$$

Part2

** All dimensions are in mm

Ratio of the height of object and its distance from the camera to the height of image and focal length is,

$$\frac{height\ of\ object}{diatance\ from\ camera} = \frac{height\ of\ image}{focal\ length\ of\ camera},$$

$$\frac{50}{20000} = \frac{height\ of\ image}{25},$$

Height of image = 0.0625 mm

Area of camera sensor = 14 x 14 = 196 mm^2 covers 5 MP,

So, 1 mm^2 covers $\frac{5\,x\,10^6}{196} = 25510.2\,pixels$.

Area of Image = $(0.0625\ mm)^2$

Therefore, number of pixels covered by the object in the image

= 25510.2 x $(0.0625)^2$ = 99.6 pixels ≅ **100 pixels.**

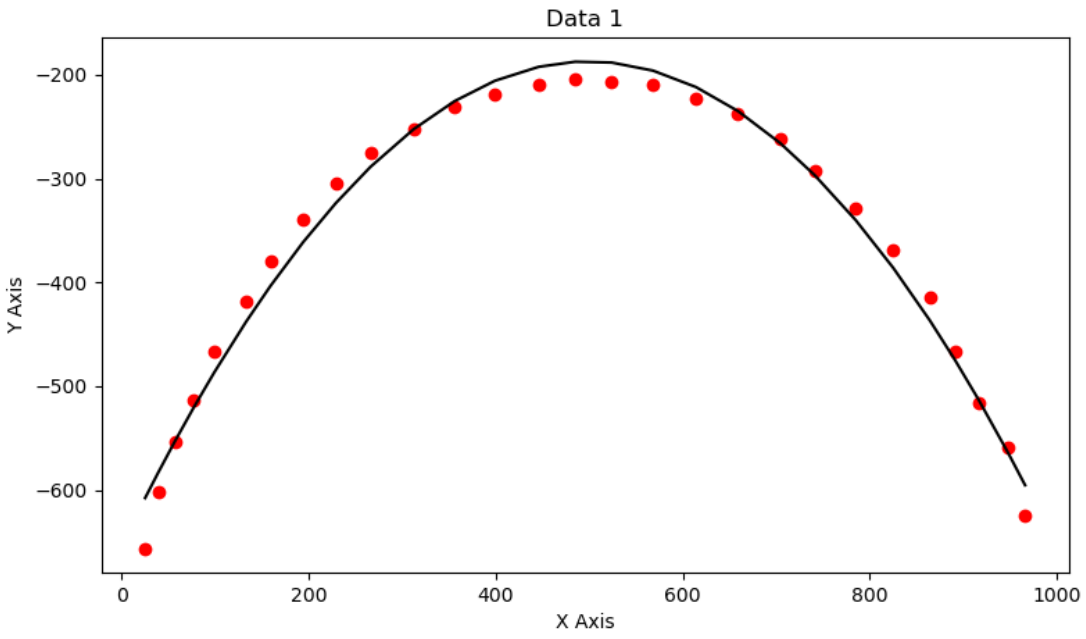
Problem 2:

In this problem, the Standard Least Squares method was used to fit curves for the given videos. Steps taken were:

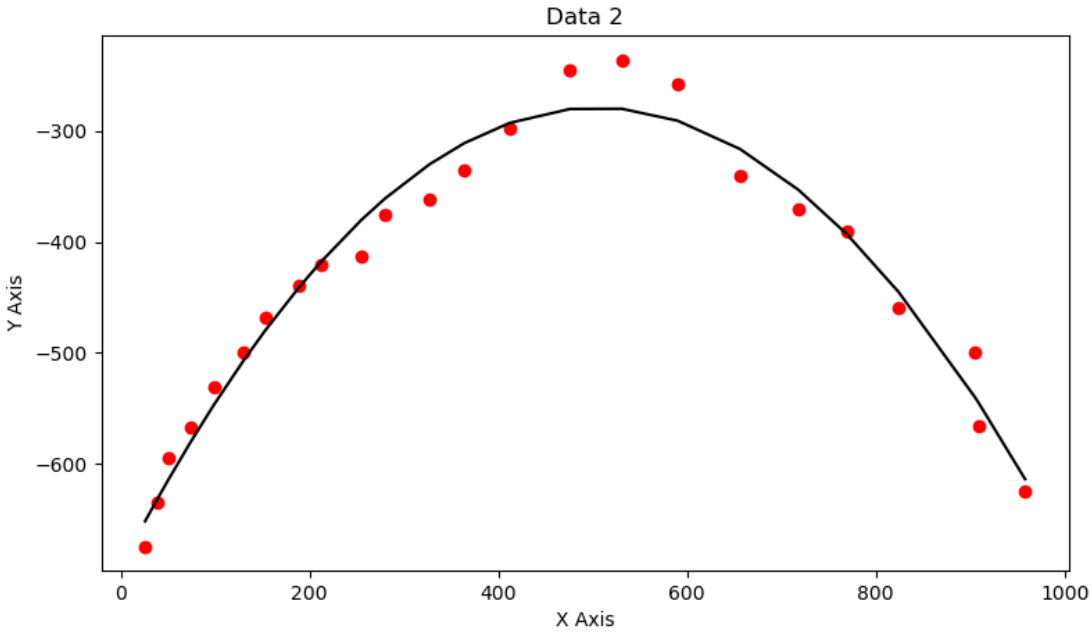
1. Read the video frame by frame.
2. Mask frame to detect red ball.
3. Find contours of the ball.
4. Using the contour area to find the center and save coordinates for each frame.
5. Save coordinates to .csv file.
6. Use that .csv file to read X and Y coordinates of the center of the ball and fit a curve.
7. Plot the graphs

Results:

For Video 1 without noise:



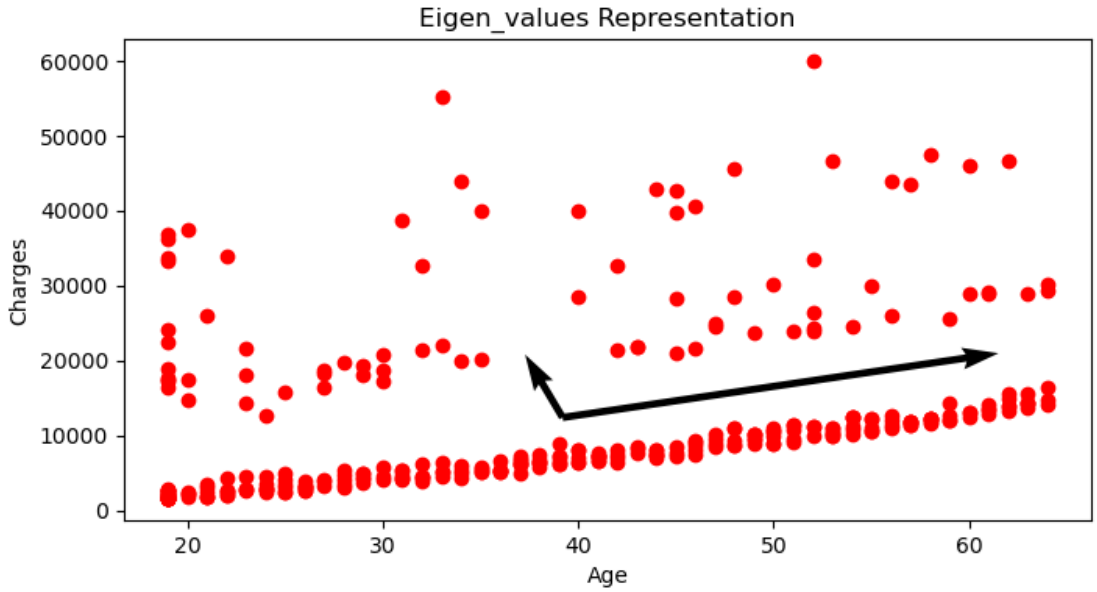
For Video 2 with noise:

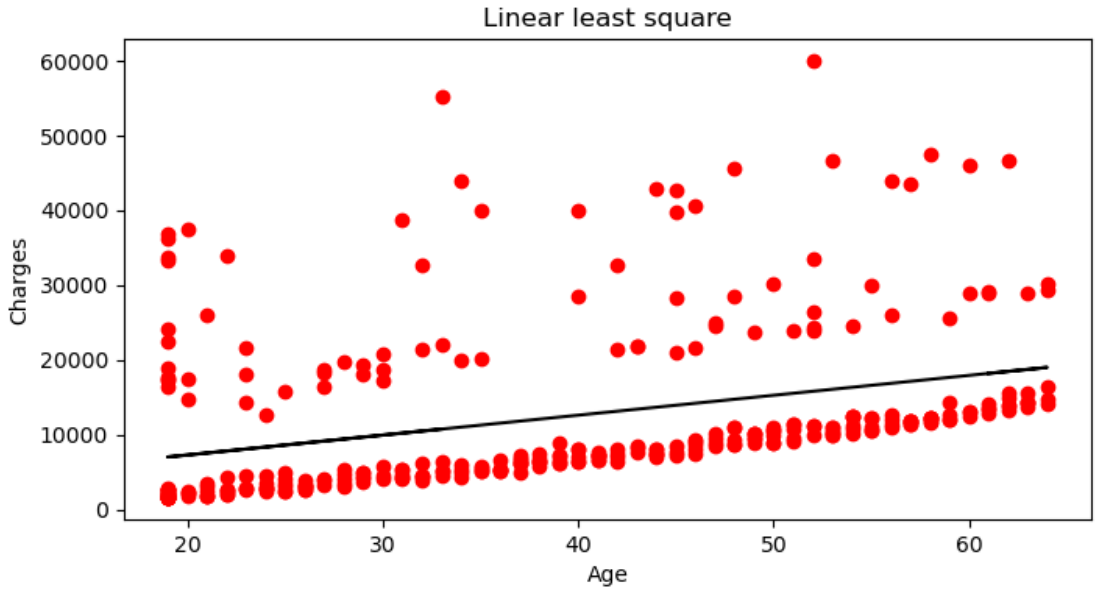


Problem 3:

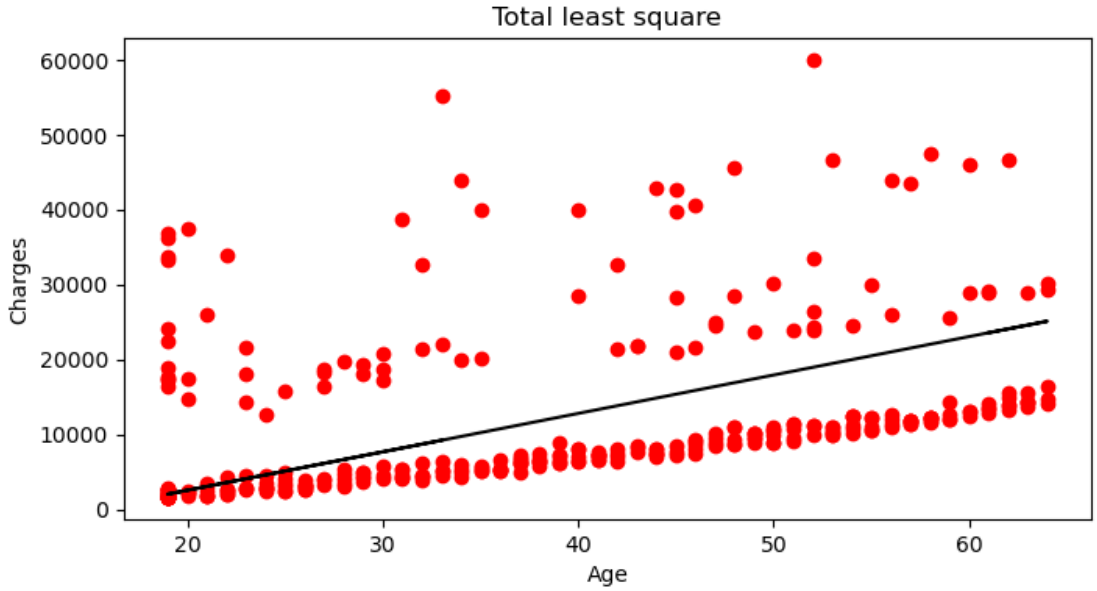
Part2

Eigen Representation

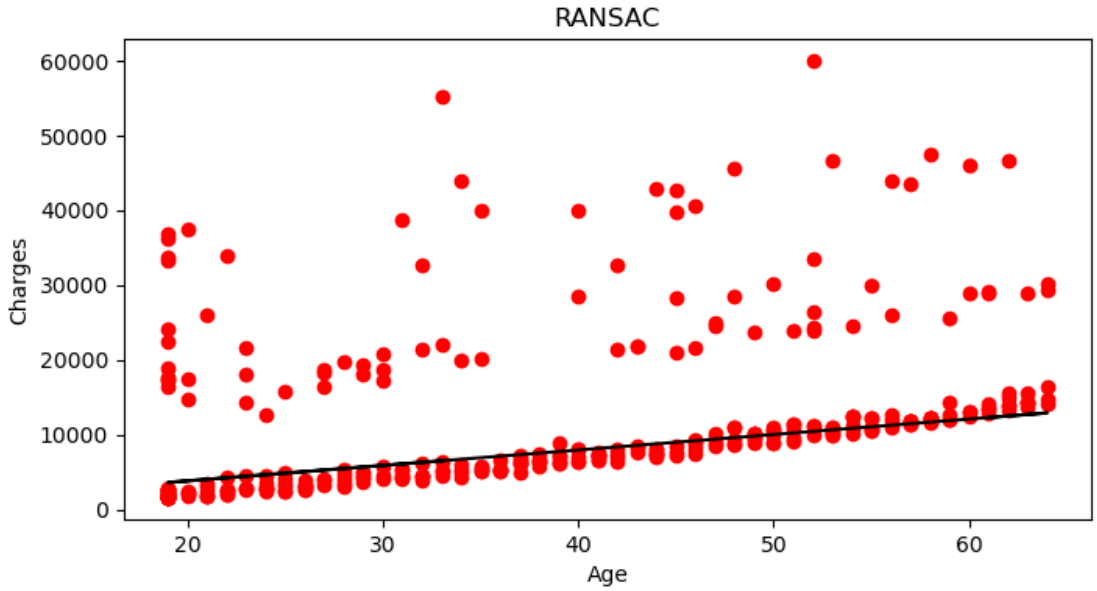




For the linear least-squares method, outliers affect the curve(line) by dragging the curve away from the ideal set of data and towards the outliers, therefore this method may not be the best fit for a line for a given set of data.



For the Total Least Square method, the line is much closer to the meaningful data points compared to the Linear least square method. In this method also, the outliers are dragging the line away from the ideal data points, this method also may not be appropriate for fitting the line for a given dataset but this fit is much better than the Linear Least Square method.



The RANSAC method lies near the viable data points, we can see that outlier data points do not influence the fitting of the line, this method of line fitting is best suited for this given set of data.

Part3

1) **Linear Least Square Method:**

Equation of line can be given by:

$$Y = mX + C$$

where ,

m= slope of the line.

C = constant

By solving the equation for m and C we can fit a line in given points.

2) **Total Least Square Method:**

Using the homogeneous equation $U^TUN=0$ where U is the mean matrix of the above equation.

Where the solution of the homogeneous equation can be solved by SVD (Single Value Decomposition).

3) RANSAC- Random Sample Consensus:

1) Select any two random points from the data.
2) Fit the curve using the Total Least Square Method.
3) Calculate the error using function- $(Y-m*X-c)^2$.
4) Continue the process till your error is less than your specified error.

5) Iteration can be calculated using:

$$N = \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

Where,

e = probability that a point is an outlier.
s = number of points in a sample.
N = number of samples.
p= desired accuracy

Problem 4:

1) According to Singular Value Decomposition, a **m**x**n** matrix A can be factorized as,
A = UΣV^T ,
where **U** is m x m, **Σ** is m x n, and **V** is n x n.

The columns of **U** are the orthogonal eigenvectors (left singular vectors) of **AA^T**,
The columns of **V** are the orthogonal eigenvectors (right singular vectors) of **A^{T}A**, and
Σ is a diagonal matrix with elements $\boldsymbol{\sigma_i}$ ($\boldsymbol{\sigma_1} > \boldsymbol{\sigma_2} > \boldsymbol{\sigma_3} \ldots$), which are the square root of the eigenvalues of **AA^T** or **A^{T}A.**

To mathematically compute SVD,

a) Find the eigenvalues and eigenvectors of **AA^T** using **numpy.linalg.eig()** and then sort the eigenvalues in descending order and rearrange the columns of the eigenvector matrix as per the sorted eigenvalues to get the **U** matrix.
b) Find the eigenvalues and eigenvectors of **A^{T}A** using **numpy.linalg.eig()** and then sort the eigenvalues in descending order and rearrange the columns of the eigenvector matrix as per the sorted eigenvalues to get the **V** matrix.
c) To get the **Σ**, use eigenvalues of **AA^T** or **A^{T}A**, sort them in descending order, compute their square roots and form a diagonal matrix of order m x n.

$$\boldsymbol{\sigma}_i = \sqrt{\lambda_i}$$

$$\sum = diag(\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3 \ldots\ldots \boldsymbol{\sigma}_i)$$

d) To compute the homography matrix **H** we take the last row of the V matrix or the last column of the V matrix and then reshape it to a 3 x 3 matrix and finally normalize it to make the last element 1.

V Matrix:

```
VT =  [[ 2.84043894e-03  2.42121739e-03  2.20891154e-05  1.09109680e-03
   1.63479471e-03  1.33908907e-05 -6.96053715e-01 -7.17950893e-01
  -6.16016024e-03]
 [ 3.14430147e-03 -1.28321626e-03  1.13495064e-05  1.17416448e-03
  -2.90636016e-03 -1.14077892e-05 -7.17961695e-01  6.96067270e-01
   2.29933343e-05]
 [-2.46384735e-01 -3.77000733e-01 -2.37217168e-03  6.61240940e-01
   5.74279813e-01  5.80190908e-03 -7.57487350e-05  1.62813209e-03
  -1.73453679e-01]
 [-1.58554932e-01  1.76600215e-01 -3.65660431e-03  3.41172744e-01
  -7.10405325e-02 -2.15181510e-03 -3.79564118e-03 -3.77529395e-03
   9.06741660e-01]
 [-1.75245114e-01  6.89508147e-01  5.19584361e-03  5.01749740e-01
  -3.14549320e-01  2.88147957e-03  2.50334194e-03  2.49754245e-03
  -3.78319687e-01]
 [ 1.76705635e-01  5.90273326e-01  7.52000216e-03 -2.32499365e-01
   7.49883366e-01 -5.73504703e-03 -1.50223526e-04  3.65599795e-03
   6.21986452e-02]
 [ 9.13738625e-01 -5.29344506e-02  6.59901592e-02  3.72052359e-01
  -6.19835401e-02 -1.22489909e-01  4.37766998e-03 -6.00114914e-04
   2.52338778e-02]
 [-1.20261073e-01 -2.23230964e-03  7.85970680e-01 -4.25903575e-02
   4.58785873e-03 -6.04930528e-01 -5.55196291e-04  3.48250731e-05
  -2.47794676e-03]
 [ 5.31056350e-02 -4.91718844e-03  6.14648552e-01  1.77018784e-02
  -3.93375075e-03  7.86750146e-01  2.36025045e-04 -4.91718843e-05
   7.62164205e-03]]
```

U Matrix:

```
U =  [[ 1.17519867e-02  3.44207228e-04 -5.15532162e-02 -4.66128587e-01
  -2.60345896e-01 -6.78428560e-02  1.08122929e-02 -8.41087769e-01]
 [ 1.17517760e-02  3.43641967e-04 -8.72103737e-02 -4.59351955e-01
  -2.49098952e-01 -8.85591890e-02  7.65455993e-01  3.54169470e-01]
 [ 3.58735699e-01  6.54942912e-01  1.34538659e-02 -4.65084492e-01
   1.70101644e-01  2.93617516e-01 -2.78385484e-01  1.82289869e-01]
 [ 1.43494223e-01  2.61976394e-01 -4.45383120e-01  1.36060221e-01
  -5.00795526e-01 -5.87488150e-01 -2.73099287e-01  1.52897236e-01]
 [ 7.74962678e-01  2.27117371e-02  4.08516159e-01  2.84937362e-01
   3.19642679e-02 -2.35211438e-01  2.62688692e-01 -1.59658351e-01]
 [ 2.81806634e-01  8.24745878e-03 -6.92167142e-01  3.15915567e-01
   1.14149714e-02  5.01908806e-01  2.46628160e-01 -1.69560615e-01]
 [ 1.84643411e-01 -3.16806256e-01  2.48466337e-01 -3.46544961e-02
  -6.98268275e-01  4.67261587e-01 -2.52393736e-01  1.81630339e-01]
 [ 3.69278450e-01 -6.33614920e-01 -2.88917222e-01 -3.93333286e-01
   3.18917542e-01 -1.75016528e-01 -2.61429000e-01  1.52633610e-01]]
```

Homography Matrix:

```
Homography Matrix =  [[ 6.96774195e+00 -6.45161293e-01  8.06451613e+01]
 [ 2.32258065e+00 -5.16129035e-01  1.03225806e+02]
 [ 3.09677420e-02 -6.45161292e-03  1.00000000e+00]]
```