Arunava Basu
117720617

ENPM673
Homework 1

Problem 1

1. The field of view θ, can be calculated as,

$$\theta = 2 \times \tan^{-1}(f/2d)$$

where, f and d is the focal length and sensor width respectively. Since the image sensor is mentioned to be square, the sensor width is equal to the sensor height.

Substituting the values of f = 15mm and d = 14mm,

$$\theta = 2 \times 15.692 = 31.284°$$

2. The pixel density of the camera, $P_d = \frac{Resolution}{Area\ of\ sensor} = \frac{5 \times 10^6}{14 \times 14}\ p/mm^2$

The relationship between the image height and the object height is given as,

$$h_i = f \times \frac{h_o}{D}$$

where h_i, f, h_o, and D are the image height, focal length, and the object height respectively.

Substituting the values from the question we obtain $h_i = \frac{25 \times 5 \times 10}{20 \times 1000} = \frac{1}{16}\ mm$
Area of the image, $A_i = h_i \times h_i$
Hence, the pixels occupied by the image $P = P_d \times A_i = 99.649 \sim 100\ pixels$

Arunava Basu
117720617

Problem 2

This problem was split into two halves:

- Generation of points from the two videos
- Fitting a parabola on two sets of points using the Standard Least Squares method

Generations of points from the videos

The following pipeline was followed to generate a set of points for each video:

- Each video was read frame by frame and each frame was converted into a grayscale image.
- Aggressive inverse binary thresholding was performed between a grayscale value of 200 and 255.
- Indices from the image were extracted where the pixel values exceed 200. This provides us with a blob that contains the ball.
- Computing the maximum and minimum of both the x and y indices gives use the location of the bounding box around the ball.
- Considering the maximum and minimum points of the bounding box as x_{max}, y_{max} and x_{min}, y_{min} respectively, we get the top and bottom most pixels P_{top}, P_{bottom}, of the ball in every frame as:

$$P_{top} = (\frac{x_{max} + x_{min}}{2}, y_{min})$$

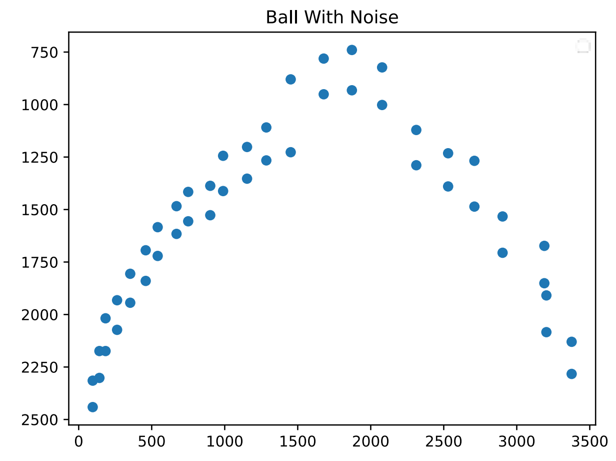
$$P_{bottom} = (\frac{x_{max} + x_{min}}{2}, y_{max})$$

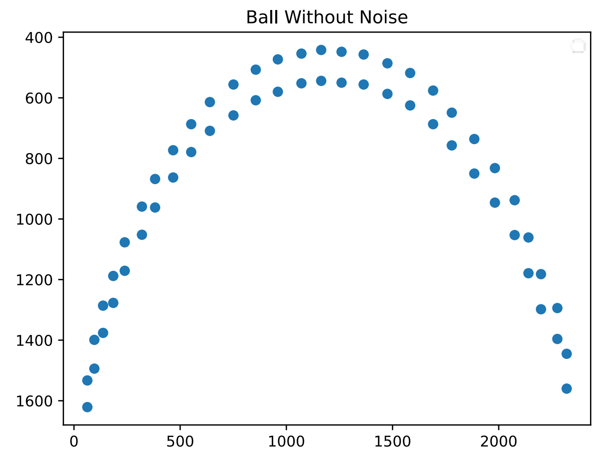
- Hence, the top and bottom most pixels of the ball from each frame is then written into a JSON file as x and y co-ordinate lists for easy access.
- This process was repeated for both the videos.

The data is stored in the ***/data*** folder as ***ball_without_noise.json*** and ***ball_with_noise.json***.

The python implementation of the above pipeline is present in the file ***gen_data_from_video.py***.

The points obtained are as follows:





Arunava Basu
117720617

Fitting a parabola using the Standard Least Squares method

The equation of a parabola is modelled as,

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

The error in the standard least square method is considered only in the y direction. Hence, the error function can be written as,

$$E = \sum_1^n (y_i - ax_i^2 - bx_i - c)^2$$

where n is the total number of recorded points.
Considering $X = [\, x_i^2,\ x_i, 1\,]$, $Y = [y_i]$, and $A = [\, a,\ b, c\,]^T$. Then,

$$E = ||Y - XA||^2 = (Y - XA)^T(Y - XA)$$

We need to find a value of A such that the error function E is a minimum. We differentiate E with respect to A and set it to zero.

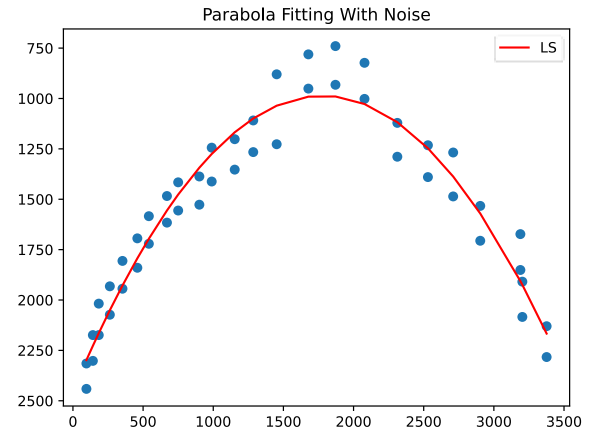
$$\frac{\partial E}{\partial A} = 0$$

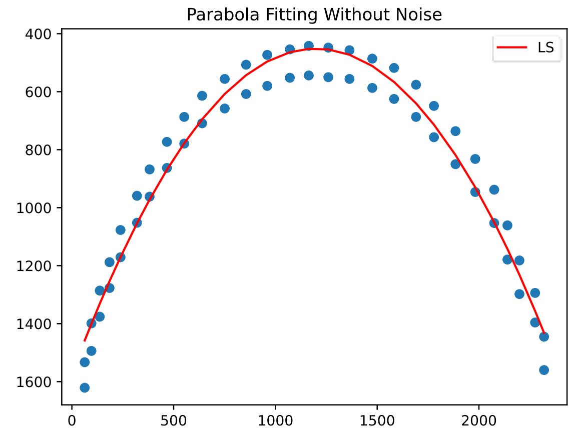
We can solve this for A as follows [3],

$$X^T XA = X^T Y$$
$$A = (X^T X)^{-1} X^T Y$$

The term $(X^T X)^{-1} X^T$ is known as the pseudo-inverse of X and can be now utilized to solve for this over-determined system of equations.

The curves fit by using the above discussed method are:





The python implementation of the above pipeline is present in the file ***P2.py***.

Arunava Basu
117720617

Problem 3

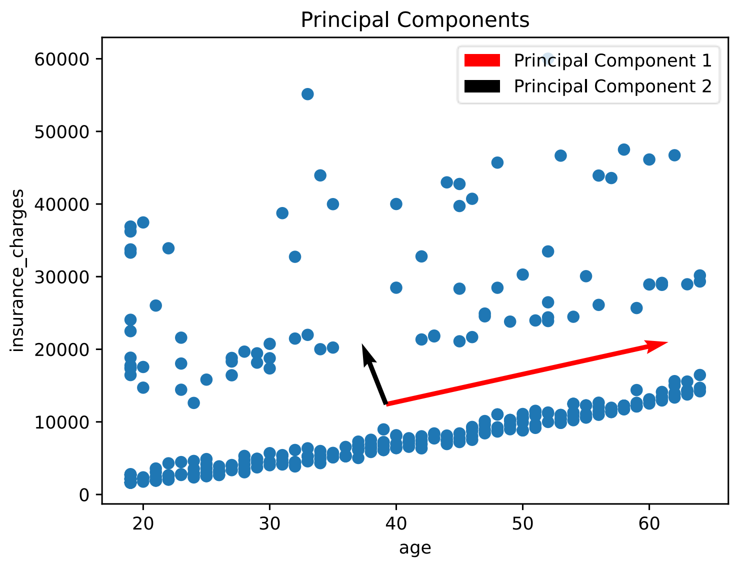
1. The data provided here has very different scales for age and insurance cost. Hence the data has been normalized using minimum maximum normalization. The covariance matrix of the original and the normalized data is calculated using,

$$Cov_M = \begin{bmatrix} var(x) & cov(x,y) \\ cov(x,y) & var(y) \end{bmatrix}$$

where, Cov_M is the covariance matrix for two variables x and y and $var(x)$, $var(y)$, and $cov(x,y)$ are the variance of x, variance of y and the covariance of x and y respectively.

For the next calculations, the normalized data was used.

The eigen vectors and the eigen values were computed using numpy. To visualize this on the original data, the vectors were scaled back and multiplied with their corresponding eigen values. The resulting plot is shown below.



A few key observations can be made here:

- The eigen vector corresponding to the largest eigen value denotes the direction of largest variation. This is called the first principal component of our data.
- The eigen vector corresponding to the second largest eigen value therefore denotes the direction of the second largest variation. This is called the second principal component of our data.
- Since the covariance matrix is a symmetric matrix, the two vectors are orthonormal to each other. This has been proved in code by seeing that the dot product of the two vectors equals zero.

The python implementation of the above pipeline is present in the *Q1* section of the file ***P3.py***.

Arunava Basu
117720617

2. **Linear Least Square Line Fitting**: The equation of a straight line is modelled as,

$$y = bx + a$$

The error in the standard least square method is considered only in the y direction. Hence, the error function can be written as,

$$E = \sum_1^n (y_i - bx_i - a)^2$$

where n is the total number of recorded points.

We need to find the value of *b* and *a* such that the error function E is a minimum. We differentiate E with respect to *b* and *a* and set it to zero.

$$\frac{\partial E}{\partial a} = 0$$

$$\frac{\partial E}{\partial b} = 0$$

From the above two operations, we get two equations,

$$\sum_1^n y_i = na + b\sum_1^n x_i$$

$$\sum_1^n x_i y_i = a\sum_1^n x_i + b\sum_1^n x_i^2$$

Solving these two equations gives us the values of *a* and *b* as,

$$a = \bar{y} - b\bar{x}$$

$$b = \frac{SS_{xy}}{SS_{xx}}$$

where $SS_{xy} = \sum_1^n x_i^2 - \frac{(\sum_1^n x)^2}{n}$, $SS_{xy} = \sum_1^n x_i y_i - \frac{\sum_1^n x_i \sum_1^n y_i}{n}$ and $\bar{x}$ and $\bar{y}$ are the mean of x and y respectively.

The advantage of linear least square is that it is scale invariant. However, it is not rotation invariant and fails completely for vertical lines.

Total Least Square Line fitting: The equation of a straight line is modelled as,

$$ax + by = d$$

The error in the total least square method is considered by taking the normal distance to the parametric model line. Hence, the error function can be written as,

$$E = \sum_1^n (ax_i + by_i - d)^2$$

where n is the total number of recorded points.

We need to find the value of *d* such that the error function E is a minimum. We differentiate E with respect to *d* and set it to zero.

$$\frac{\partial E}{\partial d} = 0$$

Arunava Basu
117720617

We get the value as $d = a\bar{x} + b\bar{y}$ where $\bar{x}$ and $\bar{y}$ are the mean of x and y respectively. Plugging this value of d back into the original error equation E, we get

$$E = \sum_1^n (a(x_i - \bar{x}) + b(y_i - \bar{y})^2$$

Considering $U = [\, x_i - \bar{x},\, y_i - \bar{y}\,]$, and $N = [\, a,\, b\,]^T$. Then,

$$E = (UN)^T(UN)$$

Therefore, we need to find a value of N which minimizes the above error equation. Taking partial derivate with respect to N and setting it to zero.

$$\frac{\partial E}{\partial N} = 0$$

$$(U^T U)N = 0$$

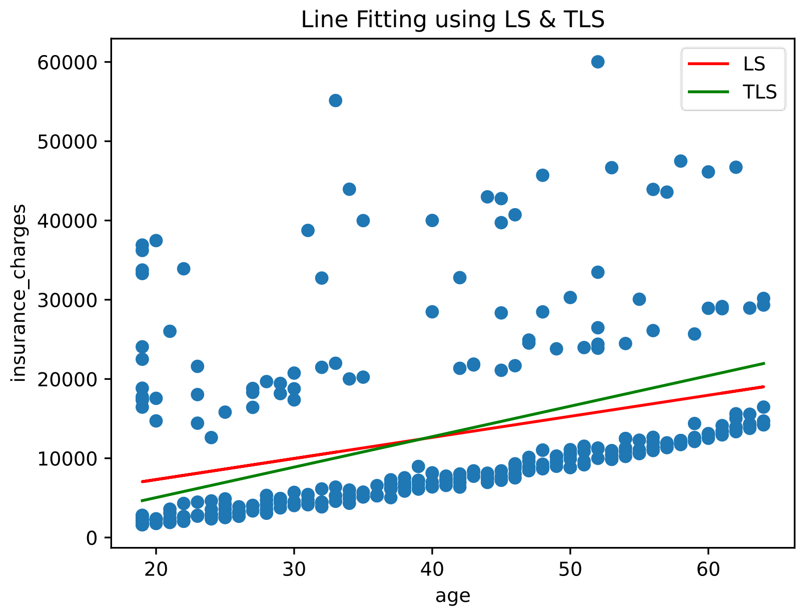
Let $U^T U\ be\ A$, then,

$$AN = 0$$

which is a homogenous overdetermined system of equations and can be solved by using the Singular Value Decomposition of A. The least square solution of this system of equations is given by the last column of the V vector obtained after performing the SVD of A.

The advantage of total least square is that it considers errors in both x and y. However, it is not scale invariant and is penalized for outliers.

The plot for Linear Least Square and Total Least Square is shown below in the same graph for better comparative visualization.



Arunava Basu
117720617

RANSAC line fitting: The RANSAC algorithm stands for **Ran**dom **Sa**mple **C**onsensus.

To fit a line using RANSAC we follow the follow steps:

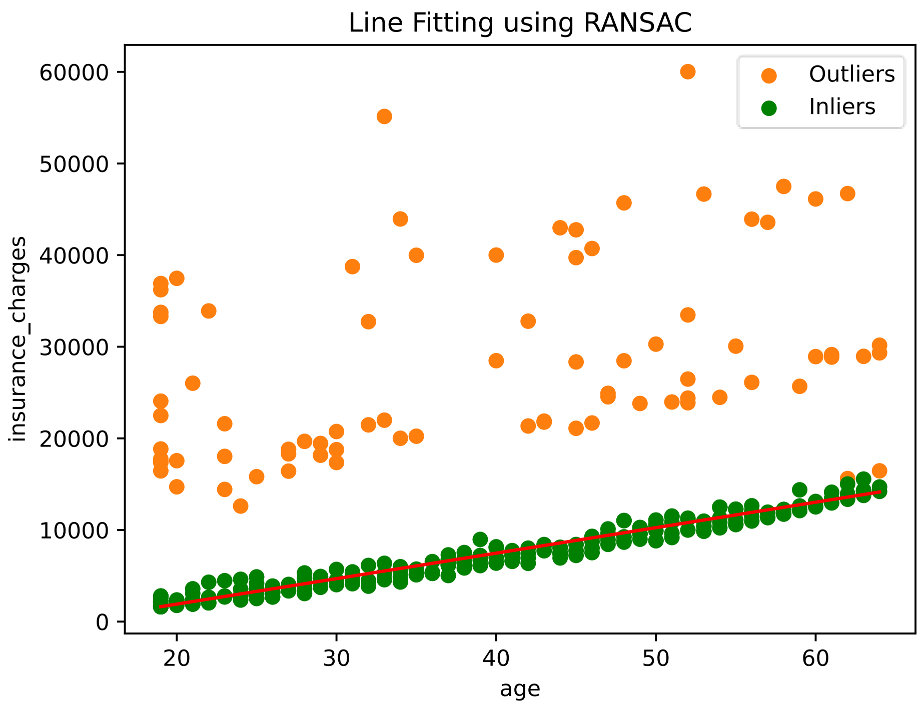
- Select the minimum number of random points required to fit a line, i.e. 2.
- Attempt to fit a curve using Linear Least Square line fitting method using the two points.
- Calculate the error for all the data points with respect to the line obtained from the above line fit as:
$$E = (y_i - bx_i - a)^2$$
- Mark all points below the error threshold as inliers and the rest as outliers
- Continue this process for N iterations and return a and b for the line which gives us the maximum number of inliers or if the desired ratio of inliers to outliers is obtained.

The number of iterations, N, is chosen high enough to ensure the probability p, that at least one of the sets of random samples does not include an outlier. If u represents the probability that any selected data point is an inlier and v = 1 – u, the probability of observing an outlier. N iterations of the minimum number of points denoted m are required, where [1]
$$1 - p = (1 - u^m)^N$$

and thus, by taking log on both sides of the equation, we get,
$$N = \frac{\log(1-p)}{\log(1-(1-v)^m)}$$

Since random points are chosen every time we run this routine, there is a possibility of minor variations in our resultant line. Therefore, an additional step can be added where we compute the least square estimated line for only the inliers after N iterations. This is implemented in code by using an optional "optimal_fit" flag which is disabled by default to facilitate a fair analysis between Linear Least Square, Total Least Square and RANSAC.

The RANSAC line fit plot is shown below.



Arunava Basu
117720617

The advantage of RANSAC is that it robust to outliers. A disadvantage of RANSAC is that there is no upper limit on the time it takes to compute these parameters. When the number of iterations computed is limited, the solution obtained may not be optimal.

The python implementation of the Linear Least Square, Total Least Square and RANSAC is present in the *Q2* section of the file ***P3.py***.

Note:

- Normalized data was used for TLS and RANSAC methods as TLS is scale variant and the use of normalized data facilitated in better threshold selection for RANSAC.
- The resultant parameters for the line model where then used to generate the best fit line which was then re-scaled to fit the original data.
- For the LS method, the original data was used as it is scale invariant

3. The steps to the solution for each of the line fitting method is outlined above and have been implemented in python.

    - Linear Least Square methods of curve/line fitting works well for data with noise. However, it does not work well with the age vs cost data due to the presence of noise and outliers. Even though it is scale invariant, it is rotation variant and fails for vertical line fitting.
    - Total Least Square method considers errors in both x and y measurement which makes it more robust to noise than linear least square. However, it is heavily penalized for outliers. It is also scale invariant.
    - RANSAC line fitting was robust for our data as it was able to reject outliers effectively. However, since the points are chosen randomly, the result can vary slightly every time we perform a line fit. Additionally, there is no guarantee that even after exhausting N iterations, we will be presented with an optimal fit. Iterations beyond N would make this method computationally more expensive.

Arunava Basu
117720617

Problem 4

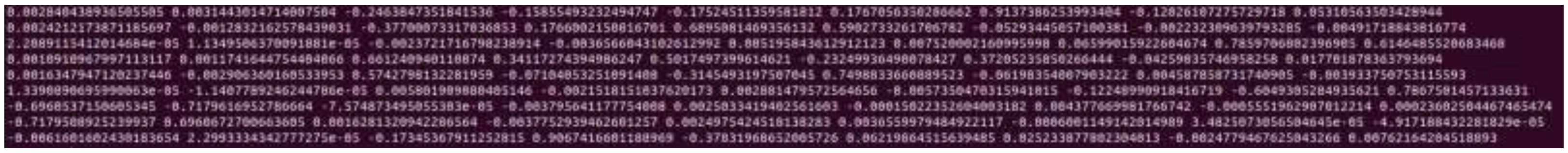
1. From the given points, A is

$$\begin{bmatrix} -5 & -5 & -1 & 0 & 0 & 0 & 500 & 500 & 100 \\ 0 & 0 & 0 & -5 & -5 & -1 & 500 & 500 & 100 \\ -150 & -5 & -1 & 0 & 0 & 0 & 30000 & 1000 & 200 \\ 0 & 0 & 0 & -150 & -5 & -1 & 12000 & 400 & 80 \\ -150 & -150 & -1 & 0 & 0 & 0 & 33000 & 33000 & 220 \\ 0 & 0 & 0 & -150 & -150 & -1 & 12000 & 12000 & 80 \\ -5 & -150 & -1 & 0 & 0 & 0 & 500 & 15000 & 100 \\ 0 & 0 & 0 & -5 & -150 & -1 & 1000 & 30000 & 20 \end{bmatrix}$$

The Singular Value Decomposition of A creates three matrices, U, S and V and the original matrix can be written as,

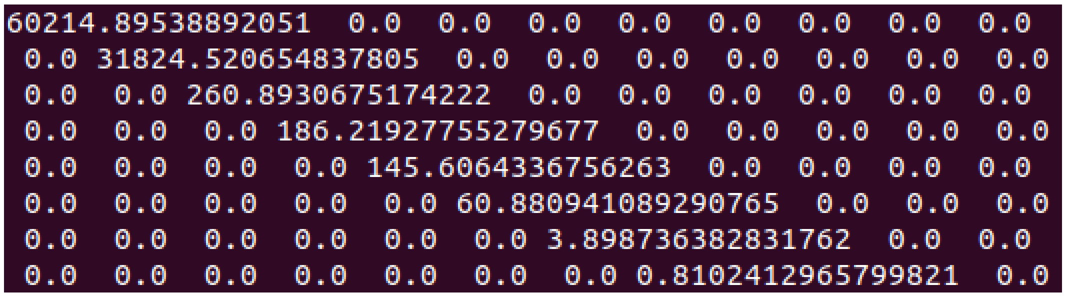
$$A = USV^T$$

The columns of V are formed by placing the right singular vectors or the orthonormal set of eigenvectors of A^TA. Therefore, V is



```
0.002840438936505505 0.0031443014714007504 -0.2463847351841536 -0.15855493232494747 -0.17524511359581812 0.1767056350286662 0.9137386253993404 -0.12826107275729718 0.05310563503428944
0.0024212173871185697 -0.0012832162578439031 -0.37700073317036853 0.1766002150816701 0.6895081469356132 0.5902733261706782 -0.05293445057100381 -0.002232309639793285 -0.00491718843816774
2.2089115412014684e-05 1.1349506370091881e-05 -0.0023721716798238914 -0.0036566043102612992 0.005195843612912123 0.007520002160995998 0.06599015922604674 0.7859706082396905 0.6146485520683468
0.0018910967997113117 0.0011741644754404866 0.061240940110874 0.34117274394986247 0.5017497399614621 -0.23249930490078427 0.37205235850266444 -0.04259835746958258 0.017701878363793694
0.0016347947120237446 -0.002906360160533953 0.5742798132281959 -0.07104053251091408 -0.3145493197507045 0.7498833660889523 -0.06198354007903222 0.004587858731740905 -0.003933750753115593
1.3398090695990063e-05 -1.1407789246244786e-05 0.005801909880405146 -0.0021518151837620173 0.002881479572564656 -0.0057350470315941815 -0.12248990918416719 -0.6049305284935621 0.7867501457133631
-0.6968537150605345 -0.7179616952786664 -7.574873495055383e-05 -0.003795641177754008 0.0025033419402561603 -0.00015022352604003182 0.004377669981766742 -0.0005551962907012214 0.00023602504467465474
-0.7179508925239937 0.6960672700663605 0.0016281320942286564 -0.0037752939462601257 0.0024975424518138283 0.0030559979484922117 -0.0006001149142014989 3.4825073056504645e-05 -4.917188432281829e-05
-0.0061601602430183654 2.2993334342777275e-05 -0.17345367911252815 0.9067416601188969 -0.37831968652005726 0.06219864515639485 0.025233877802304813 -0.0024779467625043266 0.00762164204518893
```

The eigen values of A^TA is found and arranged in the descending order. The square root of these values are the singular values, σ_i which forms the S matrix as,



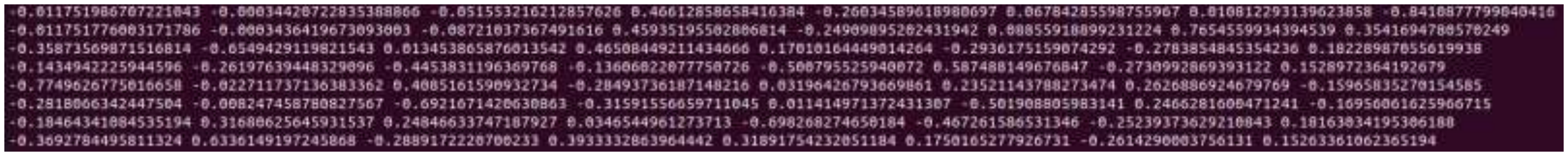
```
60214.89538892051  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0 31824.520654837805  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0 260.8930675174222  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0 186.21927755279677  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0 145.6064336756263  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0 60.880941089290765  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0 3.898736382831762  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0 0.8102412965799821  0.0
```

At this point we have found V and S. We can compute U by using the formula [2]

$$u_i = \frac{Av_i}{\sigma_i}$$

where u_i, v_i, and σ_i are the columns of the U matrix, columns of the V matrix and the singular values, respectively.

Therefore, we get the U matrix as



```
-0.011751986707221043 -0.00034420722835388866 -0.051553216212857626 0.46612858658416384 -0.26034589618980697 0.06784285598755967 0.010812293139623858 -0.8410877799040416
-0.011751776003171786 -0.0003436419673093003 -0.08721037367491616 0.45935195502806814 -0.24909895282431942 0.008559108099231224 0.7654559934394539 0.3541694788570249
-0.35873569871516814 -0.6549429119821543 0.013453865876013542 0.46508449211434666 0.17010164449014264 -0.2936175159074292 -0.2783854845354236 0.18228987055619938
-0.1434942225944596 -0.26197639448329096 -0.4453831196369768 -0.13606822077750726 -0.500795525940072 0.587488149676847 -0.2730992869393122 0.1528972364192679
-0.7749626775016658 -0.022711737136383362 0.4085161590932734 -0.28493736187148216 0.03196426793669861 0.23521143788273474 0.2626886924679769 -0.15965835270154585
-0.2818066342447504 -0.008247458780827567 -0.6921671420630863 -0.31591556659711045 0.011414971372431307 -0.501908805983141 0.2466281600471241 -0.16956061625966715
-0.18464341084535194 0.31680625645931537 0.24846633747187927 0.0346544961273713 -0.698268274650184 -0.4672615806531346 -0.25239373629210843 0.18163834195386188
-0.36927844495811324 0.6336149197245868 -0.28891722208700233 0.3933332863964442 0.31891754232051184 0.17501652779226731 -0.26142900003756131 0.15263361062365194
```

The U, S and V matrices are present in the **assets/** folder and can be viewed in a higher resolutions as there are size constraints in this document.
To compute the homography matrix, H, we must solve the homogenous overdetermined system [4],

$$Ax = 0$$

The least estimate of this solution is obtained by computing the Singular Value Decomposition of the A matrix as show above and extracting the last column of the V matrix. This column when reshaped will provide us with an estimate of the homography matrix.

Extracting and reshaping the last column, we get the homography matrix as,

$$H = \begin{bmatrix} 5.31056350\text{e}-02 & -4.91718844\text{e}-03 & 6.14648552\text{e}-01 \\ 1.77018784\text{e}-02 & -3.93375075\text{e}-03 & 7.86750146\text{e}-01 \\ 2.36025045\text{e}-04 & -4.91718843\text{e}-05 & 7.62164205\text{e}-03 \end{bmatrix}$$

Since the homography matrix is computed upto a scale and has only 8 degrees of freedom, we normalize the estimated matrix by dividing each element by H_{33} to obtain the final homography matrix,

$$H = \begin{bmatrix} 6.96774195\text{e}+00 & -6.45161293\text{e}-01 & 8.06451613\text{e}+01 \\ 2.32258065\text{e}+00 & -5.16129035\text{e}-01 & 1.03225806\text{e}+02 \\ 3.09677420\text{e}-02 & -6.45161292\text{e}-03 & 1.00000000\text{e}+00 \end{bmatrix}$$

The python implementation of the above pipeline is present in the file ***P4.py***.

Guide to code files:

1. A **gen_data_from_video.py** file is present to extract and save the top and bottom most pixels of the ball in each given video.
2. A **utils.py** file is maintained which contains custom implementations of the following functions
    - Covariance between two variables
    - A class for computing the Singular Value Decomposition of a given matrix
    - A class for Total Least Square Estimation for line fitting
    - A class for Linear Least Square Estimation for fitting a line and a parabola
    - A class for fitting a line using the RANSAC algorithm
3. Question specific python implementations can be found in **P2.py**, **P3.py** and **P4.py**.
4. The data folder contains the generated JSON files, csv data, and the original videos

Arunava Basu
117720617

5. Instructions to run the code is present in the README.md

References

[1] http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf
[2] https://www.d.umn.edu/~mhampton/m4326svd_example.pdf
[3] http://pillowlab.princeton.edu/teaching/statneuro2018/slides/notes03b_LeastSquaresRegression.pdf
[4] https://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimation/homography_estimation.pdf