Homework 1

Mrugesh J. Shah
117074109

A. JAMES CLARK
SCHOOL OF ENGINEERING

ENPM 673

—

PERCEPTION FOR

AUTONOMOUS ROBOTS

—

Dr. Mohammed Samer Charifa

Contents

Problem 1

Assume that you have a camera with a resolution of 9MP where the camera sensor is square shaped with a width of 14mm. It is also given that the focal length of the camera is 15mm.

1.1 Question 1

Compute the Field of View of the camera in the horizontal and vertical direction.

The Field of View (FOV) represents how wide a camara can capture. Higher FOV will allow camera to capture larger object. FOV is the angular size of the view cone, its value is in degrees of angle and it depends upon two factors

1. Sensor width (d)
2. Focal length of camera (f)

The formula to find FOV is,

$$FOV = 2 \times \tan^{-1} \frac{d}{2f} \qquad (1)$$

In our case,

d = 14 mm
F = 15 mm

Thus,

$$FOV = 2 \times \tan^{-1} \frac{14}{2 \times 15}$$

$$FOV = 2 \times \tan^{-1} 0.466$$

$$FOV = 50.03°$$

1.2 Question 2

Assuming you are detecting a square shaped object with width 5cm, placed at a distance of 20 meters from the camera, compute the minimum number of pixels that the object will occupy in the image.

The size of an image formed on the image sensor can be found using the formula that consist of following parameters. Assuming that it has a thin lenses.

1. Height/width of object (h/w)
2. Focal length (f)
3. Distance to object (d)

We have values,

h/w = 5 cm (As object is square)
d = 2,000 cm
f = 15 mm

$$Image\ Size = f \times \frac{h\ or\ w}{d} \qquad (2)$$

$$Image\ Size = 15 \times \frac{5}{2{,}000}$$

$$Image\ Size = 0.0375\ mm$$

The image formed on the sensor will have an area given by this formula,

$$Area\ of\ image = image\ height \times image\ width$$

$$Area\ of\ image = 0.0375\ mm \times 0.0375\ mm$$

$$Area\ of\ image = \ 0.00140625\ mm^2$$

Area of the square sensor with length 14mm will be,

$$Area\ of\ sensor = \ Sensor\ width \times length$$

$$Area\ of\ sensor = 14\ mm\ \times\ 14\ mm$$

$$Area\ of\ sensor = 196\ mm^2$$

Minimum number of pixels occupied by the image of the object on image sensor will be,

$$Pixels\ occupied = Resolution \times \frac{Area\ of\ image}{Area\ of\ camera\ sensor} \qquad (3)$$

$$Pixels\ occupied = 9 \times 10^6 \times \frac{0.00140625}{196}$$

$$Pixels\ occupied = 64.57 \cong 65$$

Thus, the minimum pixels occupied will be 65.

Problem 2

In this problem, we have to implement the algorithms for curve fitting for the points generated by the object in the given videos.

2.1 Introduction

There are two videos given in this problem, each having a ball shaped object thrown with white background. In one video, i.e., part 1, the ball has shape of a circle in all the frames and the motion of the ball is smooth parabola like. But, in part 2, the ball will change its shape in few frames along with deviating from the parabolic path.

Objective here is to get the location of that ball in each frame in the form of point or coordinates and applying curve fitting algorithm to these points to get the trajectory of these objects. The challenge here is that object in second video will generate a few coordinates that will not align with the majority of data.

We will have to implement OpenCV library to do the image processing on these videos and store the location of ball in each frame to do the further processing such as plotting the scatter plot of the points and implementing algorithms.

2.2 What is OpenCV?

OpenCV is an open-source library developed by intel and used for computer vision, image processing as well as machine learning. OpenCV can be implemented in major programming languages including Python, C++, Java, and MATLAB.

In this problem, we have used OpenCV with Python 3.6. To install the library in Linux based operating system, open the terminal and run the following command. It is important to note that the module import name for OpenCV is cv2 for python.

```
sudo apt install python3-opencv
```

It is important to note that OpenCV uses NumPy library to compute as it is easier to integrate with other libraries such as Matplotlib and SciPy. Apart from that NumPy has syntax like the syntax of MATLAB. Thus, it is recommended to have proficiency in NumPy to use OpenCV.

It is important to note that the coordinates of Y will be flipped in OpenCV generated frames. So, we need to take this into consideration while plotting the curve and generating parameters of path of objects.

2.3 Obtaining data from video

The OpenCV library has a function called cv2.VideoCapture which will take the path of the video as parameter in string format. This function will convert the .mp4 video format to the NumPy array format for each frame. The array will include the pixel location and its BGR (Blue, Green, and Red) values ranging from 0 to 255. Note that the format is BGR not the conventional RGB (Red, Green, Blue), as cv2.VideoCapture returns the color in BGR format.

Then, we have to read that video frame by frame, that's when the cv2.read function will come handy. This function will return two parameters, one Boolean and one NumPy array. The first parameter confirms if the frame is successfully received or not. Since this function will return the next frame only when it is called once again, we have to put it into a while loop. This will take the frame data and do the required image processing for that frame. But, when the last frame is shown and there are no other frames left in this video, at this time there will be a False flag on the first parameter. If it is false, we can break the loop to do further processing.

Now, we have the video frame by frame. We can detect the object in each frame now. There are several methods to do so. One of them is to look for the red colored pixel in each frame starting from the top pixels to the bottom most pixel, which will give the top point of the object in each frame.

Another approach for this problem is to use the cv2 function named cv2.findContours, which will take each frame as arguments and return the points of contours which contain the object in the frame and a hierarchy matrix which shows parent child relationship of each objects. Since our video does not contain an object inside of an object, there is no need of the hierarchy matrix. The function has been fed more arguments as well,

```
contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL,
                      cv2.CHAIN_APPROX_NONE)
```

The cv2.RETR_EXTERNAL suggest that we are only interested in the outer most contour in each frame as there are no child elements in data, and cv2.CHAIN_APPROX_NONE will make sure that the function returns the value of all the all points of contour in each frame as we must find the centroid of each element.

But, the find contour function will work well only if the frame is in binary format for the gray scale image, with the intensity (color) value of the object 255 (White colored) and back ground should be black. To implement this, we can create a mask for the object that will assign value 255 to each object with color red and value 0 (Black) to all the other colored object.

It is easy to find the HSV (Hue, Saturation, Value) for red and it is from [170, 120, 70] to [180, 255, 255]. We can apply this mask using cv2.inRange function, with given arguments being the frame, lower and upper end of the HSV spectrum for red. The returning matrix will be that frame with all the other pixel valued at 0 (black) and the red colored pixels having value 255 (white).

Now, we can use the mask to get the contours using the aforementioned function named find contours.

Note that we have just found the contour points as of now. We need the centroid of object for each frame. The centroid will give better results as it will be the center of mass of object in each frame. The centroid can be found by using the 'Moments' function, which will take each contour points and generate a moment matrix. The X and Y coordinates based on contour points will be given respectively by the ratio of 2^{nd} row 1^{st} column element and 1^{st} row 1^{st} column element for X and Y will be ratio of 1^{st} row 2^{nd} column element and 1^{st} row 1^{st} column element of the moments' matrix.

Now, the coordinates cX and cY for each frame can be appended to a list named myPoints for the further processing. The generated data set in this case will be the X and Y coordinates of centroid of object in each frame. Also, in the python program the is a function with name cv2.circle which will put circular dots on each centroid so we can visually see the centroids real time with each frame. The final outcome will look like Figure 1.

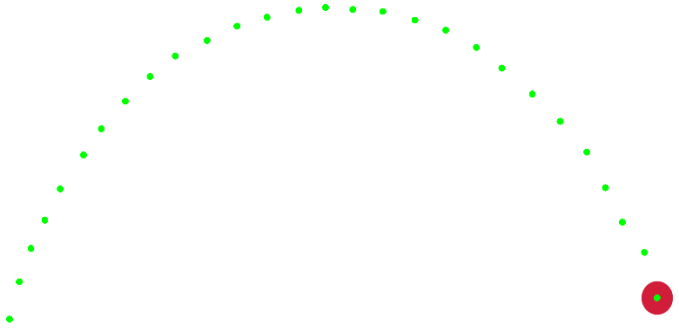

Fig 1. Path of the centroid of the object

2.4 What is Matplotlib?

Matplotlib is a library that is used to plot the functions or points on a coordinate system. We have used this library to plot the Scatterplot of our data set and the graph plot of the equation of the parabola that we derive from the curve fitting. It convenient to use it as it is based on NumPy. This library can be install using following command in the terminal for ubuntu.

```
Python -m pip install -U matplotlib
```

The module import name for matplotlib is matplotlib itself, but usually it is imported under plt name. This will make easier to call the functions based on matplotlib module.

We will use only two functions of the library in our code namely plt.scatter and plt.plot. The plt.scatter is used to plot the points on the axis while plt.plot is used to plot the equations graphs. The syntax for each of them is mentioned below.

```
plt.plot(x, y, label=`label_str`, linewidth=`linewidth_int`)
```

```
plt.scatter(x, y, c='color', marker='shape')
```

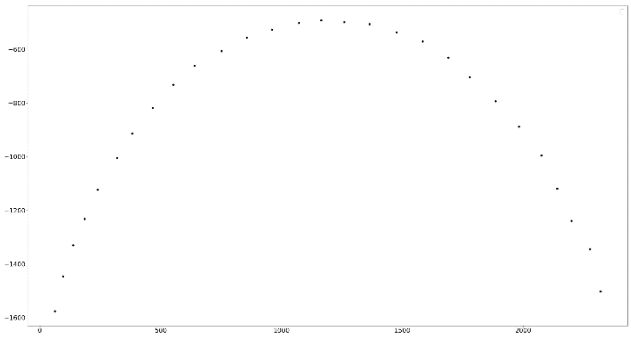

Fig. 2 Scatter plot for Video 1

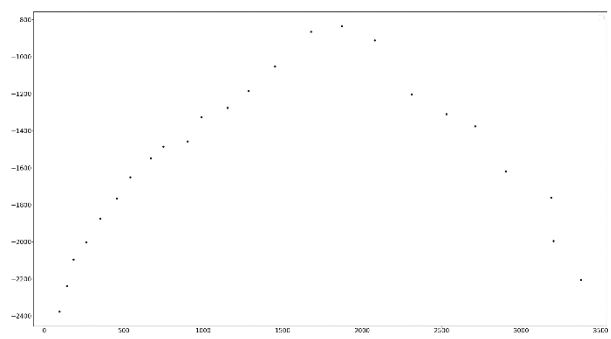

Fig. 3 Scatter plot for Video

There are several other arguments for each function, but they are mostly for changing the color of the plot, type of the plane where the graph is being plotted and many more.

2.5 What is Eigen Value Decomposition?

The Eigen Value Decomposition is used for several image processing algorithms including but not limited to image compression, homography, image denoising and finding pseudoinverse for applications such as Least Squares (LS) and Total Least Squares (TLS) method.

EVD converts the matrix to a multiplication of three different matrices with first and last matrices are the singular matrices and the middle matrix is the diagonal matrix with diagonal values are singular values of that matrix, in this case matrix A

$$EVD(A) = U \wedge U^T \qquad (4)$$

Where U = Eigen vectors of A

U^T = Eigenvectors of A^T

$\wedge$ = Diagonal matrix with values $\lambda_1, \lambda_2, \ldots \lambda_n$. Where, $\lambda_i \geq \lambda_{i+1}$.

From theoretical perspective the EVD will be converting the given matrix to three distinct matrices which are U, $\wedge$, and U^T that corresponds to final rotation, scaling along with axis, and first rotation, respectively.

While this method can be used to find the pseudoinverse as well to find the solution for the over defined or under defined system. In out case the data set is much larger than three points. Thus, this will be over defined system that will generate a tall matrix A_{MxN} where M>>N.

2.6 Curve fitting methods for Nonlinear systems

2.6.1 Least Squares (LS) error

The Least Squares (LS) error correction method will minimize the error in vertical axis only. The error for each point is squared hence the name Least Squared error. This method is also known as Ordinary Least Squares (OLS) method.

Least Squares method takes into account only the error in vertical axis. Thus, the error in X-axis is not taken into consideration. One of the major effect of this inconsideration will be visible when the curve is a vertical one or has high slope. The major error in this case will be in the X-axis, which are not accounted for in this method.

In this method we are supposed to minimize the error between the curve and the data points. The error E can be found using the following formula for a nonlinear system like a parabola.

$$E = \sum_{i=1}^{n} (y_i - ax_i^2 - bx_i - c)^2 \qquad (5)$$

Where the E is the square of the error from points Xi, Yi to the parabola with parameters a, b, and c. We can apply this function for each point in our data set. This can be easily implemented using the matrices.

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \dots & \dots & \dots \\ x_n^2 & x_n & 1 \end{bmatrix} \quad B = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

We will have such matrices to compute with. After putting the above mentioned equation in the matrix format, we get the following equation.

$$E = \|Y - XB\|^2$$

$$E = (Y - XB)^T (Y - XB)$$

$$E = Y^T Y - (2(XB)^T Y) + \left((XB)^T (XB)\right) \qquad (6)$$

This is the equation (6) of the error. We need to differentiate this to find the minima and maxima. By differentiating this equation with respect to matrix B, and solving it for zero will give us the parameters.

$$\frac{dE}{dB} = (2X^T XB) - (2X^T Y) = 0$$

$$X^T XB = X^T Y$$

This form of equation is similar to XB = Y, where A and B are known matrices while x is the unknow matrix, in this case the parameter matrix B. We will solve this by taking pseudo inverse of X. Pseudo, because A is not a square matrix and thus it will be generating a matrix that is closest to matrix X^{-1}. It is important to note that matrix A consists of only 1 axis points which are X-axis points while matrix Y consists points for Y-axis. Since both are in separate matrices, the error will be counted for only the one axis.

$$B = (X^T X)^{-1}(X^T Y) \qquad (7)$$

Implementation of LS method in python will be to generate the matrix X and matrix Y at first from the data points taken from the video. The use of NumPy library will make it easier for generating squared columns and stacking them side by side. But, in my programme I have used 2D lists to form these matrices and then using np.dot, np.transpose, and np.inverse functions on the lists. Either of the approach will work fine.

The B matrix will yield the parameters in column form and the curve generated by these parameters a, b, and c will give us a parabola that is best fit for the given set of data points.

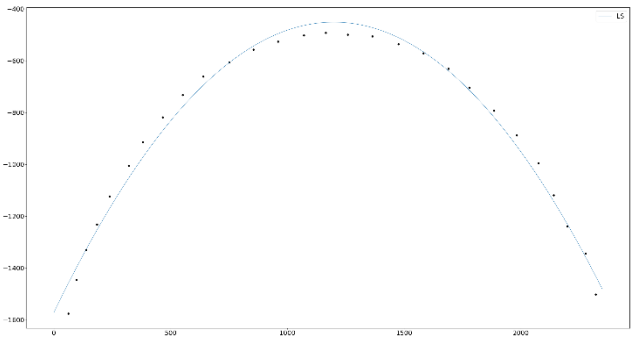

Fig. 4 LS for Video 1

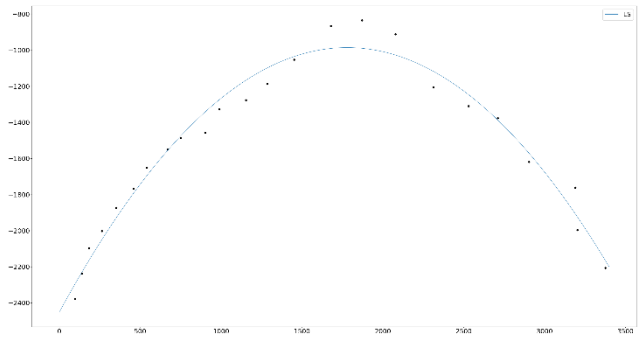

Fig. 5 LS for Video 2

The figure 3 will give the generated curve for the parameters from LS curve fitting for the first video that has no noise, while figure 4 will give the generated curve for the parameters from LS curve fitting for the video that has a lot of noise. This can be seen with the scatterplot in the background.

2.6.2 Total Least Squares (TLS) error

The Total Least Squares (TLS) method will overcome the constrains of the LS method. TLS will consider the error in both the axis. The LS method was implementing Ax=B format where matrix A will have all the X parameters, matrix B will have all the Y parameters, while x will be a column matrix of parameters.

TLS will combine Y-axis points with the X-axis points and generate matrix A. The equation for this method will be, Ax=0. Which is a homogeneous form of equation. Below is the discussion on how this method works.

The error between the data point and parabola can be found using equation (8)

$$E = (ax^2 + bx + cy - d) \qquad (8)$$

Where the condition is satisfied, $a^2 + b^2 + c^2 = 1$. The sum of squared error for all the point will be given by this equation.

$$E^2 = \sum_{i=1}^{n} (ax_i^2 + bx_i + cy_i - d)^2 \qquad (9)$$

We have to differentiate the error (equation (9)) and solve the equation when it equates to zero.

$$2\frac{\partial E}{\partial d} = \sum_{i=1}^{n} -2(ax_i^2 + bx_i + cy_i - d) = 0$$

$$\sum d = \sum (ax_i^2 + bx_i + cy_i)$$

$$d = \frac{a}{n}\sum x_i^2 + \frac{b}{n}\sum x_i + \frac{c}{n}\sum y_i$$

$$d = a\overline{x_i^2} + b\bar{x} + c\bar{y} \qquad (10)$$

From equation (10), we can put the value of d in equation (9).

$$E = \sum_{i=1}^{n} \left(a(x_i^2 - \overline{x^2}) + b(x_i - \bar{x}) + c(y_i - \bar{y})\right) = 0 \qquad (11)$$

Creating matrix A from the equation (11) that will give us the form Ax=0. By solving this, we will have the value of parameter matrix x.

$$E = \begin{bmatrix} x_1^2 - \overline{x^2} & x_1 - \bar{x} & y_1 - \bar{y} \\ x_2^2 - \overline{x^2} & x_2 - \bar{x} & y_2 - \bar{y} \\ \cdots & \cdots & \cdots \\ x_n^2 - \overline{x^2} & x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 0$$

$$A = \begin{bmatrix} x_1^2 - \overline{x^2} & x_1 - \bar{x} & y_1 - \bar{y} \\ x_2^2 - \overline{x^2} & x_2 - \bar{x} & y_2 - \bar{y} \\ \cdots & \cdots & \cdots \\ x_n^2 - \overline{x^2} & x_n - \bar{x} & y_n - \bar{y} \end{bmatrix}$$

Now, we will do further processing on the matrix A. The method that we are goin to implement here is called Eigne Value Decomposition. For that first find the A^TA matrix and find its eigen values and eigen vectors. Then we have to sort the eigen values in descending order. The new order of eigenvalues will have a new eigen vector matrix. The eigen vector corresponding to the least eigenvalue will be our solution. This method is known as Eigen Value Decomposition. We can cross verify the answer by checking if $a^2 + b^2 + c^2 = 1$ condition is satisfied or not.

The derived values of a, b, and c can be used to find the value of d using equation (10). After finding d, we have all the parameters to plot the parabola based on TLS method. The following equation (12) has to be used to plot the curve.

$$y = -\frac{a}{c}x^2 - \frac{b}{c}x + \frac{d}{c} \qquad (12)$$

Implementing the TLS method for both the videos will yield the following results.

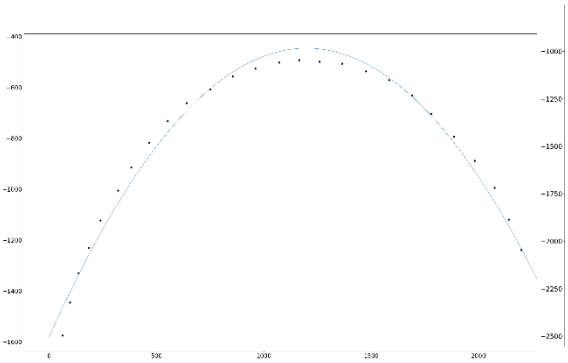

Fig. 6 TLS for Video 1

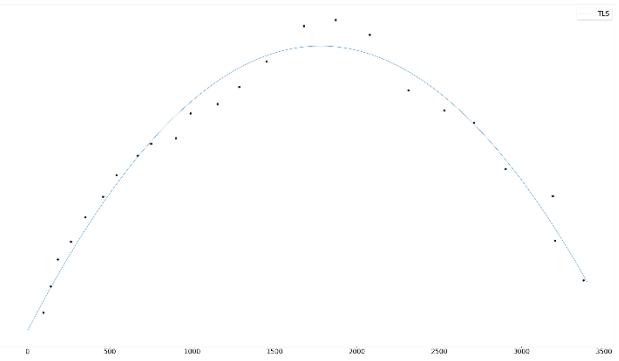

Fig. 7 TLS for Video 2

2.6.3 Random Sample Consensus (RANSAC)

The Random Sample Consensus (RANSAC) method is used when there is a lot of noise. To understand the RANSAC, we need to investigate some drawbacks of TLS method. The TLS method will consider the normal error from all the points from the data set. So, if there is a data point that has ridiculously high error and can be considered completely false data will have an impact on the final output.

What RANSAC does is that it selects required random sample from the data set and creates a model from this and checks the perpendicular error from each data point to the curve of the model generated from the samples. The samples will be taken randomly, and number of samples will be three. Then we will check if the error is less than the defined threshold or not, if the error is higher than the threshold, it will be considered as a outlier and if the error is less than the specified threshold, then the point will considered as an inlier. This whole process is a single iteration.

There will be N number of iterations, and the model generates a list of inliers and outliers. Each iteration will have a different number of inliers and outliers. The best fit model will give us the highest number of inliers with the threshold as a constrain. We now have the inliers from our curve and we can implement any of the previous curve fitting method to these inliers and will generate the curve that is not influenced by the extreme errors. It is important to have minimum number of iterations that the probability (p) of having at least one group of samples is free from inliers (p = 0.99). This can be assumed by the given equation.

$$N = \log(1-p) \Big/ \log(1- (1-e)^s) \qquad (13)$$

Where, s = Number of points in the sample
e = Probability of point being outlier (We must visually inspect this)
N = Number of samples\iterations

In nutshell, the RANSAC algorithm will draw a line between the inliers and outliers. The inliers and outliers depend upon the specified threshold. Below are the figures of derived plot from the videos. Figure 7 is the RANSAC curve fitting for the first video while figure 8 is the RANSAC curve fitting for the second video. It can be observed that the red dots in both the figures are outliers while black dots are considered as an outliers.

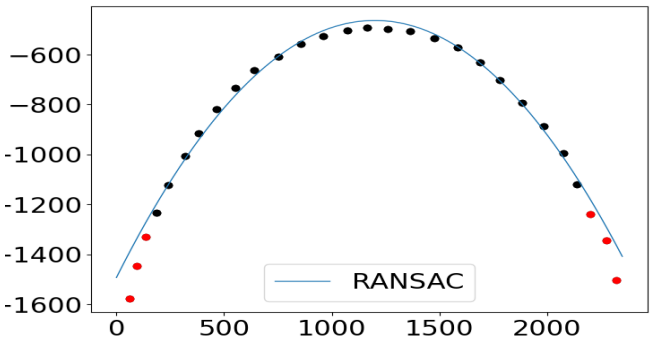

Fig. 8 RANSAC for Video 1

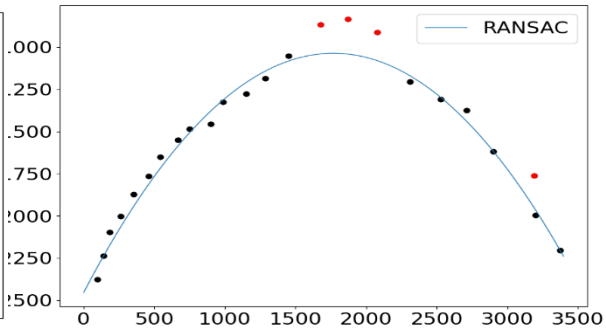

Fig. 9 RANSAC for Video 2

2.7 Comparison of these three methods

All three methods have their advantages and disadvantages. Least Squares (LS) error method is tiny bit faster compared to Total Least Squares (TLS) and less complex to implement for nonlinear systems, but LS does not consider the horizontal errors while TLS does.

Both the LS and TLS can not differentiate between noise and real data. While RANSAC can differentiate between noise and real data. But primary drawback of RANSAC is that it will take much larger time period to compute the plot, as it has to usually compute much larger number of iterations for some type of data set.

To conclude, I would recommend understanding the data set to which you want to implement the curve fitting method to. If the data is reliable and you want fast results while accuracy is not your prime concern, LS is the best option. If you have reliable data set and want accurate curve and the time to compute is not a tight constrain TLS is best option. But, if you have data set that is full of noise and errors, I recommend using RANSAC as it will generate more reliable results compared to LS and TLS.

Here are the plots that I received by implementing all three algorithms on both the videos.

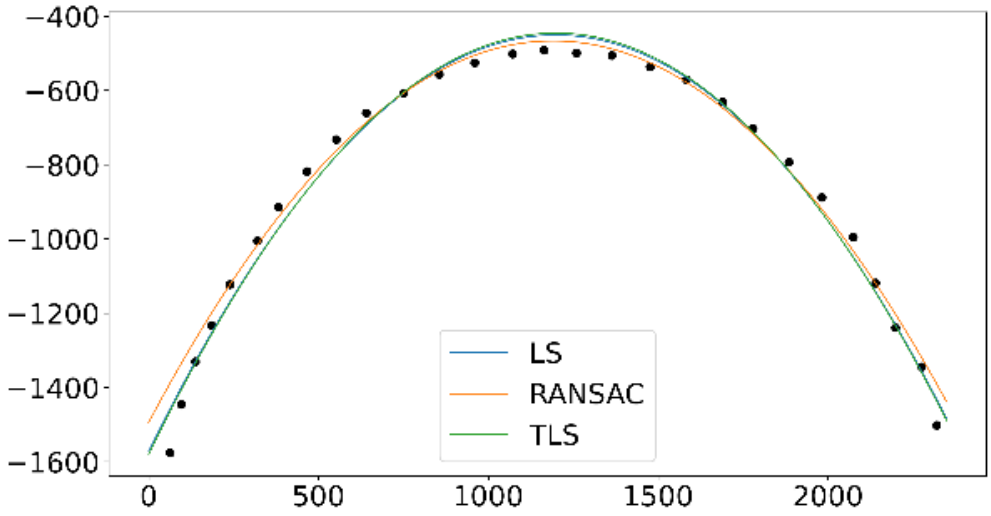

Fig. 10 All three curves from Video 1

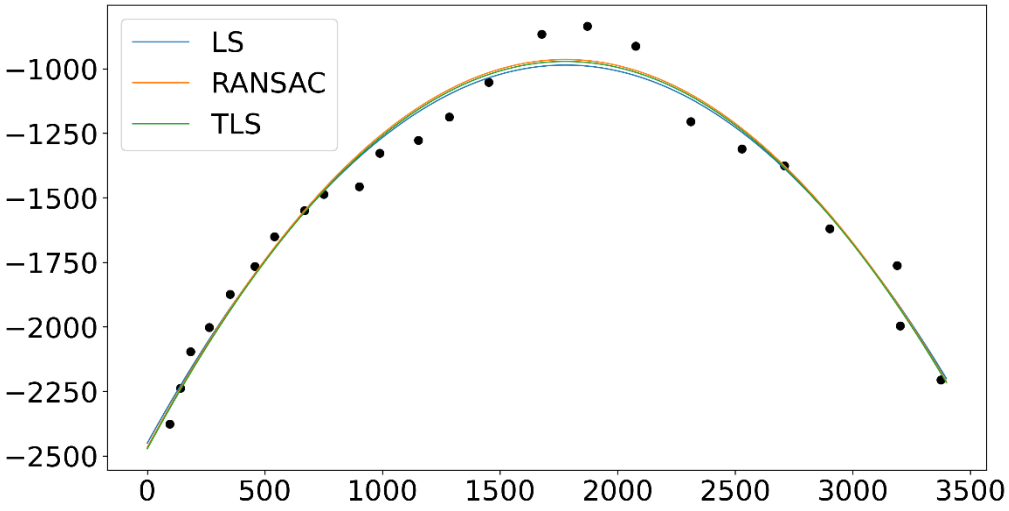

Fig. 11 All three curves from Video 2

Figure 9 shows the curves for all three methods for curve fitting for first video and Figure 10 is same for second video.

Here we can see that LS and TLS methods generates much similar results in both the cases, it can be explained by the fact that key difference between TLS and LS is that TLS considered the X-axis or horizontal error as well as vertical error. In this problem, in first case, there is not much horizontal error, thus there is not much difference between both the curves. While in second video, where there is much significant error, the error is near the vertex of the parabola. At vertex, longitudinal distance is nearly same as the Y-axis difference.

For RANSAC curve, we can visually see that it is much more accurate. I have tried different values of threshold and choose the one that generated the best fitting curve. Although RANSAC is much slower method for larger data sets, to get accurate results for the data with significant noise, it is recommended to use RANSAC.

Problem 3

3.1 Introduction to Homography matrix

Homography matrix is used to project an image on a plane that is not aligning with the current frame of reference either parallelly or perpendicularly. To put it into simpler words, let's say we have an image in rectangular shape, and we want it to place on a plain that extends to horizon. The shape of image will now become trapezoidal.

It is used to add the 3D effect to a 2D image as well. This matrix will transform the image to give it a perspective that matches with the scene. One of the major field that uses Homography matrix is Augmented Reality (AR).

3.2 What is Singular Value Decomposition (SVD)?

The Singular Value Decomposition (SVD) is used for several image processing algorithms including but not limited to image compression, homography, image denoising and finding pseudoinverse for applications such as Least Squares (LS) and Total Least Squares (TLS) method.

SVD converts the matrix to a multiplication of three different matrices with first and last matrices are the singular matrices and the middle matrix is the diagonal matrix with diagonal values are singular values of that matrix, in this case matrix A

$$SVD(A) = U\sum V^T \qquad (14)$$

Where U = Orthogonal vectors of AA^T
V = Orthogonal vectors of A^TA
$\sum$ = Diagonal matrix with values $\sigma_{1,}\ \sigma_{2,\ \ldots}\ \sigma_r$

From theoretical perspective the SVD will be converting the given matrix to three distinct matrices which are U, $\sum$, and V^T that corresponds to final rotation, scaling along with axis, and first rotation, respectively.

It is required to sort the eigenvalues of the A^TA, and then putting the eigen vectors in that order. We can do this in python code by using np.argsort function provided by NumPy. This will return the indices of the eigenvalues that are in descending order. We can use these indices to sort the eigen vector matrix for further calculation of the V^T matrix.

While this method can be used to find the pseudoinverse as well to find the solution for the over defined or under defined system. Here in the case of homography matrix $M \neq N$. Thus, we need to use the SVD to calculate the column matrix consisting values of homography matrix.

3.3 How SVD is useful?

To understand how SVD is useful, we need to look into how the Homography Matrix is calculated.

Below is the matrix formation of the data points and the homography matrix.

$$A = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1 * x_{p1} & y_1 * x_{p1} & x_{p1} \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1 * y_{p1} & y_1 * y_{p1} & y_{p1} \\ -x_1 & -x_1 & -1 & 0 & 0 & 0 & x_2 * y_{p2} & y_2 * x_{p2} & x_{p2} \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2 * y_{p2} & y_2 * y_{p2} & y_{p2} \\ -x_1 & -x_1 & -1 & 0 & 0 & 0 & x_3 * x_{p3} & y_3 * x_{p3} & x_{p3} \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3 * y_{p3} & y_3 * y_{p3} & y_{p3} \\ -x_1 & -x_1 & -1 & 0 & 0 & 0 & x_4 * y_{p4} & y_4 * x_{p4} & x_{p4} \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4 * y_{p4} & y_4 * y_{p4} & y_{p4} \end{bmatrix}, \quad x = \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix}$$

We have been given the parameters in the question which are,

	x	y	xp	yp
1	5	5	100	100
2	150	5	200	80
3	150	150	220	80
4	5	150	100	200

The homography matrix will need to have the V^T. The last column of that matrix will give us the values of the elements of the homography matrix.

The homography matrix is in a (3x3) format which is shown below, we need to use np.reshape function to convert it to (3x3) shape. Which will look like following matrix.

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix}$$

3.4 How to find SVD?

Suppose we have a matrix A, of which we want to find the SVD. The final three matrices after implementing SVD will be U, $\sum$, and V^T. Refer equation (14) for reference.

$U^TU = I$ and $V^TV = I$ as both are orthogonal marices.

We need to find the A^TA and AA^T to calculate the V^T and U. First we have to calculate the eigen values and eigen vectors of the matrix A^TA. Then we have to sort the eigen values of that matrix in descending order. Then assemble the eigenvector matrix corresponding to those eigenvalues. The transpose of this matrix will be out V^T matrix.

Now, we have to create a diagonal matrix with values of sigma (σ) with relation between eigenvalues (λ) being defined by,

$$\sigma = \sqrt{\lambda}$$

We also have to adjust the dimensions of the ∑ matrix. We need to add columns or rows to the diagonal matrix such that it has shape like matrix A.

The U matrix is simply AA^T. All the multiplication, transposes, and matrix formation is supported by different NumPy library.

3.5 Solution

The solution for homography matrix will be the last column of the V^T matrix. After running the python code for the homography matrix, following was the generated matrix H.

$$H = \begin{bmatrix} 0.05310563 & -.000491719 & 0.61464855 \\ 0.01770188 & -0.00393375 & 0.78675015 \\ 0.00023603 & -0.00004917 & 0.00762164 \end{bmatrix}$$

References

Singular Value Decomposition (SVD) tutorial

Singular Value Decomposition (SVD) tutorial. (2021). Retrieved 15 February 2021, from https://web.mit.edu/be.400/www/SVD/Sing

Finding Homography Matrix using Singular-value Decomposition and RANSAC in OpenCV and Matlab - Robotics with ROS

Finding Homography Matrix using Singular-value Decomposition and RANSAC in OpenCV and Matlab - Robotics with ROS. (2017). Retrieved 15 February 2021 from http://ros-developer.com/2017/12/26/finding-homography-matrix-using-singular-value-decomposition-and-ransac-in-opencv-and-matlab/

Least squares

Least squares. (2021). Retrieved 15 February 2021, from https://en.wikipedia.org/wiki/Least_squares

Total least squares

Total least squares. (2021). Retrieved 15 February 2021, from https://en.wikipedia.org/wiki/Total_least_suares

Least Squares Fitting -- from Wolfram MathWorld

Least Squares Fitting -- from Wolfram MathWorld. (2021). Retrieved 15 February 2021, from https://mathworld.wolfram.com/LeastSquaresFitting.html

Singular value decomposition

Singular value decomposition. (2021). Retrieved 15 February 2021, from https://en.wikipedia.org/wiki/Singular_value_decomposition

Neto, J.

Neto, J. (2021). Singular Value Decomposition. Retrieved 15 February 2021, from http://www.di.fc.ul.pt/~jpn/r/svd/svd.html

Installation Guide — Matplotlib 3.3.4 documentation

Installation Guide — Matplotlib 3.3.4 documentation. (2021). Retrieved 15 February 2021, from https://matplotlib.org/stable/users/installing.htm

OpenCV: Introduction to OpenCV-Python Tutorials

OpenCV: Introduction to OpenCV-Python Tutorials. (2021). Retrieved 15 February 2021, from https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html

OpenCV: Install OpenCV-Python in Ubuntu

OpenCV: Install OpenCV-Python in Ubuntu. (2021). Retrieved 15 February 2021, from https://docs.opencv.org/master/d2/de6/tutorial_py_setup_in_ubuntu.html

Homography (computer vision)

Homography (computer vision). (2021). Retrieved 15 February 2021, from https://en.wikipedia.org/wiki/Homography