ENPM673 Perception for Autonomous Robots

HOMEWORK 1

Aditya Khopkar (116911627)
Nalin Das (116698290)
Nidhi Bhojak (116787529)

Contents

List of Figures

1 Problem 1

Assume that you have a camera with a resolution of 5MP 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. Compute the Field of View of the camera in the horizontal and vertical direction.
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.

Solution:
Given: f=15mm
Height/Width of the sensor = h = 14mm
Resolution= 5MP
Width/Height of the object = 15mm
Distance from the object= 20m
We know that the Field of View is calculated using the formula,

$$F.O.V = 2 \times arctan\frac{h}{2f}$$

Hence, the Field of View can be determined by substituting values in the above equation,

$$F.O.V = 2 \times arctan\frac{14}{30} = 2 \times arctan(0.4467) = 50.02°$$

Image size can be determined using the equation,

$$\frac{Height/width\ of\ the\ object}{Distance\ to\ object} = \frac{Image\ Size}{Focal\ length}$$

Substituting the values we get,

$$\frac{50}{20000} = \frac{x}{15}$$

Hence, the image size is 0.0375mm Hence, area of the image is

$$0.0375 \times 0.0375 = 0.00140mm^2$$

We know that the area of the camera sensor is

$$14 \times 14 = 196mm^2$$

Now, minimum number of pixels the object will occupy in the image is given by,

$$\frac{Area\ of\ image}{Area\ of\ camera\ sensor} \times Resolution\ of\ camera$$

$$= \frac{0.001408}{196} \times 5 \times 10^6$$

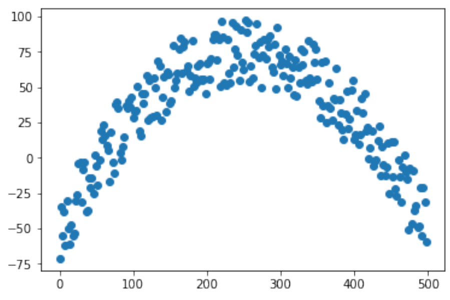
$$= 35.91\ pixels \cong 36\ pixels$$

2 Problem 2

Two files of 2D data points are provided in the form of CSV files (Dataset_1and Dataset_2). The data represents measurements of a projectile with different noise levels and is shown in figure 1. Assuming that the projectile follows the equation of a parabola,

- Find the best method to fit a curve to the given data for each case. You have to plot the data and your best fit curve for each case. Submit your code along with the instructions to run it.
- Briefly explain all the steps of your solution and discuss why your choice of outlier rejection technique is best for that case.

Solution: Curve fitting is a process of constructing a curve, or mathematical function that has the best fit to the series of data points, possibly subject to constraints. It is observed in the dataset plots that, dataset 1 has more ordered data than dataset 2. The outliers in case of dataset 2 is higher compared to that of the dataset 1.


(a) Dataset 1

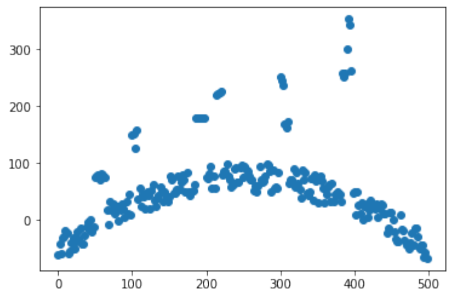

(b) Dataset 2

Figure 1: Plots of Dataset1 and Dataset2

2.1 Curve fitting method

The two datasets majorly require different methods to optimally fit the curve. While the method of 'Line fitting by Least Square error' was used in the case of dataset 1, a robust fitting technique such as RANSAC was implemented to take into consideration the effect of outliers on the curve.

Curve fitting with Least Square Error

It can be seen that dataset 1 has more ordered flow of data than dataset 2. A method of curve fitting by Least Square error was implemented. Since, the curve follows the equation of a parabola, the equation is as follows:

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

Where, a,b,c are the model parameters. We can thus form a system of non-linear equation following: $y = XA$.

Where,

$$A = \text{model paramters matrix; order} = (3\text{x}1)$$

$$X = \text{Input data matrix; order} = (m\text{x}3)$$

$$y = \text{Output data matrix; order} = (m\text{x}1)$$

Where, m = number of data points. We depict X as: $\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$, A = $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$, y = $\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \end{bmatrix}$

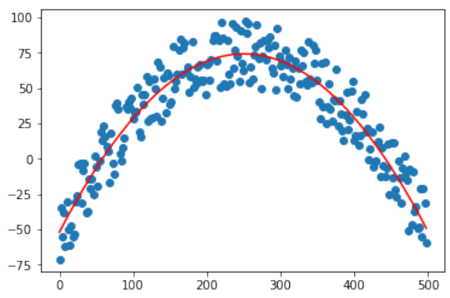
By, implementing least square error method for building the model for the curve to fit the datasets, we have the equation which represents the equation of the model with least square error.

$$X^T X A = X^T y$$

We aim to find the A matrix which gives us the model parameters. Thus, it is given by:

$$A = \left(X^T X\right)^{-1} X^T y$$

On applying this method to both the datasets, we get the output as in fig.2. It can be observed, the


(a) LSE for Dataset1

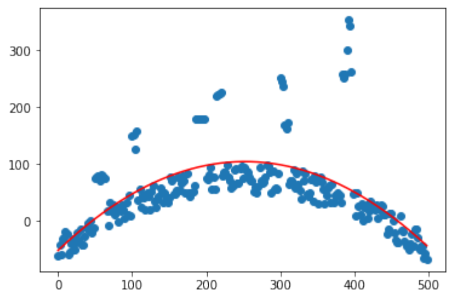

(b) LSE for Dataset2

Figure 2: Plots for Dataset1 and Dataset2 for Least Square Error

red line which represents the curve, fits the dataset 1 perfectly, while is severely penalized under the method of LSE in case of dataset 2 due to the presence of outliers. To optimize this problem, we use the implementation of RANSAC.

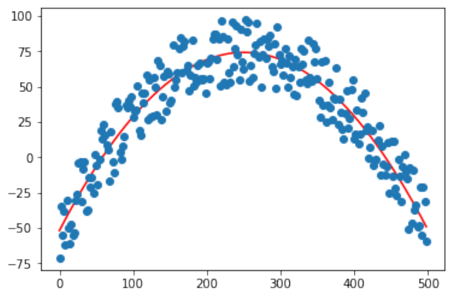
Curve fitting using RANSAC

While in case of outliers, Least square error might not be the best method because the outliers penalize the curve heavily and thus, there is a need of adopting a robust fitting method such as RANSAC (= random sample consensus). RANSAC is an iterative fitting technique to find the best fit model for the dataset. RANSAC algorithm is as follows:

- Hypothesize a model from a random set of minimum number of points (,say 3) to build a curve for the data.
- Verify the model against a threshold value using some error function.
- Update the points to be a part of the consensus set (i.e., inliers).
- Build a model using this consensus set, verify again.

- Repeat the process 'N' number of times till the best fit model is obtained.

The advantage of RANSAC is that due to the iterative procedure, with a minimum number of sample points, an optimum or a best fit model is obtained. The algorithm produces a good fit for both the datasets. However, the process is computationally expensive and thus is not preferred where the outliers are less (like in dataset 1) refer fig.[1a]. Unlike in LSE for dataset 2, RANSAC


(a) RANSAC for Dataset1

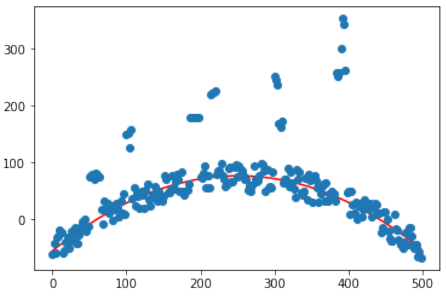

(b) RANSAC for Dataset2

Figure 3: Plots of Dataset1 and Dataset2 for RANSAC

perfectly fits the curve for dataset 2.

2.2 Optimum solution and Outlier Rejection Technique

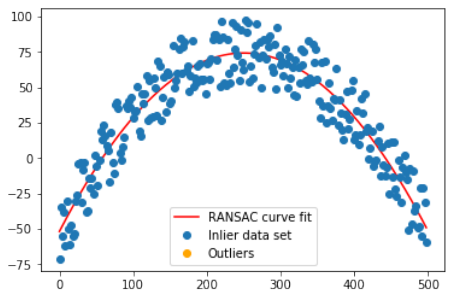
While in case of Least square error method, outlier rejection technique is not profound as we majorly deal with the dataset which have maximum inliers naturally, it is more profound in case of RANSAC. RANSAC requires definition of some of the key hyper-parameters. In this question, the following hyper-parameters have been used to find the optimum solution, these are:

- Threshold: The filter to realize the inliers and outliers. The error function is used to filter the inliers and outliers from the set of data points taken into consideration. If the error is less than the threshold, the data points are considered as inliers else, outliers. (= 45)

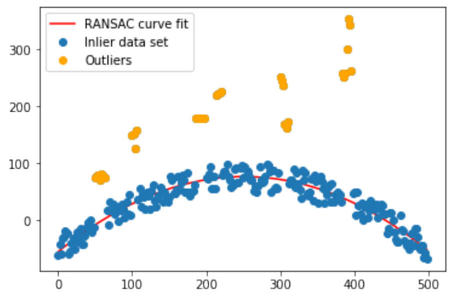
- Iteration Number: The number of times the algorithm is iterated. It was observed for less number of iteration number (< 100), optimum solution was not obtained. By increasing the iteration number, the optimum solution is obtained much efficiently, principally because, with a higher number of iteration, the model is hypothesized and verified that many times and thus, ensuring a higher accuracy. (= 1000)

- RANSAC Ratio: This hyper parameter is for tuning the algorithm to obtain the best fit model. It denotes the probability of inliers in the dataset taken into consideration. If the ratio is reached, indicates that the desired number of inliers are obtained to find the best fit model. (= 0.9)

It was also observed, that sometimes the algorithm threw an exception of obtaining a 'Singular Matrix' on calling the function of build model. It was essentially because, a matrix was passed to the function which was non-invertible, i.e., the determinant of the matrix was zero or when an all zero matrix was passed. This problem was solved by increasing the threshold of the algorithm, as



(a) Outliers for Dataset1



(b) Outliers for Dataset2

Figure 4: Plots of comparison for Dataset1 and Dataset2 w.r.t outliers

because of very low threshold, the inliers could not be defined, resulting in an empty set.

The outlier rejection can be optimized by tuning the hyper parameters. By increasing the number of iterations, the results are observed to be highly tuned and the outlier rejection is more aggressive. However, by increasing the threshold too much, the real outliers may be considered as inliers, which would reduce the efficiency of the outlier rejection technique.

3 Problem 3

3.1 Problem Description

The concept of homography in Computer Vision is used to understand, explain and study visual perspective,and, specifically, the difference in appearance of two plane objects viewed from different points of view. This concept will be taught in more detail in the coming lectures. For now, you just need to know that given4 corresponding points on the two different planes, the homography between them is computed using the following system of equations Ax = 0, where:

$$\text{A is given as} \begin{bmatrix} -x1 & -y1 & -1 & 0 & 0 & 0 & x1 \times xp1 & y1 \times xp1 & xp1 \\ 0 & 0 & 0 & -x1 & -y1 & -1 & x1 \times yp1 & y1 \times yp1 & yp1 \\ -x2 & -y2 & -1 & 0 & 0 & 0 & x2 \times xp2 & y2 \times xp2 & xp2 \\ 0 & 0 & 0 & -x2 & -y2 & -1 & x2 \times xp2 & x2 \times yp2 & xp2 \\ x3 & y3 & 1 & 0 & 0 & 0 & x3 \times xp3 & y3 \times xp3 & xp3 \\ 0 & 0 & 0 & x3 & y3 & 1 & x3 \times yp3 & y3 \times yp3 & yp3 \\ x4 & y4 & 1 & 0 & 0 & 0 & x4 \times xp4 & y4 \times xp4 & xp4 \\ 0 & 0 & 0 & x4 & y4 & 1 & x4 \times yp4 & y4 \times yp4 & yp4 \end{bmatrix}$$

- Show mathematically how you will compute the SVD for the matrix A.
- Write python code to compute the SVD.

3.2 Problem Solution

Solution: Substituting the values in the A matrix we get,

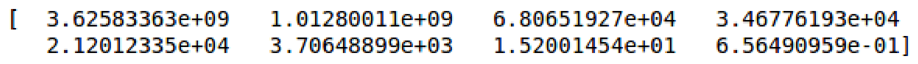
$$\text{A=} \begin{bmatrix} -5 & -5 & -1 & 0 & 0 & 0 & 500 & 500 & 100 \\ 0 & 0 & 0 & -5 & -5 & -1 & 500 & 500 & 100 \\ -150 & -5 & -1 & 0 & 0 & 0 & 30000 & 1000 & 200 \\ 0 & 0 & 0 & -150 & -5 & -1 & 120000 & 400 & 80 \\ -150 & -150 & -1 & 0 & 0 & 0 & 33000 & 33000 & 220 \\ 0 & 0 & 0 & -150 & -5 & -1 & 12000 & 400 & 80 \\ -5 & -150 & -1 & 0 & 0 & 0 & 500 & 15000 & 100 \\ 0 & 0 & 0 & -5 & -150 & -1 & 1000 & 30000 & 200 \end{bmatrix}$$

Now, $A * A^T$ is

$$\begin{bmatrix} 510051 & 510000 & 15520776 & 6208000 & 33023501 & 12008000 & 7760776 & 15520000 \\ 510000 & 510051 & 15520000 & 6208776 & 33022000 & 12009501 & 7760000 & 15520776 \\ 15520776 & 15520000 & 901062526 & 360416000 & 1023067251 & 372016000 & 30021501 & 60040000 \\ 6208000 & 6208776 & 360416000 & 144188926 & 409217600 & 148829651 & 12008000 & 24017501 \\ 33023501 & 33022000 & 1023067251 & 409217600 & 2178093401 & 792017600 & 511545251 & 1023044000 \\ 12008000 & 12009501 & 372016000 & 148829651 & 792017600 & 288051401 & 186008000 & 372039251 \\ 7760776 & 7760000 & 30021501 & 12008000 & 511545251 & 186008000 & 225282526 & 450520000 \\ 15520000 & 15520776 & 60040000 & 24017501 & 1023044000 & 372039251 & 450520000 & 901062526 \end{bmatrix}$$

The sorted eigen values for $A * A^T$ are:

The reason for sorting the eigen values in descending order is that for SVD we need to have the common eigen values and corresponding eigen vectors of U, S and V^T aligned together.



```
[  3.62583363e+09   1.01280011e+09   6.80651927e+04   3.46776193e+04
   2.12012335e+04   3.70648899e+03   1.52001454e+01   6.56490959e-01]
```

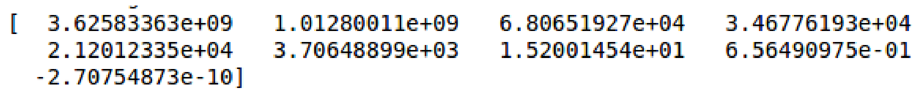
Hence the sorted eigen vectors corresponding to the sorted eigen values for $A * A^T$ is given as:

$$\begin{bmatrix} 0.01175199 & 0.00034421 & -0.05155322 & -0.46612859 & -0.2603459 & -0.06784286 & 0.01081229 & -0.84108777 \\ 0.01175178 & 0.00034364 & -0.08721037 & -0.45935196 & -0.24909895 & -0.08855919 & 0.76545599 & 0.35416947 \\ 0.3587357 & 0.65494291 & 0.01345387 & -0.46508449 & 0.17010164 & 0.29361752 & -0.27838548 & 0.18228987 \\ 0.14349422 & 0.26197639 & -0.44538312 & 0.13606022 & -0.50079553 & -0.58748815 & -0.27309929 & 0.15289724 \\ 0.77496268 & 0.02271174 & 0.40851616 & 0.28493736 & 0.03196427 & -0.23521144 & 0.26268869 & -0.15965835 \\ 0.28180663 & 0.00824746 & -0.69216714 & 0.31591557 & 0.01141497 & 0.50190881 & 0.24662816 & -0.16956061 \\ 0.18464341 & -0.31680626 & 0.24846634 & -0.0346545 & -0.69826827 & 0.46726159 & -0.25239374 & 0.18163034 \\ 0.36927845 & -0.63361492 & -0.28891722 & -0.39333329 & 0.31891754 & -0.17501653 & -0.261429 & 0.15263361 \end{bmatrix}$$

Now, $A^T * A$ is

$$\begin{bmatrix} 45050 & 24025 & 310 & 0 & 0 & 0 & -9455000 & -5177500 & -64000 \\ 24025 & 45050 & 310 & 0 & 0 & 0 & -5177500 & -7207500 & -49500 \\ 310 & 310 & 4 & 0 & 0 & 0 & -64000 & -49500 & -620 \\ 0 & 0 & 0 & 45050 & 24025 & 310 & -3607500 & -2012500 & -25500 \\ 0 & 0 & 0 & 24025 & 45050 & 310 & -2012500 & -6304500 & -42900 \\ 0 & 0 & 0 & 310 & 310 & 4 & -25500 & -42900 & -460 \\ -9455000 & -5177500 & -64000 & -3607500 & -2012500 & -25500 & 2278750000 & 1305800000 & 15530000 \\ -5177500 & -7207500 & -49500 & -2012500 & -6304500 & -42900 & 1305800000 & 2359660000 & 16052000 \\ -64000 & -49500 & -620 & -25500 & -42900 & -460 & 15530000 & 16052000 & 171200 \end{bmatrix}$$

The sorted eigen values for $A^T * A$ are:



```
[  3.62583363e+09   1.01280011e+09   6.80651927e+04   3.46776193e+04
   2.12012335e+04   3.70648899e+03   1.52001454e+01   6.56490975e-01
  -2.70754873e-10]
```

Here we can see that the eigen values of $A * A^T$ and $A^T * A$ are common, except that there is one extra eigen value (the last value) for $A^T * A$ which is equal to zero.

Hence the sorted eigen vectors corresponding to the sorted eigen values for $A^T * A$ is given as:

$$\begin{bmatrix} 0.00284044 & 0.0031443 & -0.24638474 & -0.15855493 & -0.17524511 & 0.17670564 & 0.91373863 & -0.12026107 & 0.05310564 \\ 0.00242122 & -0.00128322 & -0.37700073 & 0.17660022 & 0.68950815 & 0.59027333 & -0.05293445 & -0.00223231 & -0.00491719 \\ 0.00002209 & 0.00001135 & -0.00237217 & -0.0036566 & 0.00519584 & 0.00752 & 0.06599016 & 0.78597068 & 0.61464855 \\ 0.0010911 & 0.00117416 & 0.66124094 & 0.34117274 & 0.50174974 & -0.23249936 & 0.37205236 & -0.04259036 & 0.01770188 \\ 0.00163479 & -0.00290636 & 0.57427981 & -0.07104053 & -0.31454932 & 0.74988337 & -0.06198354 & 0.00458786 & -0.00393375 \\ 0.00001339 & -0.00001141 & 0.00580191 & -0.00215182 & 0.00288148 & -0.00573505 & -0.12248991 & -0.60493053 & 0.78675015 \\ -0.69605372 & -0.7179617 & -0.00007575 & -0.00379564 & 0.00250334 & -0.00015022 & 0.00437767 & -0.0005552 & 0.00023603 \\ -0.71795089 & 0.69606727 & 0.00162813 & -0.00377529 & 0.00249754 & 0.003656 & -0.00060011 & 0.00003483 & -0.00004917 \\ -0.00616016 & 0.00002299 & -0.17345368 & 0.90674166 & -0.37831969 & 0.06219865 & 0.02523388 & -0.00247795 & 0.00762164 \end{bmatrix}$$

To compute the Singular Value Decomposition (SVD) of any matrix A, we use the following expression:

$$\boxed{A = U * \Sigma * V^T}$$

where,

U = Column matrix of Eigen Vectors of $A * A^T$
V = Column matrix of Eigen Vectors of $A^T * A$
Σ = Diagonal Matrix of square roots of the common eigen values of $A * A^T$ and $A^T * A$

Hence U and V are the sorted column eigen vector matrices of $A * A^T$ and $A^T * A$ as given above respectively.

Σ is given as follows:

$$\Sigma = \begin{bmatrix} 60214.89538892 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 31824.52065484 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 260.89306752 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 186.21927755 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 145.60643368 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 60.88094109 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3.89873639 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8102413 & 0 \end{bmatrix}$$

Now we know that the eigen values satisfy the equation,

$$\boxed{A * x = \lambda * X}$$

Comparing it with the given system of equations,

$$A * x = 0$$

We have that λ must be equal to zero.

Hence we can find the eigen value=0 for $A^T * A$ and the corresponding eigen vector is the solution (x) for the above equation.

The eigen vector corresponding to eigen value= 0 for $A^T * A$ is given by the last eigen vector in the eigen vector matrix for $A^T * A$. This is because it had been sorted in descending order such that the eigen value=0 was at the last.

Hence x is given by,

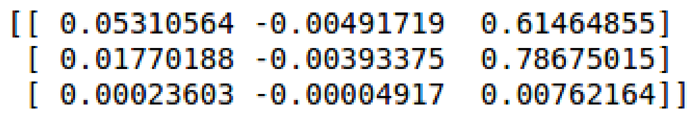
x =

```
[ 0.05310564 -0.00491719  0.61464855  0.01770188 -0.00393375  0.78675015
  0.00023603 -0.00004917  0.00762164]
```

The final obtained Homography Matrix is therefore given by,

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

H =



```
[[ 0.05310564 -0.00491719  0.61464855]
 [ 0.01770188 -0.00393375  0.78675015]
 [ 0.00023603 -0.00004917  0.00762164]]
```

3.3 Problems Encountered

1. The common eigen values and eigen vectors of $A * A^T$ and $A^T * A$ were not aligned in U and V respectively.

2. The sigma matrix was only 8x8 after creating a diagonal matrix of square roots of eigen values. Hence this created a problem while multiplying with U (8x8) and V^T (9x9) to get A matrix.

3. The given equation Ax=0 could not be solved using A inverse, since A is not square. Also while trying to solve with python's numpy equation solver, it only accepts a square A matrix. Making the A matrix square by adding a zero row resulted in a singular matrix, rendering it unsolvable. Least square method would also not work since it is an homogeneous equation, and therefore we can't multiply by pseudo-inverse.

3.4 Solutions Implemented

1. For problem 1, sorted the eigen values and with the same indices sorted the eigen vectors in U and V.

2. For problem 2, concatenated a zero column.

3. For problem 3, x is given by the eigen vector corresponding to eigen value=0.