HOMEWORK 1

ENPM 673

PERCEPTION OF AUTONOMOUS ROBOTS

Prof. Dr. Mohammed Charifa

UNIVERSITY OF MARYLAND

Nikhil Lal Kolangara (116830768)
Kartik Venkat (116830751)
Kushagra Agrawal(116700191)

This paper represents our own work in accordance with University regulations.

Contents

1 Question 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: focal length(f)=15mm
Height/Width of the sensor(h)= 14mm
Distance from the object= 20m
Width and Height of the object = 15mm
Resolution= 5MP
The Field of View is calculated using the formula,

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

Hence, the Field of View is 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,

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

Substituting the values we get,

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

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

$$0.0375 \times 0.0375 = 0.00140625mm^2$$

We know that the area of the camera sensor is

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

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

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

$$= \frac{0.00140625}{196} \times 5 \times 10^6$$
$$= 35.8735\ pixels \cong 36\ pixels$$

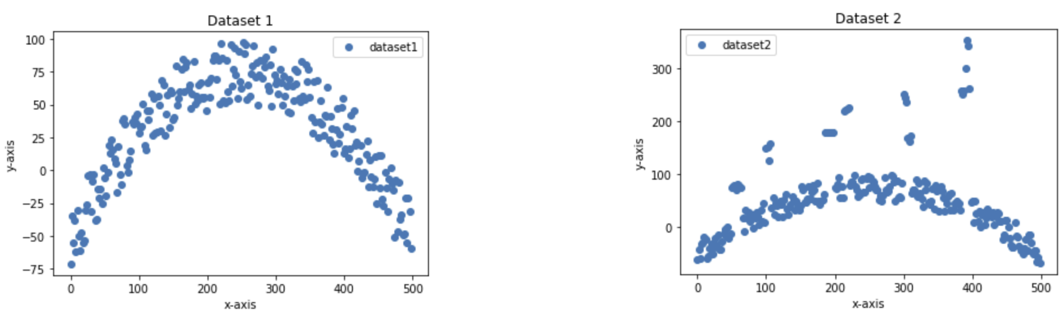
2 Question 2

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:

Often while analysing data, we are interested in the general trend (inliers) of the data more than the actual data values itself. So, we need to find a method to compute the relationship between variables and find the general trend. Hence, we use the method of curve fitting. It is a method of finding a model that provides the best fit for specific curves in the dataset. after visual analysis of the data provided in this problem, it is observed that the first dataset is more clean, ie. less noise and outliers, as compared with the second dataset which has a lot of outliers.



(a) Dataset 1

(b) Dataset 2

Figure 1: Input Datasets

2.1 Least Mean Square Error Method

The general steps to fit a curve involve determining an equation and finding its model parameters by minimizing an error function. The most commonly used error function is the Least Mean Square Error function. Since, it is given that the data follows a parabolic equation, we use quadratic equation given by:

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

Here, a,b,c are the model parameters. We get a system of non-linear equation given by: $y = XA$.
Where,

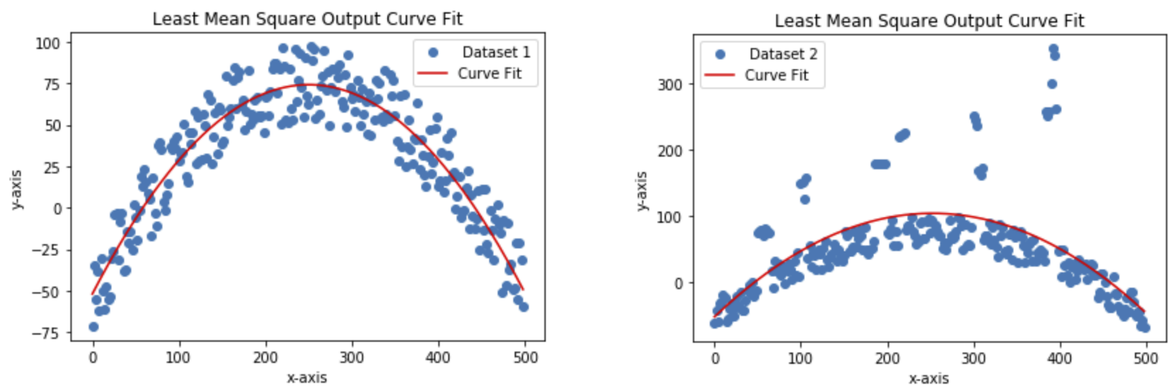
$$A = \text{model paramters matrix; order} = (3\text{x}1)$$

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

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

$$\text{Where, } X \text{ is given by: } \begin{bmatrix} n & \sum(x_i) & \sum((\mathrm{x}_i)^2 \\ \sum(x_i) & \sum((\mathrm{x}_i)^2 & \sum((\mathrm{x}_i)^3 \\ \sum((\mathrm{x}_i)^2 & \sum((\mathrm{x}_i)^3 & \sum((\mathrm{x}_i)^4 \end{bmatrix}, \ \mathrm{A} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \ \ \mathrm{y} = \begin{bmatrix} \sum(x_i) \\ \sum(x_i y_i) \\ \sum((\mathrm{x}_i)^2 \mathrm{y}_i) \end{bmatrix}$$

Hence, in the first part of our code, we perform curve fitting for both datasets using the above method (using LMSE). We then analyze the results as shown below:



(a) Dataset 1

(b) Dataset 2

Figure 2: Least Mean Square Output

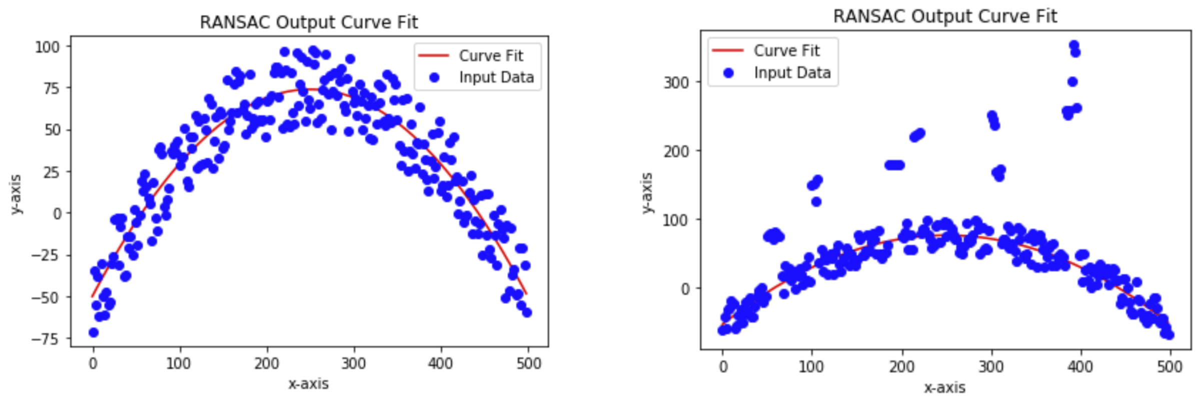
As seen above, this method provides an optimum fit for the first dataset but fails to provide a good fit for the second dataset. This deviation from the optimum fit in the second dataset is due to the amplification of the effect of outliers on the curve fit. To tackle this issue, we need to use an algorithm that is capable if handling outliers, ie. remove the effect of outliers on the curve fit. We can solve this problem by using the RANSAC algorithm.

2.2 RANSAC Method

The RANSAC algorithm works as follows:

1. Select three data points randomly from the dataset.
2. Determine the model parameters of the quadratic equation from those three data points.
3. Compare all the datapoints in the dataset with the predicted model output and classify them as inliers or outliers.
4. Select a model that maximizes the ratio of inliers to outliers.
5. Generate a curve fit from the final model.

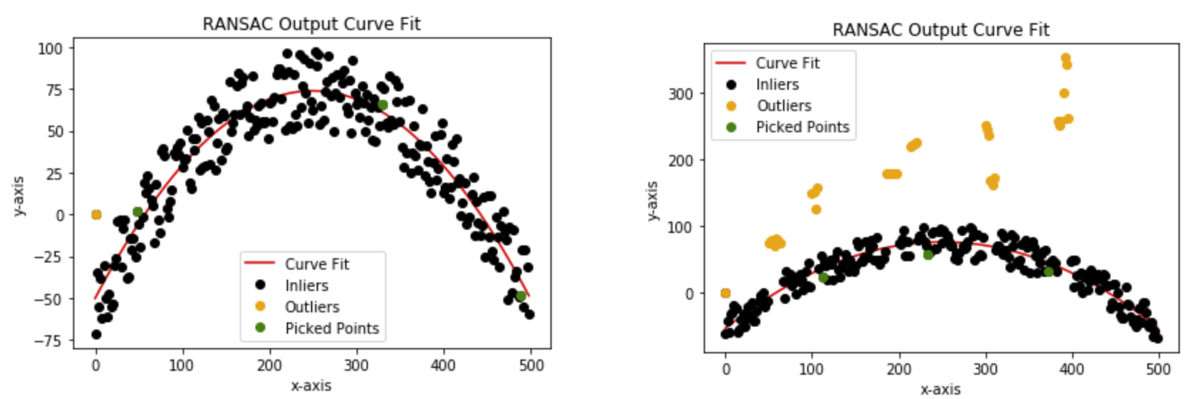
The output of the RANSAC algorithm is shown below:



(a) Dataset 1 (b) Dataset 2

Figure 3: RANSAC algorithm output

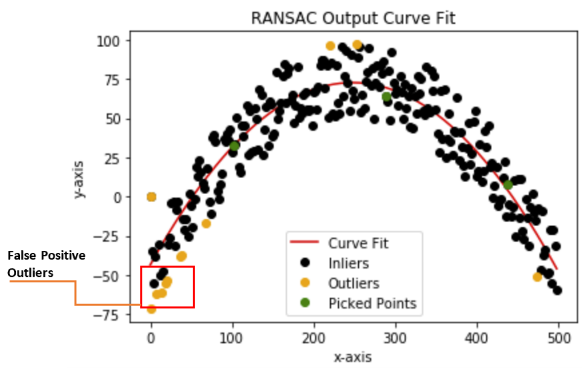
In the above images we see that this algorithm provides a good fit for both the datasets. We plot the inliers and outliers of the final model chosen by the RANSAC algorithm below to further analyse this algorithm.



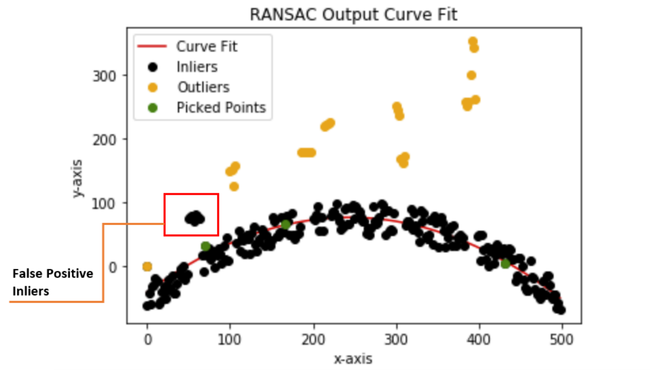
(a) Dataset 1 (b) Dataset 2

Figure 4: RANSAC algorithm detailed output

Here, we can clearly see the outlier rejection of RANSAC in action. We can also see that the accuracy of the model (more no. of inliers) can be increased by varying the minimum accuracy parameter. In case of the second dataset, we see that the threshold of the distance function plays an important role in classifying the data points as inliers and outliers. We see the effect of this parameter below:



(a) Dataset 1 with t=30



(b) Dataset 2 with t=60

Figure 5: RANSAC algorithm output showing effect of parameter ’t’

Here, we see that if we reduce the value of t, the model considers lesser points as inliers and may even wrongly classify inliers as outliers. Similarly, if the value of threshold is increased, lesser number of outliers get rejected, ie. outliers get wrongly classified as inliers.

Overall, the RANSAC algorithm does a good job rejecting outliers and its only drawback is that we need to run the algorithm long enough so as to find the best fit.

2.3 Problems encountered and Solutions

1. Singular matrix:- this problem was occurring because the np.random.randint was not generating unique values which resulted in a singular martix which cannot be inverted. The problem was solved using the random package.

2. int is not subscriptable:- This error occurred because the code failed to find a solution within the defined number of iterations. It was solved by tweaking the algorithm to handle this error.

3 Question 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: First we will substitute the given values in matrix A,

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

Now calculating, $A * A^T$ 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 eigen values for $A * A^T$ after sorting them are:

```
Eigen value of A times A transpose is:
 [  3.62583363e+09   1.01280011e+09   6.80651927e+04   3.46776193e+04
    2.12012335e+04   3.70648899e+03   6.56490959e-01   1.52001454e+01]
```

The eigen values are sorted in descending order because for SVD, the eigen values for $A * A^T$ and $A^T * A$ should be same and the corresponding eigen vectors should be alligned together.

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

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

Nowcalculating, $A^T * A$ 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 eigen values for $A^T * A$ after sorting them are:

It can be seen that the eigen values of $A * A^T$ and $A^T * A$ are same, except that there is one extra eigen value

```
Eigen value of A transpose times A is:
 [  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]
```

(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.

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

We know that the eigen values satisfy the equation,

$$\boxed{A * x - \lambda * x = 0}$$

Comparing it with the given system of equations,

$$A * x = 0$$

We can see that λ is 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($\lambda = 0) for$A$^T * A$ is given by the last eigen vector in the eigen vector matrix for $A^T * A$. This is because it has been sorted in descending order such that the eigen value($\lambda = 0) was placed at the last.$
$Therefore\ x\ is\ given\ by,$

x =

```
[ 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]
```

The final obtained Homography Matrix is given by,

```
The homography matrix is:
 [[  5.31056351e-02  -4.91718843e-03   6.14648552e-01]
 [  1.77018784e-02  -3.93375075e-03   7.86750146e-01]
 [  2.36025045e-04  -4.91718843e-05   7.62164205e-03]]
```

4 Appendix

4.1 README

ENPM 673 Perception for Autonomous Robots

@ Author Kartik Venkat, Kushagra Agrawal, Nikhil Lal Kolangara

Instructions to run the code:

1. Using Command Prompt:
   python ...PATH...97$_hw$11.py // use python3 if using Linux based OS

2. Using Spyder or any other IDE:
   Open the file and Run.

3. Using Jupyter Notebook:
   Open Assignment1.ipynb file and run it in Jupyter Notebook IDE.

Special Instructions:

1. Install all package dependencies before running the code.

2. Update pip and all the packages to the latest versions.

4.2 CODE:

```

# coding: utf-8

# # ENPM 673 Perception for Autonomous Robots

# # Assignment 1

# Question 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:  Answer in the report

# Question 2:
# Two files of 2D data points are provided in the form of CSV files (Dataset_1 and 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,
#
# 1. 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.
# 2. Briefly explain all the steps of your solution and discuss why your choice of outlier rejection technique is best
# for that case.

# First step:
#
# We import the required libraries and read the dataset. We use the read_csv function from the Pandas library to access the csv file of the dataset.

# In[1]:
print("######################################################################################")
print("                                          Solution 2:")
print("######################################################################################")

#Import Dataset and read
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math
import random
data_1 = pd.read_csv(r"C:\Users\Kartik\Documents\ENPM673\Assignment 1\Dataset\data_1.csv")
data_2 = pd.read_csv(r"C:\Users\Kartik\Documents\ENPM673\Assignment 1\Dataset\data_2.csv")


# Next step:
#
# We convert the dataset into a list and then split the data into separate x and y variable lists.

# In[2]:


data_1 = data_1.values.tolist()
data_2 = data_2.values.tolist()
#Split Data
x_data1 =[]
y_data1 =[]
x_data2 =[]
y_data2 =[]
for i in range(len(data_1)):
    x_data1.append(data_1[i][0])
    y_data1.append(data_1[i][1])
for i in range(len(data_2)):
    x_data2.append(data_2[i][0])
    y_data2.append(data_2[i][1])


# Next step:
#
# We plot both the datasets using functions from the matplotlib library.

```

```
# In[3]:



plt.figure()
plt.plot(x_data1, y_data1,'o', label= 'dataset1')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title("Dataset 1")
plt.legend()

plt.figure()
plt.plot(x_data2, y_data2,'o', label = 'dataset2')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title("Dataset 2")

plt.legend()
plt.show()


# Next step:
#
# We define a function to compute the power of a list.

# In[4]:


def calc_power(list_val,power):
    out = []
    for i in range(len(list_val)):
        out.append(pow(list_val[i],power))
    return out


# Next step:
#
# We then define a function to determine the model parameters a, b, c of the quadratic equation so that we can predict the output y for any given input x.

# In[5]:



def build_Model(x,y):
    model_params=[]
    n= len(x)

    X = np.array([[       n           ,        sum(x)      ,sum(calc_power(x,2))],
                 [      sum(x)        ,sum(calc_power(x,2)),sum(calc_power(x,3))],
                 [sum(calc_power(x,2)),sum(calc_power(x,3)),sum(calc_power(x,4))]])
    xy  = [np.dot(x,y)]
    x2y = [np.dot(calc_power(x,2),y)]
    Y   = np.array([[(sum(y))],[(sum(xy))],[(sum(x2y))]])
```

```python

    model_params = np.dot(np.linalg.inv(X),Y)

    return model_params


# Next step:
#
# We define a function to predict the output y using the model parameters and the input data
     x.

# In[58]:


def predictOutput(x,y,A1):
    y_predict = A1[2]*calc_power(x,2)+A1[1]*x+A1[0]
    return y_predict



# Next step:
#
# Plot the output curve fit using the LMSE method.

# In[59]:


plt.figure()
plt.plot(x_data1, y_data1,'o', label= ' Dataset 1')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title("Least Mean Square Output Curve Fit")

plt.plot(x_data1,predictOutput(x_data1,y_data1,build_Model(x_data1,y_data1)), color='red',
    label= 'Curve Fit')
plt.legend()
plt.show()

plt.figure()
plt.plot(x_data2, y_data2,'o', label= ' Dataset 2')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title("Least Mean Square Output Curve Fit")
plt.plot(x_data2,predictOutput(x_data2,y_data2,build_Model(x_data2,y_data2)), color='red',
    label= 'Curve Fit')
plt.legend()

plt.show()


# Next step:
#
# We then define a function that implements the RANSAC algorithm.
#
```

```python
# RANSAC algorithm:
#     1. Select three data points randomly from the dataset.
#     2. Determine the model parameters for the quadratic from those three data points.
#     3. Compare all the datapoints with the predicted model equation and classify them as
    inliers or outliers.
#     4. Select a model that maximizes the ratio of inliers to outliers.
#     5. Generate a curve fit from the final model.

# In[55]:


def ransac(x_data,y_data, n, t, success_threshold):
    final_inliers_x=[]
    final_inliers_y=[]
    final_outliers_x=[]
    final_outliers_y=[]
    worstfit= 0
    prev_inliers=0
    #Number of iterations
    # n=10000

    #Threshold value
    # t=55

    #Worst possible error is infinite error
    worst_error = np.inf

    for i in range(n):

        dataPoints = random.sample(range(len(x_data)), 3)
        #print(dataPoints)
        possible_inliers_x=[]
        possible_inliers_y=[]

        for i in dataPoints:
            possible_inliers_x.append(x_data[i])
            possible_inliers_y.append(y_data[i])
        test_Model = build_Model(possible_inliers_x,possible_inliers_y)
        y_predict = predictOutput(x_data,y_data,test_Model)
        print(possible_inliers_x)
        print(possible_inliers_y)

        num_inliers =0
        num_outliers =0
        valid_inliers_x=[0]
        valid_inliers_y=[0]
        valid_outliers_x=[0]
        valid_outliers_y=[0]

        for i in range(len(x_data)):

            if abs(y_data[i]-y_predict[i]) < t:
                valid_inliers_x.append(x_data[i])
                valid_inliers_y.append(y_data[i])
```

```
                num_inliers+=1
            else:
                valid_outliers_x.append(x_data[i])
                valid_outliers_y.append(y_data[i])
                num_outliers+=1

        if num_inliers > worstfit:
            worstfit=num_inliers

            print("###############################    Better Model Found ######################################")
            #Update chosen starting points

            input_points_x= possible_inliers_x
            input_points_y= possible_inliers_y

            #Update the model parameters
            update_model = build_Model(valid_inliers_x,valid_inliers_y)
            op= predictOutput(valid_inliers_x,valid_inliers_y,update_model)
            final_model = update_model

            #Update temperary variables to preserve data corresponding to the final chosen model.

            fin_inlier=num_inliers
            fin_outlier=num_outliers
            final_inliers_x=valid_inliers_x.copy()
            final_inliers_y=valid_inliers_y.copy()
            final_outliers_x=valid_outliers_x.copy()
            final_outliers_y=valid_outliers_y.copy()

            success_rate= (worstfit/len(x_data))*100

            if success_rate >= success_threshold:
                break
            print(num_inliers,num_outliers)

    print(fin_inlier,fin_outlier)

    print('Worstfit=',worstfit)

    plt.figure()
    plt.xlabel('x-axis')
    plt.ylabel('y-axis')
    plt.title("RANSAC Output Curve Fit")
    plt.plot(x_data,predictOutput(x_data,y_data,final_model), color='red',label='Curve Fit')
    plt.plot(x_data,y_data,'o', color='blue',label='Input Data')
    plt.legend()
    plt.show()

    plt.figure()
    plt.xlabel('x-axis')
    plt.ylabel('y-axis')
    plt.title("RANSAC Output Curve Fit")
```

```python
    plt.plot(x_data,predictOutput(x_data,y_data,final_model), color='red',label='Curve Fit')
    plt.plot(final_inliers_x,final_inliers_y,'o', color='black',label='Inliers')
    plt.plot(final_outliers_x,final_outliers_y,'o', color='orange',label='Outliers')
    plt.plot(input_points_x,input_points_y,'o', color='green',label='Picked Points')
    plt.legend()
    plt.show()


# Next step:
#
# We plot the output curve fit for the first dataset using the RANSAC algorithm.

# In[56]:


ransac(x_data1,y_data1, 10000, 45,95)


# Next step:
#
# We plot the output curve fit for the second dataset using the RANSAC algorithm.

# In[57]:


ransac(x_data2,y_data2, 10000, 45,95)


# Question 3:
#
# 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 given
# 4 corresponding points on the two different planes, the homography between them is computed using the
# following system of equations Ax = 0, where:
#
# 1. Show mathematically how you will compute the SVD for the matrix A.
# 2. Write python code to compute the SVD.

# Solution:
#
# 1. Solved in the Report
# 2. Code to compute SVD:
#

# In[71]:

print("########################################################################################")
print("                                          Solution 3:")
```

```python
print("
    #####################################################################################")

import numpy as np
from numpy import linalg as LA
import pprint

#variable declaration and initialization

x1 =5
y1 = 5
xp1 = 100
yp1 = 100
x2 = 150
y2 = 5
xp2 = 200
yp2 = 80
x3 = 150
y3 = 150
xp3 = 220
yp3 = 80
x4 = 5
y4 = 150
xp4 = 100
yp4 = 200

# Matrix A
A = np.array([[-x1, -y1, -1, 0, 0, 0, x1*xp1, y1*xp1, xp1],
             [0, 0, 0, -x1, -y1, -1, x1*yp1, y1*yp1, yp1],
             [-x2, -y2, -1, 0, 0, 0, x2*xp2, y2*xp2, xp2],
             [0, 0, 0, -x2, -y2, -1, x2*yp2, y2*yp2, yp2],
             [-x3, -y3, -1, 0, 0, 0, x3*xp3, y3*xp3, xp3],
             [0, 0, 0, -x3, -y3, -1, x3*yp3, y3*yp3, yp3],
             [-x4, -y4, -1, 0, 0, 0, x4*xp4, y4*xp4, xp4],
             [0 , 0, 0, -x4, -y4, -1, x4*yp4, y4*yp4, yp4]],dtype='float64')

print("Matrix A is:\n",A)

# A transpose
At = np.transpose(A)
print("A transpose is given as:\n",At)

# A times A transpose
AAt = np.matmul(A,At)
print("A times A transpose is:\n",AAt)

# Eigen values and Eigen vectors of A times A transpose
eigenval_AAt, eigenvec_AAt = LA.eig(AAt)
print("Eigen value of A times A transpose is:\n",eigenval_AAt)

# A transpose times A
AtA = np.matmul(At,A)
print("A transpose times A is:\n",AtA)

```

```python
#Eigen values and Eigen vectors of A transpose times A
eigenval_AtA, eigenvec_AtA = LA.eig(AtA)
print("Eigen value of A transpose times A is:\n",eigenval_AtA)

# the columns of U are the left singular vectors
U = eigenvec_AAt
print("The U matrix is:\n",U)

# V transpose has rows that are the right singular vectors
Vt = eigenvec_AtA
print("The V transpose matrix is:\n",Vt)

# S is a diagonal matrix containing singular values
S = np.diag(np.sqrt(eigenval_AAt))
S = np.concatenate((S,np.zeros((8,1))), axis = 1)
print("S matrix is given as:\n",S)

# The Homography matrix
H = Vt[:,8]
H = np.reshape(Vt[:,8],(3,3))
print("The homography matrix is:\n",H)
```