ENPM673 : Perception for Autonomous Robots **(Due: 02/11/20)**

Submission Homework #1

Samer Charifa, Abhinav Modi, Prateek Arora *Name:* Aditya Goswami*, UID:* 116951968

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. [10 POINTS]

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. [10 POINTS]

Solution:-
Given:
The camera sensor is square of width 14 mm. Since it is square so the vertical and horizontal length is same.
The focal length is 15 mm.
The resolution of the camera is 5 MP.

(a) The field of vision is given by the formula:

$$\theta = 2tan^{-1}\left(\frac{h}{2f}\right)$$

plugging in the values:

$$\theta = 2tan^{-1}\left(\frac{14}{2(15)}\right)$$
$$\theta = 2(25.032575)$$
$$\theta = 50.06515$$

Since Horizontal and vertical field of vision is same due to the square shape of lens.

$$FOV = 50.06515$$

(b) Now according to the second part of the question:
The height of the object is 5 cm or 50 mm.
The distance of object is 20 m or 20,000 mm.

So in order to find the minimum number of pixels that the object will occupy in the image, we need to find the image size first, which is given by the following formula :

$$\left(\frac{Height\ of\ Object}{Distance\ between\ object\ and\ lens}\right) = \left(\frac{Image\ Size}{Focal\ Length}\right)$$
$$\left(\frac{50}{20000}\right) = \left(\frac{Image\ Size}{15}\right)$$
$$Image\ Size = 0.0375\ mm$$

This now gives the area of image (image size).
When we have the resolution of camera, sensor size and image size, we can find the Resolution of Image (Pixels

occupied by the object in the image) by the following formula.

$$\left(\frac{Resolution\ (c)}{Area\ (Sensor)}\right) = \left(\frac{Resolution\ (I)}{Area\ (Image\ Size *\ Image\ Size)}\right)$$

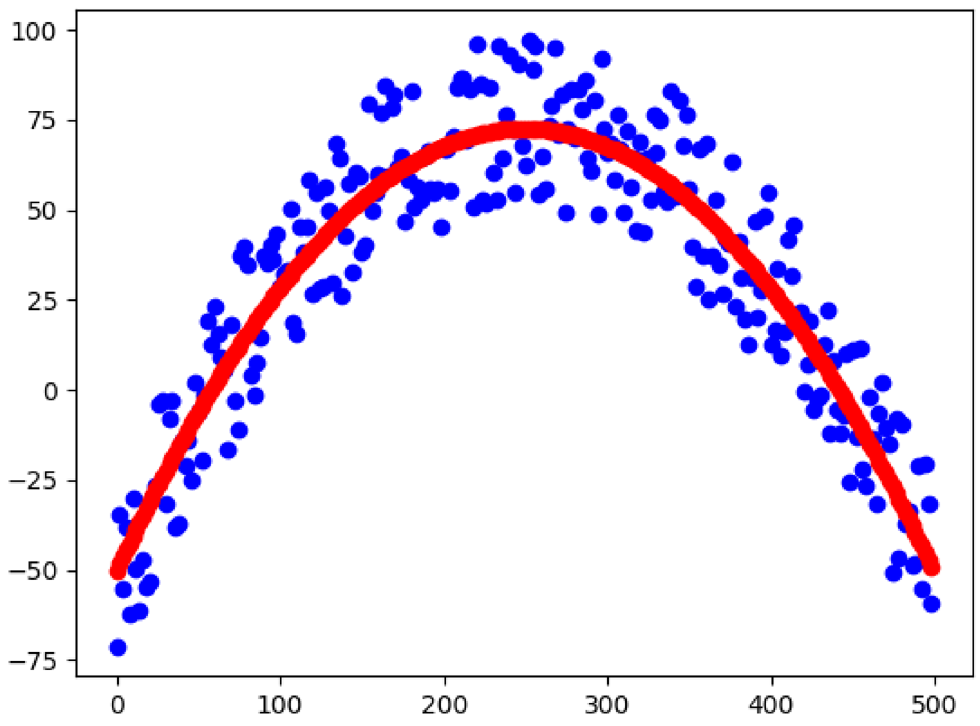
$$\left(\frac{5\ MP}{14 * 14}\right) = \left(\frac{Resolution\ (I)}{(0.0375 * 0.0375)}\right)$$

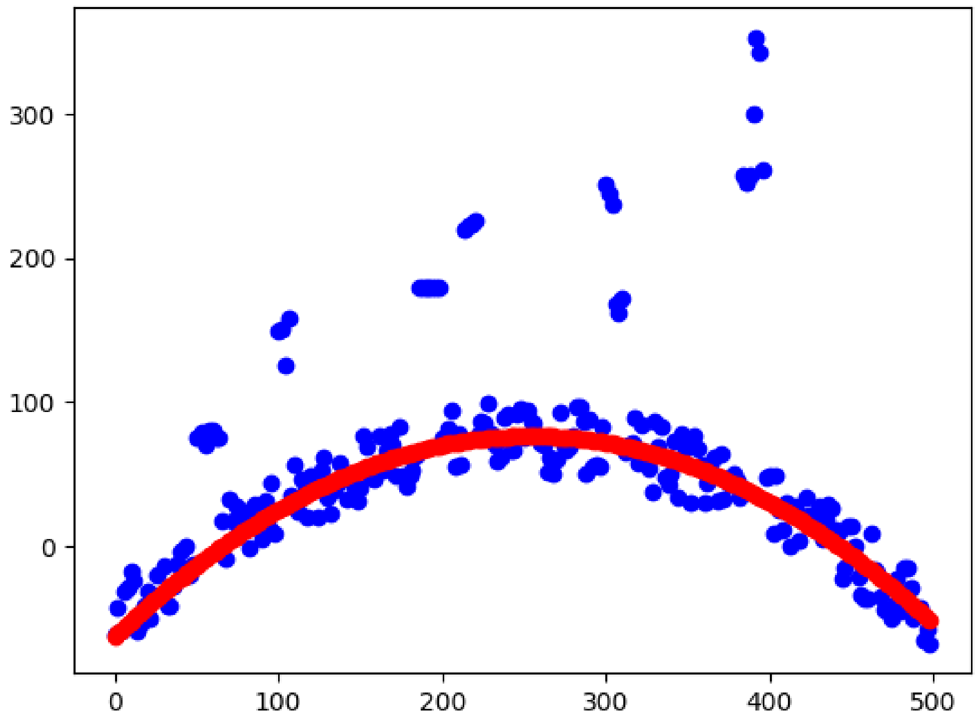
$$\left(\frac{4915,200}{196\ mm^2}\right) = \left(\frac{Resolution\ (I)}{(0.0375^2\ mm^2)}\right)$$

$$Image\ Size = 0.0375\ mm$$

Problem 2:

(a) Here Random Sampling Consensus (RANSAC) algorithm to find the best fit to the curve. Following are the steps taken to implement the algorithm:
• First a few points are required for sampling; thus, 3 random points are chosen in each iteration to form a parabola. Iter – It is the number of iterations, taken to be 3590 where the threshold to be 27 to find the most fitting points.
• Random samples are taken to form uniform data.
• Then the polynomial equation is used to best fit this set of points and keep the count of the points.
• Finally plot the best fit from this data which cover most of the Inliers. Following is the code used:





(b) Since there are no outliers in data1.csv, least square method is used to find the best fit for the parabola. Since it is a quadratic equation (parabola), polynomial features are included.

Now in data2.csv, outliers are present, therefore RANSAC algorithm is used to best fit the curve despite the presence of outliers. RANSAC provides the best method to cover the curve without having the need to clean the outliers.
Out of these two, polynomial regression throws large squared error which does not give the right curve to fit the data, which is not the case in RANSAC algorithm.
Therefore the choice – RANSAC algorithm.

Following is the code for that:

Listing 1: Python code.

```
import numpy as np
import matplotlib.pyplot as plt
import random
import argparse
import csv


#To check if the points which are randomly picked aren't repeatedly picked
def Repeating_or_not(PointsList, PointsSet):
        check = 0
        for p in PointsSet:
                if (PointsList[0] in p) and (PointsList[1] in p) and (PointsList[2] in p):
                        check = 1
        return check

#To check whether a point lies within the threshold
```

```
#Thus finding the inliers and outliers
def check_whether_outlier_inlier(a,b,c,xi,yi,th):
    dist = yi -(a*(xi**2)) - (b*xi) - c
    if abs(dist) <= th:             # if the absolute distance <= threshold -> It is an in
        return 1
    else:
        return 0

if __name__== '__main__':

    threshold = 27
    iter = 3590        # Number of iterations
    x=[]
    y=[]
    with open("c:/Users/Aditya Goswami/Downloads/problem_2_ransac/data_1.csv") as csvfile:
        d = csv.reader(csvfile, delimiter = ',')
        for r in d:
            x.append(r[0])
            y.append(r[1])

    x=np.array(x[1:],dtype=np.float32)
    y=np.array(y[1:],dtype=np.float32)

    #Creating a matrix of each type to further compute
    mat_a = y
    mat_b = []

    for i in range(len(y)):
            mat_b.append([x[i]*x[i],x[i],1])

    y = np.array(mat_a)
    x = np.array(mat_b)

    x_cord = []
    y_cord = mat_a
    for xi in x:
        x_cord.append(xi[1])

    Number_Points = len(y)
    PointsSet = []
    max_inliers = 0

    # This while statement is tailored to work best for this dataset

    for _ in range(iter):

        #Selecting three random points to generate the plot for the Parabola
        PointsList = random.sample(range(len(mat_a)), 3)
        while Repeating_or_not(PointsList, PointsSet):
                PointsList = random.sample(range(len(mat_a)), 3)
        PointsSet.append([PointsList])
        mat_x = []
        mat_y = []

        for i in PointsList:
                mat_y.append(mat_a[i])
                mat_x.append(mat_b[i])
        y = np.array(mat_y)
        x = np.array(mat_x)

        one = np.linalg.inv(x)
        two = y

        B = np.matmul(one, two)

        pres_inliers = 0

        for pointID in range(Number_Points):

                if check_whether_outlier_inlier(B[0], B[1], B[2], x_cord[pointID], y_cord[
                        pres_inliers += 1

        if pres_inliers > max_inliers:
                max_inliers = pres_inliers
                best_model = [pres_inliers, B[0], B[1], B[2]]


    print(best_model)

    yi = []
    for xi in mat_b:
        yi.append(xi[0]*best_model[1] + xi[1]*best_model[2] + xi[2]*best_model[3])

    plt.plot(x_cord, mat_a, 'bo')
    plt.plot(x_cord, yi, 'ro')
    plt.show()
```

Problem 3:

(a) To compute the SVD of the Matrix A:- We are given:

$$A \in \mathbb{R}^{mxn}$$

Here we have a 8x9 Matrix, A. Decomposition of A results in

$$A = U * S * V^{\mathrm{T}}$$

Therefore, the matrices are :

$$U \in \mathbb{R}^{\mathrm{mxm}} \quad (An\ Orthogonal\ Matrix)$$
$$S \in \mathbb{R}^{\mathrm{mxn}} \quad (A\ Rectangular\ Diagonal\ Matrix)$$
$$V \in \mathbb{R}^{\mathrm{nxn}} \quad (An\ Orthogonal\ Matrix)$$

Formulas Applied in our Algorithm :

$$A^{\mathrm{T}} * A = V * (S^{\mathrm{T}} * S) * V^{\mathrm{T}} \tag{0.1}$$

$$A * V = U * S \tag{0.2}$$

$$A * A^{\mathrm{T}} = U * (S * S^{\mathrm{T}}) * U^{\mathrm{T}} \tag{0.3}$$

Steps:
1. First we need to calculate the dot product of A'*A 2. Square root of eigen values of A'*A in descending

order are placed on the principal diagonal of S. All other entries are zeroes. 3. Eigen vectors are placed in the column of V which should mapped with values in S. 4. Each non-zero singular values in S correspond to the column of the U matrix and satisfy the equations. 5. Then form the remaining matrix U such that homogenous system of equations is satisfied by them.

(b)
The code is following:

Listing 2: Python code.

```
import numpy as np
import matplotlib.pyplot as plt
import random
import argparse
import csv


#To check if the points which are randomly picked aren't repeatedly picked
def Repeating_or_not(PointsList, PointsSet):
        check = 0
        for p in PointsSet:
                if (PointsList[0] in p) and (PointsList[1] in p) and (PointsList[2] in p):
                        check = 1
        return check

#To check whether a point lies within the threshold
#Thus finding the inliers and outliers
def check_whether_outlier_inlier(a,b,c,xi,yi,th):
    dist = yi -(a*(xi**2)) - (b*xi) - c
    if abs(dist) <= th:            # if the absolute distance <= threshold -> It is an in
        return 1
    else:
        return 0

if __name__== '__main__':

    threshold = 27
    iter = 3590        # Number of iterations
    x=[]
    y=[]
    with open("c:/Users/Aditya Goswami/Downloads/problem_2_ransac/data_1.csv") as csvfile:
        d = csv.reader(csvfile, delimiter = ',')
        for r in d:
            x.append(r[0])
            y.append(r[1])

    x=np.array(x[1:],dtype=np.float32)
    y=np.array(y[1:],dtype=np.float32)

    #Creating a matrix of each type to further compute
    mat_a = y
    mat_b = []

    for i in range(len(y)):
            mat_b.append([x[i]*x[i],x[i],1])
    y = np.array(mat_a)
    x = np.array(mat_b)

    x_cord = []
    y_cord = mat_a
    for xi in x:
        x_cord.append(xi[1])

    Number_Points = len(y)
    PointsSet = []
    max_inliers = 0

    # This while statement is tailored to work best for this dataset

    for _ in range(iter):

        #Selecting three random points to generate the plot for the Parabola
        PointsList = random.sample(range(len(mat_a)), 3)
        while Repeating_or_not(PointsList, PointsSet):
                PointsList = random.sample(range(len(mat_a)), 3)
        PointsSet.append([PointsList])
        mat_x = []
        mat_y = []

        for i in PointsList:
                mat_y.append(mat_a[i])
                mat_x.append(mat_b[i])

        y = np.array(mat_y)
        x = np.array(mat_x)

        one = np.linalg.inv(x)
        two = y

        B = np.matmul(one, two)

        pres_inliers = 0

        for pointID in range(Number_Points):

                if check_whether_outlier_inlier(B[0], B[1], B[2], x_cord[pointID], y_cord[
                        pres_inliers += 1

        if pres_inliers > max_inliers:
                max_inliers = pres_inliers
                best_model = [pres_inliers, B[0], B[1], B[2]]


    print(best_model)

    yi = []
    for xi in mat_b:
        yi.append(xi[0]*best_model[1] + xi[1]*best_model[2] + xi[2]*best_model[3])

    plt.plot(x_cord, mat_a, 'bo')
    plt.plot(x_cord, yi, 'ro')
    plt.show()
```