CSE 573: INTRODUCTION TO COMPUTER VISION AND IMAGE PROCESSING

(FALL 2018)

Dr. Junsong Yuan

University at Buffalo

PROJECT 2 REPORT

Submitted by-

ANIRUDDHA SINHA (UB Person No - 50289428)

Date: November 5, 2018

Task 1. Image Features and Homography

Aim

Find similarity between two images, match their features and warp one image with respect to the other based on the homography between the two images.

Process

This task finds the similarity between two images based on their features and keypoints that are found using **Scale Invariant Feature Transform (SIFT)**.

Further, we find the best matches between these keypoints using **K-Nearest Neighbour algorithm for nearest two neighbours (k=2)** and draw the corresponding matches of all keypoints in both images.

Next, we compute the homography matrix ***H*** using RANSAC method of comparison for all the keypoints in the first image to the second image. This computation gives us a projective space of a list of all **inliers –** points that are very close to the projective lines and relates the transformation between two planes.

We visualize the matches by drawing 10 random matches **using only inliers.**

Finally, we create a warping of the first image w.r.t the second image in a panoramic form.

Implementation

The above objective has currently been implemented in openCV in the following manner:

1. The two images given to us are `mountain1.jpg` and `mountain2.jpg`, shown in Fig.1.1 and Fig.1.2.

Fig.1.1 Original left image (mountain1.jpg)

Fig.1.2 Original right image (mountain2.jpg)

2. We read the two images `mountain1.jpg` and `mountain2.jpg`, and compute their keypoints using the inbuilt openCV library function `cv2.xfeatures2d.SIFT_create()`. The two resulting images are written to disk. Results are shown in Fig.1.3 and Fig.1.4. The keypoints are shown in multi-coloured circles on the two images.

Fig.1.3 Keypoints on the left image (mountain1.jpg)

Fig.1.4 Keypoints on the right image (mountain2.jpg)

3. Next, we detect the keypoint descriptors to identify each keypoint's orientation and characteristics that help us in finding K-nearest neighbours.

4. We apply the K-Nearest Neighbour algorithm on the keypoints from both images and match them. For every keypoint in left image, we find the best two matches in the right image, where distance of first match should be less than 0.75 times the distance of the second match from the keypoint in the left image. For this, I have used the **Fast Library for Approximate Nearest Neighbors (FLANN)** which is a highly fast K Nearest Neighbour calculation algorithm.

   Once, the matches are found, we draw them on both images. Results are shown in Fig.1.5.

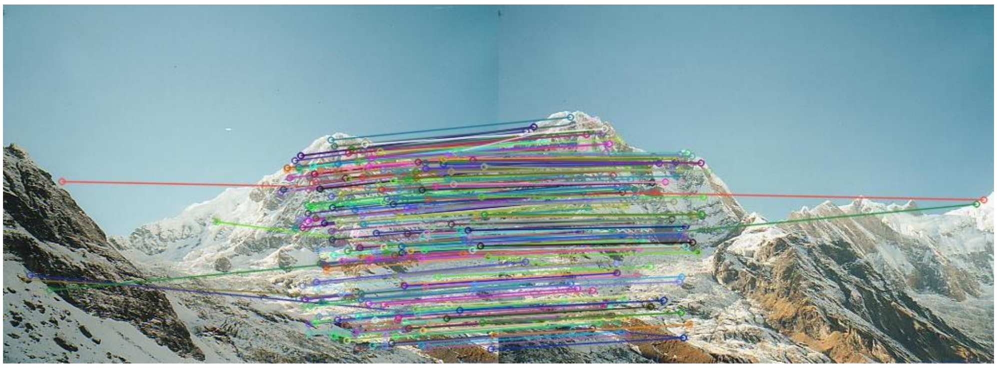

Fig.1.5. Drawn matches of keypoints on both left and right images

5. Next, we compute the homography matrix H, using the inbuilt openCV library function, `cv2.findHomography`, which takes the set of keypoints in both images as first two arguments, followed by the method of calculation, which is `RANSAC` here, followed by a threshold. We have set our `threshold = 5.0`. It returns us a 3x3 Homography matrix and a set of mask of matches between the two perspective planes.

The values of the Homography matrix are shown below –

$$\mathbf{H} = \begin{matrix} \mathbf{1.5893} & \mathbf{-2.9155 \times 10^{-1}} & \mathbf{-3.959 \times 10^{2}} \\ \mathbf{4.4942 \times 10^{-1}} & \mathbf{1.4311} & \mathbf{-1.9061 \times 10^{2}} \\ \mathbf{1.2126 \times 10^{-3}} & \mathbf{-6.287 \times 10^{-5}} & \mathbf{1} \end{matrix}$$

6. We select any random 10 inliers from the set of inliers found from the mask of matches from homography computation and use them for matching between two images. Results are shown in Fig.1.6.

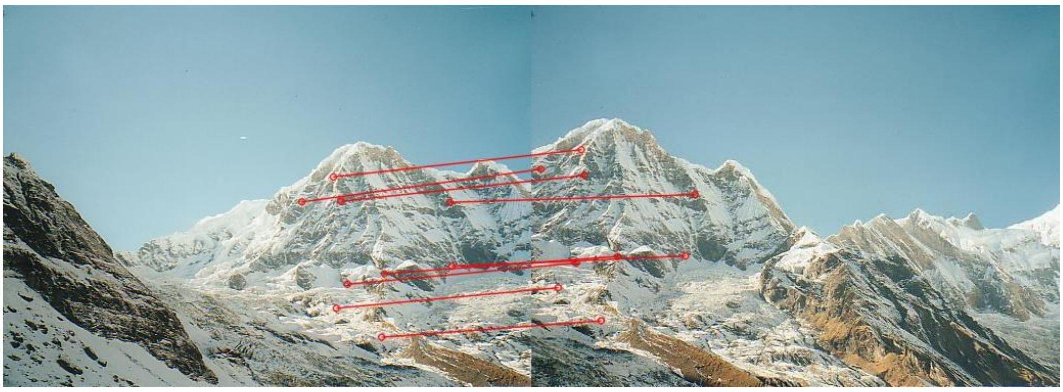

Fig.1.6 Matches between 10 random inliers on mountain1 (left) and mountain2 (right) images.

7. Finally, we warp the first image w.r.t the second image based on its orientation without losing any pixels. For this, we use the inbuilt openCV library function `cv2.warpPerspective()`. We translate the homography matrix such that no pixels of either images are lost while mapping them on to a new frame or canvas where left is warped w.r.t right. Results are shown in Fig.1.7.

Fig.1.7 Warped first image to the second image using H

<u>Conclusion:</u>

After implementation of this task, we learn the means by which SIFT can be used for feature detection in two or more different images of the same frame and how homography can help us in manipulating and transforming those images with respect to one another.

Source Code for Task1:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 29 11:29:06 2018
@author: Aniruddha Sinha
UB Person Number = 50289428
UBIT = asinha6@buffalo.edu

"""
UBIT = '<asinha6>'; import numpy as np; np.random.seed(sum([ord(c) for c in UBIT]))

import cv2
import numpy as np
import random
import math
import time

'function to write the image to disk'
def writeImage(img, imageName):
    cv2.imwrite("results/" + imageName + ".jpg", img)

'Start function execution'
t = time.time()

'Read the two images, first in colour and then in grayscales'
left_img = cv2.imread('data/mountain1.jpg')
rt_img = cv2.imread('data/mountain2.jpg')

mountain1 = cv2.imread("data/mountain1.jpg", 0)
mountain2 = cv2.imread("data/mountain2.jpg", 0)

############################# PART 1
##################################################
print('\n\n###################### Starting execution of Task 1 part 1
##########################################')

''' The following code for computing SIFT features and drawing keypoints has been
referenced and partly copied from '''
'''' https://docs.opencv.org/3.4.3/da/df5/tutorial_py_sift_intro.html '''

'Compute the keypoints and features of the two images'
sift = cv2.xfeatures2d.SIFT_create()
keypoints1 = sift.detect(mountain1, None)
keypoints2 = sift.detect(mountain2, None)

'Draw the keypoints on the images and write to disk'
task1_sift1 = cv2.drawKeypoints(left_img, keypoints1, None)
task1_sift2 = cv2.drawKeypoints(rt_img, keypoints2, None)
writeImage(task1_sift1, "task1_sift1")
print('\n\n******************** Image task1_sift11 successfully written to drive
**************************')
writeImage(task1_sift2, "task1_sift2")
print('\n\n******************** Image task2_sift2 successfully written to drive
**************************')
print('\n\n################ Task 1 part 1 successfully
completed.#####################\nTime taken for Task 1 part 1 = ',time.time()-t,'
seconds')

############################# END PART 1
################################################

########################## PART 2
###################################################
print('\n\n###################### Starting execution of Task 1 part 2
##########################################')
'Find the keypoints and the descriptors of the keypoints'
```

```
''' The following code for computing the K Nearest Neighbours and drawing matches
 has been referenced and party copied from -
 https://docs.opencv.org/3.0-
beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html '''

keypoint_mountain1, descriptor_left_img = sift.detectAndCompute(mountain1, None)
keypoint_mountain2, descriptor_rt_img = sift.detectAndCompute(mountain2, None)

'Using FLANN matching technique for finding K Nearest Neighbours'
FLANN_INDEX_KDTREE = 0

check_index_params = dict(algorithm = FLANN_INDEX_KDTREE, tress = 5)
check_search_params = dict(checks=100)

'''Create object for FLANN matching'''
doFlann = cv2.FlannBasedMatcher(check_index_params, check_search_params)
NearestNeighborsMatches = doFlann.knnMatch(descriptor_left_img, descriptor_rt_img,
k=2)


'Calculate the better neighbour of the two best matches of a keypoint of left image
in the right image'
keepKeypoints = []
for m,n in NearestNeighborsMatches:
    if m.distance < 0.75*n.distance:
        keepKeypoints.append(m)

'Draw the matched keypoints of the two images'
task1_matches_knn = cv2.drawMatches(left_img, keypoint_mountain1, rt_img,
keypoint_mountain2, keepKeypoints, None, flags=2)
writeImage(task1_matches_knn, "task1_matches_knn")
print('\n\n******************** Image task1_matches_knn successfully written to
drive ************************')
print('\n\n############### Task 1 part 2 successfully
completed.####################\nTime taken for Task 1 part 2 = ',time.time()-t,'
seconds')

######################### END PART 2
####################################################

################################## PART 3
###############################################
# Draw the matches between 10 random points in the two images
#task1_matches_knn = cv2.drawMatchesKnn(mountain1, keypoint_mountain1, mountain2,
keypoint_mountain2, NearestNeighborsMatches, None, **draw_parameters)
print('\n\n#################### Starting execution of Task 1 part 3
##########################################')

''' The following code for finding Homography and computing the inliers has been
referenced and
party been copied from https://docs.opencv.org/3.0-
beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html
'''

MIN_NUM_MATCHES = 10

if len(keepKeypoints) > MIN_NUM_MATCHES:
    coord1 = np.array([ keypoint_mountain1[m.queryIdx].pt for m in keepKeypoints ])
    np.reshape(coord1, (-1,1,2))
    coord1 = np.float32(coord1)
    coord2 = np.array([ keypoint_mountain2[m.trainIdx].pt for m in keepKeypoints ])
    np.reshape(coord2, (-1,1,2))
    coord2 = np.float32(coord2)

    'Find homography matrix and list of inliers after RANSAC algorithm'
    H, homographyStatus = cv2.findHomography(coord1, coord2, cv2.RANSAC, 5.0)
#
```

```
    print('\n\nHomography matrix is',H)
#    print('Homography status', homographyStatus)

    print('\n\n############### Task 1 part 3 successfully
completed.######################\nTime taken for Task 1 part 3 = ',time.time()-t,'
seconds')
############################ END PART 3 ##########################################

########################## PART 4 ########################################
    print('\n\n##################### Starting execution of Task 1 part 4
##########################################')
    'convert the inliers to a 1D list'
    maskOfMatches = homographyStatus.ravel().tolist()
else:
    print('\n\nMatches found are less than 10')
    maskOfMatches = None

'Select 10 random inliers after shuffling the array of inliers'
tenMatches = np.where(np.array(maskOfMatches)==1)[0]
np.random.shuffle(tenMatches)
tenRandomMatches = tenMatches[1:10].tolist()
#matchesMask= np.array(maskOfMatches)
matchesMask = []

for pt in range(len(maskOfMatches)):
    if pt in tenRandomMatches:
        matchesMask.append(1)
    else:
        matchesMask.append(0)
#print('\n\n\n10 random inlier locations are:')
print(tenRandomMatches)
#print(len(matchesMask), len(maskOfMatches), len(keepKeypoints))

'Draw the matched 10 random inliers in both images'
task1_matches = cv2.drawMatches(left_img, keypoint_mountain1, rt_img,
keypoint_mountain2, keepKeypoints, None, (0, 0, 255), None, matchesMask, flags=2)
writeImage(task1_matches, "task1_matches")
print('\n\n******************** Image task1_matches successfully written to drive
**************************')
print('\n\n############### Task 1 part 4 successfully
completed.######################\nTime taken for Task 1 part 4 = ',time.time()-t,'
seconds')

########################### END PART 4 #########################################

########################### PART 5 ############################################
print('\n\n##################### Starting execution of Task 1 part 5
##########################################')

''' The following code for computing the translation and rotation of image corners
 and warping one image to the other has been referenced from
https://docs.opencv.org/3.4.3/da/d6e/tutorial_py_geometric_transformations.html'''

'Read the shapes of the two images'
mountain1_h, mountain1_w = mountain1.shape
mountain2_h, mountain2_w = mountain2.shape

'Save the corners of each image in a form of list of list'
points_m1 = np.float32([[0,0], [0,mountain1_h], [mountain1_w, mountain1_h],
[mountain1_w,0]]).reshape(-1,1,2)
points_m2 = np.float32([[0,0], [0, mountain2_h], [mountain2_w, mountain2_h],
[mountain2_w,0]]).reshape(-1,1,2)

'Compute the translation and rotation of each corner of left image w.r.t the right
image using Homography matrix'
points_m1_modify = cv2.perspectiveTransform(points_m1,H)
#print('points_m1_modify',points_m1_modify)
```

```
'''Add the translated corners of the left image to the corners of the right image
in the form of a
list of list for each corner location on the canvas. We add alon axis=0, i.e. along
the rows'''
all_mPoints = np.concatenate((points_m2, points_m1_modify), axis=0)

'''Find the minimum and maximum values from all the corners to create the size of
the frame at a
 distance of +-1 '''
[pano_xmin, pano_ymin] = np.int32(all_mPoints.min(axis=0).ravel() - 1.0)
[pano_xmax, pano_ymax] = np.int32(all_mPoints.max(axis=0).ravel() + 1.0)

'Calculate the left upper most and lower most corners of the frame'
transformationM = [-pano_ymin, -pano_xmin]

'Compute the Translated matrix for the new frame, w.r.t which our H will be
translated'
translatedH = np.array([[1,0,transformationM[1]], [0,1,transformationM[0]],
[0,0,1]])

'warp the left image w.r.t the right image'
task1_pano = cv2.warpPerspective(left_img, translatedH.dot(H), (pano_xmax -
pano_xmin, pano_ymax - pano_ymin))
task1_pano[transformationM[0]:mountain2_h + transformationM[0],
transformationM[1]:mountain2_w + transformationM[1]] = rt_img
writeImage(task1_pano,"task1_pano")
print('\n\n******************* Image task1_pano successfully written to drive
*************************')
print('\n\n############### Task 1 part 5 successfully
completed.####################\nTime taken for Task 1 part 5 = ',time.time()-t,'
seconds')
```

Task 2. Epipolar Geometry

Aim

Find the epipolar lines and epipoles for given two different images of the same frame.

Process

First, we find the keypoints in two images, and compute which features in both images are similar. Once, we have found that, the corresponding matching features are matched using lines and the image is saved.

Next, to calculate the amount of epipolarity between two images, it is necessary to calculate the **fundamental matrix *F*** to draw inference about the translation and rotation of two images, also with their intrinsic parameters. This is provided by F.

After calculating F, we search for 10 random inlier epipolar points for both images. For each epipolar point in the left image, we compute the epiline and draw it on the right image and do it likewise for the right image and draw the epiline on the left image.

While taking a photo of a 3D space by a single camera, the depth information of the 3D space is lost when it is mapped to a 2D space on the camera's sensor. Hence, epipolarity helps us in finding the depth of a subject on a 2D frame.

Finally, to find the depth and disparity of the images, we calculate the disparity map for both of them that gives us the depth information about the subjects in the frame and infer the disparities in the two images.

Implementation

The required task has been implemented using the following steps:

1) The two images given to us are `tsucuba_left.png` and `tsucuba_right.png`, as shown in Fig.2.1 and Fig.2.2.

Fig.2.1 Original image, tsucuba_left.png

Fig.2.2 Original image, tsucuba_right.png

2) The two images are read and we compute the keypoints for both images by using the inbuilt openCV library function `cv2.xfeatures2d.SIFT_create()`. The two resulting images are written to disk. Results are shown in Fig.2.3 and Fig.2.4. The keypoints are shown in red points on the images.

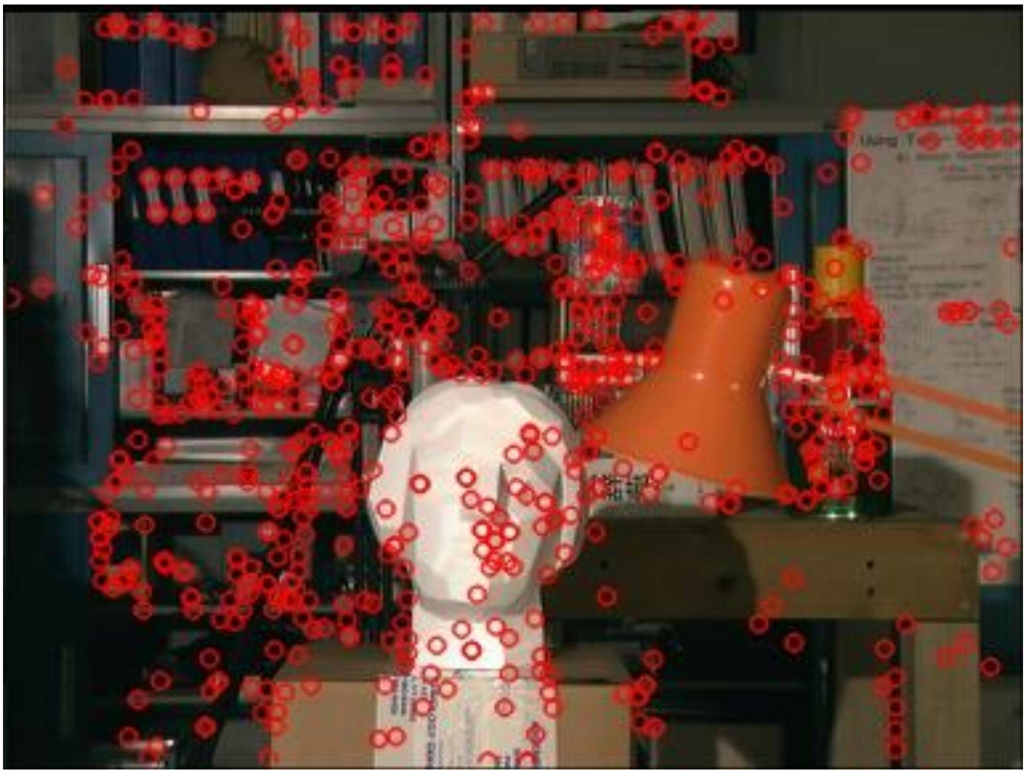

Fig.2.3 Keypoints of tsucuba_left.png

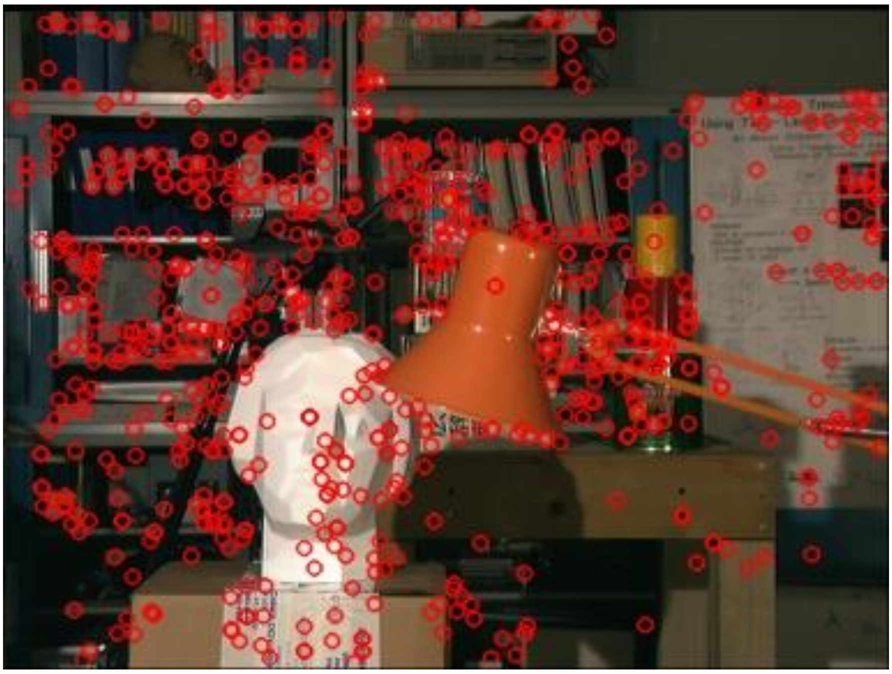

Fig.2.4 Keypoints of tsucuba_right.png

3) Next, we use K Nearest Neighbour Algorithm to find matching keypoints and plot them on the image, shown in Fig.2.5. The keypoints are shown in green, and the matches are shown in green.

Fig.2.5 K-Nearest Neighbours matching of keypoints in the tscuba_left.png and tsucuba_right.png

4) Next, we calculate the fundamental matrix *F* that gives us the values about how keypoints in one image are changed in perspective w.r.t the other image. We do this by using the inbuilt openCV library function, `cv2.findFundamentalMat` and setting the method of computation as `cv2.FM_RANSAC` with a **threshold of minimum distance =** `20.0`. The values of the fundamental matrix are given below:

$$\textbf{F (for RANSAC = 20.0)} = \begin{matrix} -1.5793 \times 10^{-6} & -4.1279 \times 10^{-4} & 8.4494 \times 10^{-2} \\ 3.9447 \times 10^{-4} & -9.8901 \times 10^{-5} & -1.3811 \times 10^{-2} \\ -8.0523 \times 10^{-2} & -1.3576 \times 10^{-1} & 1.0000 \times 10^{0} \end{matrix}$$

5) Further, we randomly select **10 inlier match pairs and compute epilines in right image for each keypoint in the left image and draw them, and vice-versa.** This task is achieved using the inbuilt openCV library function, `cv2.computeCorrespondEpilines`. The epilines for

different match pairs are drawn using different colours and the same colour for epilines with same match pairs on the left and right image, as shown in Fig.2.6 and Fig.2.7.

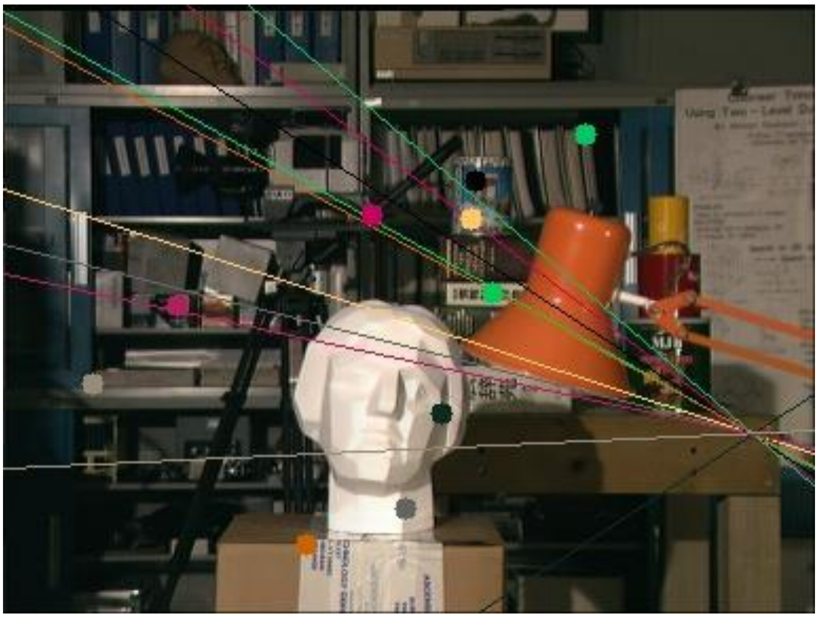

Fig.2.6 Epilines drawn on the right image, for each keypoint in the left image

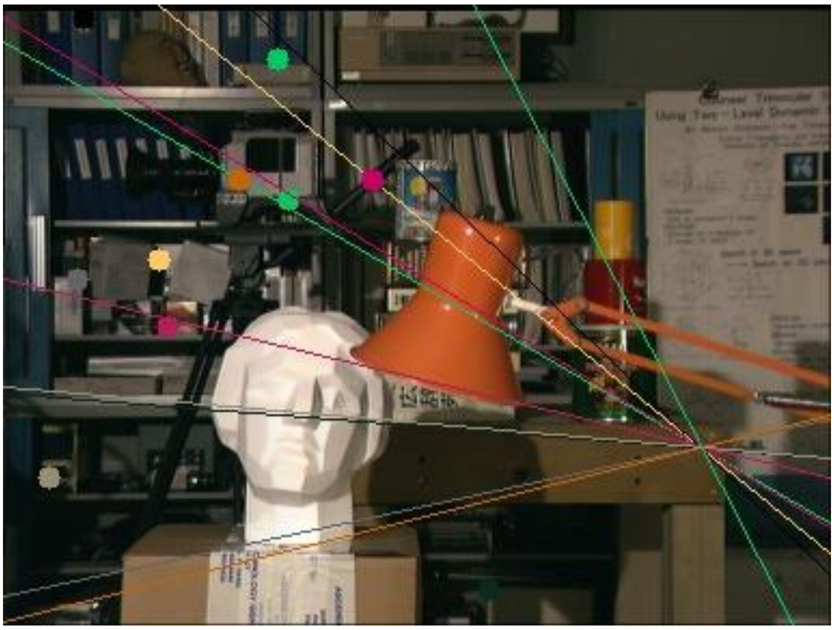

Fig.2.7 Epilines drawn on left image, for each keypoint in the left image

6) Finally, to compute the depth details of the two tsucuba images, we compute the disparity map for the left and the right image. For this we have used the inbuilt openCV library function, `cv2.StereoBM_create()` with default arguments, that gives us the disparity for a stereo vision model. The resulting image is shown in Fig. 2.8.

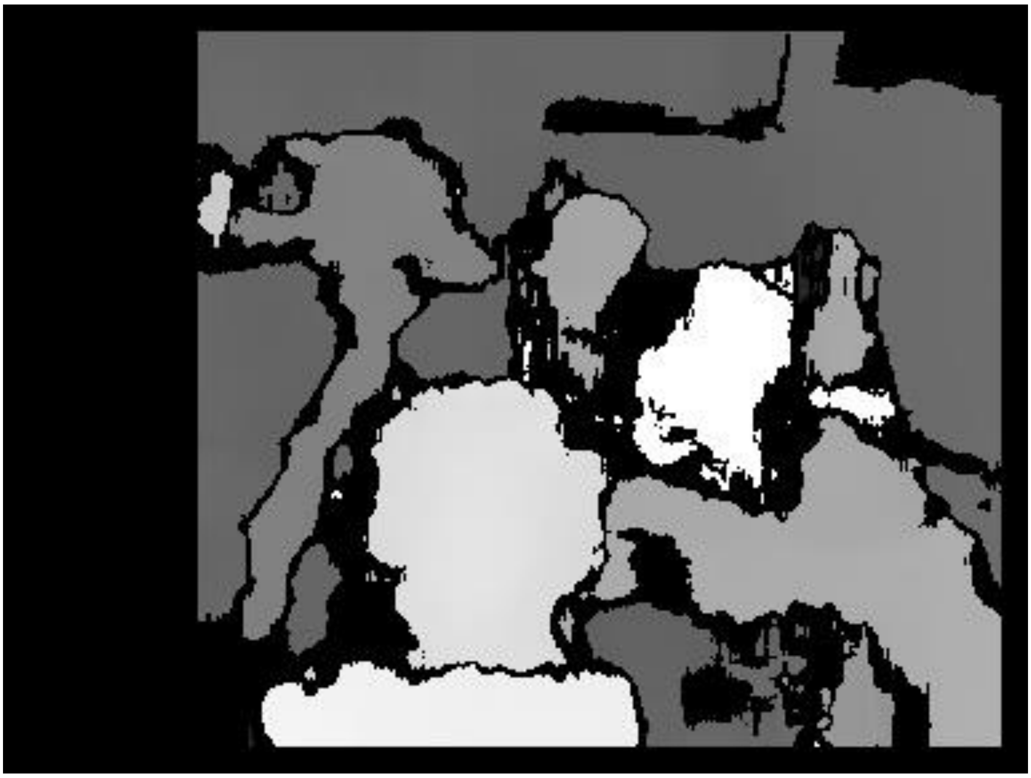

Fig.2.8 Disparity map image for tsucuba_left.png and tsucuba_right.png

Conclusion:

After implementation of this task, we gathered knowledge about stereo vision and how epipolarity plays an important role in understanding how images can be transformed to gather their features and depth information. Also, it tells us the importance of the fundamental matrix.

Source code for Task 2:

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov  2 03:46:01 2018

@author: Aniruddha Sinha
UB Person Number = 50289428
UBIT = asinha6@buffalo.edu
"""

#UBIT = '<asinha6>'; import numpy as np; np.random.seed(sum([ord(c) for c in
UBIT]))

import cv2
import numpy as np
import random
from matplotlib import pyplot as plt
import time

'function to write the image to disk'
def writeImage(img, imageName):
    cv2.imwrite("results/" + imageName + ".jpg", img)

''' The following functions for drawing the epipolar lines and epipolar points and
finding epilines
```

```
 have been referenced and partly been copied from
https://docs.opencv.org/3.2.0/da/de9/tutorial_py_epipolar_geometry.html'''

def drawEpipolarLines(image1, image2, epilines, epipts1, epipts2, num):
    h, w = image1.shape

    colors = [[100, 200, 300], [1,3,2], [32, 43, 12], [90, 200, 12], [100, 100,
100], [0, 100, 200], [100, 0, 200], [00, 200, 0], [90, 30, 180], [120, 140, 150]]

    for i, r, point1, point2 in zip(range(10), epilines, epipts1, epipts2):
        color = tuple(colors[i])
        x0, y0 = map(int, [0, -r[2]/r[1]] )
        x1, y1 = map(int, [w, -(r[2] + r[0]*w)/r[1]] )

        global img1
        global img2

        if num == 1:
            image1 = cv2.line(img1, (x0,y0), (x1,y1), color, 1)
            image1 = cv2.circle(img1, tuple(point1), 5, color, -1)
            image2 = cv2.circle(img2, tuple(point2), 5, color, -1)
        else:
            image1 = cv2.line(img2, (x0,y0), (x1,y1), color, 1)
            image1 = cv2.circle(img2, tuple(point1), 5, color, -1)
            image2 = cv2.circle(img1, tuple(point2), 5, color, -1)
    return image1

def findEpilines(left_img, right_img, pts1, pts2, F):

    #Find epilines corresponding to points in the right image and
    #draw their lines on left image
    lines_left = cv2.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2, F)
    lines_left = lines_left.reshape(-1,3)
    epiLeftImg = drawEpipolarLines(left_img, right_img, lines_left, pts1, pts2, 1)


    #Find epilines corresponding to points in the left image and
    #draw their lines on right image
    lines_right = cv2.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1, F)
    lines_right = lines_right.reshape(-1,3)
    epiRightImg = drawEpipolarLines(right_img,left_img, lines_right, pts2, pts1, 2)

    return epiRightImg, epiLeftImg


################### Start Main #######################
'Start function execution'
t= time.time()

'Read the two images, first in colour and then in grayscales'
img1 = cv2.imread("data/tsucuba_left.png")
img2 = cv2.imread("data/tsucuba_right.png")

tsucuba_left = cv2.imread("data/tsucuba_left.png", 0)
tsucuba_right = cv2.imread("data/tsucuba_right.png", 0)

############################ PART 1
################################################
print('\n\n#################### Starting execution of Task 2 part 1
#########################################')

''' The following code for computing SIFT features and drawing keypoints has been
referenced and partly copied from '''
''''https://docs.opencv.org/3.4.3/da/df5/tutorial_py_sift_intro.html'''''

'Compute the keypoints and features of the two images'
sift = cv2.xfeatures2d.SIFT_create()
keypoints1 = sift.detect(tsucuba_left, None)
```

```
keypoints2 = sift.detect(tsucuba_right, None)

'Draw the keypoints on the images and write to disk'
task2_sift1 = cv2.drawKeypoints(img1, keypoints1, None ,(255,0,0))
task2_sift2 = cv2.drawKeypoints(img2, keypoints2, None, (255,0,0))
writeImage(task2_sift1, "task2_sift1")
print('\n\n******************* Image task2_sift_1 successfully written to drive
*************************')
writeImage(task2_sift2, "task2_Sift2")
print('\n\n******************* Image task2_sift_2 successfully written to drive
*************************')


''' The following code for computing the K Nearest Neighbours and drawing matches
 has been referenced and party copied from -
 https://docs.opencv.org/3.0-
beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html '''


'Find the keypoints and the descriptors of the keypoints'
keypoint_left_img, descriptor_left_img = sift.detectAndCompute(tsucuba_left, None)
keypoint_rt_img, descriptor_rt_img = sift.detectAndCompute(tsucuba_right, None)

'Using FLANN matching technique for finding K Nearest Neighbours'
FLANN_INDEX_KDTREE = 0
check_index_params = dict(algorithm = FLANN_INDEX_KDTREE, tress = 5)
check_search_params = dict(checks=100)

'''Create object for FLANN matching'''
doFlann = cv2.FlannBasedMatcher(check_index_params, check_search_params)
NearestNeighborsMatches = doFlann.knnMatch(descriptor_left_img, descriptor_rt_img,
k=2)

'Calculate the better neighbour of the two best matches of a keypoint of left image
in the right image'
keepKeypoints = []
tsc_pts1 = []
tsc_pts2 = []
for m,n in NearestNeighborsMatches:
    if m.distance < 0.75*n.distance:
        keepKeypoints.append(m)
        tsc_pts2.append(keypoint_left_img[m.trainIdx].pt)
        tsc_pts1.append(keypoint_rt_img[m.queryIdx].pt)

'Draw the matched keypoints of the two images'
task2_matches_knn = cv2.drawMatches(img1, keypoint_left_img, img2, keypoint_rt_img,
keepKeypoints, None, (0,255,0), (255,0,0) ,flags=2)
writeImage(task2_matches_knn, "task2_matches_knn")
print('\n\n******************* Image task2_matches_knn successfully written to
drive *************************')
print('\n\n############## Task 2 part 1 successfully
completed.####################\nTime taken for Task 2 part 1 = ',time.time()-t,'
seconds')
############################# END PART 1
################################################


########################## PART 2
##################################################
print('\n\n#################### Starting execution of Task 2 part 2
#########################################')

''' The following code for finding Homography and computing the inliers has been
referenced and
party been copied from https://docs.opencv.org/3.0-
beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html
'''
```

```
MIN_NUM_MATCHES = 10

if len(keepKeypoints) > MIN_NUM_MATCHES:

    coord_left = np.array([ keypoint_left_img[m.queryIdx].pt for m in keepKeypoints
])
    np.reshape(coord_left, (-1,1,2))
    coord_left = np.int32(coord_left)
    coord_rt = np.array([ keypoint_rt_img[m.trainIdx].pt for m in keepKeypoints ])
    np.reshape(coord_rt, (-1,1,2))
    coord_rt = np.int32(coord_rt)

    'Find fundamental matrix and list of inliers after RANSAC algorithm'
    F, fundamentalStatus = cv2.findFundamentalMat(coord_left, coord_rt,
cv2.FM_RANSAC, 20.0)
    print('\n\nFundamental matrix is',F)
#    print(fundamentalStatus)#    print(fundamentalStatus)
print('\n\n############### Task 2 part 2 successfully
completed.####################\nTime taken for Task 2 part 2 = ',time.time()-t,'
seconds')
########################## END PART 2
####################################################


''''''''''''''''''''''''''''''''''''''' START PART 3
''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''''
'''''''''
print('\n\n#################### Starting execution of Task 2 part 3
#########################################')

'Select the inliers and convert the inliers to a 1D list'
# Select only inlier points
'convert the inliers to a 1D list'
# Select only inlier points
'Select 10 random inliers after shuffling the array of inliers'
inliers_1 = coord_left[fundamentalStatus.ravel()==1].tolist()
inliers_left = random.sample(inliers_1,10)
inliers_left = np.array(inliers_left)

inliers_2 = coord_rt[fundamentalStatus.ravel()==1].tolist()
inliers_right = random.sample(inliers_2,10)
inliers_right = np.array(inliers_right)

'Draw the epilines for 10 random inliers in both images'
task2_epi_right, task2_epi_left = findEpilines(tsucuba_left, tsucuba_right,
inliers_left, inliers_right, F)
#task2_epi_right, task2_epi_lt = findEpilines(tsucuba_right, tsucuba_left,
inliers_right, inliers_left, F)
writeImage(task2_epi_right, "task2_epi_right")
writeImage(task2_epi_left, "task2_epi_left")
print('\n\nImages task2_epi_right.jpg and task2_epi_left.jpg succesfully written to
disk.')
##
print('\n\n############### Task 2 part 3 successfully
completed.####################\nTime taken for Task 2 part 3 = ',time.time()-t,'
seconds')
################################# END PART 3 ####################################



########################## START PART 4 #########################################
print('\n\n#################### Starting execution of Task 2 part 4
#########################################')
imgleft = cv2.imread('data/tsucuba_left.png', 0)
imgright = cv2.imread('data/tsucuba_right.png', 0)
```

```
'Create the object of stereo BM for computing disparity'
task2_stereo = cv2.StereoBM_create()
task2_disparity = task2_stereo.compute(imgleft, imgright)
writeImage(task2_disparity, "task2_disparity")
plt.imshow(task2_disparity, 'gray')
plt.savefig('./results/task2_disparity.jpg')
print('\n\n******************** Image task2_disparity successfully written to drive
*************************')
print('\n\n############### Task 2 part 4 successfully
completed.#####################\nTime taken for Task 2 part 4 = ',time.time()-t,'
seconds')
```

<u>Task 3</u>. K-means Clustering

Aim

Through this task, we try to understand and implement the K-means algorithm for clustering any random unarranged dataset. The metric used in this project is the Euclidean distance as the distance function.

Process

We are provided with a random dataset and a set of three random centroids for that dataset. Now, we compute the distances of all samples in the dataset with the centroids and calculate the minimum of the Euclidean distances of each centroid for every point. Subsequently, the samples are clustered based on the centroid they are closest to. We repeat the same algorithm for each centroid and update it by taking the average of the new clustered datasets. That way, we go on clustering the entire dataset and will be left with all data points clustered properly. This is the K-Means algorithm.

Implementation

For this task, we are provided by a dataset, X of size 10x2 and three centroids, each of size 1x2. The plot of the initial dataset is shown in Fig.3.1

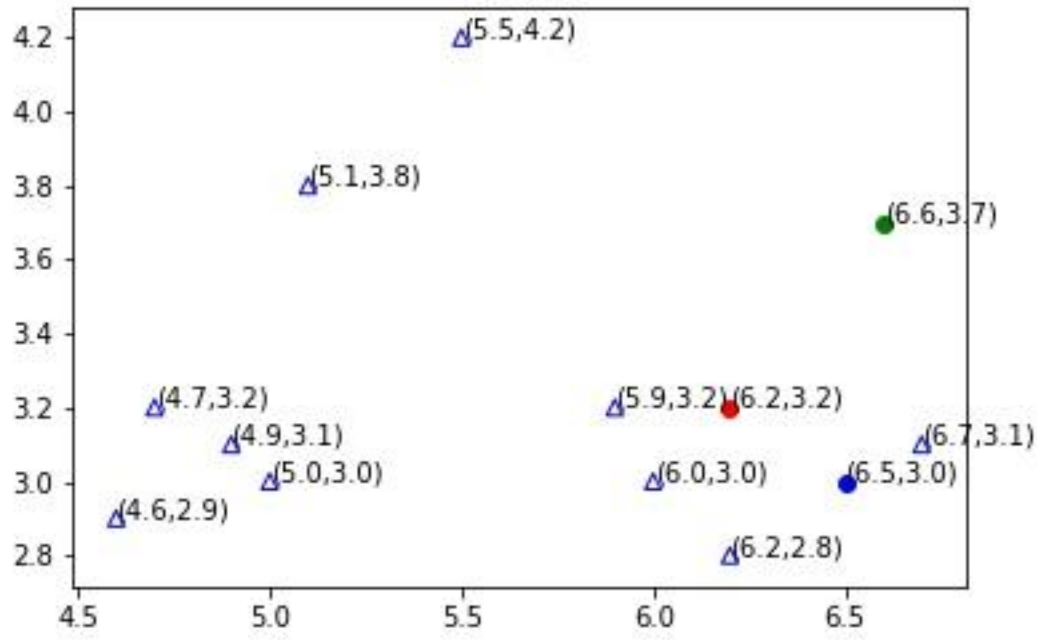

Fig.3.1. Original plot of the given data points

1) We classify the 10 samples according to the nearest centroid and plot the results by colorizing the empty triangle in red, blue or green, as shown in Fig.3.2. Also, we display the classification vector which tell which data point corresponds to which centroid. The classification vector is shown below-

```
Classification vector for 0th iteration:
['Mu1', 'Mu1', 'Mu3', 'Mu1', 'Mu2', 'Mu1', 'Mu1', 'Mu3', 'Mu1', 'Mu1']
```

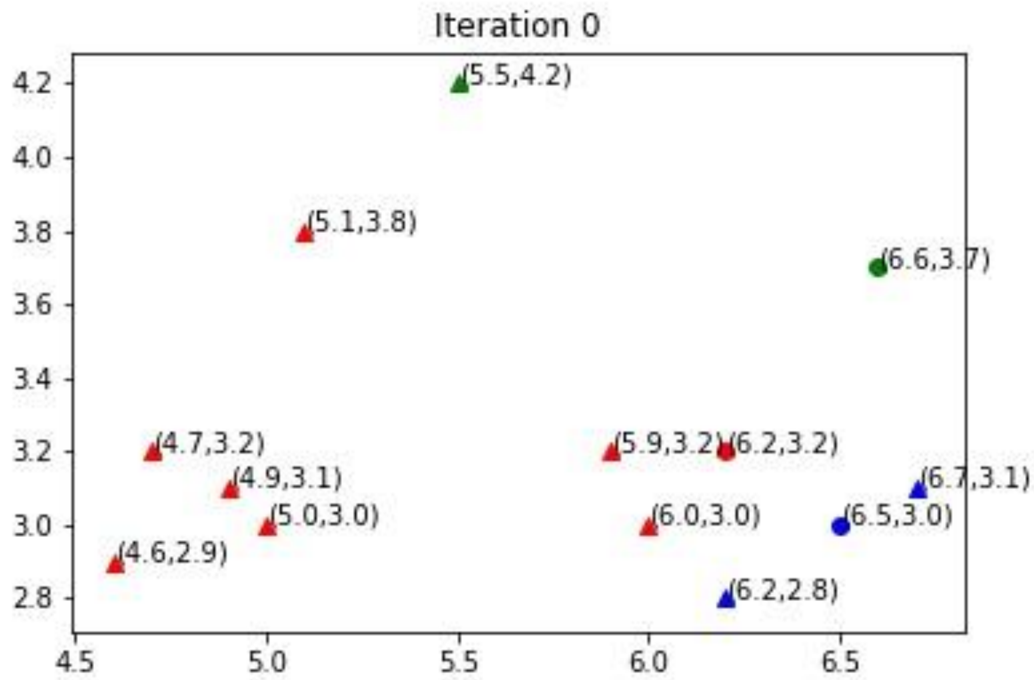

Fig.3.2 Classification plot based on the initial centroids

2) Next, we recompute all centroids based on the classification of all data points. The updated centroids are plotted, as shown in Fig.3.3 and their values are given below –

$$\boldsymbol{\mu_1} = (\mathbf{5.2, 3.2}),\ \boldsymbol{\mu_2} = (\mathbf{5.5, 4.2}),\ \boldsymbol{\mu_3} = (\mathbf{6.4, 3.0}),$$

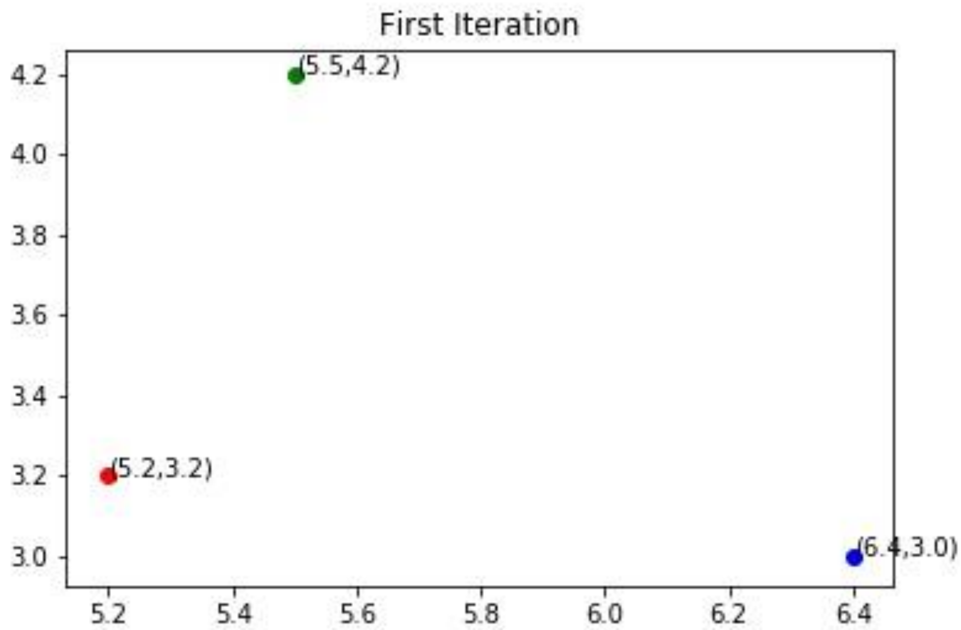

Fig.3.3 Plot of updated $\mu_i$after 1st iteration

3) Further, we classify the data points based on the updated centroids and plot the new classified data points based on these centroids as shown in Fig.3.4.

```
Classification vector for 0th iteration:
['Mu3', 'Mu1', 'Mu3', 'Mu1', 'Mu2', 'Mu1', 'Mu1', 'Mu3', 'Mu2', 'Mu3']
```

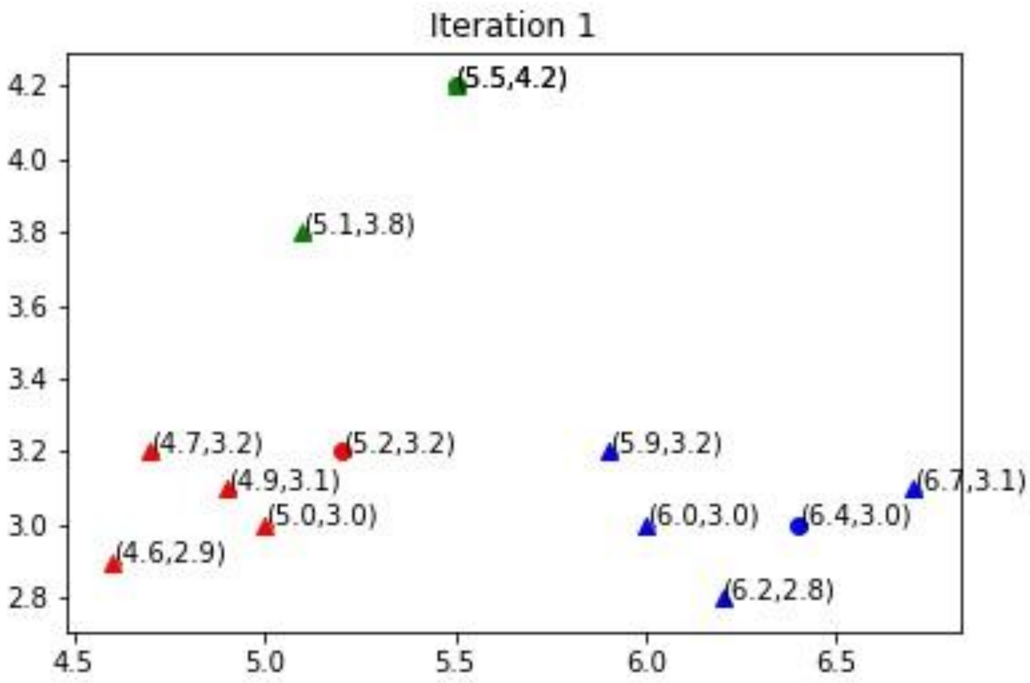

Fig.3.4. Classification plot for second iteration

4) The centroids are again updated and the new centroids after the second iteration are given below-

$$\boldsymbol{\mu_1} = (\mathbf{4.8}, \mathbf{3.0}),\ \boldsymbol{\mu_2} = (\mathbf{5.3}, \mathbf{4.0}),\ \boldsymbol{\mu_3} = (\mathbf{6.2}, \mathbf{3.0})$$

The plot of the updated centroids is shown in Fig.3.5.

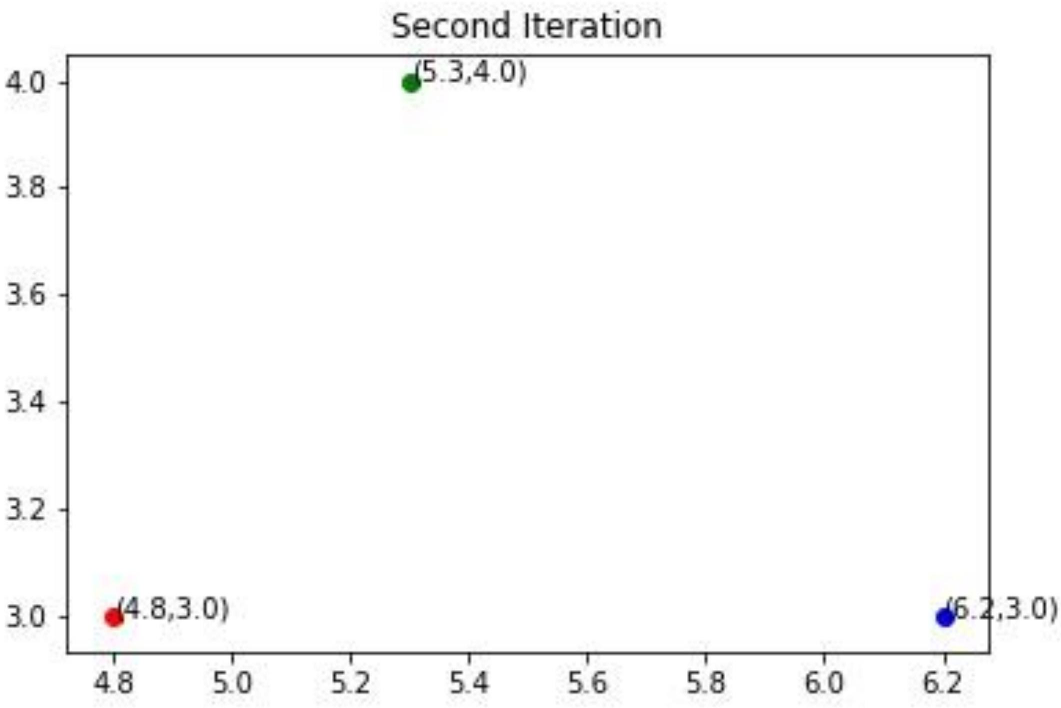

Fig.3.5 Updated μ_i plot for the second iteration

5) Finally, we have been provided with an image, called `baboon.jpg` shown in Fig.3.6 on which we applied K-Means clustering for image quantization. We have been provided with a set of clusters, namely k = [3, 5, 10, 20].

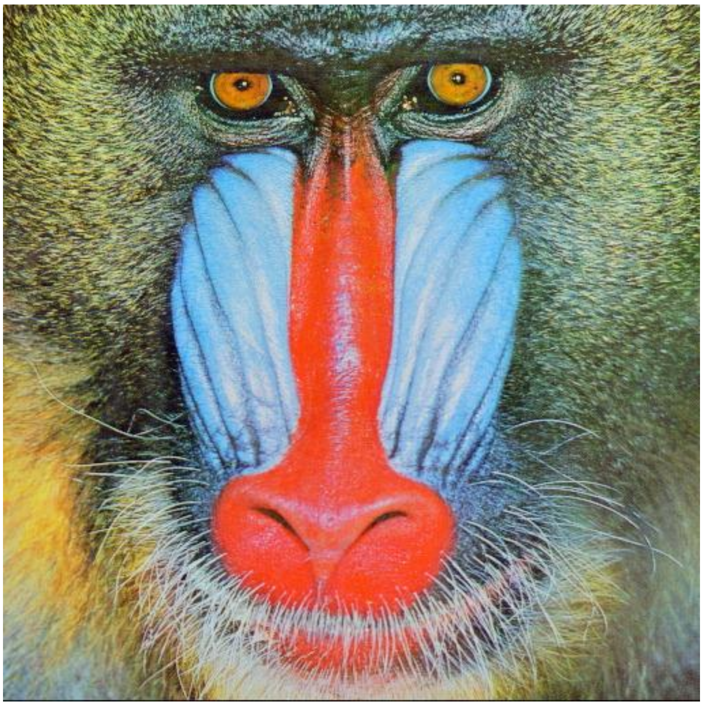

Fig.3.6 Input baboon.jpg image

6) The images are quantized for these k-values, i.e. the number of clusters and the program is run 20 time to achieve convergence of the centroids. The resulting images are shown in Fig. 3.7, Fig.3.8, Fig.3.9 and Fig.4.0 respectively.

Fig.3.7. K = 3

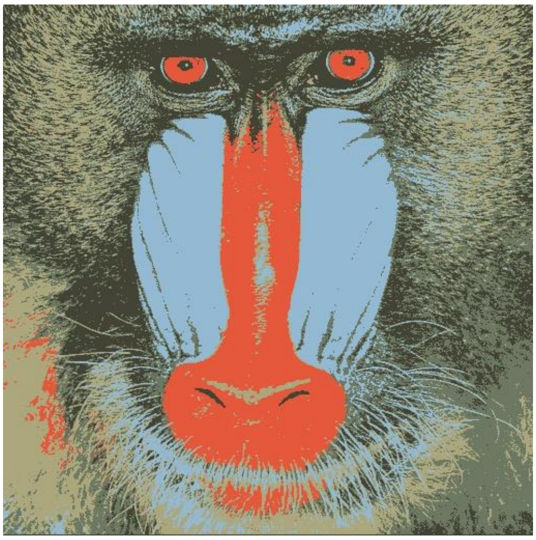

Fig.3.8. K =5

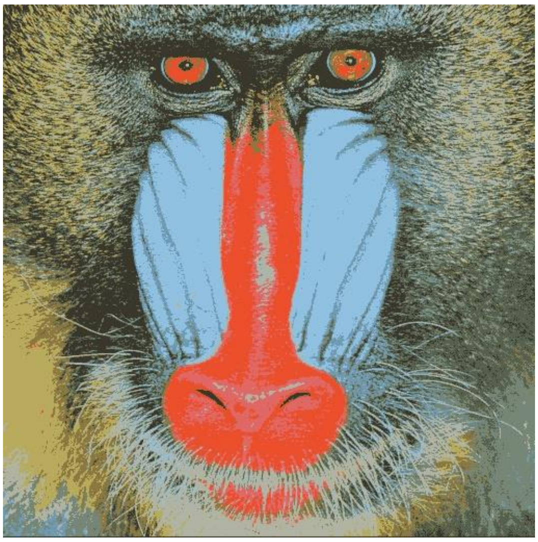

Fig.3.9. K = 10

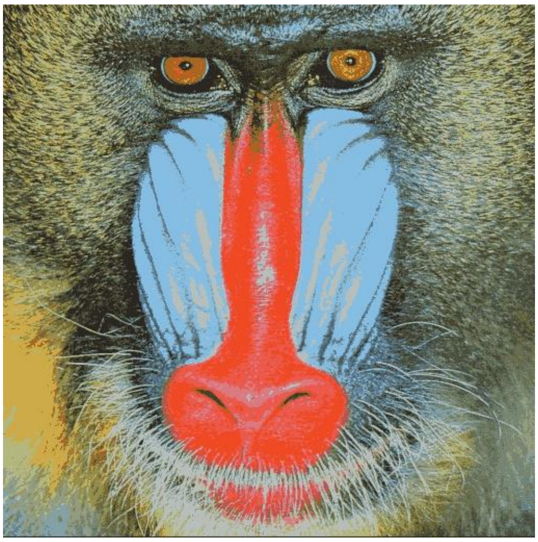

Fig.4.0. K = 20

BONUS

In this part, we implement a variation of K-Means clustering, the Gaussian mixture models (GMM) that calculates the distance of each data point from the given centroids based on the probability density function (Gaussian function) of the data point. We also bring to use the covariance in this case.

Implementation

5. (a)

1. The GMM is run on the given dataset X which is of dimension 10x2 matrix. We have three initial centroids given to us and three covariance matrices.
2. After the first iteration, the centroids are updated and are found as follows:

$$\boldsymbol{\mu_1 = (5.171, 3.171)},\ \boldsymbol{\mu_2 = (5.5, 4.2)},\ \boldsymbol{\mu_3 = (6.45, 2.95)}$$

5. (b)

1. We read the Old Faithful dataset from a CSV file and load the [x: eruptions and y: waiting] values in the form of a numpy array.

2. The initial covariances and centroids are given to us. We apply the algorithm of K-Means with GMM to update the centroids and covariances every time and plot the clustered points with a population shown using ellipse around them.

3. The program is ran for 5 iterations and the results are shown in Fig.B.1, Fig.B.2, Fig.B.3, Fig.B.4 and Fig.B.5.

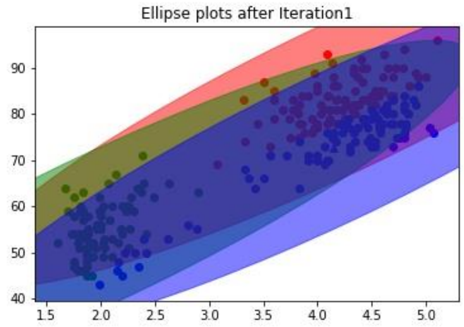

Fig.B.1 Clusters after 1st iteration

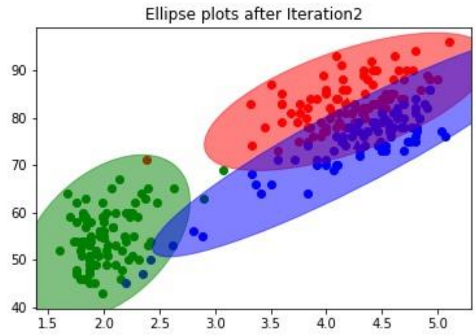

Fig.B.2 Clusters after 2nd iteration

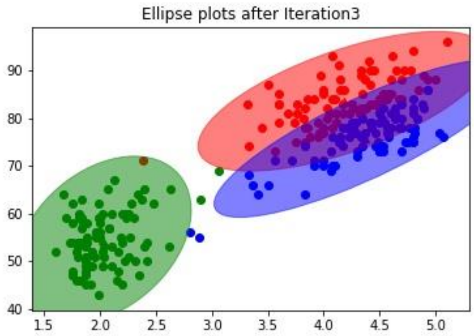

Fig.B.3 Clusters after 3rd iteration

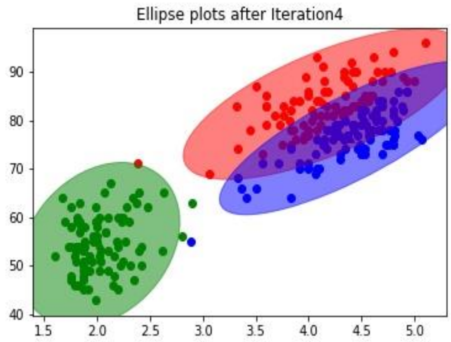

Fig.B.4 Clusters after 4th iteration

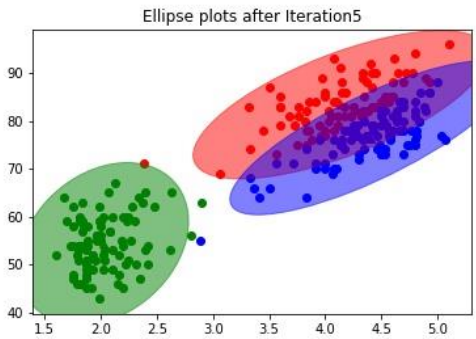

Fig.B.5 Clusters after 5th iteration

Conclusion:
Hence, after having computed K-Means clustering on various models, I can draw the following conclusions-

- K-Means helps in quickly clustering a large chunk of data.
- Updating the centroids is a good strategy for computing clusters.
- The centroids almost converge after 15-20 iterations, and K=5-10 is a good number of clusters for clustering the data.
- In image quantization, as we go increasing the size of K, the more refined quantized image is generated.
- GMM helps us to calculate the clusters based on their probability which gives us a good distribution about the data.

Source code for Task 3:

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  3 11:52:05 2018

@author: Aniruddha Sinha
UB Person Number = 50289428
UBIT = asinha6@buffalo.edu
"""

UBIT = '<asinha6>'; import numpy as np; np.random.seed(sum([ord(c) for c in UBIT]))

import cv2
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import multivariate_normal
import math
import time
import csv
from matplotlib.patches import Ellipse

def classifyPoints(X, Mu, Cov, pdf):
    C1 = []
    C2 = []
    C3 = []
    classification_vector = []

    pdf1 = []
    pdf2 = []
    pdf3 = []

    if pdf:
        for i in range(len(X)):
            L2_C1 = multivariate_normal.pdf(X[i], mean = Mu[0,:], cov=Cov)
            pdf1.append(L2_C1)

            L2_C2 = multivariate_normal.pdf(X[i], mean = Mu[1,:], cov=Cov)
            pdf2.append(L2_C2)

            L2_C3 = multivariate_normal.pdf(X[i], mean = Mu[2,:], cov=Cov)
            pdf3.append(L2_C3)

            if max(L2_C1,L2_C2,L2_C3) == L2_C1:
                C1.append(X[i])
                classification_vector.append("Mu1")
            elif max(L2_C1,L2_C2,L2_C3) == L2_C2:
                C2.append(X[i])
                classification_vector.append("Mu2")
            else:
#                print("L2_C3", L2_C3)
                C3.append(X[i])
```

```
                classification_vector.append("Mu3")
    else:
        for i in range(len(X)):
            L2_C1 = math.sqrt((X[i][0] - Mu[0][0])**2 + (X[i][1] - Mu[0][1])**2)

            L2_C2 = math.sqrt((X[i][0] - Mu[1][0])**2 + (X[i][1] - Mu[1][1])**2)

            L2_C3 = math.sqrt((X[i][0] - Mu[2][0])**2 + (X[i][1] - Mu[2][1])**2)

            if min(L2_C1,L2_C2,L2_C3) == L2_C1:
                C1.append([ X[i][0], X[i][1] ])
                classification_vector.append("Mu1")
            elif min(L2_C1,L2_C2,L2_C3) == L2_C2:
                C2.append([ X[i][0], X[i][1] ])
                classification_vector.append("Mu2")
            else:
                C3.append([ X[i][0], X[i][1] ])
                classification_vector.append("Mu3")

#    print('\n\nPDF for Mu1:', pdf1, '\n\nPDF for Mu2:', pdf2, '\n\nPDF for Mu3:',
pdf3)
    return C1, C2, C3, classification_vector


def plotWithText(Arr):
   for i in range(len(Arr)):
    s = '('+str(Arr[i][0])+','+str(Arr[i][1])+')'
    plt.text(Arr[i][0], Arr[i][1], s)

def plotMu(Mu):
    plt.scatter(Mu[0,0], Mu[0,1], c = 'r', marker="o", edgecolors=None,
facecolor='none')
    plt.scatter(Mu[1,0], Mu[1,1], c = 'g', marker="o", edgecolors=None, facecolor=
'none')
    plt.scatter(Mu[2,0], Mu[2,1], c = 'b', marker="o", edgecolors= None, facecolor=
'none')

def updateMu(classifiedPoints):

    xSum= 0
    ySum = 0
    for i in range(len(classifiedPoints)):
        xSum += classifiedPoints[i][0]
        ySum += classifiedPoints[i][1]

    xAvg = xSum/len(classifiedPoints)
    xAvg = round(xAvg,1)

    yAvg = ySum/len(classifiedPoints)
    yAvg = round(yAvg,1)

    return [xAvg,yAvg]

def plotClassified(C1,C2,C3,Mu, filename):

    np_C1 = np.array(C1)
    np_C2 = np.array(C2)
    np_C3 = np.array(C3)

    plt.figure()
    plotMu(Mu)
    plotWithText(Mu)

    if np_C1.shape[0]>0:
        plt.scatter(np_C1[:,0], np_C1[:,1], c='r', marker="^", edgecolors='r',
facecolor='none')
        plotWithText(np_C1)
```

```
    if np_C2.shape[0]>0:
        plt.scatter(np_C2[:,0], np_C2[:,1], c='g', marker="^", edgecolors='g',
facecolor='none')
        plotWithText(np_C2)

    if np_C3.shape[0]>0:
        plt.scatter(np_C3[:,0], np_C3[:,1], c='b', marker="^", edgecolors='b',
facecolor='none')
        plotWithText(np_C3)

    plt.savefig('./results/'+filename+'.jpg')

def part1(X, Mu):

    C1, C2, C3, classification_vector = classifyPoints(X, Mu, None, False)
    print('\n\nFirst cluster for Mu1: ', C1)
    print('\n\nSecond cluster for Mu2:', C2)
    print('\n\nThird cluster for Mu3:', C3)

    print('\n\nClassification Vector is:\n', classification_vector)

    plotClassified(C1,C2,C3,Mu,'task3_iter1_a')

def part2(X,Mu):

    C1, C2, C3, classification_vector = classifyPoints(X,Mu, None, False)

    updatedMu = []

    updatedMu.append(updateMu(C1))
    updatedMu.append(updateMu(C2))
    updatedMu.append(updateMu(C3))
    updatedMu = np.array(updatedMu)

    plt.figure()
    plt.scatter(X[:,0], X[:,1], marker="^", edgecolor = 'b', facecolor='None')
    plotWithText(X)
    plotMu(updatedMu)
    plotWithText(updatedMu)
    plt.savefig('./results/part3_iter1_b.jpg')

    print('\n\nUpdated Mu after iteration 1: ', updatedMu)

    return updatedMu

def part3(X,updatedMu):

    C1, C2, C3, classification_vector = classifyPoints(X,updatedMu, None, False)

    print('\n\nUpdated cluster for Mu1: ', C1)
    print('\n\nUpdated cluster for Mu2:', C2)
    print('\n\nUpdated cluster for Mu3:', C3)

    print('\n\nUpdated Classification Vector is:\n', classification_vector)

    plotClassified(C1,C2,C3,updatedMu, 'task3_iter2_a')

    updated2Mu = []

    updated2Mu.append(updateMu(C1))
    updated2Mu.append(updateMu(C2))
    updated2Mu.append(updateMu(C3))
    updated2Mu = np.array(updated2Mu)

    plt.figure()
    plt.scatter(X[:,0], X[:,1], marker="^", edgecolor = 'b', facecolor='None')
    plotWithText(X)
    plotMu(updated2Mu)
```

```
    plotWithText(updated2Mu)
    plt.savefig('./results/task3_iter2_b.jpg')

    print('\n\nUpdated Mu after iteration 2: ', updated2Mu)

def updateMuBaboon(classifiedPoints):

    xSum= 0
    ySum = 0
    zSum = 0
    for i in classifiedPoints:
        xSum += i[0]
        ySum += i[1]
        zSum += i[2]

    xAvg = xSum/len(classifiedPoints)
    xAvg = round(xAvg)

    yAvg = ySum/len(classifiedPoints)
    yAvg = round(yAvg)

    zAvg = zSum/len(classifiedPoints)
    zAvg = round(zAvg)

    return xAvg,yAvg,zAvg

def part4(baboon):

    h,w,c = baboon.shape
    reshapedBaboon = baboon.reshape(-1,3)

    Mu = [3,5,10,20]

    for m in Mu:
        Mu_baboon = reshapedBaboon[5:m+5]
        muLen = Mu_baboon.shape[0]
        muLen2 = Mu_baboon.shape[1]

        muPixels = []

        for K in range(10):

            classification_vector = []
            C_matrix = []

            for num_clust in range(m):
                C_matrix.append([])

            MuList = []
            for pixel in range(reshapedBaboon.shape[0]):

                temp = np.zeros((muLen))

                for i in range(muLen):
                    tempMu = 0
                    for j in range(muLen2):
                        tempMu += (reshapedBaboon[pixel][j] - Mu_baboon[i][j])**2
                    temp[i] = math.sqrt(tempMu)

                temp = temp.tolist()
                muIndex = temp.index(min(temp))
                MuList.append(muIndex)

                C_matrix[muIndex].append(reshapedBaboon[pixel])

                classification_vector.append(Mu_baboon[muIndex])

            muPixels = []
```

```
            for muInd in range(muLen):
                pixelsofThisMu = [i for i, e in enumerate(MuList) if e == muInd]
                muPixels.append(reshapedBaboon[pixelsofThisMu])

            Mu_baboon = []
            for i in range(len(muPixels)):
                Mu_baboon.append(updateMuBaboon(muPixels[i]))

            Mu_baboon = np.array(Mu_baboon)
#            print(Mu_baboon, Mu_baboon.shape)

            classification_vector = np.array(classification_vector)
            output = classification_vector.reshape(h,w,c)
        global t
        cv2.imwrite("results/task3_baboon_"+str(m)+".jpg",output)
        print('\n\n**************** task3_baboon'+str(m)+'.jpg written to 
file.**************************')
        print('\n\nTime taken for K='+str(m)+' is ', time.time()-t)


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''
'''''''''''''''''''' FUNCTIONS FOR BONUS START 
''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''''''''''
def getFaithfulData(filePath):
    t = []
    count = 0
    with open(filePath, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            if count == 0:
                count = 1
                continue
            t.append(np.float32(row))
    return t

def plotClassifiedforBonus(C1,C2,C3,Mu):

    plotMu(Mu)
    if C1.shape[0]>0:
        plt.scatter(C1[:,0], C1[:,1], c='r', marker="o", edgecolors='r', 
facecolor='none')

    if C2.shape[0]>0:
        plt.scatter(C2[:,0], C2[:,1], c='g', marker="o", edgecolors='g', 
facecolor='none')

    if C3.shape[0]>0:
        plt.scatter(C3[:,0], C3[:,1], c='b', marker="o", edgecolors='b', 
facecolor='none')


"""

The functions plotPointsEllipse have been copied from 
https://github.com/joferkington/oost_paper_code/blob/master/error_ellipse.py


"""
def plotPointsEllipse(cov, pos, nstd=2, ax=None, **kwargs):
    """
    Plots an `nstd` sigma error ellipse based on the specified covariance
    matrix (`cov`). Additional keyword arguments are passed on to the 
    ellipse patch artist.
    Parameters
    ----------
```

```
        cov : The 2x2 covariance matrix to base the ellipse on
        pos : The location of the center of the ellipse. Expects a 2-element
            sequence of [x0, y0].
        nstd : The radius of the ellipse in numbers of standard deviations.
            Defaults to 2 standard deviations.
        ax : The axis that the ellipse will be plotted on. Defaults to the
            current axis.
        Additional keyword arguments are pass on to the ellipse patch.
    Returns
    -------
        A matplotlib ellipse artist
    """
    def eigsorted(cov):
        vals, vecs = np.linalg.eigh(cov)
        order = vals.argsort()[::-1]
        return vals[order], vecs[:,order]

    if ax is None:
        ax = plt.gca()

    vals, vecs = eigsorted(cov)
    theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))

    # Width and height are "full" widths, not radius
    width, height = 2.5 * nstd * np.sqrt(vals)
    ellip = Ellipse(xy=pos, width=width, height=height, angle=theta, **kwargs)

    ax.add_artist(ellip)
    return ellip


def part5_bonus():
    X = np.array([[5.9,3.2], [4.6,2.9], [6.2,2.8], [4.7,3.2], [5.5,4.2], [5.0,3.0],
[4.9,3.1], [6.7,3.1], [5.1,3.8], [6.0,3.0]])
    Mu = np.array([[6.2,3.2], [6.6,3.7], [6.5,3.0]])
    Cov = [[0.5,0], [0,0.5]]

    C1, C2, C3, classification_vector = classifyPoints(X,Mu, Cov, True)

    updatedMu = []

    updatedMu.append(updateMu(C1))
    updatedMu.append(updateMu(C2))
    updatedMu.append(updateMu(C3))
    updatedMu = np.array(updatedMu)

    print('\nUpdated Mu after 1st iteration on given dataset is:', updatedMu)

    faithfulData = np.array(getFaithfulData('data/faithful.csv'))
    print('\n\n',faithfulData.shape, type(faithfulData), faithfulData[1,:])

    computeFaithful =  faithfulData[:,1:]
    Mu_faithful = np.array([[4.0, 81], [2.0, 57], [4.0,71]])
    Cov_f = np.array([[1.30, 13.98], [13.98, 184.82]])

    print('\n\n################# Starting computation on the faithful data:
######################\n\n')
    for i in range(5):
        f_C1, f_C2, f_C3, classification_faith = classifyPoints(computeFaithful,
Mu_faithful, Cov_f, True)
        Mu_faithful = []

        Mu_faithful.append(updateMu(f_C1))
        Mu_faithful.append(updateMu(f_C2))
        Mu_faithful.append(updateMu(f_C3))
        Mu_faithful = np.array(Mu_faithful)

        print('\n\nUpdated Mu after iteration '+str(i+1)+' is:', Mu_faithful)
```

```
        f_C1 = np.array(f_C1)
        f_C2 = np.array(f_C2)
        f_C3 = np.array(f_C3)

        if i==0:
            f_c1 = f_C1.mean(axis=0)
            cov_f1 = Cov_f

            f_c2 = f_C2.mean(axis=0)
            cov_f2 = Cov_f

            f_c3 = f_C3.mean(axis=0)
            cov_f3 = Cov_f
        else:
            f_c1 = f_C1.mean(axis=0)
            cov_f1 = np.cov(f_C1, rowvar=False)

            f_c2 = f_C2.mean(axis=0)
            cov_f2 = np.cov(f_C2, rowvar=False)

            f_c3 = f_C3.mean(axis=0)
            cov_f3 = np.cov(f_C3, rowvar=False)

        plt.figure()
        plt.title('Ellipse plots after Iteration'+str(i+1))
        plotClassifiedforBonus(f_C1, f_C2, f_C3,Mu_faithful)
        plotPointsEllipse(cov_f1, f_c1, nstd=3, alpha=0.5, color='red')
        plotPointsEllipse(cov_f2, f_c2, nstd=3, alpha=0.5, color='green')
        plotPointsEllipse(cov_f3, f_c3, nstd=3, alpha=0.5, color='blue')
        plt.savefig('./results/task3_gmm_iter'+str(i+1)+'.jpg')
        plt.show()

def kmeans_implement():

    X = np.array([[5.9,3.2], [4.6,2.9], [6.2,2.8], [4.7,3.2], [5.5,4.2], [5.0,3.0],
[4.9,3.1], [6.7,3.1], [5.1,3.8], [6.0,3.0]])
    Mu = np.array([[6.2,3.2], [6.6,3.7], [6.5,3.0]])

    plt.figure()
    plt.scatter(X[:,0], X[:,1], marker="^", edgecolor = 'b', facecolor='None')
    plotWithText(X)
    plotMu(Mu)
    plotWithText(Mu)
    plt.savefig('./results/task3_original.jpg')

    global t
    ############################ START PART 1 ####################################
    print('\n\n##################### Starting execution of Task 3 part 1
##########################################')
    part1(X,Mu)
    print('\n\n############### Task 3 part 1 successfully
completed.#####################\nTime taken for Task 3 part 1 = ',time.time()-t,'
seconds')

    ############################ END PART 1 #######################################

    ############################ START PART 2 #####################################
    print('\n\n##################### Starting execution of Task 3 part 2
##########################################')
    updatedMu = part2(X,Mu)
    print('\n\n############### Task 3 part 2 successfully
completed.#####################\nTime taken for Task 3 part 2 = ',time.time()-t,'
seconds')

    ####################### END PART 2 ############################################

    ####################### START PART 3 ##########################################
```

```
    print('\n\n###################### Starting execution of Task 3 part 3
##########################################')
    part3(X, updatedMu)
    print('\n\n################ Task 3 part 3 successfully
completed.#####################\nTime taken for Task 3 part 3 = ',time.time()-t,'
seconds')
    ###################### END PART 3 ############################################

    ########################## START PART4 #######################################
    print('\n\n###################### Starting execution of Task 3 part 4
##########################################')
    baboon = cv2.imread('data/baboon.jpg')
    part4(baboon)
    print('\n\n################ Task 3 part 4 successfully
completed.#####################\nTime taken for Task 3 part 4 = ',time.time()-t,'
seconds')

    ###################### START BONUS: PART 5
#########################################
    print('\n\n###################Starting execution of Task 3: BONUS
###########################')
    print('\n\n************ Task 3: Bonus , Part 1, started execution:
*********************** ')
    part5_bonus()
    print('\n\n################ Task 3 part 5 - BONUS successfully
completed.#####################\nTime taken for Task 3 BONUS:part 5 =
',time.time()-t,' seconds')


if __name__=='__main__':
    try:
        global t
        t = time.time()
        kmeans_implement()
    except:
        pass
```

References:

- https://docs.opencv.org/3.3.1/d9/d0c/group__calib3d.html#ga8e25cb8c64d8baa4749ca28ff1b0866a

- https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

- http://answers.opencv.org/question/182587/how-to-draw-epipolar-line/

- https://docs.opencv.org/3.1.0/da/de9/tutorial_py_epipolar_geometry.html

- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html

- https://docs.opencv.org/2.4.1/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#stereobm-stereobm

- https://stackoverflow.com/questions/13063201/how-to-show-the-whole-image-when-using-opencv-warpperspective/20355545#20355545

- https://lmcaraig.com/color-quantization-using-k-means/

- https://en.wikipedia.org/wiki/K-means_clustering

- https://www.datascience.com/blog/k-means-clustering

- https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1