CSE -573 Intro to Computer Vision and Image Processing
Project 2

Manish Reddy Challamala
Department of Computer Science
University at buffalo
UBIT Name: manishre
Person Number : 50289714
manishre@buffalo.edu

November 5, 2018

1 Task1 - Image Features and Homography

Abstract

The goal of this task is stated point wise below:

1. To find the SIFT feature between two images and plot the key points in the images.
2. Compute all the k-nearest neighbours according to the given condition stated in the task and draw all the matches.
3. Compute the Homo graphic Matrix using RANSAC.
4. Plot 10 random matches using only inliners.
5. wrap the image 2 with image 1 using computed homography matrix.

1.1 Experimental set-up:

1. For this task, we are using the following libraries:
   1. Cv2, numpy, matplotlib
2. Firstly, calculating the keypoints and descriptors of the given two images mountain1.jpg ans mountain2.jpg and plotting the respective key-points on the image.
3. For the task 1.2 calculating the matches using the FlannBasedMatcher function by sending the calculated keypoints of both images as input.
4. Plotting all the macthes found between two images by using drawMatchesKnn built in fucntion ofcv2 library.
5. Computing the homography matrix and displaying the results on the screen.

6. selecting 10 random macthes using inliners and plotting them along the images.
7. Finally, wrapping the image by first transforming the prespective of the image2 and and aligning it according to the image 1 and merging the both images together by decreasing the distance between them.

1.2 Code:

```
import cv2
import random
UBIT ='manishre'
import numpy as np
np.random.seed(sum([ord(c) for c in UBIT]))
from matplotlib import pyplot as plt
import matplotlib.image as mpimg

Reading the Images
image1 = cv2.imread('mountain1.jpg')
image2 = cv2.imread('mountain2.jpg')

# SIFT OBJECT CREATION
sift = cv2.xfeatures2d.SIFT_create()
# CALUCLATING the keypoints and descriptors
detected_points1, descriptors1 = sift.detectAndCompute(image1,None)
detected_points2, descriptors2 = sift.detectAndCompute(image2,None)
# displaying the keypoints of the images
detect_image1 =cv2.drawKeypoints(image1,detected_points1,None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
detect_image2 = cv2.drawKeypoints(image2,detected_points2,None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

#====================================part2=========================================
# Feature Matching
Flann_index = 0
index_parameters = dict(algorithm = Flann_index, trees = 5)
search_parameters = dict(checks=50)   # or pass empty dictionary
flann = cv2.FlannBasedMatcher(index_parameters,search_parameters)
matches = flann.knnMatch(descriptors1,descriptors2,k=2)
matchesMask = [[0,0] for i in range(len(matches))]
#Apply ratio test
good = []
good_pts =[]
for i,(m,n) in enumerate(matches):
    if m.distance < 0.75*n.distance:
        good.append([m])
        good_pts.append(m)
featureMacthing = cv2.drawMatchesKnn(image1,detected_points1,image2,
detected_points2,good,None,flags=2)
```

```
#========================================part3==========================================
src_pts = np.float32([ detected_points1[m.queryIdx].pt for m in good_pts ])
.reshape(-1,1,2)
dst_pts = np.float32([ detected_points2[m.trainIdx].pt for m in good_pts ])
.reshape(-1,1,2)
print(src_pts.shape)
print(dst_pts.shape)
homography, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC)

#======================================part4============================================
inliner =[]
for i in range(len(mask)):
    if(mask[i]==1):
        inliner.append(good[i])

__10match = []
__10match = random.sample(inliner,10)
__10inlinerMatch = cv2.drawMatchesKnn(image1,detected_points1,image2,
detected_points2,__10match,None,(255,0,0),flags=2)

#=====================================part5============================================
row1 = image1.shape[0]
row2 = image2.shape[0]
col1 = image1.shape[1]
col2 = image2.shape[1]
points_1 = np.float32([[0,0], [0,row1], [col1, row1], [col1,0]]).reshape(-1,1,2)
temp = np.float32([[0,0], [0,row2], [col2, row2], [col2,0]]).reshape(-1,1,2)
points_2 = cv2.perspectiveTransform(temp, homography)
points = np.concatenate((points_1, points_2), axis=0)
[x_min, y_min] = np.int32(points.min(axis=0).ravel() - 0.5)
[x_max, y_max] = np.int32(points.max(axis=0).ravel() + 0.5)
translation_dist = [-x_min, -y_min]
H_translation = np.array([[1, 0, translation_dist[0]],
[0, 1, translation_dist[1]], [0,0,1]])
output_img = cv2.warpPerspective(image1, H_translation.dot(homography),
(x_max - x_min, y_max - y_min))
output_img[translation_dist[1]:row1+translation_dist[1],
translation_dist[0]:col1+translation_dist[0]] = image2
wrappedImage = cv2.warpPerspective(image1, homography, (image2.shape[1],image2.shape[0]))

print("======HOMOGRAPHY MATRIX=====")
print(homography)
cv2.imwrite('task1_sift1.jpg',detect_image1)
cv2.imwrite('task1_sift2.jpg',detect_image2)
cv2.imwrite('task1_matches_knn.jpg',featureMacthing)
cv2.imwrite('task1_matches.jpg',__10inlinerMatch)
cv2.imwrite('task1_pano.jpg',output_img)
```

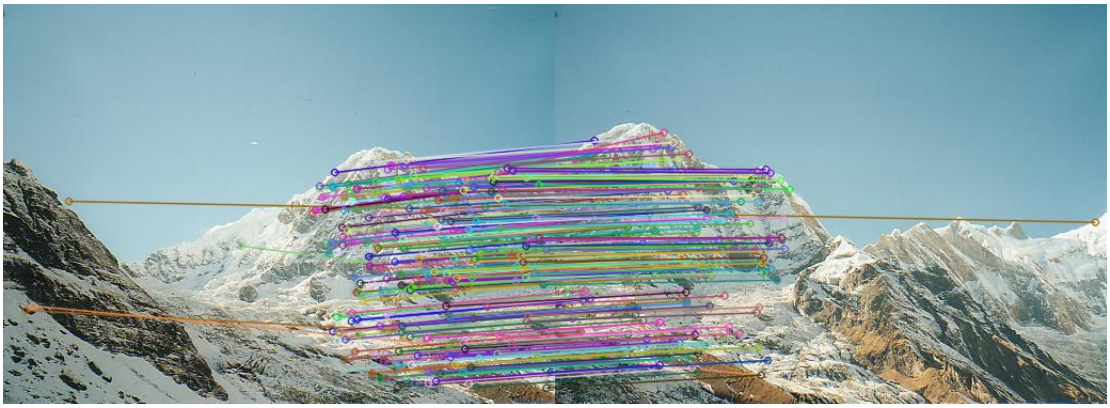
1.3 Result:

```
======HOMOGRAPHY MATRIX=====
[[ 1.58840616e+00 -2.91461287e-01 -3.95621049e+02]
 [ 4.45312759e-01  1.43782325e+00 -1.90624961e+02]
 [ 1.19636606e-03 -3.75347069e-05  1.00000000e+00]]
```

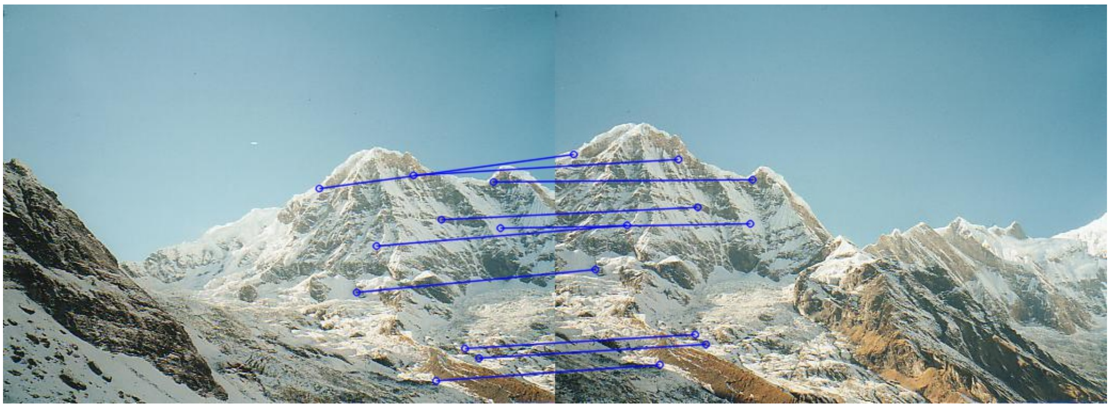
Key point detection: Images task1sift1.jpg and task1sift2.jpg

$fig[1]: \ Sift\ key\ detection\ of\ image\ mountain1$

$fig[2]:\ Sift\ key\ detection\ of\ image\ mountain2$



$fig[3]:\ All\ key-point\ matching$



$fig[4]:\ 10\ random\ matches\ using\ inliners$

$fig[1]:\ Wrapped\ image\ of\ image1\ and\ image2\ [panorama\ effect]$

2 Task 2:

Abstract

The goal of this task is stated point wise below:

1. To find the SIFT feature between two images and plot the key points in the images ans also Compute all the k-nearest neighbours according to the given condition stated in the task and draw all the matches.
2. Compute the Fundamental Matrix using RANSAC.
3. Plot 10 random matches using only inliners. and for each keypoint in right compute and draw the epiline on the left image. and vice-versa
4. Compute the disparity map for both left and right images.

2.1 Experimental set-up:

1. For this task, we are using the following libraries:
   1. Cv2, numpy, matplotlib
2. Firstly, calculating the keypoints and descriptors of the given two images tsucubaleft.png ans tsucubaleft.png and plotting the respective key-points on the image.
3. For the task 2.1 calculating the matches using the FlannBasedMatcher function by sending the calculated keypoints of both images as input.
4. Plotting all the macthes found between two images by using drawMatchesKnn built in fucntion ofcv2 library.
5. Computing the Fundamental Matrix and displaying the results on the screen.
6. selecting 10 random macthes using inliners and for each keypoint compute the epiline and plot the line with respect o the image.
7. Compute the disparity map:
   1.Create stereoBMcreate object with numDisparites =64 and block size =21 has parameters.
   2. use this object and compute the disparity map for two given images

2.2 Code:

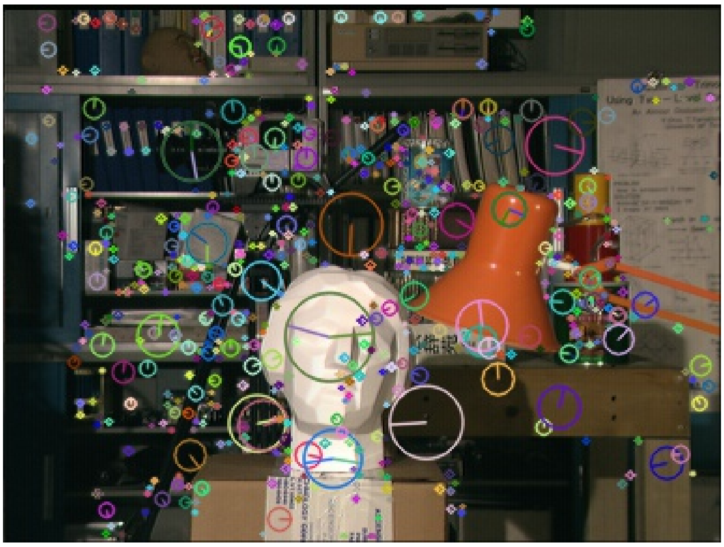
```
import numpy as np
import cv2
import random
from matplotlib import pyplot as plt
UBIT ='manishre'
np.random.seed(sum([ord(c) for c in UBIT]))

def epipolarLines(left_image,right_image,lines,src_pts,des_pts):
    r,c = left_image.shape[:2]

    for r,pt1,pt2 in zip(lines,src_pts,des_pts):
        # picking random color for each line
        color = tuple(np.random.randint(0,255,3).tolist())
        x0,y0 = map(int, [0, -r[2]/r[1] ]) #
        x1,y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
        left_image = cv2.line(left_image, (x0,y0), (x1,y1), color,1)
        left_image = cv2.circle(left_image,tuple(pt1),5,color,-1)
        right_image = cv2.circle(right_image,tuple(pt2),5,color,-1)

    return left_image,right_image

image1 = cv2.imread('tsucuba_left.png')
image2 = cv2.imread('tsucuba_right.png')
# System is not supporting the cv2.colorB2G function so reading the files again
image3 = cv2.imread('tsucuba_left.png',0)
image4 =cv2. imread('tsucuba_right.png',0)

sift = cv2.xfeatures2d.SIFT_create()
# find the keypoints and descriptors with SIFT
detected_points1, descriptors1 = sift.detectAndCompute(image1.copy(),None)
detected_points2, descriptors2 = sift.detectAndCompute(image2.copy(),None)
detect_image1 =cv2.drawKeypoints(image1,detected_points1,None,flags=
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
detect_image2 = cv2.drawKeypoints(image2,detected_points2,None,flags=
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(descriptors1,descriptors2,k=2)
good = []
good_pts =[]
pts1 = []
pts2 = []
# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.75*n.distance:
        good.append(m)
```

```
        good_pts.append([m])
        pts2.append(detected_points2[m.trainIdx].pt)
        pts1.append(detected_points1[m.queryIdx].pt)
featureMacthing = cv2.drawMatchesKnn(image1.copy(),detected_points1,image2.copy(),
detected_points2,good_pts,None,flags=2)

#======================================PART2=========================================
pts1 = np.int32(pts1)
pts2 = np.int32(pts2)

fundamentalMatrix, mask = cv2.findFundamentalMat(pts1,pts2,cv2.RANSAC)
print('===FUNDAMENTAL MATRIX===')
print(fundamentalMatrix)
# We select only inlier points
pts1 = pts1[mask.ravel()==1]
pts2 = pts2[mask.ravel()==1]
print(len(pts1))
key=[]
for i in range(10):
     key.append(random.randint(1,100))
print('key',key)
pts3 =[]
pts4 =[]
for i in key:
    pts3.append(pts1[i])
    pts4.append(pts2[i])

pts3 =np.asarray(pts3)
pts4 =np.asarray(pts4)

#============================================Part3==========================================
inliners_left = cv2.computeCorrespondEpilines(pts3.reshape(-1,1,2), 2,fundamentalMatrix)
inliners_left = inliners_left.reshape(-1,3)
image_l2r,image_l2rp = epipolarLines(image1.copy(),image2.copy(),inliners_left,pts3,pts4)

inliners_right =cv2.computeCorrespondEpilines(pts4.reshape(-1,1,2),2,fundamentalMatrix)
inliners_right = inliners_right.reshape(-1,3)
image_r2l,image_r2lp = epipolarLines(image2.copy(),image1.copy(),inliners_right,pts3,pts4)

#==============================================part4========================================
depthMap = cv2.StereoBM_create(numDisparities=64, blockSize=21)
#depthMap = cv2.createStereoBM(numDisparities=64, blockSize=21)
__disparityMap = depthMap.compute(image3,image4)

plt.imshow(__disparityMap,'gray')
plt.show()
cv2.imwrite('task2_sift1.jpg.jpg',detect_image1)
cv2.imwrite('task2_sift2.jpg',detect_image2)
cv2.imwrite('task2_matches_knn.jpg',featureMacthing)
cv2.imwrite('task2_epi_right.jpg',image_l2r)
cv2.imwrite('task2_epi_left.jpg',image_r2l)
cv2.imwrite('task2_disparity.jpg',__disparityMap)
```

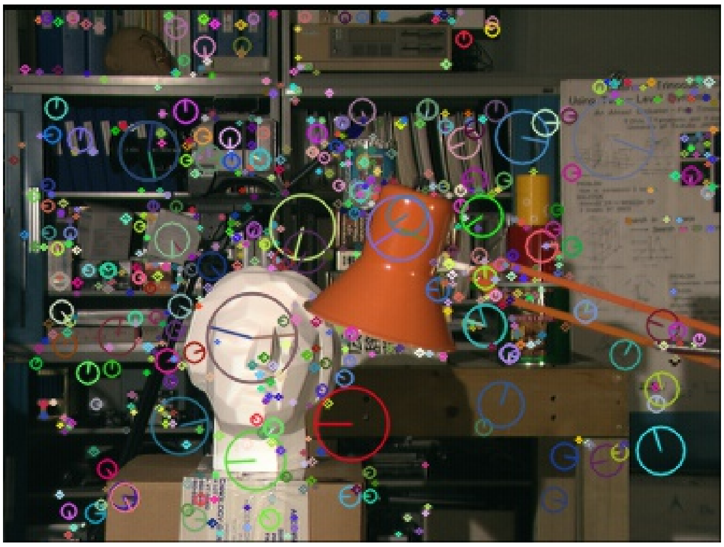
2.3 Result:

```
===FUNDAMENTAL MATRIX===
[[-2.12607354e-06 -8.10713687e-05  7.47530309e-02]
 [ 4.60726414e-05  3.79326900e-05  1.32728554e+00]
 [-7.52042326e-02 -1.32608913e+00  1.00000000e+00]]
```

Key point detection: Images task2sift1.jpg and task2sift2.jpg

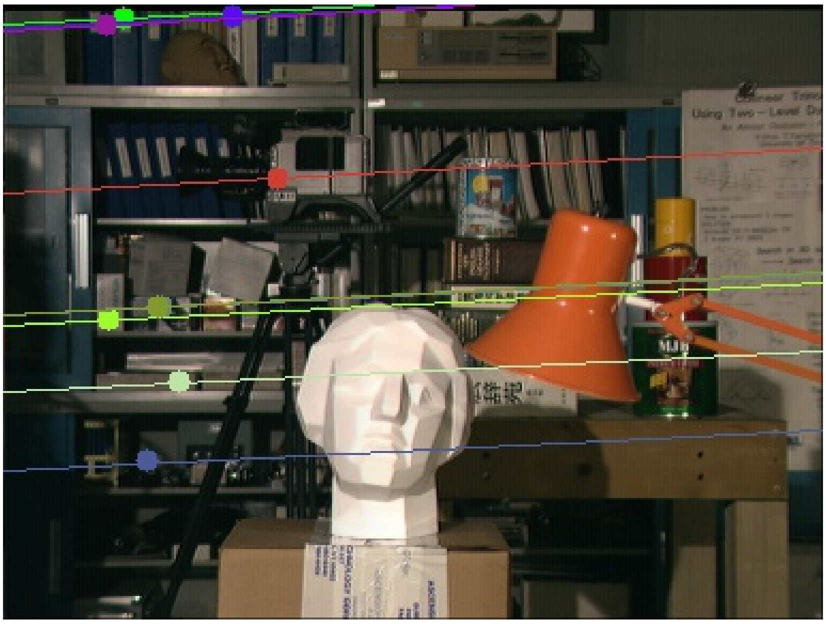


$fig[1]:\ Sift\ key\ detection\ of\ image\ mountain1$

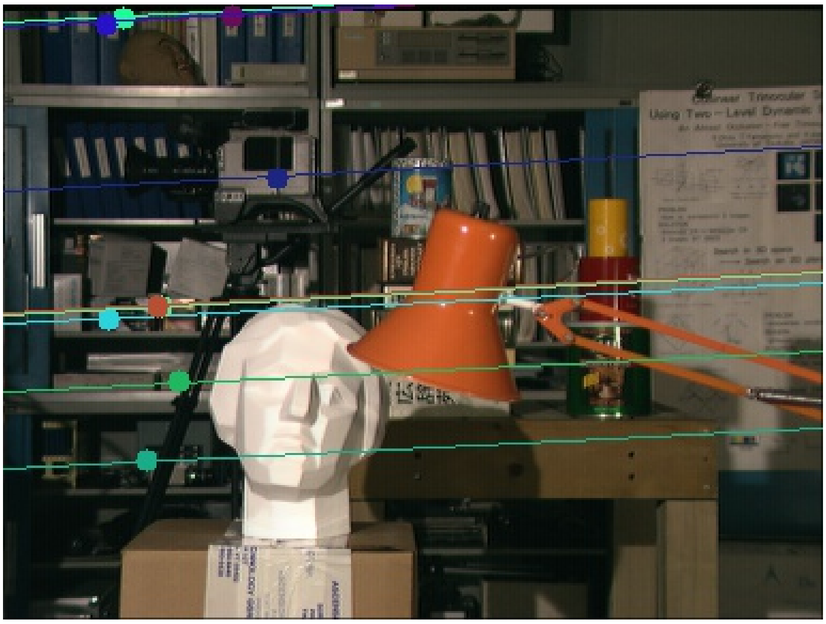


$fig[2]:\ Sift\ key\ detection\ of\ image\ mountain2$

$fig[3]:\ All\ key-point\ matching$



$fig[4]:\ plotting\ the\ epilines\ from\ left\ to\ right$



$fig[5]:\ plotting\ the\ epilines\ from\ right\ to\ left$

$fig[5]:\ Disparity\ Map$

3 Task3

Abstract

The goal of this task is stated point wise below:

1. Compute the classification vector and plot of given N samples and corresponding centroids.
2. Recompute centroid value and plot the graph of corresponding cetroids
3. Repeat the above process for 2nd iteration and plot the graph

3.1 Experimental setup:

1. calculate the euclidean distance and update the cluster points.
2. cluster the points according to there nearest centroid.
3. plot the graph of the new distance points.
4. recompute the centroid values by taking the average of new distance value of x and y.
5. plot the graph for new centroid values.
6. repeat the above steps for seconf iteration.

3.2 code:

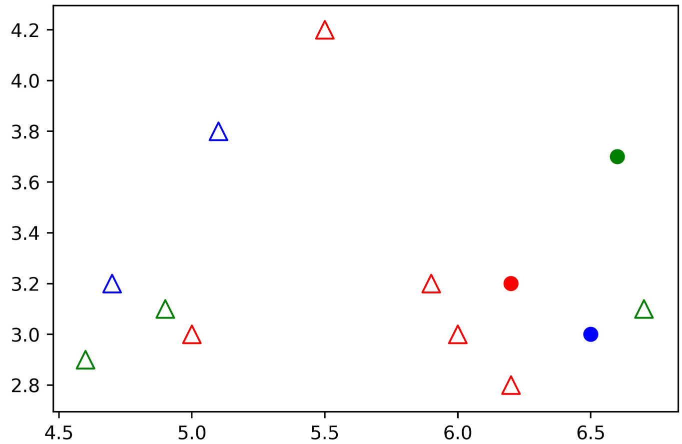
```
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt

def _clustering(a, b):
return np.linalg.norm(a - b, axis=1)

x_1 = [5.9,4.6,6.2,4.7,5.5,5.0,4.9,6.7,5.1,6.0]
y_1 = [3.2,2.9,2.8,3.2,4.2,3.0,3.1,3.1,3.8,3.0]
X = np.asarray(list(zip(x_1,y_1)))
mu_x =[6.2,6.6,6.5]
mu_y =[3.2,3.7,3.0]
mu = np.asarray(list(zip(mu_x,mu_y)))
print(mu)
new_mu = np.zeros(len(X))
for i in range(len(X)):
    new_samples = _clustering(X[i], mu)
    c =  np.argmin(new_samples)
    new_mu[i] = c
print(new_mu)
new_samples =[]
new_x =[]
colors = ['r', 'g', 'b']
figure1, axis1 = plt.subplots()
```

```
for k in range(3):
    temp = np.array([X[i] for i in range(len(X)) if new_mu[i] == k])
    new_x.append(temp)
    print("cluster" + str(k+1), temp )
    axis1.scatter(temp[:,0], temp[:,1], marker='^',
    s=90,facecolor ='#FFFFFF' ,edgecolor=colors)
axis1.scatter(mu[:,0], mu[:,1], marker='o', s=50
, facecolor=colors)
figure1.savefig('task3_iter1_a.jpg', dpi=500)

c1 =np.asarray(new_x[0])
c2 =np.asarray(new_x[1])
c3 =np.asarray(new_x[2])
c1_xaxis =c1[:,0]
c1_yaxis =c1[:,1]
c2_xaxis =c2[:,0]
c2_yaxis =c2[:,1]
c3_xaxis =c3[:,0]
c3_yaxis =c3[:,1]
#========= Part2===================
mu_1 = [(sum(c1_xaxis)+mu[0,0])/(len(c1_xaxis)+1),(sum(c1_yaxis)+mu[0,0])
/(len(c1_yaxis)+1)]
mu_2 = [(sum(c2_xaxis)+mu[0,0])/(len(c2_xaxis)+1),(sum(c2_yaxis)+mu[0,0])
/(len(c2_yaxis)+1)]
mu_3 = [(sum(c3_xaxis)+mu[0,0])/(len(c3_xaxis)+1),(sum(c3_yaxis)+mu[0,0])
/(len(c3_yaxis)+1)]
print(mu_1)
print(mu_2)
print(mu_3)
figure2, axis2 = plt.subplots()
mu_x1 =[mu_1[0],mu_2[0],mu_3[0]]
mu_y1 =[mu_1[1],mu_2[1],mu_3[1]]
NewMu = np.asarray(list(zip(mu_x1,mu_y1)))
color =['r','b','g']
axis2.scatter(mu_x1, mu_y1, marker='o', s=50, c=color)
print("newmu",NewMu)
figure2.savefig('task3_iter1_b.jpg', dpi=500)

#=========part3====================
X1 = np.asarray(list(zip(x_1,y_1)))
new_mu1 = np.zeros(len(X1))
for i in range(len(X1)):
    new_samples1 = _clustering(X1[i], NewMu)
    c1 =  np.argmin(new_samples1)
    new_mu1[i] = c1
new_x1 =[]
colors = ['r', 'g', 'b']
figure3, axis3 = plt.subplots()
for k in range(3):
    temp1 = np.array([X1[i] for i in range(len(X1)) if new_mu1[i] == k])
```

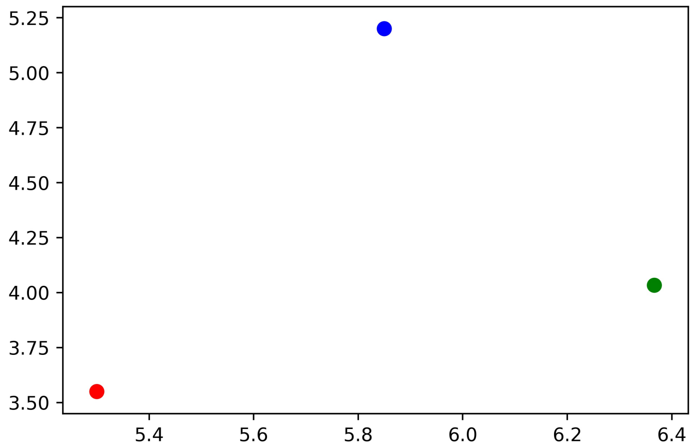
```
    new_x1.append(temp1)
    print("cluster" + str(k+1), temp )
    axis3.scatter(temp1[:,0], temp1[:,1], marker='^',
    s=90,facecolor ='#FFFFFF' ,edgecolor=colors)

c1 =np.asarray(new_x[0])
c2 =np.asarray(new_x[1])
c3 =np.asarray(new_x[2])
c1_xaxis =c1[:,0]
c1_yaxis =c1[:,1]
c2_xaxis =c2[:,0]
c2_yaxis =c2[:,1]
c3_xaxis =c3[:,0]
c3_yaxis =c3[:,1]
#========= Part2====================
mu_1 = [(sum(c1_xaxis)+mu[0,0])/(len(c1_xaxis)+1),(sum(c1_yaxis)+mu[0,0])
/(len(c1_yaxis)+1)]
mu_2 = [(sum(c2_xaxis)+mu[0,0])/(len(c2_xaxis)+1),(sum(c2_yaxis)+mu[0,0])
/(len(c2_yaxis)+1)]
mu_3 = [(sum(c3_xaxis)+mu[0,0])/(len(c3_xaxis)+1),(sum(c3_yaxis)+mu[0,0])
/(len(c3_yaxis)+1)]
print(mu_1)
print(mu_2)
print(mu_3)
figure2, axis2 = plt.subplots()
mu_x1 =[mu_1[0],mu_2[0],mu_3[0]]
mu_y1 =[mu_1[1],mu_2[1],mu_3[1]]
NewMu = np.asarray(list(zip(mu_x1,mu_y1)))
color =['r','b','g']
axis2.scatter(mu_x1, mu_y1, marker='o', s=50, c=color)
print("newmu",NewMu)
figure2.savefig('task3_iter1_b.jpg', dpi=500)
```

3.3 Result:

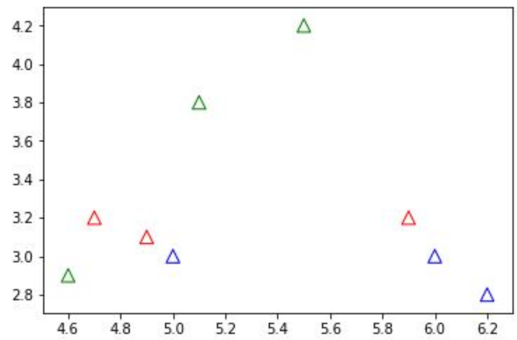
Graphs: Images task3itera.jpg and task3iterb.jpg



$fig[1]: \ graph \ for \ data \ samples$



$fig[2]: \ task3_iterb$



$fig[3] : foriteration2$

4 References:

caluclation of descriptors are done by using different feature matching techniques: like bruteforce,FLANN matchers,KNN match.
link1 : https://opencv-python-tutroals.readthedocs.i o/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html
link2 : https://docs.opencv.org/2.4/modules/features2d/ doc/common_interfaces_of_descriptor_matchers.html
link3 : https://docs.opencv.org/3.4/db/d39/classcv_1_1D escriptorMatcher.html --> bruteforce and FLANN matchers
link4 : https://docs.opencv.org/3.0-beta/doc/py_tutorials /py_feature2d/py_matcher/py_matcher.html --> knn matcher program
link5 : https://opencv-python-tutroals.readthedocs.io/en/latest /py_tutorials/py_feature2d/ py_sift_intro/py_sift_intro.html
link6 : https://docs.opencv.org/3.4.1/d9/dab/tutorial_homography.html