SCHOOL OF COMPUTER SCIENCE AND ENGINEERING, UNIVERSITY AT BUFFALO

Computer Vision and Image Processing
CSE 573: Project 2

Sagnik Ghosh [UB person number : 50289151]
November 5, 2018

1. IMAGE FEATURES AND HOMOGRAPHY (5PT)

- SOURCE CODE:

```
UBIT = 'sagnikgh'
import numpy as np
np.random.seed(sum([ord(c) for c in UBIT]))

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

##############################################PROBLEM 1.1, 1.2, 1.4#####################

img1      = cv2.imread('mountain1.jpg',0)
img2      = cv2.imread('mountain2.jpg',0)

sift          = cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
```

```
kp1        = sift.detect(img1,None)
kp2        = sift.detect(img2,None)

img1 =
cv2.drawKeypoints(img1,kp1,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2 =
cv2.drawKeypoints(img2,kp2,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

good_match= []
gm_random = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good_match.append([m])

gm_random = random.sample(good_match, 10)

img4=cv2.drawMatchesKnn(img1,kp1,img2,kp2,gm_random,None,flags=2)

img3=cv2.drawMatchesKnn(img1,kp1,img2,kp2,good_match,None,flags=2)

cv2.imwrite('task1_sift1.jpg',img1)
cv2.imwrite('task1_sift2.jpg',img2)
cv2.imwrite('task1_matches_knn.jpg',img3)
cv2.imwrite('task1_matches.jpg.jpg',img4)

################################################PROBLEM 1.3, 1.5#############################

def get_stitched_image(img1, img2, M):

        w1,h1 = img1.shape[:2]
        w2,h2 = img2.shape[:2]

        img1_dims =
        np.float32([ [0,0], [0,w1], [h1, w1], [h1,0] ]).reshape(-1,1,2)
        img2_dims_temp =
        np.float32([ [0,0], [0,w2], [h2, w2], [h2,0] ]).reshape(-1,1,2)

        img2_dims = cv2.perspectiveTransform(img2_dims_temp, M)

        result_dims = np.concatenate( (img1_dims, img2_dims), axis = 0)
```

```
        [x_min, y_min] = np.int32(result_dims.min(axis=0).ravel() - 0.5)
        [x_max, y_max] = np.int32(result_dims.max(axis=0).ravel() + 0.5)

        transform_dist = [-x_min,-y_min]
        transform_array =
        np.array([[1, 0, transform_dist[0]], [0, 1, transform_dist[1]], [0,0,1]])

        result_img = cv2.warpPerspective
        (img2, transform_array.dot(M), (x_max-x_min, y_max-y_min))
        result_img[transform_dist[1]:w1+transform_dist[1],
                                transform_dist[0]:h1+transform_dist[0]] = img1

        return result_img

def get_sift_homography(img1, img2):

        sift = cv2.xfeatures2d.SIFT_create()

        k1, d1 = sift.detectAndCompute(img1, None)
        k2, d2 = sift.detectAndCompute(img2, None)

        bf = cv2.BFMatcher()
        matches = bf.knnMatch(d1,d2, k=2)

        good_match = []
        for m1,m2 in matches:
                if m1.distance < 0.75 * m2.distance:
                        good_match.append(m1)

        min_matches = 8
        if len(good_match) > min_matches:

                img1_pts = []
                img2_pts = []

                for match in good_match:
                        img1_pts.append(k1[match.queryIdx].pt)
                        img2_pts.append(k2[match.trainIdx].pt)
                img1_pts = np.float32(img1_pts).reshape(-1,1,2)
                img2_pts = np.float32(img2_pts).reshape(-1,1,2)

                M, mask = cv2.findHomography(img1_pts, img2_pts, cv2.RANSAC, 5.0)
                return M
```

```
        else:
                print ('Error: Not enough matches')
                exit()

def main():

        img1 = cv2.imread('mountain1.jpg',0)
        img2 = cv2.imread('mountain2.jpg',0)


        M =  get_sift_homography(img1, img2)

        with open('homographymatrix.txt','w') as f:
                for line in M:
                        np.savetxt(f, line, fmt='%.2f')

        result_image = get_stitched_image(img2, img1, M)

        cv2.imwrite('task1_pano.jpg',result_image)

if __name__=='__main__':
        main()
```

- OUTPUT IMAGES:

Figure 0.1: task1 sift1

Figure 0.2: task1 sift2

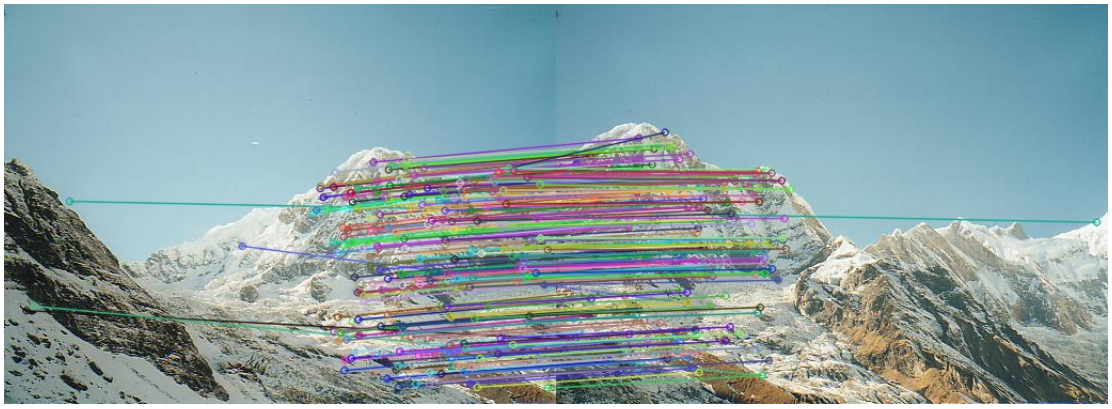

Figure 0.3: task1 matches knn

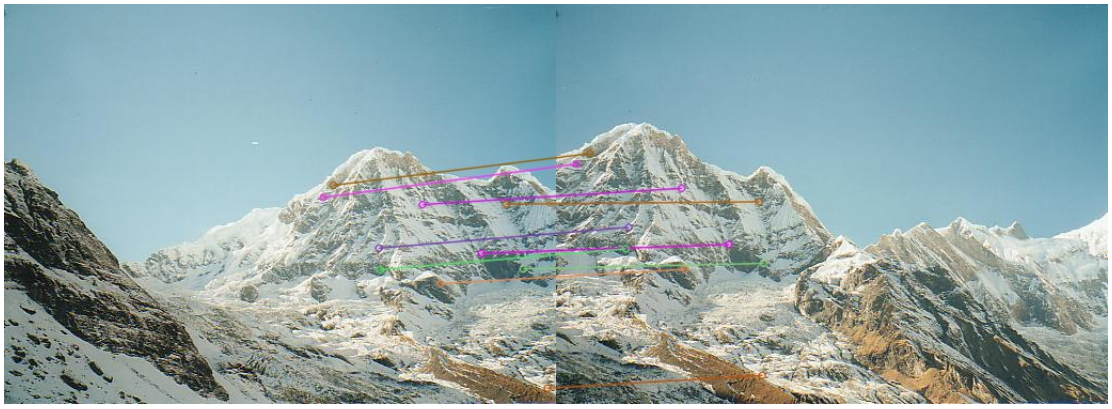

Figure 0.4: task1 matches

- Homography Matrix:

1.59	-0.29	-395.97
0.45	1.43	-190.61
0.00	-0.00	1.00

Figure 0.5: task1 pano

2 Epipolar Geometry (5 pt)

- SOURCE CODE:

```
UBIT = 'sagnikgh';
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 random import choices

##############################################PROBLEM 2.1#################################

img1 = cv2.imread('tsucuba_left.png',0)
img2 = cv2.imread('tsucuba_right.png',0)

sift = cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

img1  =
```

```
cv2.drawKeypoints(img1,kp1,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2  =
cv2.drawKeypoints(img2,kp2,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

good_match = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good_match.append([m])

img3 =
cv2.drawMatchesKnn(img1,kp1,img2,kp2,good_match,None,flags=2)

cv2.imwrite('task2_sift1.jpg',img1)
cv2.imwrite('task2_sift2.jpg',img2)
cv2.imwrite('task2_matches_knn.jpg',img3)

################################################PROBLEM 2.2, 2.3#############################

img1 = cv2.imread('tsucuba_left.png',0)
img2 = cv2.imread('tsucuba_right.png',0)
sift = cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

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(des1,des2,k=2)
good = []
pts1 = []
pts2 = []
pts11 = []
pts22 = []

for i,(m,n) in enumerate(matches):
    if m.distance < 0.75*n.distance:
        good.append(m)
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)
```

```
pts1 = np.int32(pts1)
pts2 = np.int32(pts2)
F, mask = cv2.findFundamentalMat(pts1,pts2,cv2.FM_LMEDS)

with open('fundmentalmatrix.txt','w') as f:
                for line in F:
                        np.savetxt(f, line, fmt='%.2f')

pts1 = pts1[mask.ravel()==1]
pts2 = pts2[mask.ravel()==1]

rnd = np.random.choice(len(pts1), 10)
pts11 = pts1[rnd]
pts22 = pts2[rnd]

def drawlines(img1,img2,lines,pts11,pts22,color):
    r,c = img1.shape
    img1 = cv2.cvtColor(img1,cv2.COLOR_GRAY2BGR)
    img2 = cv2.cvtColor(img2,cv2.COLOR_GRAY2BGR)
    for r,pt1,pt2,color in zip(lines,pts11,pts22,color):
        x0,y0 = map(int, [0, -r[2]/r[1] ])
        x1,y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
        img1 = cv2.line(img1, (x0,y0), (x1,y1), color,1)
        img1 = cv2.circle(img1,tuple(pt1),5,color,-1)
        img2 = cv2.circle(img2,tuple(pt2),5,color,-1)
    return img1,img2

lines1 = cv2.computeCorrespondEpilines(pts22.reshape(-1,1,2), 2,F)
lines1 = lines1.reshape(-1,3)
color = tuple(np.random.randint(0,255, size=(10, 3)).tolist())
img5,img6 = drawlines(img1,img2,lines1,pts11,pts22,color)

lines2 = cv2.computeCorrespondEpilines(pts11.reshape(-1,1,2), 1,F)
lines2 = lines2.reshape(-1,3)
img3,img4 = drawlines(img2,img1,lines2,pts22,pts11,color)

cv2.imwrite('task2_epi_left.jpg',img5)
cv2.imwrite('task2_epi_right.jpg',img3)

##############################################PROBLEM 2.4#################################

imgL = cv2.imread('tsucuba_left.png',0)
imgR = cv2.imread('tsucuba_right.png',0)
```

```
window_size = 3

StereoMatcher = cv2.StereoSGBM_create(
        minDisparity = 16,
        numDisparities = 48,
        blockSize = 9,
        P1 = 8*3*window_size**2,
        P2 = 32*3*window_size**2,
        disp12MaxDiff = 1,
        uniquenessRatio = 10,
        speckleWindowSize = 50,
        speckleRange = 32)

disparity = StereoMatcher.compute(imgL, imgR)
plt.imshow(disparity,'gray')
plt.savefig('task2_disparity.jpg')
```

- OUTPUT IMAGES:

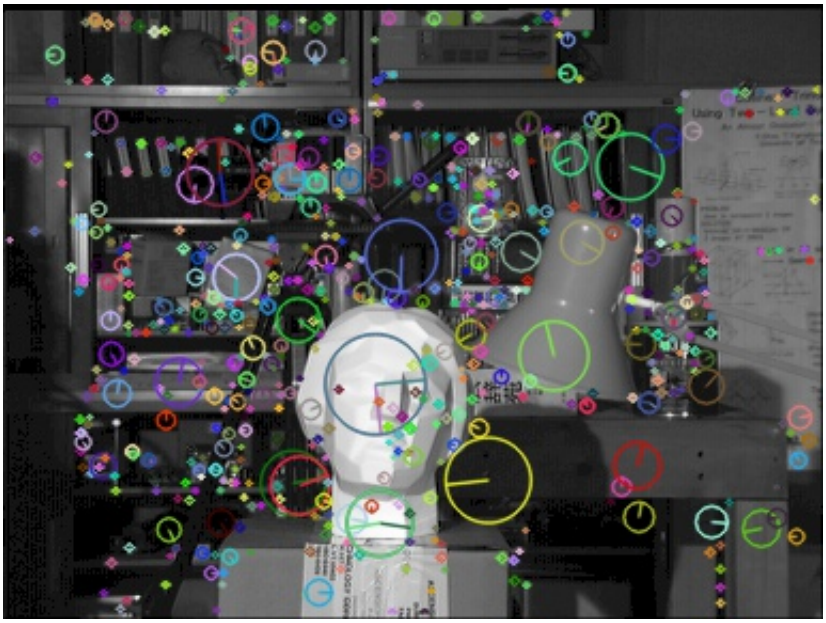

Figure 0.6: task2 sift1

Figure 0.7: task2 sift2

Figure 0.8: task2 matches knn

- FUNDAMENTAL MATRIX:

0.00	-0.00	-0.00
0.00	0.00	37218935642416.00
-0.00	-37218935642416.01	1.00

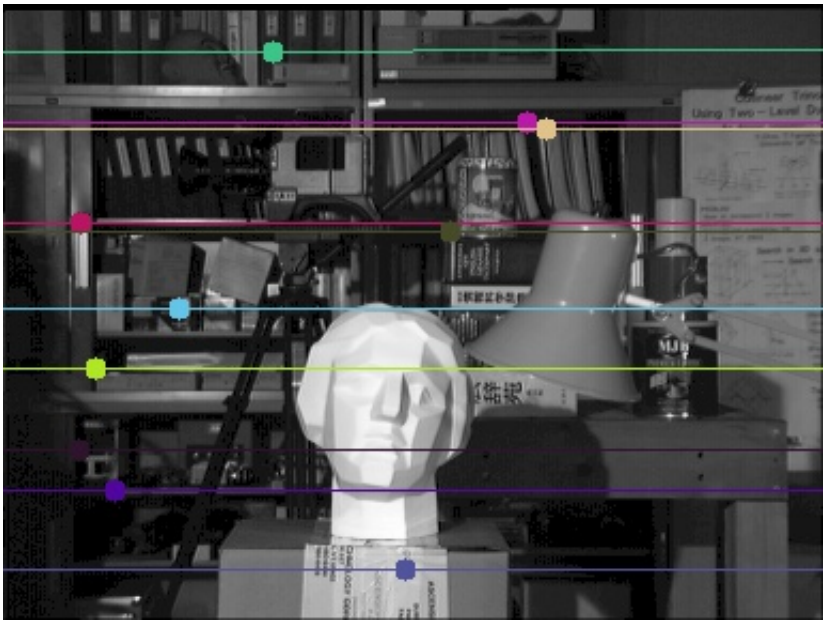

Figure 0.9: task2 epi left

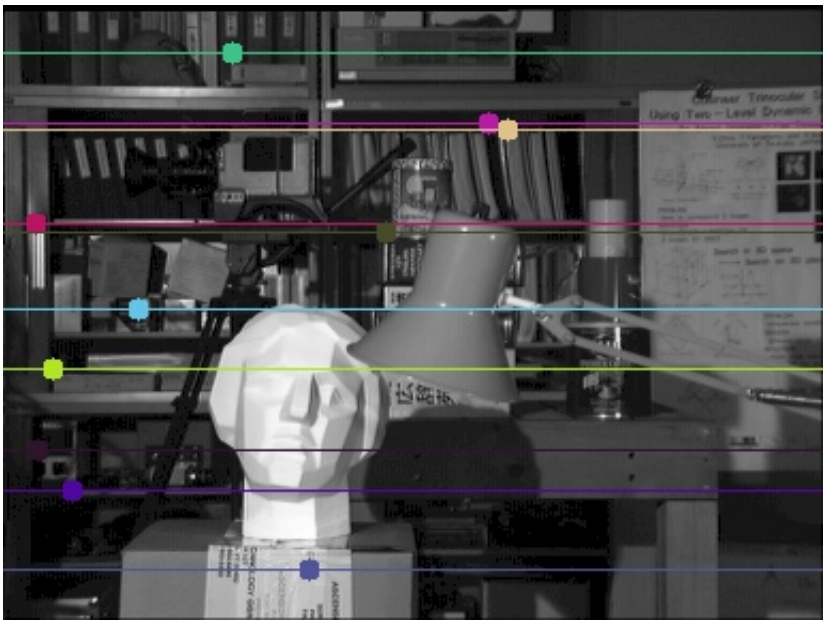

Figure 0.10: task2 epi right

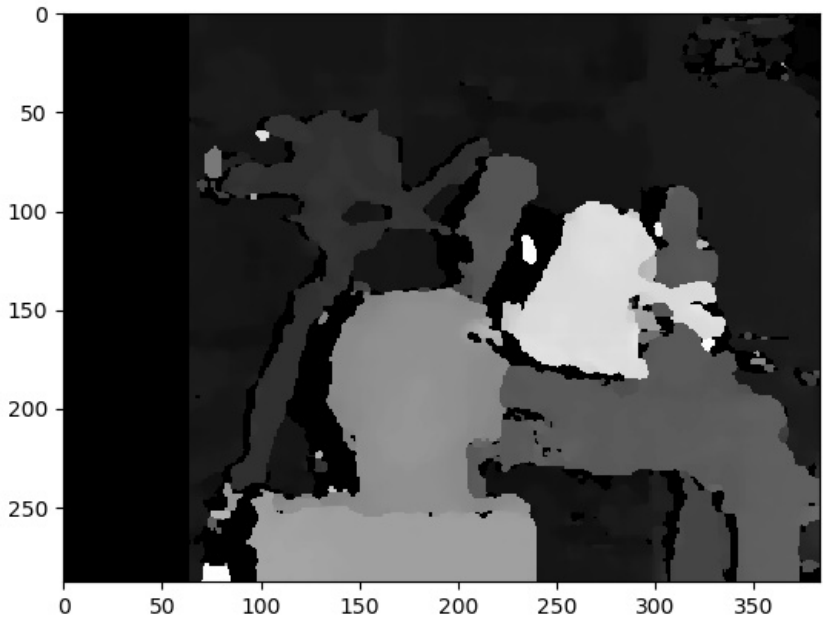

Figure 0.11: task2 disparity

3. K-MEANS CLUSTERING (5 + 3 PT)

- SOURCE CODE

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

from copy import deepcopy
import math
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from copy import deepcopy
import cv2
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

##############################################PROBLEM 3.1, 3.2, 3.3#######################
```

```
k=3
fig = plt.figure()
ax = fig.add_subplot(111)

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

f1 = [5.9,4.6,6.2,4.7,5.5,5.0,4.9,6.7,5.1,6.0]
f2 = [3.2,2.9,2.8,3.2,4.2,3.0,3.1,3.1,3.8,3.0]
X = np.array(list(zip(f1, f2)))

C_x = [6.2,6.6,6.5]
C_y = [3.2,3.7,3.0]
C = np.array(list(zip(C_x, C_y)))

plt.scatter(f1, f2, s=80, facecolors='none', edgecolors='b', marker="^")
plt.scatter(C_x, C_y, color=['red','green','blue'])
for xy in zip(f1, f2):
    ax.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
for xy in zip(C_x, C_y):
    ax.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.savefig('task3_iter0.jpg')
plt.show()

dis = []
cluster_red = []
cluster_green = []
cluster_blue = []
c_r_new = []
c_g_new = []
c_b_new = []

for m in range(k):
        for i in range(len(X)):
                for j in range(len(C)):
                        d  = (X[i][0]-C[j][0])**2 + (X[i][1]-C[j][1])**2
                        dd = math.sqrt(d)
                        dis.append(dd)
                if(dis.index(min(dis)) == 0):
                        cluster_red.append(X[i])
                elif(dis.index(min(dis)) == 1):
                        cluster_green.append(X[i])
                elif(dis.index(min(dis)) == 2):
                        cluster_blue.append(X[i])
```

```
            dis = []
        cluster_red = np.array(cluster_red)
        cluster_green = np.array(cluster_green)
        cluster_blue = np.array(cluster_blue)
        c_x = sum(cluster_red[:,0])/len(cluster_red)
        c_r_new.append(c_x)
        c_y = sum(cluster_red[:,1])/len(cluster_red)
        c_r_new.append(c_y)
        c_x = sum(cluster_green[:,0])/len(cluster_green)
        c_g_new.append(c_x)
        c_y = sum(cluster_green[:,1])/len(cluster_green)
        c_g_new.append(c_y)
        c_x = sum(cluster_blue[:,0])/len(cluster_blue)
        c_b_new.append(c_x)
        c_y = sum(cluster_blue[:,1])/len(cluster_blue)
        c_b_new.append(c_y)
        C = []
        C = [c_r_new,c_g_new,c_b_new]
        C = np.array(C)
        C_x = C[:,0]
        C_y = C[:,1]
        r_x = cluster_red[:,0]
        r_y = cluster_red[:,1]
        plt.scatter(r_x, r_y, s=80, facecolors='r', edgecolors='r', marker="^")
        g_x = cluster_green[:,0]
        g_y = cluster_green[:,1]
        plt.scatter(g_x, g_y, s=80, facecolors='g', edgecolors='g', marker="^")
        b_x = cluster_blue[:,0]
        b_y = cluster_blue[:,1]
        plt.scatter(b_x, b_y, s=80, facecolors='b', edgecolors='b', marker="^")
        plt.scatter(C_x, C_y, color=['red','green','blue'])
        plt.savefig('task3_iter'+str(m+1)+'.jpg')
        plt.show()
        cluster_red = []
        cluster_green = []
        cluster_blue = []
        c_r_new = []
        c_g_new = []
        c_b_new = []

##############################################PROBLEM 3.4#################################


k = 5
```

```
c = 3

image = cv2.imread('baboon.jpg')
image_new = np.zeros(image.shape)
rep_mat = np.zeros((image.shape[0],image.shape[1]))

clus = [[[0,0,0],[0,0,0],[0,0,0]]]*c

centroid = np.random.randint(0,255, size=(c, 3))
dm = []

for n in range(k):
        print(n)
        for i in range (len(image)):
                for j in range(len(image)):
                        for k in range (len(centroid)):
                                d = (image[i][j]-centroid[k])
                                d = math.sqrt(d[0]**2 + d[1]**2 + d[2]**2)
                                dm.append(d)
                        x = dm.index(min(dm))
                        rep_mat[i][j] = x
                        clus[x].append(image[i][j])
                        dm = []
                for m in range(len(centroid)):
                        centroid[m] = np.mean(clus[m], axis=0)
        for i in range(len(image_new)):
                for j in range(len(image_new)):
                        z = int(rep_mat[i][j])
                        image_new[i][j] = centroid[z]

cv2.imwrite('task3_baboon_'+str(k)+'.jpg',image_new)
```

- OUTPUT IMAGES:

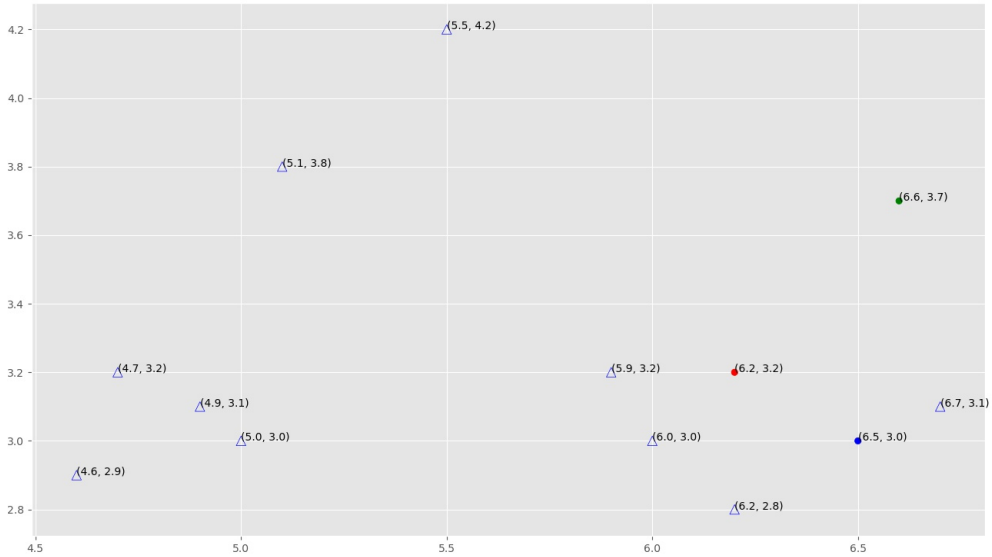

Figure 0.12: task3 iter 1 a

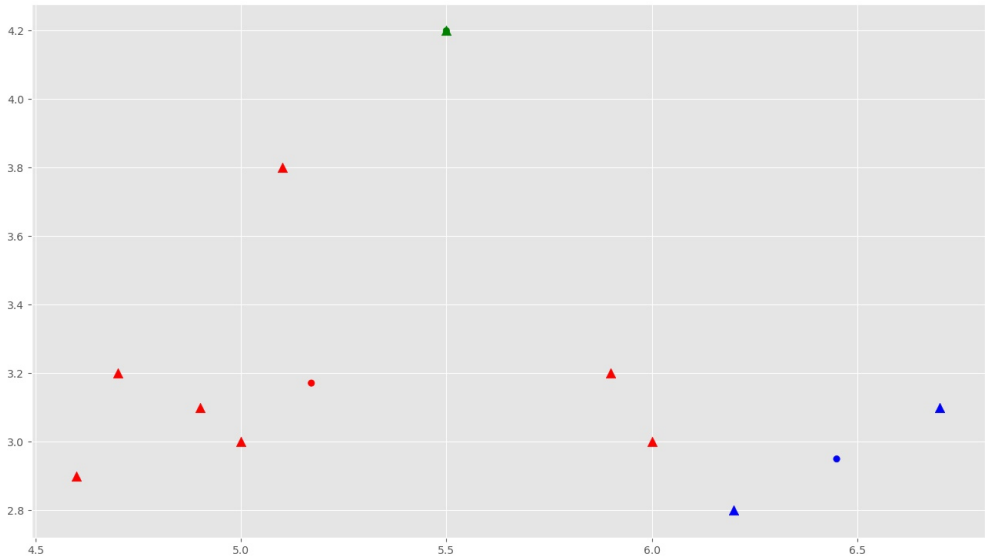

Figure 0.13: task3 iter 1 b

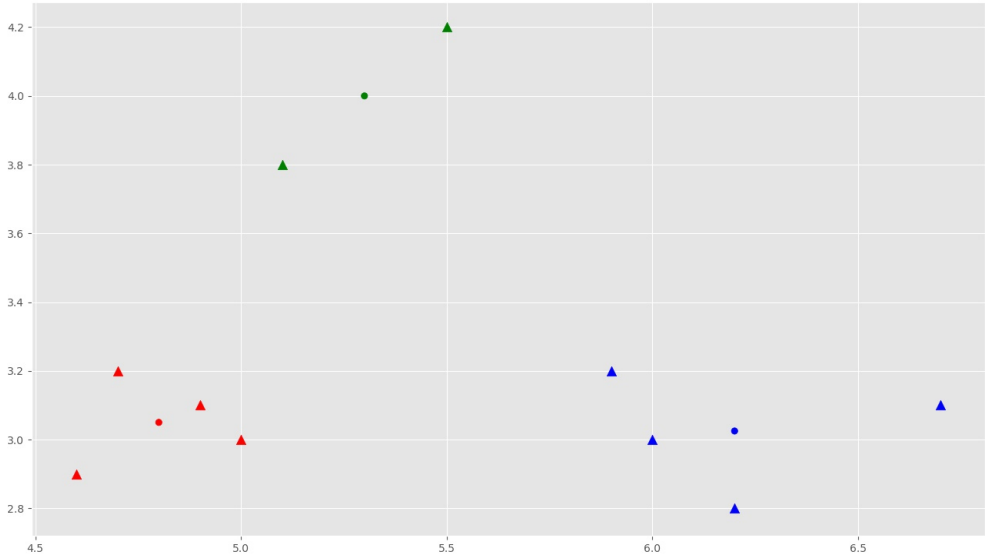

Figure 0.15: task3 iter 2 a

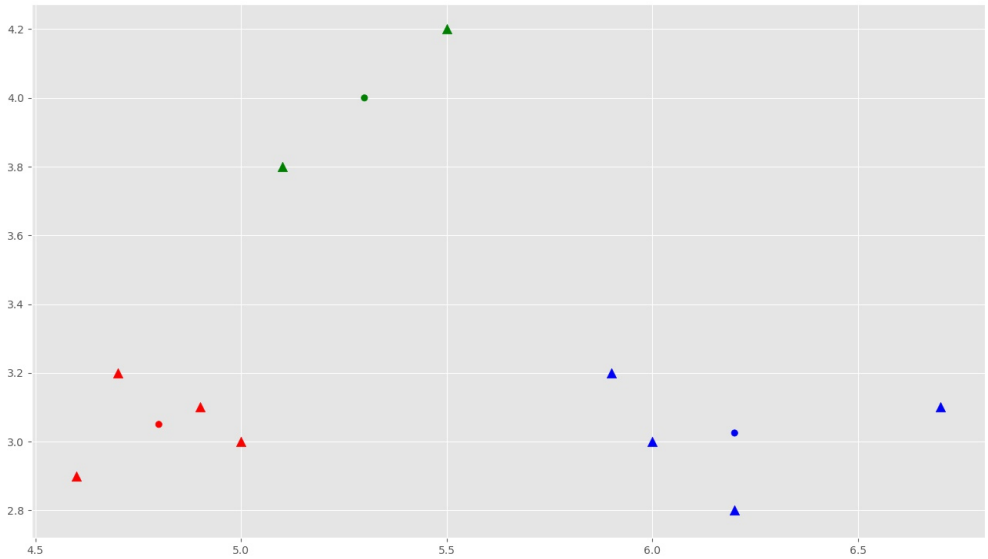

Figure 0.14: task3 iter 2 b

Figure 0.16: task3 baboon 3

N.B. It is taking long time to execute with large K values, which is why only one test case is attached where, k=3.