CSE 573: Introduction to Computer Vision and Image Processing
(Fall 2018)
Instructor: Junsong Yuan

Project 2

November 5, 2018

Report By:
Siddheswar Chandrasekhar

Objective

The objective is to perform three independent tasks: Image Features and Homography, Epipolar Geometry and K-means Clustering

Task 1: Image Features and Homography

Code [1][2][3]

```
UBIT = 'siddhesw'
import numpy as np
np.random.seed(sum([ord(c) for c in UBIT]))
import cv2
import random
def task1():
    img1_color = cv2.imread(r'data/mountain1.jpg')
    img2_color = cv2.imread(r'data/mountain2.jpg')
    img1_gray = cv2.imread(r'data/mountain1.jpg', 0)
    img2_gray = cv2.imread(r'data/mountain2.jpg', 0)
    sift = cv2.xfeatures2d.SIFT_create()
    kp1, des1 = sift.detectAndCompute(img1_gray, None)
    kp2, des2 = sift.detectAndCompute(img2_gray, None)
```

```
key_points_img1 = cv2.drawKeypoints(img1_color, kp1, outImage = np.array([]), flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

key_points_img2 = cv2.drawKeypoints(img2_color, kp2, outImage = np.array([]), flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imwrite('task1_sift1.jpg', key_points_img1)

cv2.imwrite('task1_sift2.jpg', key_points_img2)

matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)

matches_knn = matcher.knnMatch(des1, des2, 2)

ratio_thresh = 0.75

good_matches = []

pts1 = []

pts2 = []

for m,n in matches_knn:

    if m.distance < ratio_thresh * n.distance:

        good_matches.append(m)

        pts1.append(kp1[m.queryIdx].pt)

        pts2.append(kp2[m.trainIdx].pt)

task1_matches_knn = cv2.drawMatches(img1_color, kp1, img2_color, kp2, good_matches, None, matchColor = (0, 255, 0), flags = cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)

cv2.imwrite('task1_matches_knn.jpg', task1_matches_knn)

H, mask =  cv2.findHomography(np.asarray(pts1), np.asarray(pts2), cv2.RANSAC)

print (H)

iH = np.linalg.inv(H)

matchesMask = mask.ravel().tolist()

task1_matches = cv2.drawMatches(img1_color, kp1, img2_color, kp2, np.random.choice(good_matches,10), None, matchColor = (0, 255, 0), matchesMask = random.sample(matchesMask, 10), flags = cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)

cv2.imwrite('task1_matches.jpg', task1_matches)

rows1, cols1 = img1_gray.shape

rows2, cols2 = img2_gray.shape

lp1 = np.float32([[0, 0], [0, rows1], [cols1, rows1], [cols1, 0]]).reshape(-1, 1, 2)

temp = np.float32([[0, 0], [0, rows2], [cols2, rows2], [cols2, 0]]).reshape(-1, 1, 2)

lp2 = cv2.perspectiveTransform(temp, H)

lp = np.concatenate((lp1, lp2), axis = 0)
```

```
    [x_min, y_min] = np.int32(lp.min(axis = 0).ravel() - 0.5)
    [x_max, y_max] = np.int32(lp.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]])
    result = cv2.warpPerspective(img1_color, H_translation.dot(H), (x_max - x_min, y_max - y_min))
    result[translation_dist[1]:rows1+translation_dist[1], translation_dist[0]:cols1+translation_dist[0]] = img2_color
    cv2.imwrite('task1_pano.jpg', result)
if __name__ == '__main__':
    task1()
```

Output

1. Extract SIFT features and draw keypoints for both the images.

Fig 1.1.1 task1_sift1

Fig 1.1.2 task1_sift2

2. Match the keypoints using k-nearest neighbor

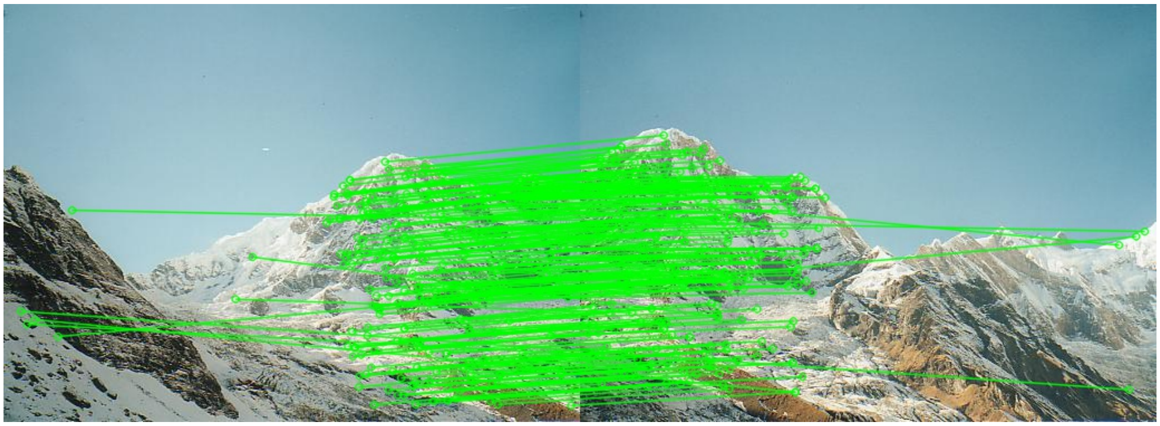

Fig 1.2 task1_matches_knn

3. Compute the Homography Matrix H

1.56236239e+00	-2.89992534e-01	-3.87985294e+02
4.33925807e-01	1.41004277e+00	-1.84226933e+02
1.15959801e-03	-6.30039813e-05	1.00000000e+00

Fig 1.3 Homography Matrix (H)

4. Draw the match image for around 10 random matches using only inliers.

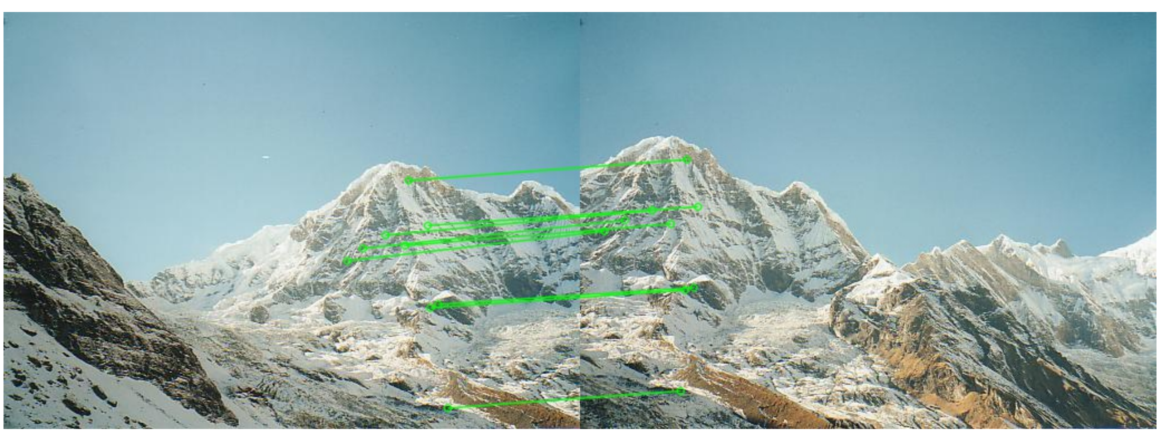

Fig 1.4 task1_matches

5. Warp the first image on the second image using H.

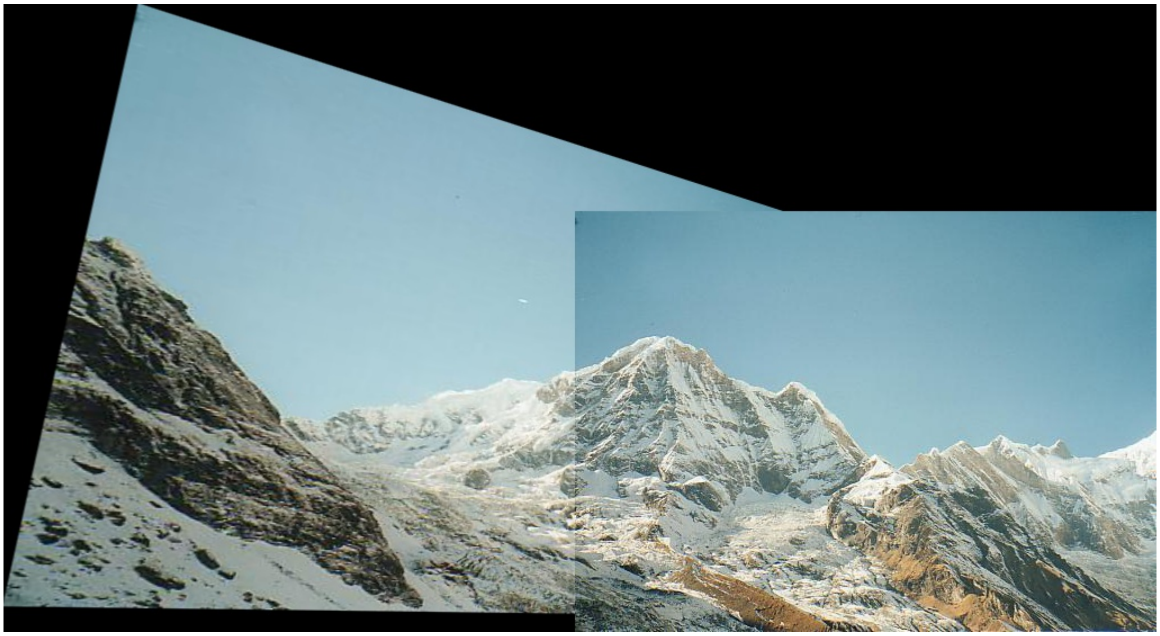

Fig 1.5 task1_pano

Task 2: Epipolar Geometry

Code [1][6]

```
UBIT = 'siddhesw'
import numpy as np
np.random.seed(sum([ord(c) for c in UBIT]))
import cv2
def get_epilines(img, lines, pts1, pts2):
  row, col = img.shape
  img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
  R = 200
  G = 255
  B = 17
  for r, pt1, pt2 in zip(lines, pts1, pts2):
     color = tuple([R, G, B])
     xInit, yInit = map(int, [0, -r[2] / r[1]])
     xNew, yNew = map(int, [col, -(r[2] + r[0] * col) / r[1]])
     img = cv2.line(img, (xInit, yInit), (xNew, yNew), color, 1)
     img = cv2.circle(img, tuple(pt1), 5, color, -1)
     R += 30
     G -= 20
     B += 5
  return img
def task2():
   # reading images
   imgL_color = cv2.imread(r'data/tsucuba_left.png')
   imgR_color = cv2.imread(r'data/tsucuba_right.png')
   imgL_gray = cv2.imread(r'data/tsucuba_left.png', 0)
   imgR_gray = cv2.imread(r'data/tsucuba_right.png', 0)
   sift = cv2.xfeatures2d.SIFT_create()
   kp1, des1 = sift.detectAndCompute(imgL_color, None)
   kp2, des2 = sift.detectAndCompute(imgR_color, None)
   # draw key points on the images
```

```
key_points_imgL = cv2.drawKeypoints(imgL_color, kp1, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
key_points_imgR = cv2.drawKeypoints(imgR_color, kp2, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imwrite('task2_sift1.jpg', key_points_imgL)
cv2.imwrite('task2_sift2.jpg', key_points_imgR)
matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)
matches = matcher.knnMatch(des1, des2, k = 2)
good_matches = []
ptsL = []
ptsR = []
for m, n in matches:
    if m.distance < 0.75*n.distance:
        good_matches.append([m])
        ptsL.append(kp1[m.queryIdx].pt)
        ptsR.append(kp2[m.trainIdx].pt)
matches_knn = cv2.drawMatchesKnn(imgL_color, kp1, imgR_color, kp2, good_matches, outImg=None, matchColor = (0, 255, 0), flags = cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imwrite('task2_matches_knn.jpg', matches_knn)
ptsL = np.int32(ptsL)
ptsR = np.int32(ptsR)
# Calculate fundamental matrix
F, mask = cv2.findFundamentalMat(ptsL, ptsR, cv2.RANSAC)
print(F)
# Select inlier points
ptsL = ptsL[mask.ravel() == 1]
ptsR = ptsR[mask.ravel() == 1]
inliersL = []
inliersR = []
# selecting 10 inlier match pairs
for i in [np.random.randint(0, len(ptsL) - 1) for x in range (10)]:
    inliersL.append(ptsL[i])
    inliersR.append(ptsR[i])
inliersL = np.int32(inliersL)
inliersR = np.int32(inliersR)
```

```
    linesR = (cv2.computeCorrespondEpilines(inliersL.reshape(-1, 1, 2), 1, F)).reshape(-1, 3)
    img1 = get_epilines(imgR_gray, linesR, inliersR, inliersL)
    linesL = (cv2.computeCorrespondEpilines(inliersR.reshape(-1, 1, 2), 2, F)).reshape(-1, 3)
    img2 = get_epilines(imgL_gray, linesL, inliersL, inliersR)
    cv2.imwrite('task2_epi_right.jpg', img1)
    cv2.imwrite('task2_epi_left.jpg', img2)
    stereo = cv2.StereoBM_create(numDisparities = 64, blockSize = 27)
    imageDisparity = stereo.compute(imgL_gray, imgR_gray)
    cv2.imwrite('task2_disparity.jpg', imageDisparity)
if __name__ == '__main__':
    task2()
```

Output

1. Extract SIFT features and draw keypoints for both the images.

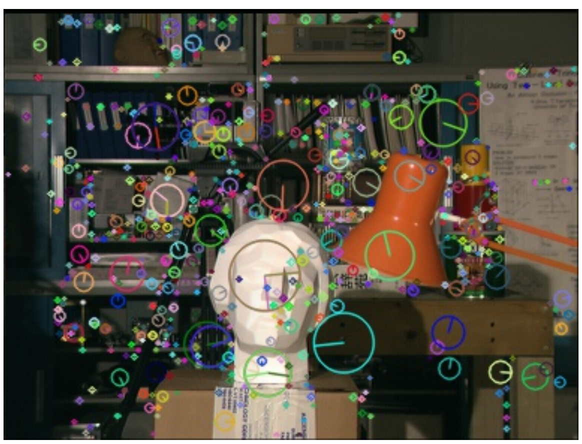

Fig 2.1.1 task2_sift1

Fig 2.1.2 task2_sift2

Match the keypoints using k-nearest neighbor

Fig 2.1.3 task2_matches_knn

2. Compute the Fundamental Matrix F

5.89928037e-08	-1.41101397e-04	4.49345742e-02
1.55556164e-04	-7.40742830e-05	4.57524587e-01
-4.99017487e-02	-4.41969595e-01	1.00000000e+00

Fig 2.2 Fundamental Matrix (F)

3. Randomly select 10 inlier match pairs. For each keypoint in the left image, compute the epiline and draw on the right image. For each keypoint in the right image, compute the epiline and draw on the left image.

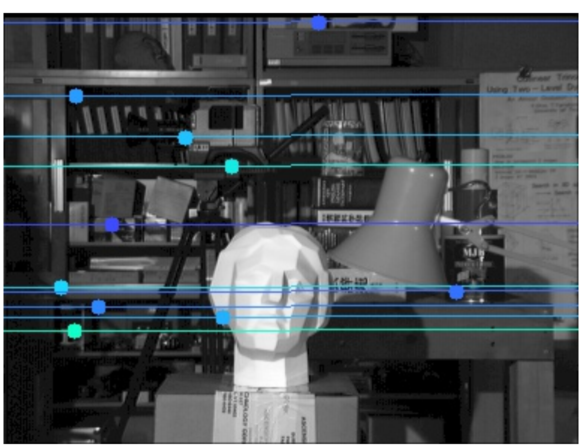

Fig 2.3.1 task2_epi_left

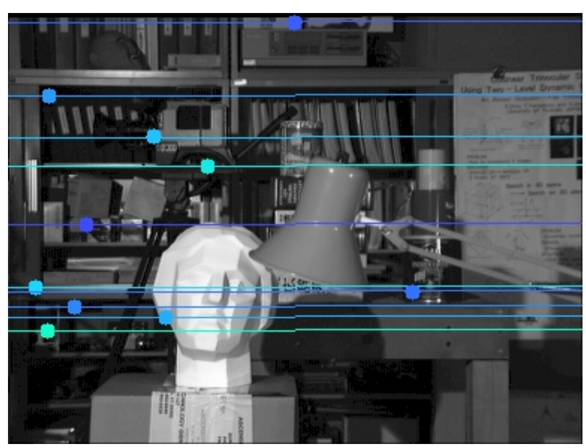

Fig 2.3.2 task2_epi_right

4. Compute the disparity map

Fig 2.4 task2_disparity

Task 3: K-Means Clustering

Code

```
for i in range (len(X)):
    x1,y1 = X[i]
    min = (x1 + y1) ** 2
    minmu = 10
    for j in range (len(mu)):
        x2,y2 = mu[j]
        euclidean = ((x2 - x1) ** 2) + ((y2 - y1) ** 2)
        euclidean = math.sqrt(euclidean)
        if (euclidean < min):
            min = euclidean
            minmu = j
    if (minmu == 0):
        cluster0.append(X[i])
        plt.scatter(x1 , y1, edgecolor = "red", facecolor = "white", marker = "^")
        plt.text(x1, y1 + 0.05, '%s, %s' % (str(x1), str(y1)), fontsize=7)
    elif (minmu == 1):
        cluster1.append(X[i])
        plt.scatter(x1 , y1, edgecolor = "blue", facecolor = "white", marker = "^")
        plt.text(x1, y1 + 0.05, '%s, %s' % (str(x1), str(y1)), fontsize=7)
    elif (minmu == 2):
        cluster2.append(X[i])
        plt.scatter(x1 , y1, edgecolor = "green", facecolor = "white", marker = "^")
        plt.text(x1, y1 + 0.05, '%s, %s' % (str(x1), str(y1)), fontsize=7)
plt.scatter(mu[0][0], mu[0][1], edgecolor = "red", facecolor = "red", marker = "o")
plt.text(mu[0][0], mu[0][1] + 0.05, '%s, %s' % (str(mu[0][0])[:3], str(mu[0][1])[:3]), fontsize=7)
plt.scatter(mu[1][0], mu[1][1], edgecolor = "blue", facecolor = "blue", marker = "o")
plt.text(mu[1][0], mu[1][1] + 0.05, '%s, %s' % (str(mu[1][0])[:3], str(mu[1][1])[:3]), fontsize=7)
plt.scatter(mu[2][0], mu[2][1], edgecolor = "green", facecolor = "green", marker = "o")
plt.text(mu[2][0], mu[2][1] + 0.05, '%s, %s' % (str(mu[2][0])[:3], str(mu[2][1])[:3]), fontsize=7)
plt.savefig('task3_iter2_a.jpg')
```

```
plt.clf()
for i in range (len(X)):
    x1,y1 = X[i]
    min = (x1 + y1) ** 2
    minmu = 10
    for j in range (len(mu)):
        x2,y2 = mu[j]
        euclidean = ((x2 - x1) ** 2) + ((y2 - y1) ** 2)
        euclidean = math.sqrt(euclidean)
        if (euclidean < min):
            min = euclidean
            minmu = j
    if (minmu == 0):
        cluster0.append(X[i])
        plt.scatter(x1 , y1, edgecolor = "red", facecolor = "white", marker = "^")
        plt.text(x1, y1 + 0.05, '%s, %s' % (str(x1), str(y1)), fontsize=7)
    elif (minmu == 1):
        cluster1.append(X[i])
        plt.scatter(x1 , y1, edgecolor = "blue", facecolor = "white", marker = "^")
        plt.text(x1, y1 + 0.05, '%s, %s' % (str(x1), str(y1)), fontsize=7)
    elif (minmu == 2):
        cluster2.append(X[i])
        plt.scatter(x1 , y1, edgecolor = "green", facecolor = "white", marker = "^")
        plt.text(x1, y1 + 0.05, '%s, %s' % (str(x1), str(y1)), fontsize=7)
# Update MU
avgx = 0
avgy = 0
for x,y in cluster0:
    avgx += x
    avgy += y
mu[0] = (avgx / len(cluster0), avgy / len(cluster0))
avgx = 0
avgy = 0
```

```
for x,y in cluster1:
    avgx += x
    avgy += y
mu[1] = (avgx / len(cluster1), avgy / len(cluster1))
avgx = 0
avgy = 0
for x,y in cluster2:
    avgx += x
    avgy += y
mu[2] = (avgx / len(cluster2), avgy / len(cluster2))
plt.scatter(mu[0][0], mu[0][1], edgecolor = "red", facecolor = "red", marker = "o")
plt.text(mu[0][0], mu[0][1] + 0.05, '%s, %s' % (str(mu[0][0])[:3], str(mu[0][1])[:3]), fontsize=7)
plt.scatter(mu[1][0], mu[1][1], edgecolor = "blue", facecolor = "blue", marker = "o")
plt.text(mu[1][0], mu[1][1] + 0.05, '%s, %s' % (str(mu[1][0])[:3], str(mu[1][1])[:3]), fontsize=7)
plt.scatter(mu[2][0], mu[2][1], edgecolor = "green", facecolor = "green", marker = "o")
plt.text(mu[2][0], mu[2][1] + 0.05, '%s, %s' % (str(mu[2][0])[:3], str(mu[2][1])[:3]), fontsize=7)
plt.savefig('task3_iter2_b.jpg')
plt.clf()
```

Output

1. Classifying samples according to nearest centroid

   Classification Vector = [0, 0, 1, 0, 2, 0, 0, 1, 0, 0]

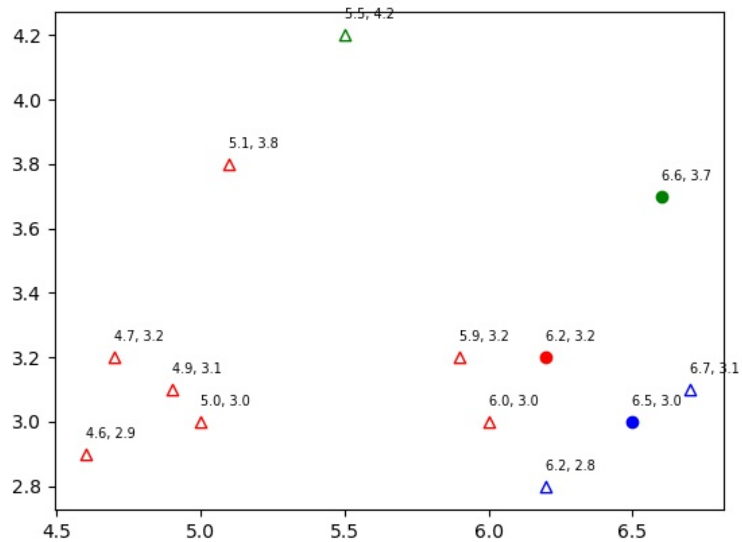

Fig 3.1 task3_iter1_a

2. Recompute the centroids

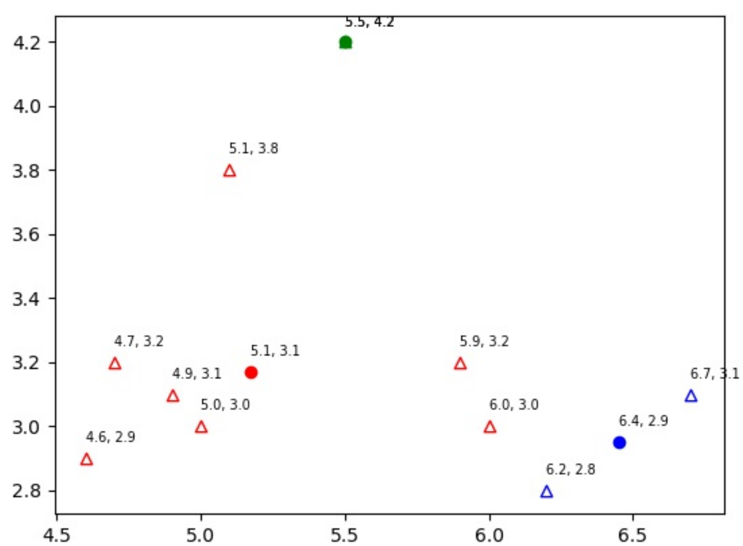

Fig 3.2 task3_iter1_b

3. 2nd iteration

Classification Vector = [1, 0, 1, 0, 2, 0, 0, 1, 2, 1]

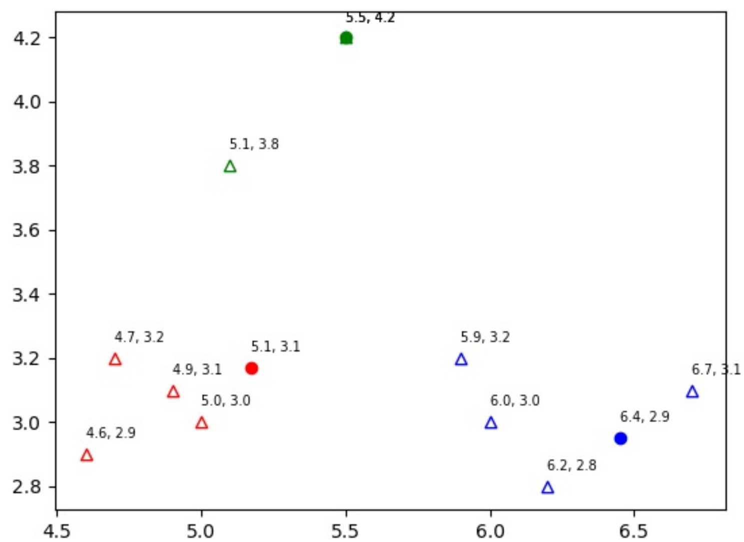

Fig 3.3.1 task3_iter2_a

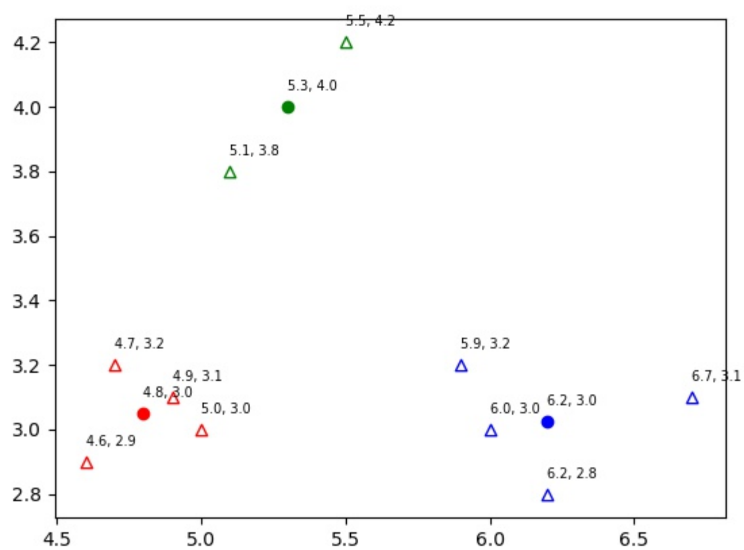

Fig 3.3.2 task3_iter2_b

4. Apply K-means to image color quantization

Code

```
UBIT = 'siddhesw'
import numpy as np
np.random.seed(sum([ord(c) for c in UBIT]))
import cv2
import random
import math
def gen_mu(k):
    mu = []
    for i in range (k):
        mu.append([np.random.randint(0,255), np.random.randint(0,255), np.random.randint(0,255)])
    return mu
def update_mu(mu, clusters, point_list):
    newmu = []
    for i, val in enumerate (mu):
        r = 0
        g = 0
        b = 0
        ctr = 0
        for j,k in zip (clusters, point_list):
            if (i == j):
                r += k[0]
                g += k[1]
                b += k[2]
                ctr += 1
        if (ctr == 0):
            newmu.append(k)
        else:
            r = int(r/ctr)
            g = int(g/ctr)
            b = int(b/ctr)
```

```
        newmu.append([r,g,b])
    return newmu
def clustering(img, mu):
    clusters = []
    point_list = []
    for i in img:
        for j in i:
            r,g,b = j[0], j[1], j[2]
            point_list.append([r,g,b])
    for i in range(len(point_list)):
        min = None
        minmu = 30
        for j in range(len(mu)):
            euclidean = ((mu[j][0] - point_list[i][0]) ** 2) + ((mu[j][1] - point_list[i][1]) ** 2) + ((mu[j][2] - point_list[i][
2]) ** 2)
            euclidean = math.sqrt(euclidean)
            if (min == None or euclidean < min):
                min = euclidean
                minmu = j
        clusters.append(minmu)
    return point_list, clusters
def update_img(mu, clusters, point_list, img):
    newimg = np.zeros([512,512,3])
    ctr = 0
    for j,k in zip (clusters, point_list):
            newimg[int(ctr/512)][int(ctr%512)] = mu[j]
            ctr += 1
    return newimg
def task3_4():
    img = cv2.imread(r'data/baboon.jpg')
    iterations = 25
    for k in [3, 5, 10, 20]:
        mu = gen_mu(k)
```

```
        for i in range (iterations):
            point_list, clusters = clustering(img, mu)
            mu = update_mu(mu, clusters, point_list)
        newimg = update_img(mu, clusters, point_list, img)
        cv2.imwrite('task3_baboon_' + str(k) + '.jpg', newimg)
if __name__ == '__main__':
    task3_4()
```

Output


Fig 3.4.1 task3_baboon_3


Fig 3.4.2 task3_baboon_5

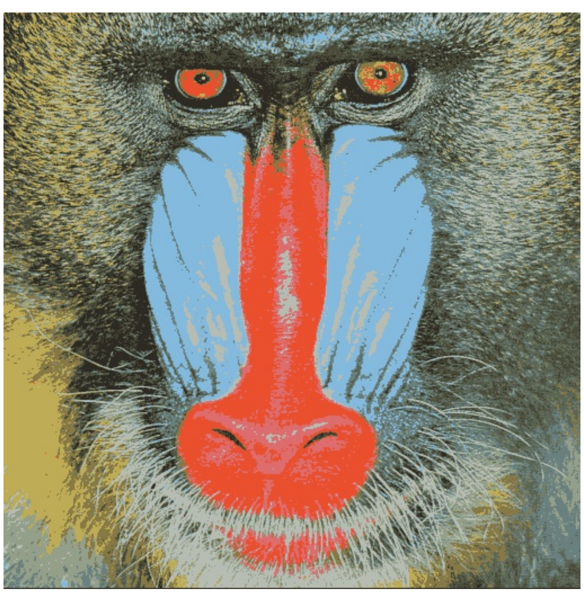
Fig 3.4.3 task3_baboon_10

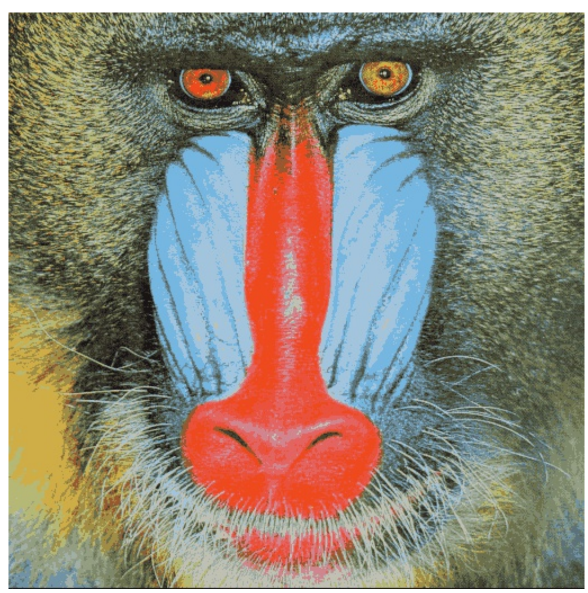
Fig 3.4.4 task3_baboon_20

References:

1. “Feature Matching”
   https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html

2. “Features 2D + Homography”
   https://docs.opencv.org/3.4.2/d7/dff/tutorial_feature_homography.html

3. Asymptote
   “Homography Estimate + Stitching two images”
   https://www.kaggle.com/asymptote/homography-estimate-stitching-two-imag

4. Scale Invariant Feature Transform
   https://docs.opencv.org/3.4.0/da/df5/tutorial_py_sift_intro.html

5. Epipolar Geometry
   https://docs.opencv.org/3.4.3/da/de9/tutorial_py_epipolar_geometry.html

6. Depth Map from Stereo Images
   https://docs.opencv.org/3.1.0/dd/d53/tutorial_py_depthmap.html

7. Computation of Fundamental Matrix
   https://www.cs.unc.edu/~blloyd/comp290-089/fmatrix/

8. *https://www.programcreek.com/python/example/89422/cv2.warpPerspective*