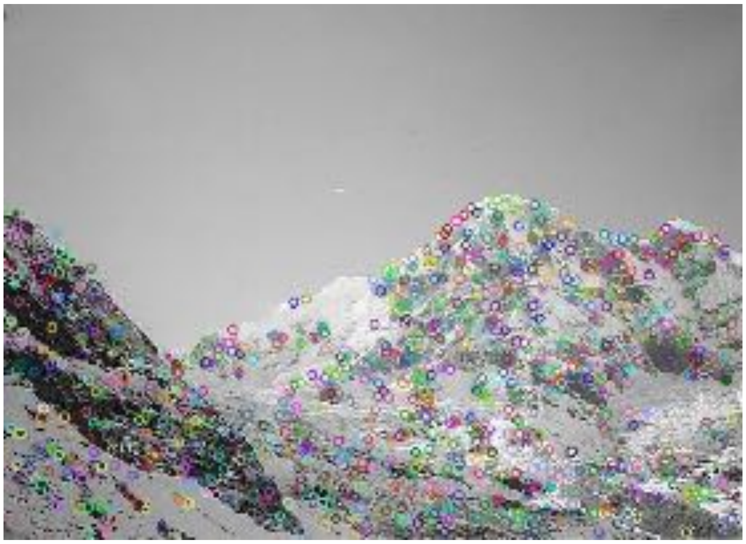
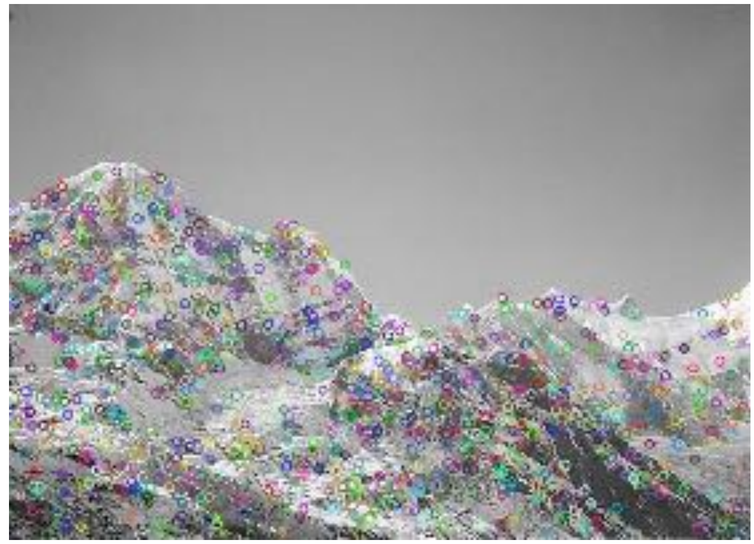
CSE 573 : PROJECT 2 REPORT

NAME : MANASI KULKARNI
UBIT NAME : mkulkarn
STUDENT NUMBER : 50288702

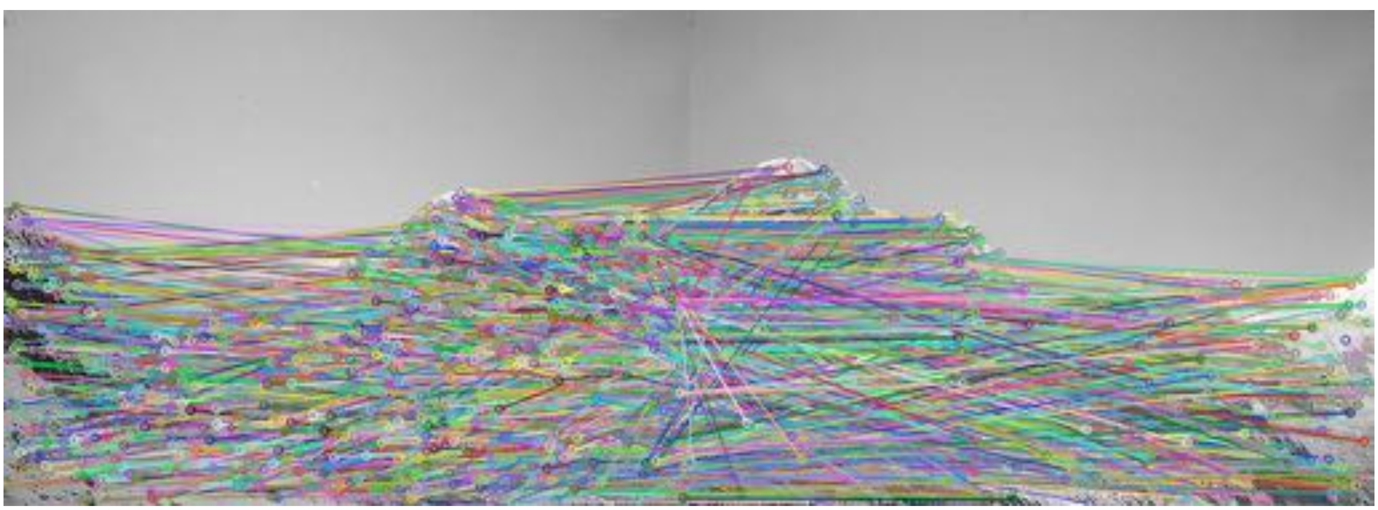
TASK 1: IMAGE FEATURES AND HOMOGRAPHY

1.1 : KEYPOINTS FOR BOTH IMAGES





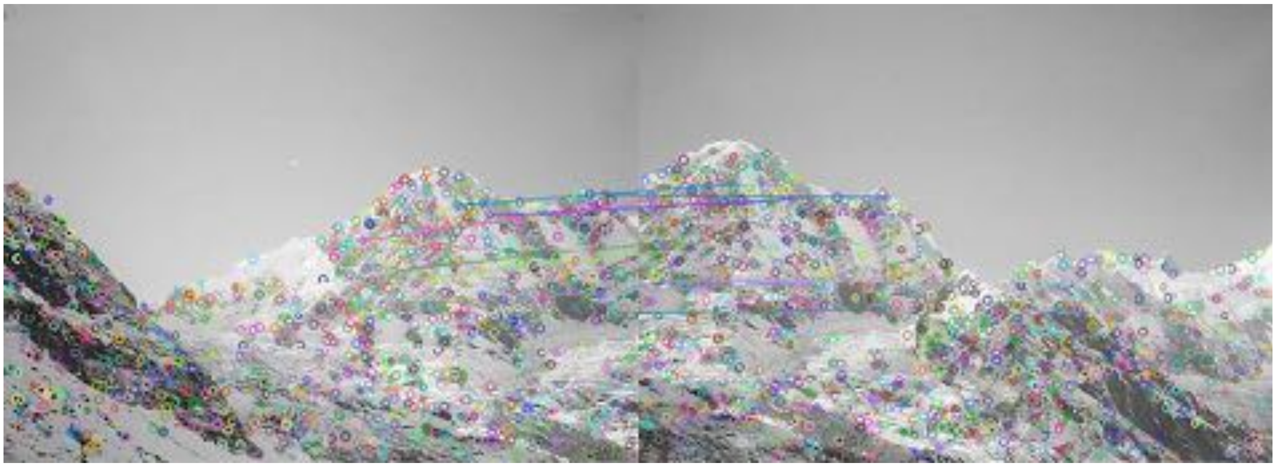
1.2 : MATCH KEYPOINTS



1.3 : HOMOGRAPHY MATRIX

[[1.58930230e+00 -2.91559040e-01 -3.95969265e+02]
 [4.49423930e-01 1.43110916e+00 -1.90613988e+02]
 [1.21265043e-03 -6.28729364e-05 1.00000000e+00]]

1.4 : MATCHES USING INLIERS:



1.5: WARPING:

I have referred the following link for warping images:
https://stackoverflow.com/questions/13063201/how-to-show-the-whole-image-when-using-opencv-warpperspective

Output:

SOURCE CODE:

```
import cv2
import numpy as np

UBIT = 'mkulkarn'
np.random.seed(sum([ord(c) for c in UBIT]))
```

```
mountain1_color = cv2.imread("/Users/manasikulkarni/Desktop/mountain1.jpg")
mountain2_color = cv2.imread("/Users/manasikulkarni/Desktop/mountain2.jpg")
mountain1 = cv2.imread("/Users/manasikulkarni/Desktop/mountain1.jpg", cv2.IMREAD_GRAYSCALE)
mountain2 = cv2.imread("/Users/manasikulkarni/Desktop/mountain2.jpg", cv2.IMREAD_GRAYSCALE)
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(mountain1, None)
kp2, des2 = sift.detectAndCompute(mountain2, None)

def drawKeypointsForBothImages():
    mountain1_kp=cv2.drawKeypoints(mountain1,kp1,mountain1)
    mountain2_kp=cv2.drawKeypoints(mountain2,kp2,mountain2)
    cv2.imwrite("task1_sift1.jpg", mountain1_kp)
    cv2.imwrite("task1_sift2.jpg", mountain2_kp)

def matchKeyPoints():
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
    mountain1_kp=cv2.drawKeypoints(mountain1,kp1,mountain1)
    mountain2_kp=cv2.drawKeypoints(mountain2,kp2,mountain2)
    good = []
    for m,n in matches:
        if m.distance < 0.75 * n.distance:
            good.append(m)
    img3 = cv2.drawMatchesKnn(mountain1_kp,kp1,mountain2_kp,kp2,matches,None,flags=2)
    cv2.imwrite("task1_matches_knn.jpg", img3)

def computeHomographyMatrix():
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
    good = []
    for m,n in matches:
        if m.distance < 0.75 * n.distance:
            good.append(m)
    src = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    H, mask = cv2.findHomography(src, dst, cv2.RANSAC,5.0)
    print(“Homography Matrix : ”, H)

def match10Inliers():
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
    mountain1_kp=cv2.drawKeypoints(mountain1,kp1,mountain1)
    mountain2_kp=cv2.drawKeypoints(mountain2,kp2,mountain2)
    good = []
    for m,n in matches:
        if m.distance < 0.75 * n.distance:
            good.append(m)
    a = np.random.permutation(good)[:10]
    img4 = cv2.drawMatches(mountain1_kp,kp1,mountain2_kp,kp2,a,None,flags=2)
    cv2.imwrite("task1_matches.jpg",img4)

def Warping():
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
    good = []
    for m,n in matches:
        if m.distance < 0.75 * n.distance:
            good.append(m)
```

```
src = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1,1,2)
dst = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1,1,2)
H, mask = cv2.findHomography(src, dst, cv2.RANSAC, 5.0)
h1,w1,c1 = mountain1_color.shape
h2,w2,c2 = mountain2_color.shape
pts1 = np.float32([[0,0],[0,h1],[w1,h1],[w1,0]]).reshape(-1,1,2)
pts2 = np.float32([[0,0],[0,h2],[w2,h2],[w2,0]]).reshape(-1,1,2)
pts1_new = cv2.perspectiveTransform(pts1, H)
finalPoints = np.concatenate((pts2, pts1_new), axis=0)
[xmin, ymin] = np.int32(finalPoints.min(axis=0).ravel() - 0.5)
[xmax, ymax] = np.int32(finalPoints.max(axis=0).ravel() + 0.5)
T = [-xmin,-ymin]
HomoT = np.array([[1,0,T[0]],[0,1,T[1]],[0,0,1]])

result = cv2.warpPerspective(mountain1_color, HomoT.dot(H), (xmax-xmin, ymax-ymin))
result[T[1]:h2+T[1],T[0]:w2+T[0]] = mountain2_color
cv2.imwrite("task1_pano.jpg", result)
```

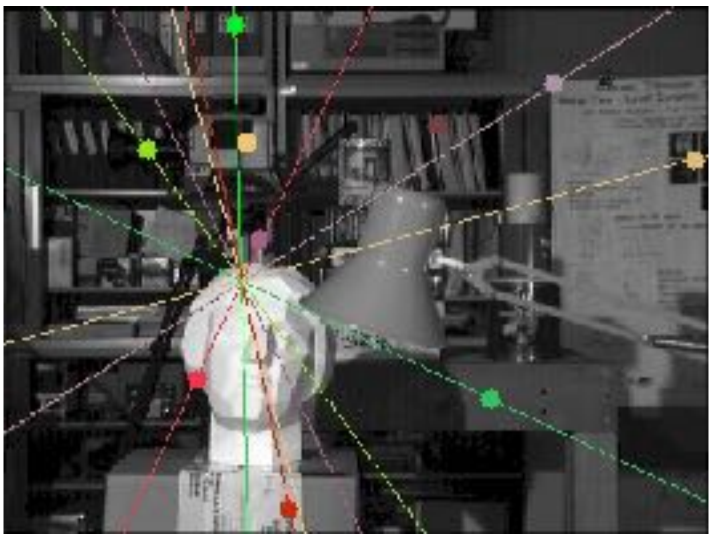
TASK 2: EPIPOLAR GEOMETRY

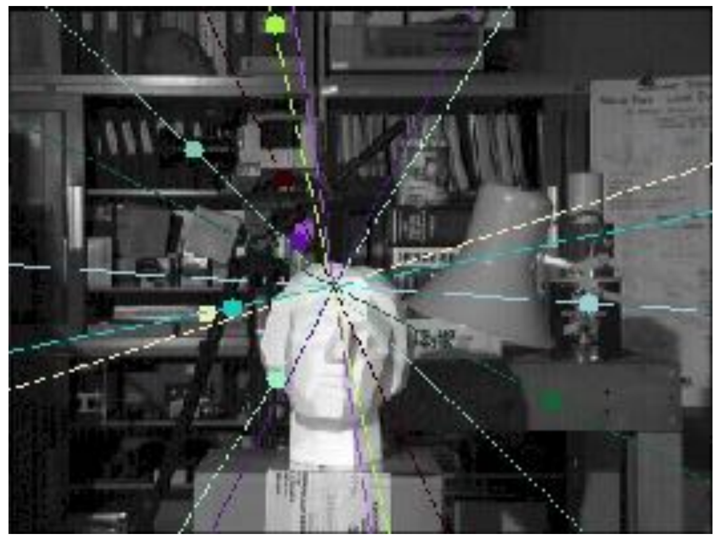
2.1: KEYPOINTS FOR BOTH IMAGES AND MATCHING KEYPOINTS

2.2 : FUNDAMENTAL MATRIX :

[[5.96046448e-08 -1.21593475e-05 -3.21960449e-03]
 [4.45842743e-05 1.31130219e-06 3.72189356e+13]
 [-1.43432617e-03 -3.72189356e+13 1.00000000e+00]]

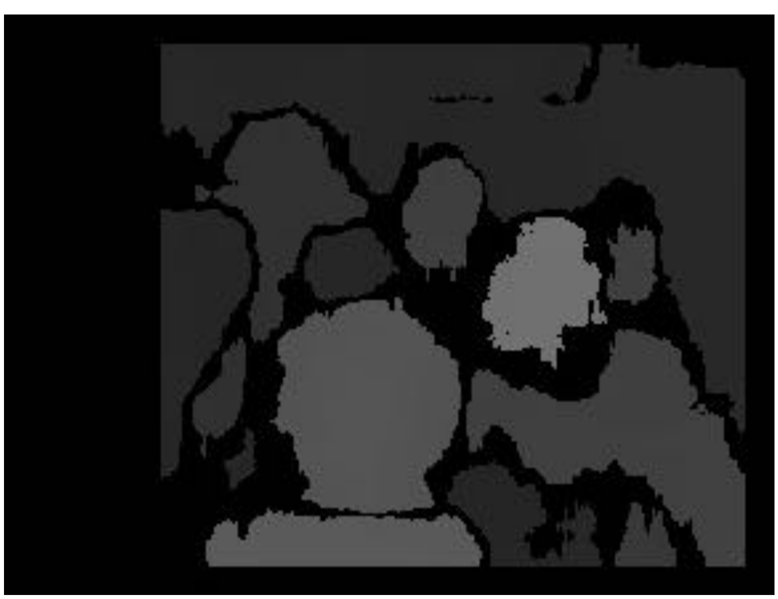
2.3 : EPILINES:





I have referred the following link for drawing epilines:
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_epipolar_geometry/py_epipolar_geometry.html

2.4 : DISPARITY MAP



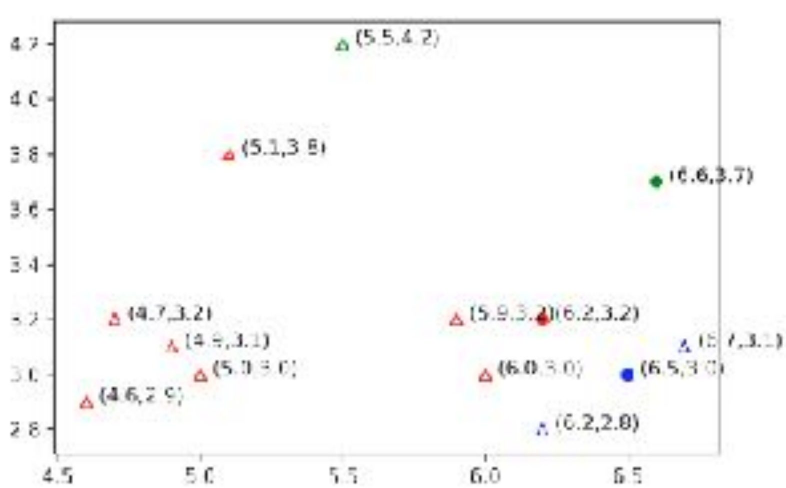
SOURCE CODE:

```
import cv2
import numpy as np

UBIT = 'mkulkarn'
np.random.seed(sum([ord(c) for c in UBIT]))
img1 = cv2.imread("/Users/manasikulkarni/Desktop/tsucuba_left.png", cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread("/Users/manasikulkarni/Desktop/tsucuba_right.png", cv2.IMREAD_GRAYSCALE)
height1, width1 = img1.shape
height2, width2 = img2.shape
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)

def drawKeypointsForBothImages():
    img1_kp=cv2.drawKeypoints(img1,kp1,img1)
    img2_kp=cv2.drawKeypoints(img2,kp2,img2)
    cv2.imwrite('task2_sift1.jpg',img1_kp)
    cv2.imwrite('task2_sift2.jpg',img2_kp)

def matchKeypoints():
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
    img1_kp=cv2.drawKeypoints(img1,kp1,img1)
    img2_kp=cv2.drawKeypoints(img2,kp2,img2)
    good = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append(m)
    img3 = cv2.drawMatches(img1_kp,kp1,img2_kp,kp2,good,None,flags=2)
    cv2.imwrite('task2_matches_knn.jpg',img3)

def computeFundamentalMatrix():
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
    good = []
    pts1 = []
    pts2 = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append(m)
            pts1.append(kp1[m.queryIdx].pt)
            pts2.append(kp2[m.trainIdx].pt)
    pts1 = np.int32(pts1)
    pts2 = np.int32(pts2)
    F, mask = cv2.findFundamentalMat(pts1,pts2,cv2.FM_LMEDS)
    print("Fundamental Matrix : ", F)

def drawlines(image1, image2, lines, pts1, pts2):
    r, c = image1.shape
    image1 = cv2.cvtColor(image1, cv2.COLOR_GRAY2BGR)
    image2 = cv2.cvtColor(image2, cv2.COLOR_GRAY2BGR)
    for r, pt1, pt2 in zip(lines, pts1, pts2):
        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] ])
        image1 = cv2.line(image1, (x0, y0), (x1, y1), color, 1)
        image1 = cv2.circle(image1, tuple(pt1), 5, color, -1)
```

```
        image2 = cv2.circle(image2, tuple(pt2), 5, color, -1)
    return image1, image2

def drawEpilines():
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
    matches_random = np.random.permutation(matches)[:10]
    good_random = []
    pts1_rand = []
    pts2_rand = []
    for m,n in matches_random:
        good_random.append(m)
        pts1_rand.append(kp1[m.queryIdx].pt)
        pts2_rand.append(kp2[m.trainIdx].pt)
    pts1_rand = np.int32(pts1_rand)
    pts2_rand = np.int32(pts2_rand)
    F1, mask1 = cv2.findFundamentalMat(pts1_rand,pts2_rand,cv2.FM_LMEDS)
    linesLeft = cv2.computeCorrespondEpilines(pts2_rand.reshape(-1, 1, 2), 2, F1)
    linesLeft = linesLeft.reshape(-1, 3)
    img5, img6 = drawlines(img1, img2, linesLeft, pts1_rand, pts2_rand)
    linesRight = cv2.computeCorrespondEpilines(pts1_rand.reshape(-1, 1, 2), 1, F1)
    linesRight = linesRight.reshape(-1, 3)
    img7, img8 = drawlines(img2, img1, linesRight, pts2_rand, pts1_rand)
    cv2.imwrite('task2_epi_left.jpg',img5)
    cv2.imwrite('task2_epi_right.jpg',img7)

def computeDisparityMap():
    stereobm = cv2.StereoBM_create(numDisparities=64, blockSize=31)
    stereobm.setSpeckleWindowSize(100)
    stereobm.setSpeckleRange(20)
    disparity = stereobm.compute(img1, img2)
    cv2.imwrite('task2_disparity.jpg', (disparity/2048) * 255)
```

TASK 3: K-MEANS CLUSTERING

3.1 : CLASSIFICATION



Classification Vector:

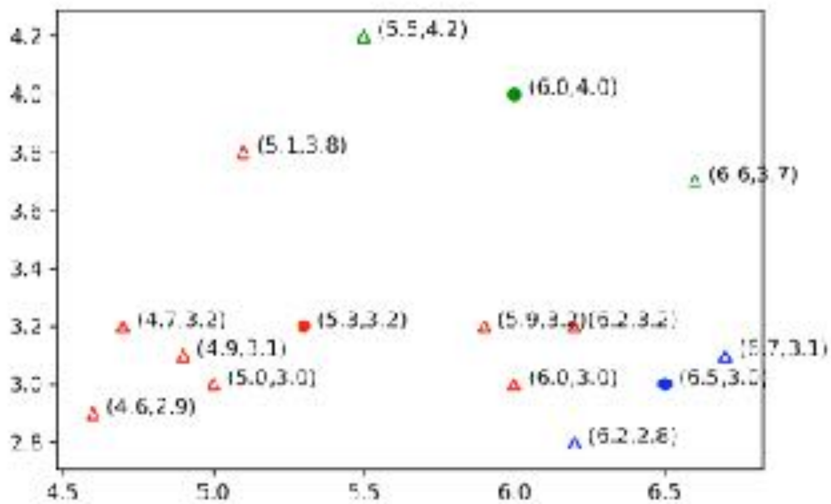
Points in Cluster 1 are : [(5.9, 3.2), (4.6, 2.9), (4.7, 3.2), (5.0, 3.0), (4.9, 3.1), (5.1, 3.8), (6.0, 3.0)]

Points in Cluster 2 are : [(5.5, 4.2)]

Points in Cluster 3 are : [(6.2, 2.8), (6.7, 3.1)]

Thus, the classification vector for [[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]] is as follows : **[1, 1, 3, 1, 2, 1, 1, 3, 1, 1]**

3.2 : RECOMPUTE CENTROIDS



Updated centroid values are as follows :

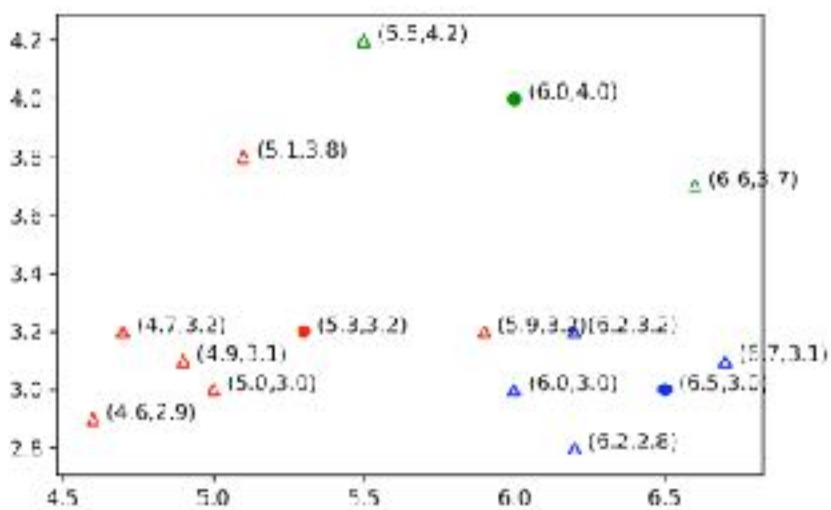
Centroid1 = [5.3, 3.2]

Centroid2 = [6.0, 4.0]

Centroid3 = [6.5, 3.0]

3.3 : SECOND ITERATION

A.



Classification Vector:

Points in Cluster 1 are : [(5.9, 3.2), (4.6, 2.9), (4.7, 3.2), (5.0, 3.0), (4.9, 3.1), (5.1, 3.8)]

Points in Cluster 2 are : [(5.5, 4.2), (6.6, 3.7)]

Points in Cluster 3 are : [(6.2, 2.8), (6.7, 3.1), (6.0, 3.0), (6.2, 3.2)]

Thus, the classification vector for [[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], [6.6, 3.7], [6.2, 3.2]] is as follows :

[1, 1, 3, 1, 2, 1, 1, 3, 1, 3, 2, 3]

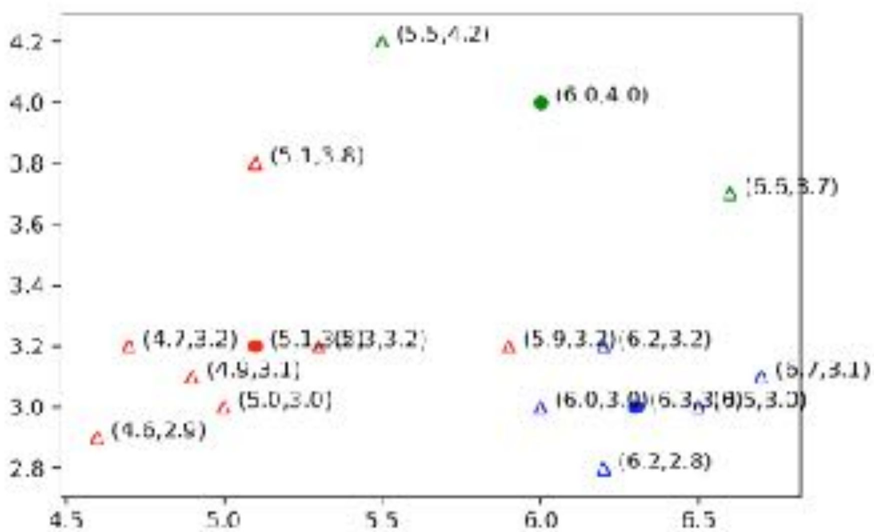
Centroid values for this classification are as follows :

Centroid1 = [5.3, 3.2]

Centroid2 = [6.0, 4.0]

Centroid3 = [6.5, 3.0]

B.



Classification Vector:

Points in Cluster 1 are : [(5.9, 3.2), (4.6, 2.9), (4.7, 3.2), (5.0, 3.0), (4.9, 3.1), (5.1, 3.8),(5.3, 3.2)]

Points in Cluster 2 are : [(5.5, 4.2), (6.6, 3.7), (6.0, 4.0)]

Points in Cluster 3 are : [(6.2, 2.8), (6.7, 3.1), (6.0, 3.0), (6.2, 3.2), (6.5, 3.0)]

Thus, the classification vector for [[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], [5.3, 3.2], [6.6, 3.7], [6.0, 4.0], [6.2, 3.2], [6.5, 3.0]] is as follows : **[1, 1, 3, 1, 2, 1, 1, 3, 1, 3]**

Updated Centroid values after this classification are as follows :

Centroid1 = [5.1, 3.2]

Centroid2 = [6.0, 4.0]

Centroid3 = [6.3, 3.0]

SOURCE CODE:

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

X = [[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]]
m1 = [6.2, 3.2]
m2 = [6.6, 3.7]
m3 = [6.5, 3.0]
colors = ["r", "g", "b"]

def FirstCluster():
```

```
    list1 = []
    list2 = []
    list3 = []
    m1_cluster = []
    m2_cluster = []
    m3_cluster = []
    for i in range(0,10):
        euc = math.sqrt(((X[i][0] - m1[0])*(X[i][0] - m1[0])) + ((X[i][1] - m1[1])*(X[i][1] - m1[1]))),math.sqrt(((X[i][0] - m2[0])*(X[i][0] - m2[0])) + ((X[i][1] - m2[1])*(X[i][1] - m2[1]))), math.sqrt(((X[i][0] - m3[0])*(X[i][0] - m3[0])) + ((X[i][1] - m3[1])*(X[i][1] - m3[1])))
        if(min(math.sqrt(((X[i][0] - m1[0])*(X[i][0] - m1[0])) + ((X[i][1] - m1[1])*(X[i][1] - m1[1]))),math.sqrt(((X[i][0] - m2[0])*(X[i][0] - m2[0])) + ((X[i][1] - m2[1])*(X[i][1] - m2[1]))), math.sqrt(((X[i][0] - m3[0])*(X[i][0] - m3[0])) + ((X[i][1] - m3[1])*(X[i][1] - m3[1])))) == euc[0]):
            m1_cluster = (X[i][0], X[i][1])
            list1.append(m1_cluster)
        elif(min(math.sqrt(((X[i][0] - m1[0])*(X[i][0] - m1[0])) + ((X[i][1] - m1[1])*(X[i][1] - m1[1]))),math.sqrt(((X[i][0] - m2[0])*(X[i][0] - m2[0])) + ((X[i][1] - m2[1])*(X[i][1] - m2[1]))), math.sqrt(((X[i][0] - m3[0])*(X[i][0] - m3[0])) + ((X[i][1] - m3[1])*(X[i][1] - m3[1])))) == euc[1]):
            m2_cluster = (X[i][0], X[i][1])
            list2.append(m2_cluster)
        elif(min(math.sqrt(((X[i][0] - m1[0])*(X[i][0] - m1[0])) + ((X[i][1] - m1[1])*(X[i][1] - m1[1]))),math.sqrt(((X[i][0] - m2[0])*(X[i][0] - m2[0])) + ((X[i][1] - m2[1])*(X[i][1] - m2[1]))), math.sqrt(((X[i][0] - m3[0])*(X[i][0] - m3[0])) + ((X[i][1] - m3[1])*(X[i][1] - m3[1])))) == euc[2]):
            m3_cluster = (X[i][0], X[i][1])
            list3.append(m3_cluster)
    plt.scatter(m1[0], m1[1], s = 30, color = colors[0], marker = "o")
    plt.scatter(m2[0], m2[1], s = 30, color = colors[1], marker = "o")
    plt.scatter(m3[0], m3[1], s = 30, color = colors[2], marker = "o")
    plt.text(m1[0], m1[1], "  (" + str(m1[0]) + "," + str(m1[1]) +")")
    plt.text(m2[0], m2[1], "  (" + str(m2[0]) + "," + str(m2[1]) +")")
    plt.text(m3[0], m3[1], "  (" + str(m3[0]) + "," + str(m3[1]) +")")
    for features1 in list1:
        plt.scatter(features1[0], features1[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[0])
        plt.text(features1[0], features1[1], "  (" + str(features1[0]) + "," + str(features1[1]) +")")
    for features2 in list2:
        plt.scatter(features2[0], features2[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[1])
        plt.text(features2[0], features2[1], "  (" + str(features2[0]) + "," + str(features2[1]) +")")
    for features3 in list3:
        plt.scatter(features3[0], features3[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[2])
        plt.text(features3[0], features3[1], "  (" + str(features3[0]) + "," + str(features3[1]) +")")
    plt.savefig("task3_iter1_a.jpg", dpi = 300)
    plt.close()
    list1.append(tuple(m1))
    list2.append(tuple(m2))
    list3.append(tuple(m3))
    ReComputeCentroid(list1, list2, list3, m1, m2, m3, False, False)
m1_old = [0,0]
m2_old = [0,0]
m3_old = [0,0]
def ReComputeCentroid(list1, list2, list3, m1, m2, m3, flag, figStatus):
    list1.sort(key=lambda t: t[0])
    list2.sort(key=lambda t: t[0])
    list3.sort(key=lambda t: t[0])
    m1_old[0] = np.round(m1[0],2)
    m1_old[1] = np.round(m1[1],2)
    m2_old[0] = np.round(m2[0],2)
    m2_old[1] = np.round(m2[1],2)
    m3_old[0] = np.round(m3[0],2)
```

```
    m3_old[1] = np.round(m3[1],2)
    a = np.mean(list1,axis=0)
    m1[0] = np.round(a[0],1)
    m1[1] = np.round(a[1],1)
    b = np.mean(list2,axis=0)
    m2[0] = np.round(b[0],1)
    m2[1] = np.round(b[1],1)
    c = np.mean(list3,axis=0)
    m3[0] = np.round(c[0],1)
    m3[1] = np.round(c[1],1)
    for i in list1:
        if(np.array_equal(i,m1)):
            list1.remove(i)
    for j in list2:
        if(np.array_equal(j,m2)):
            list2.remove(j)
    for k in list3:
        if(np.array_equal(k,m3)):
            list3.remove(k)
    plt.scatter(m1[0], m1[1], s = 30, color = colors[0], marker = "o")
    plt.scatter(m2[0], m2[1], s = 30, color = colors[1], marker = "o")
    plt.scatter(m3[0], m3[1], s = 30, color = colors[2], marker = "o")
    plt.text(m1[0], m1[1], "  (" + str(m1[0]) + "," + str(m1[1]) +")")
    plt.text(m2[0], m2[1], "  (" + str(m2[0]) + "," + str(m2[1]) +")")
    plt.text(m3[0], m3[1], "  (" + str(m3[0]) + "," + str(m3[1]) +")")
    for features1 in list1:
        plt.scatter(features1[0], features1[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[0])
        plt.text(features1[0], features1[1], "  (" + str(features1[0]) + "," + str(features1[1]) +")")
    for features2 in list2:
        plt.scatter(features2[0], features2[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[1])
        plt.text(features2[0], features2[1], "  (" + str(features2[0]) + "," + str(features2[1]) +")")
    for features3 in list3:
        plt.scatter(features3[0], features3[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[2])
        plt.text(features3[0], features3[1], "  (" + str(features3[0]) + "," + str(features3[1]) +")")
    if(figStatus):
        print(m1,m2,m3)
        plt.savefig("task3_iter2_b.jpg", dpi = 300)
        plt.close()
    else:
        plt.savefig("task3_iter1_b.jpg", dpi = 300)
        plt.close()
    if(not flag):
        ReClustering(m1, m2, m3, m1_old, m2_old, m3_old)

def ReClustering(m1, m2, m3, m1_old, m2_old, m3_old):
    list1 = []
    list2 = []
    list3 = []
    m1_cluster = []
    m2_cluster = []
    m3_cluster = []
    X.append(np.round(m1_old,1))
    X.append(np.round(m2_old,1))
    X.append(np.round(m3_old,1))
    for i in range(0,13):
        euc = math.sqrt(((X[i][0] - m1[0])*(X[i][0] - m1[0])) + ((X[i][1] - m1[1])*(X[i][1] - m1[1]))),math.sqrt(((X[i][0] -
m2[0])*(X[i][0] - m2[0])) + ((X[i][1] - m2[1])*(X[i][1] - m2[1]))), math.sqrt(((X[i][0] - m3[0])*(X[i][0] - m3[0])) + ((X[i][1] -
m3[1])*(X[i][1] - m3[1])))
```

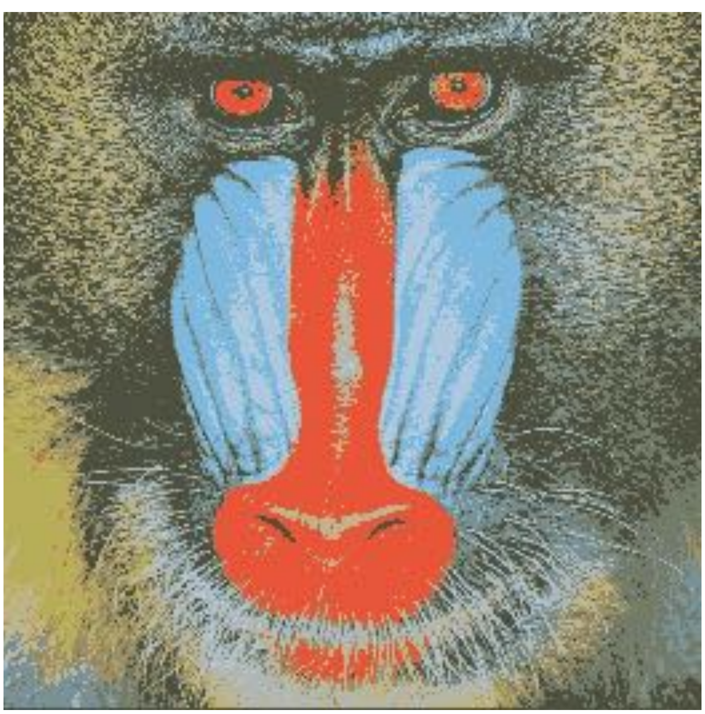
```
        if(min(math.sqrt(((X[i][0] - m1[0])*(X[i][0] - m1[0])) + ((X[i][1] - m1[1])*(X[i][1] - m1[1]))),math.sqrt(((X[i][0] -
m2[0])*(X[i][0] - m2[0])) + ((X[i][1] - m2[1])*(X[i][1] - m2[1]))), math.sqrt(((X[i][0] - m3[0])*(X[i][0] - m3[0])) + ((X[i][1] -
m3[1])*(X[i][1] - m3[1])))) == euc[0]):
            m1_cluster = (X[i][0], X[i][1])
            list1.append(m1_cluster)
        elif(min(math.sqrt(((X[i][0] - m1[0])*(X[i][0] - m1[0])) + ((X[i][1] - m1[1])*(X[i][1] - m1[1]))),math.sqrt(((X[i][0] -
m2[0])*(X[i][0] - m2[0])) + ((X[i][1] - m2[1])*(X[i][1] - m2[1]))), math.sqrt(((X[i][0] - m3[0])*(X[i][0] - m3[0])) + ((X[i][1] -
m3[1])*(X[i][1] - m3[1])))) == euc[1]):
            m2_cluster = (X[i][0], X[i][1])
            list2.append(m2_cluster)
        elif(min(math.sqrt(((X[i][0] - m1[0])*(X[i][0] - m1[0])) + ((X[i][1] - m1[1])*(X[i][1] - m1[1]))),math.sqrt(((X[i][0] -
m2[0])*(X[i][0] - m2[0])) + ((X[i][1] - m2[1])*(X[i][1] - m2[1]))), math.sqrt(((X[i][0] - m3[0])*(X[i][0] - m3[0])) + ((X[i][1] -
m3[1])*(X[i][1] - m3[1])))) == euc[2]):
            m3_cluster = (X[i][0], X[i][1])
            list3.append(m3_cluster)
    for i in list1:
        if(np.array_equal(i,m1)):
            list1.remove(i)
    for j in list2:
        if(np.array_equal(j,m2)):
            list2.remove(j)
    for k in list3:
        if(np.array_equal(k,m3)):
            list3.remove(k)
    plt.scatter(m1[0], m1[1], s = 30, color = colors[0], marker = "o")
    plt.scatter(m2[0], m2[1], s = 30, color = colors[1], marker = "o")
    plt.scatter(m3[0], m3[1], s = 30, color = colors[2], marker = "o")
    plt.text(m1[0], m1[1], "  (" + str(m1[0]) + "," + str(m1[1]) +")")
    plt.text(m2[0], m2[1], "  (" + str(m2[0]) + "," + str(m2[1]) +")")
    plt.text(m3[0], m3[1], "  (" + str(m3[0]) + "," + str(m3[1]) +")")
    for features1 in list1:
        plt.scatter(features1[0], features1[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[0])
        plt.text(features1[0], features1[1], "  (" + str(features1[0]) + "," + str(features1[1]) +")")
    for features2 in list2:
        plt.scatter(features2[0], features2[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[1])
        plt.text(features2[0], features2[1], "  (" + str(features2[0]) + "," + str(features2[1]) +")")
    for features3 in list3:
        plt.scatter(features3[0], features3[1], s = 30, marker = "^", facecolors='none', edgecolors=colors[2])
        plt.text(features3[0], features3[1], "  (" + str(features3[0]) + "," + str(features3[1]) +")")
    plt.savefig("task3_iter2_a.jpg", dpi = 300)
    plt.close()
    list1.append(tuple(m1))
    list2.append(tuple(m2))
    list3.append(tuple(m3))
    ReComputeCentroid(list1, list2, list3, m1, m2, m3, True, True)
```

3.4 : COLOR QUANTIZATION

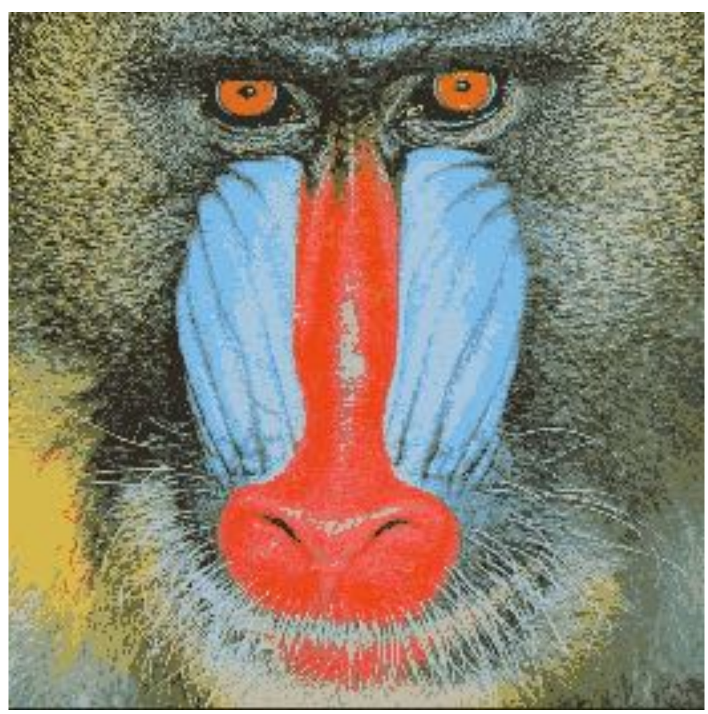
k = 3:

k = 5:

k = 10:



k = 20:



SOURCE CODE :

I have referred to following website to calculate the luminance:
https://stackoverflow.com/questions/26765875/python-pil-compare-colors/26768008

```
import cv2
import numpy as np
import math
from numpy import mean

UBIT = 'mkulkarn'
np.random.seed(sum([ord(c) for c in UBIT]))
img1 = cv2.imread("/Users/manasikulkarni/Desktop/baboon.jpg")
img_copy3 = cv2.imread("/Users/manasikulkarni/Desktop/baboon.jpg")
img_copy5 = cv2.imread("/Users/manasikulkarni/Desktop/baboon.jpg")
img_copy10 = cv2.imread("/Users/manasikulkarni/Desktop/baboon.jpg")
img_copy20 = cv2.imread("/Users/manasikulkarni/Desktop/baboon.jpg")
height, width, channels = img1.shape
centroids3 = np.array([[0 for i in range(3)] for j in range(3)],dtype = np.int)
centroids5 = np.array([[0 for i in range(3)] for j in range(5)],dtype = np.int)
centroids10 = np.array([[0 for i in range(3)] for j in range(10)],dtype = np.int)
centroids20 = np.array([[0 for i in range(3)] for j in range(20)],dtype = np.int)
for i in range(0,3):
    rand1 = np.random.uniform(0,width-2,1)
    rand2 = np.random.uniform(0,height-2,1)
    centroids3[i] = img1[int(rand1[0])][int(rand2[0])]
for i in range(0,5):
    rand1 = np.random.uniform(0,width-2,1)
    rand2 = np.random.uniform(0,height-2,1)
    centroids5[i] = img1[int(rand1[0])][int(rand2[0])]
for i in range(0,10):
    rand1 = np.random.uniform(0,width-2,1)
    rand2 = np.random.uniform(0,height-2,1)
    centroids10[i] = img1[int(rand1[0])][int(rand2[0])]
for i in range(0,20):
    rand1 = np.random.uniform(0,width-2,1)
    rand2 = np.random.uniform(0,height-2,1)
    centroids20[i] = img1[int(rand1[0])][int(rand2[0])]

def checkLuminance(point):
    return (0.299 * point[0] + 0.587 * point[1] + 0.114 * point[2])

def checkSimilarity(pt1, pt2, threshold):
    return abs(checkLuminance(pt1) - checkLuminance(pt2)) < threshold

threshold = 10
def callColor3():
    Color3(centroids3[0],centroids3[1],centroids3[2])
def callColor5():
    Color5(centroids5[0],centroids5[1],centroids5[2],centroids5[3],centroids5[4])
def callColor10():
Color10(centroids10[0],centroids10[1],centroids10[2],centroids10[3],centroids10[4],centroids10[5],centroids10[6],centroids10[7
],centroids10[8],centroids10[9])
def callColor20():
    Color20(centroids20[0], centroids20[1], centroids20[2], centroids20[3], centroids20[4], centroids20[5], centroids20[6],
centroids20[7], centroids20[8], centroids20[9],
```

```
centroids20[10],centroids20[11],centroids20[12],centroids20[13],centroids20[14],centroids20[15],centroids20[16],centroids20[1
7],centroids20[18],centroids20[19])

list1 = []
list2 = []
list3 = []
def Color3(centroids31, centroids32, centroids33):
    for x in range(0, height):
        for y in range(0, width):
            euc1 = math.sqrt((img1[x][y][0] - centroids31[0])*(img1[x][y][0] - centroids31[0]) + (img1[x][y][1] -
centroids31[1])*(img1[x][y][1] - centroids31[1]) + (img1[x][y][2] - centroids31[2])*(img1[x][y][2] - centroids31[2]))
            euc2 = math.sqrt((img1[x][y][0] - centroids32[0])*(img1[x][y][0] - centroids32[0]) + (img1[x][y][1] -
centroids32[1])*(img1[x][y][1] - centroids32[1]) + (img1[x][y][2] - centroids32[2])*(img1[x][y][2] - centroids32[2]))
            euc3 = math.sqrt((img1[x][y][0] - centroids33[0])*(img1[x][y][0] - centroids33[0]) + (img1[x][y][1] -
centroids33[1])*(img1[x][y][1] - centroids33[1]) + (img1[x][y][2] - centroids33[2])*(img1[x][y][2] - centroids33[2]))
            if(euc1 == min(euc1, euc2, euc3)):
                r1,g1,b1 = centroids31
                img_copy3[x][y] = (r1,g1,b1)
                list1.append(img1[x][y])
            elif(euc2 == min(euc1, euc2, euc3)):
                r1,g1,b1 = centroids32
                img_copy3[x][y] = (r1,g1,b1)
                list2.append(img1[x][y])
            elif(euc3 == min(euc1, euc2, euc3)):
                r1,g1,b1 = centroids33
                img_copy3[x][y] = (r1,g1,b1)
                list3.append(img1[x][y])
    ReComputeCentroids3(img_copy3, list1,list2, list3, centroids3[0],centroids3[1],centroids3[2])

centroids31_old = np.array([0 for i in range(3)],dtype = np.int)
centroids32_old = np.array([0 for i in range(3)],dtype = np.int)
centroids33_old = np.array([0 for i in range(3)],dtype = np.int)
def ReComputeCentroids3(img_copy3, list1,list2, list3, centroids31,centroids32,centroids33):
    centroids31_old[0] = centroids31[0]
    centroids31_old[1] = centroids31[1]
    centroids31_old[2] = centroids31[2]
    centroids32_old[0] = centroids32[0]
    centroids32_old[1] = centroids32[1]
    centroids32_old[2] = centroids32[2]
    centroids33_old[0] = centroids33[0]
    centroids33_old[1] = centroids33[1]
    centroids33_old[2] = centroids33[2]
    a = mean(list1,axis=0)
    centroids31 = a
    b = mean(list2,axis=0)
    centroids32 = b
    c = mean(list3,axis=0)
    centroids33 = c
    if not checkSimilarity(centroids31_old, centroids31, threshold) or not(checkSimilarity(centroids32_old, centroids32,
threshold)) or not(checkSimilarity(centroids33_old, centroids33, threshold)):
        Color3(centroids31,centroids32,centroids33)
        return

    cv2.imwrite("task3_baboon_3.jpg", img_copy3)
```

Similar code is written for k=5, k=10 and k=20

3.5 : GAUSSIAN MIXTURE MODEL

A. The means after first iteration are :

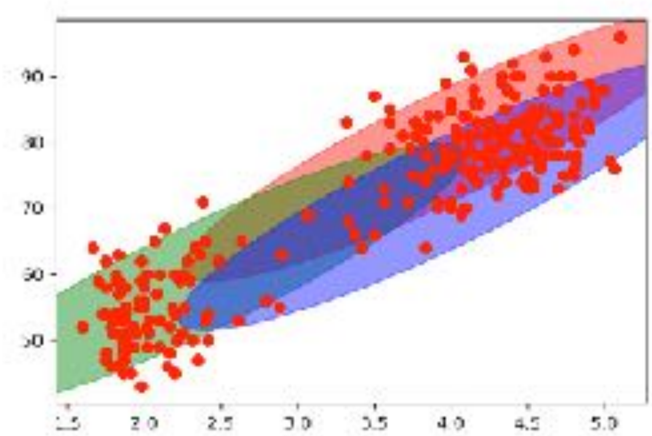
Mean1 = [5.316507899024571, 3.2152729183353053]
Mean2 = [5.611297951076313, 3.385053105024623]
Mean3 = [5.604435653845083, 3.144200610784218]

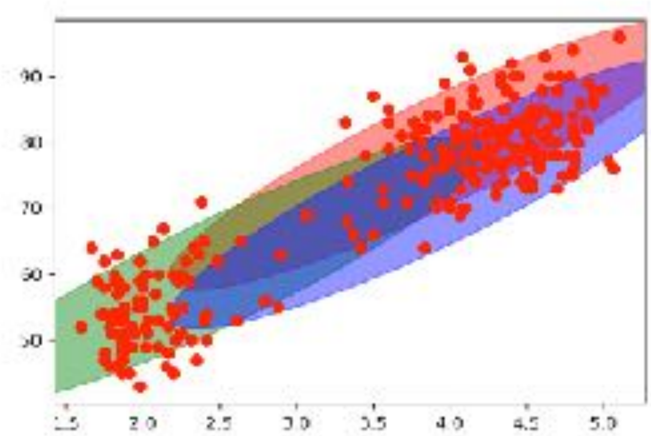
SOURCE CODE:

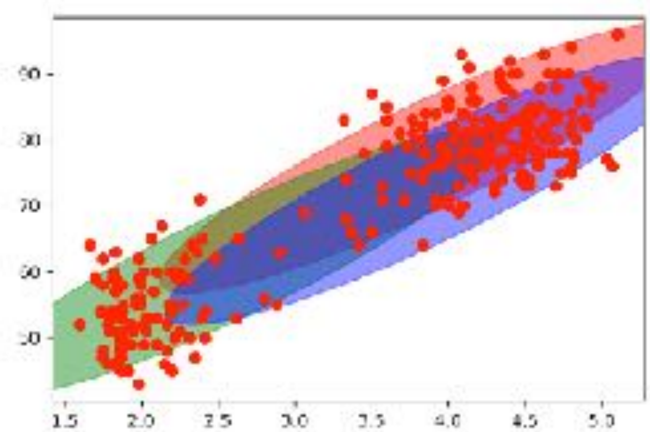
```
import cv2
from scipy.stats import multivariate_normal

X = [[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]]
mean1 = [6.2, 3.2]
mean2 = [6.6, 3.7]
mean3 = [6.5, 3.0]
cov1 = [[0.5, 0],[0, 0.5]]
cov2 = [[0.5, 0],[0, 0.5]]
cov3 = [[0.5, 0],[0, 0.5]]
pi1 = 1/3
pi2 = 1/3
pi3 = 1/3
N1 = multivariate_normal.pdf(X, mean1, cov1)
N2 = multivariate_normal.pdf(X, mean2, cov2)
N3 = multivariate_normal.pdf(X, mean3, cov3)
Numerator1 = N1 * pi1
Numerator2 = N2 * pi2
Numerator3 = N3 * pi3
Denominator = (pi1 * N1) + (pi2 * N2) + (pi3 * N3)
ResponsibilityMat1 = Numerator1 / Denominator
ResponsibilityMat2 = Numerator2 / Denominator
ResponsibilityMat3 = Numerator3 / Denominator
def ReComputeMeanValues():
    list1 = []
    list2 = []
    for i in X:
        list1.append(i[0])
        list2.append(i[1])

    a = []
    b = []
    c = []
    d = []
    e = []
    f = []
    for i in range(0,10):
        a.append(list1[i] * ResponsibilityMat1[i])
        b.append(list2[i] * ResponsibilityMat1[i])
        c.append(list1[i] * ResponsibilityMat2[i])
        d.append(list2[i] * ResponsibilityMat2[i])
        e.append(list1[i] * ResponsibilityMat3[i])
        f.append(list2[i] * ResponsibilityMat3[i])

    mean1_new = [sum(a) / sum(ResponsibilityMat1),sum(b) / sum(ResponsibilityMat1)]
    mean2_new = [sum(c) / sum(ResponsibilityMat2),sum(d) / sum(ResponsibilityMat2)]
    mean3_new = [sum(e) / sum(ResponsibilityMat3),sum(f) / sum(ResponsibilityMat3)]
```

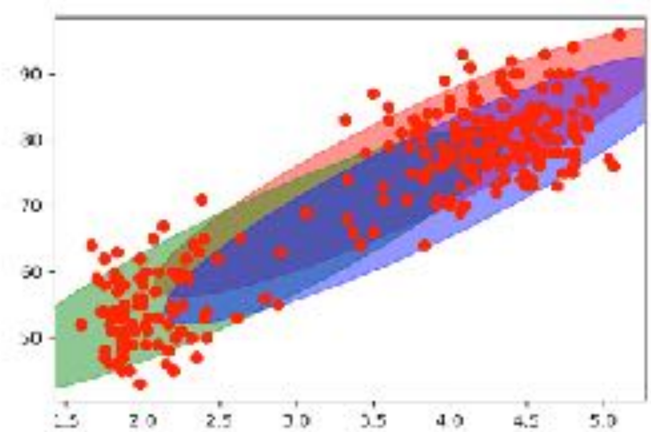
```
print("Recomputed Mean1 : ", mean1_new)
print("Recomputed Mean2 : ", mean2_new)
print("Recomputed Mean3 : ", mean3_new)
```

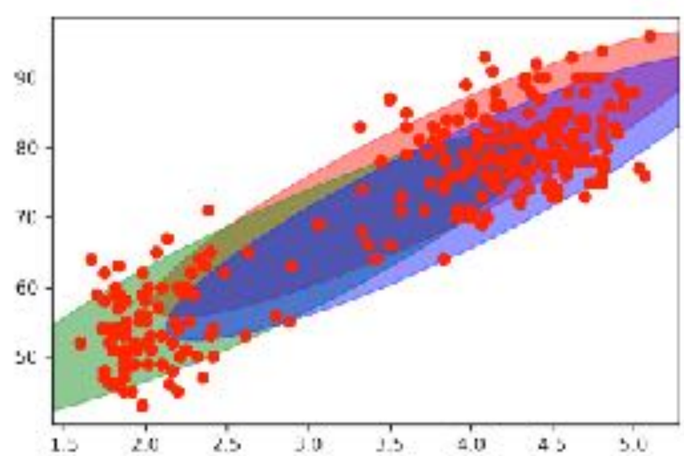
B. GMM TO OLD FAITHFUL DATASET OUTPUT:











SOURCE CODE:

```
import cv2
import numpy as np
from scipy.stats import multivariate_normal
#import matplotlib.pyplot as plt
#from matplotlib.patches import Ellipse

eruptions = np.genfromtxt('old-faithful.csv', delimiter=',', skip_header=1, usecols=(1))
waiting = np.genfromtxt('old-faithful.csv', delimiter=',', skip_header=1, usecols=(2))
def callGMM():
    X = []
    for i in range(len(eruptions)):
        X.append((eruptions[i],waiting[i]))
    mean1 = [4.0, 81]
    mean2 = [2.0, 57]
    mean3 = [4.0, 71]
    cov1 = [[1.30, 13.98],[13.98, 184.82]]
    cov2 = [[1.30, 13.98],[13.98, 184.82]]
    cov3 = [[1.30, 13.98],[13.98, 184.82]]
    pi1 = 0.33
    pi2 = 0.33
    pi3 = 0.33
    for x in range (0,5):
        N1 = multivariate_normal.pdf(X, mean1, cov1)
        N2 = multivariate_normal.pdf(X, mean2, cov2)
        N3 = multivariate_normal.pdf(X, mean3, cov3)
        Numerator1 = N1 * pi1
        Numerator2 = N2 * pi2
        Numerator3 = N3 * pi3
        Denominator = (pi1 * N1) + (pi2 * N2) + (pi3 * N3)
        ResponsibilityMat1 = Numerator1 / Denominator
        ResponsibilityMat2 = Numerator2 / Denominator
        ResponsibilityMat3 = Numerator3 / Denominator
        list1 = []
        list2 = []
        for i in eruptions:
            list1.append(i)
        for j in waiting:
            list2.append(j)
        a = []
        b = []
        c = []
        d = []
        e = []
        f = []
        for i in range(0,272):
            a.append(list1[i] * ResponsibilityMat1[i])
            b.append(list2[i] * ResponsibilityMat1[i])
            c.append(list1[i] * ResponsibilityMat2[i])
            d.append(list2[i] * ResponsibilityMat2[i])
            e.append(list1[i] * ResponsibilityMat3[i])
            f.append(list2[i] * ResponsibilityMat3[i])
        mean1 = [sum(a) / sum(ResponsibilityMat1),sum(b) / sum(ResponsibilityMat1)]
        mean2 = [sum(c) / sum(ResponsibilityMat2),sum(d) / sum(ResponsibilityMat2)]
        mean3 = [sum(e) / sum(ResponsibilityMat3),sum(f) / sum(ResponsibilityMat3)]
        a1 = []
        b1 = []
```

```
    c1 = []
    d1 = []
    e1 = []
    f1 = []
    for i in range(0,272):
        a1.append(ResponsibilityMat1[i] * ((list1[i] - mean1[0])**2))
        b1.append(ResponsibilityMat1[i] * ((list2[i] - mean1[1])**2))
        c1.append(ResponsibilityMat2[i] * ((list1[i] - mean2[0])**2))
        d1.append(ResponsibilityMat2[i] * ((list2[i] - mean2[1])**2))
        e1.append(ResponsibilityMat3[i] * ((list1[i] - mean3[0])**2))
        f1.append(ResponsibilityMat3[i] * ((list2[i] - mean3[1])**2))
    cov1 = [sum(a1) / sum(ResponsibilityMat1), sum(b1) / sum(ResponsibilityMat1)]
    cov2 = [sum(c1) / sum(ResponsibilityMat2), sum(d1) / sum(ResponsibilityMat2)]
    cov3 = [sum(e1) / sum(ResponsibilityMat3), sum(f1) / sum(ResponsibilityMat3)]
    pi1 = sum(ResponsibilityMat1) / len(X)
    pi2 = sum(ResponsibilityMat2) / len(X)
    pi3 = sum(ResponsibilityMat3) / len(X)
    x1, y1 = np.transpose(X)
    plt.plot(x1, y1, 'ro')

    plot_cov_ellipse(np.cov(X,rowvar = False), mean1, nstd=1.5, alpha = 0.5, color = 'r')
    plot_cov_ellipse(np.cov(X,rowvar = False), mean2, nstd=1.5, alpha = 0.5, color = 'g')
    plot_cov_ellipse(np.cov(X,rowvar = False), mean3, nstd=1.5, alpha = 0.5, color = 'b')
    plt.savefig("task3_gmm_iter"+str(x)+".jpg", dpi = 300)
    plt.close()
    print("Recomputed Means : ", mean1, mean2, mean3)
    print("Recomputed Weights : ", pi1, pi2, pi3)
```

Included a file(task3.py) in which all these functions for task 3 are called:

```
import task3_1 as T3_1
import task3_4 as T3_4
import task3_5 as T3_5
import task3_5_2 as T3_5_2

# Task 3.1, 3.2, 3.3 functions
T3_1.FirstCluster()
# Task 3.4 functions
T3_4.callColor3()
T3_4.callColor5()
T3_4.callColor10()
T3_4.callColor20()
# Task 3.5_1 functions
T3_5.ReComputeMeanValues()
# Task 3.5_1 functions
T3_5_2.callGMM()
```