# Project 2 - report

## Task 1 – Image Features and Homography

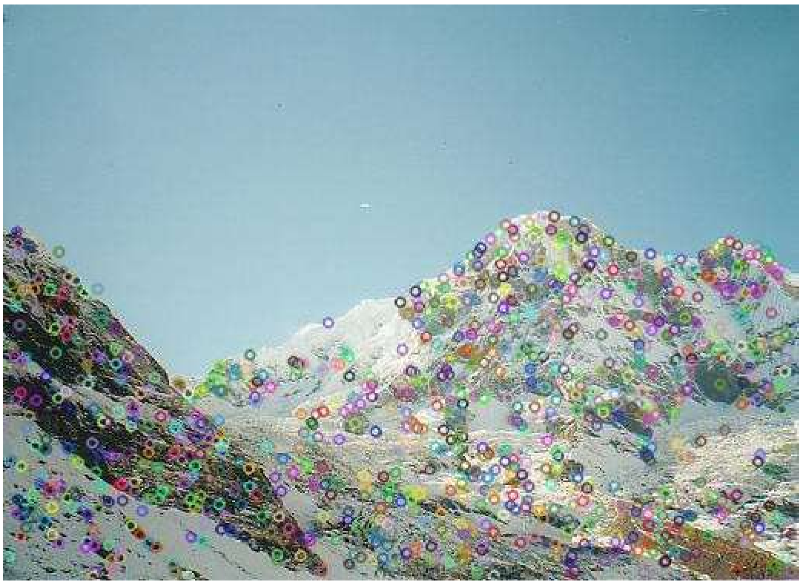
Code :

```
 import numpy as np
import cv2 as cv

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

# read images
img1 = cv.imread('mountain1.jpg',1)
img2 = cv.imread('mountain2.jpg',1)


# key point detection (1)
sift = cv.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
sift_1=cv.drawKeypoints(img1,kp1,None)
sift_2=cv.drawKeypoints(img2,kp2,None)
cv.imwrite('task1_sift1.jpg',sift_1)
cv.imwrite('task1_sift2.jpg',sift_2)


# BFMatcher with default params
bf = cv.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)


# Apply ratio test
good = []
good_list = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])
        good_list.append(m)

# cv.drawMatchesKnn expects list of lists as matches.
img3 = cv.drawMatches(img1,kp1,img2,kp2,good_list,None,flags=2)

src_pts = np.float32([ kp1[m.queryIdx].pt for m in good_list ]).reshape(-1,1,2)
dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good_list ]).reshape(-1,1,2)

M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)

matchesMask = mask.ravel().tolist()
```

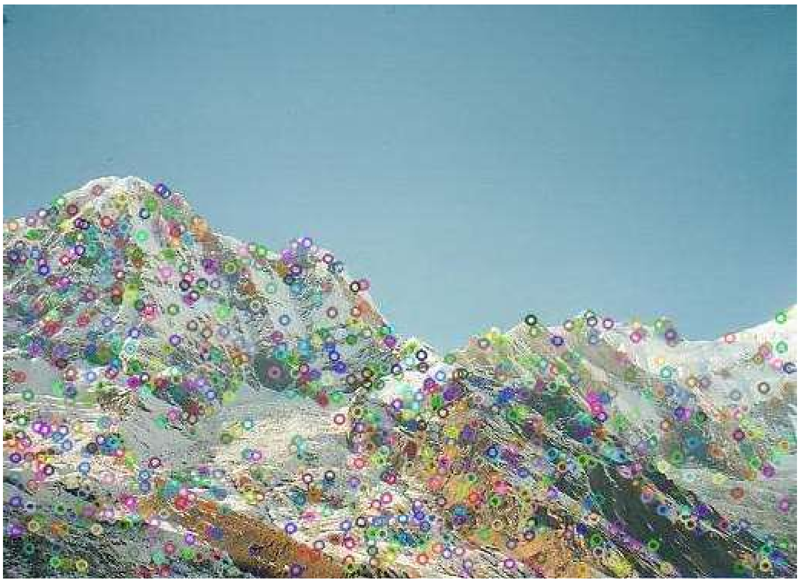
```
indices = [i for i, x in enumerate(matchesMask) if x == 1]
random_indices = np.random.randint(low=0, high=len(indices), size=10)
random_list = []
for i in range(len(random_indices)):
        random_list.append(good_list[random_indices[i]])
print("random_list - ",random_list)

h,w,c = img1.shape
pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
#dst = cv.perspectiveTransform(pts,M)

img5 = cv.warpPerspective(img1, M, (img2.shape[1] + h,img2.shape[0]))

#print(mask)

draw_params = dict(matchColor = (0,0,0), # draw matches in green color
                   singlePointColor = None,
                   flags = 2)

img4 = cv.drawMatches(img1,kp1,img2,kp2,random_list,None,**draw_params)

# img5 = cv.polylines(img2,[np.int32(dst)],True,255,3, cv.LINE_AA)


print(M)

cv.imwrite('task1_matches_knn.jpg',img3)

cv.imwrite('task1_matches.jpg',img4)

# cv.imwrite('task1_pano.jpg',img5)

def warpImages(img1, img2, H):
    h1,w1 = img1.shape[:2]
    h2,w2 = img2.shape[:2]
    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)
    pts2_ = cv.perspectiveTransform(pts2, H)
    pts = np.concatenate((pts1, pts2_), axis=0)
    [min_x, min_y] = np.int32(pts.min(axis=0).ravel() - 0.5)
    [xmax, ymax] = np.int32(pts.max(axis=0).ravel() + 0.5)
    t = [-min_x,-min_y]
    Ht = np.array([[1,0,t[0]],[0,1,t[1]],[0,0,1]]) # translate

    wrap = cv.warpPerspective(img2, Ht.dot(H), (xmax-min_x, ymax-min_y))
    wrap[t[1]:h1+t[1],t[0]:w1+t[0]] = img1
    return wrap

wrap = warpImages(img2, img1, M)
cv.imwrite('task1_pano.jpg',wrap)
```

Output:

1.1 [task1_sift1.jpg]

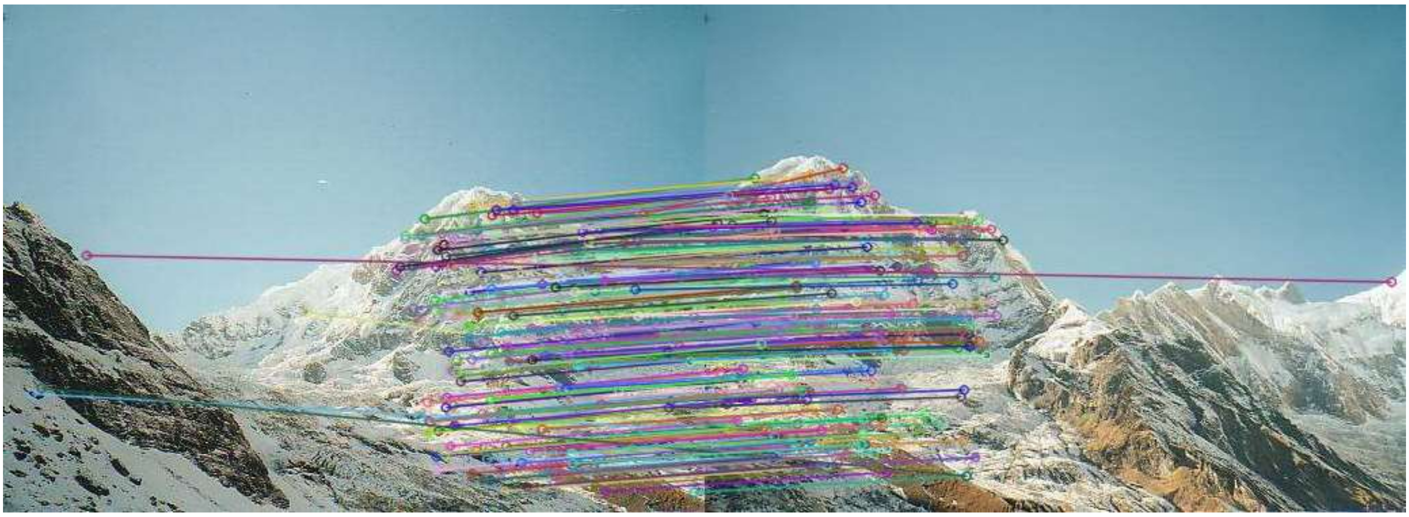


1.1 [task1_sift2.jpg]



1.2 [task1_matches_knn.jpg]

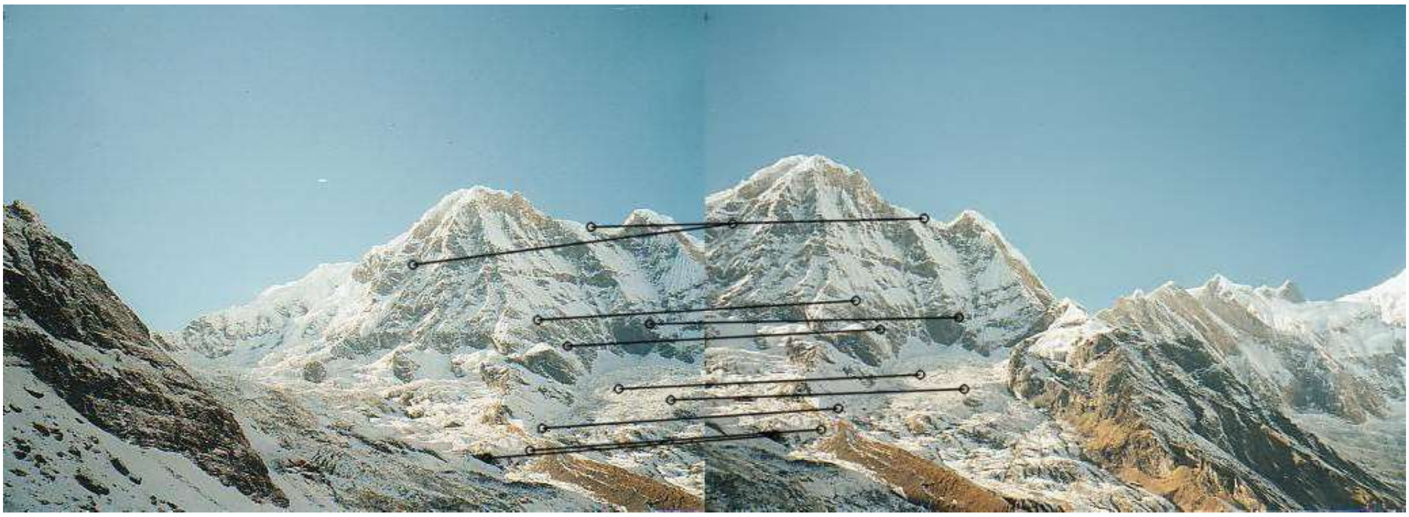
All matching key points are drawn.



1.3 [ Homography matrix]

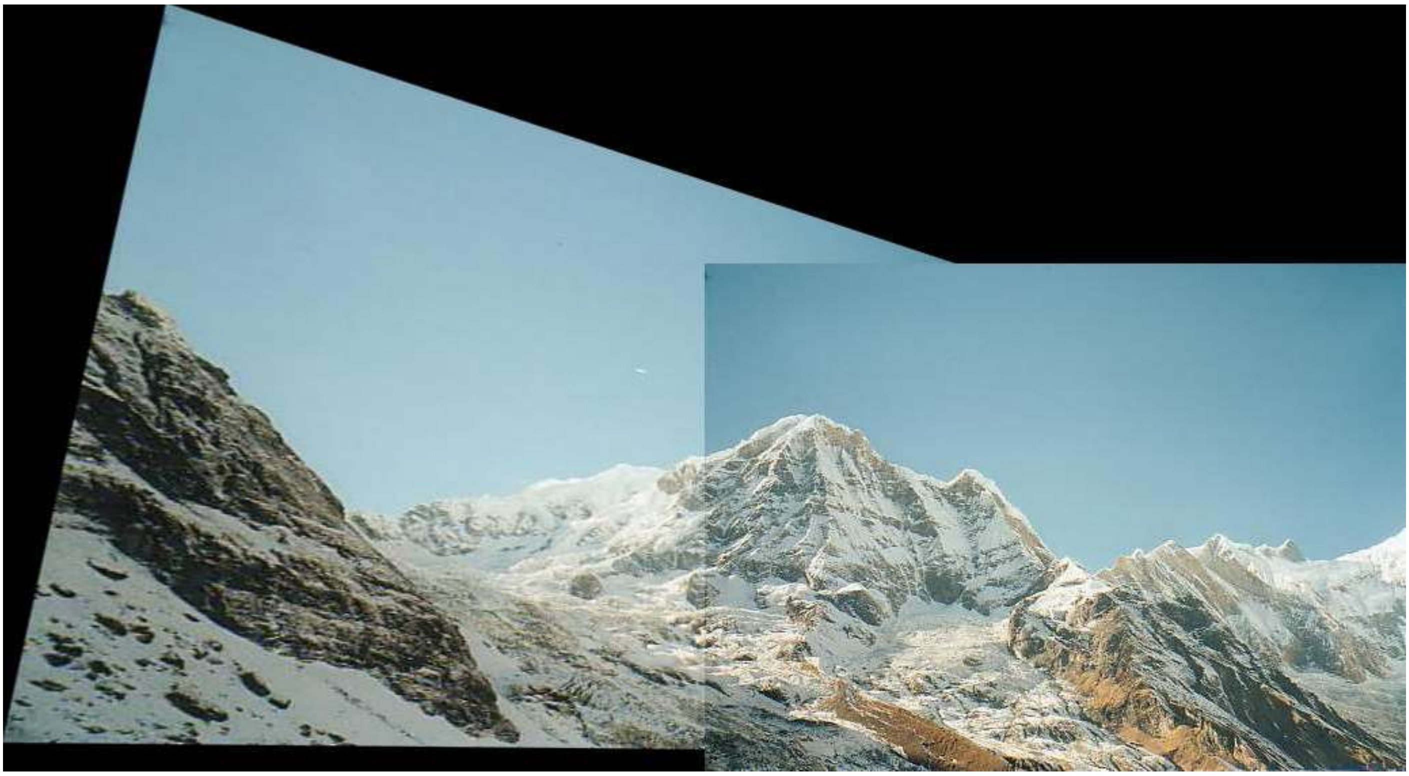
```
[[ 1.59244390e+00   -2.92375000e-01      -3.96659544e+02]

 [ 4.50346484e-01     1.43504596e+00      -1.91264661e+02]

 [ 1.21747684e-03   -5.85892685e-05       1.00000000e+00]]
```

1.4 [ task1_matches.jpg ]



1.5 [task1_pano.jpg]

Warping two images



References:

https://docs.opencv.org/3.0-beta/modules/cudawarping/doc/warping.html

Task 2 - Epipolar Geometry

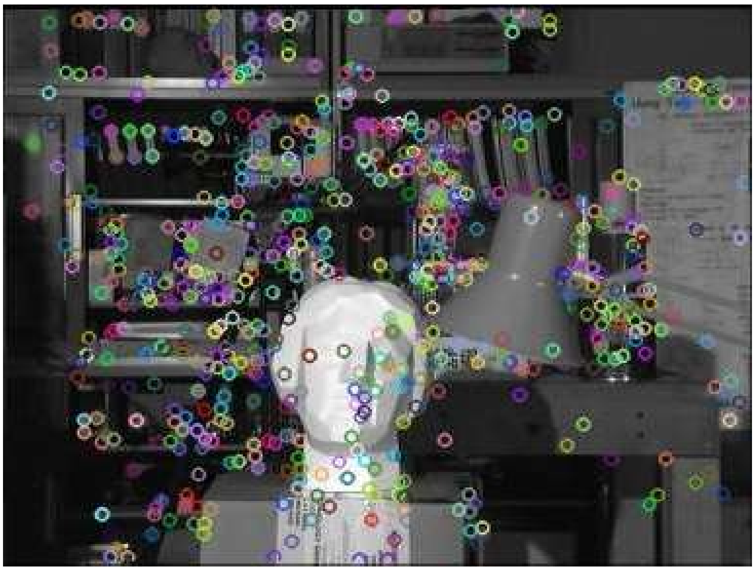
Code:

```
import numpy as np
import cv2 as cv



# read images
img_left = cv.imread('tsucuba_left.png',1)
img_right = cv.imread('tsucuba_right.png',1)

UBIT = 'ruturajt'

# key point detection (1)
sift = cv.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img_left,None)
kp2, des2 = sift.detectAndCompute(img_right,None)
sift_1=cv.drawKeypoints(img_left,kp1,None)
sift_2=cv.drawKeypoints(img_right,kp2,None)
cv.imwrite('task2_sift1.jpg',sift_1)
cv.imwrite('task2_sift2.jpg',sift_2)

# BFMatcher with default params
bruteForceMatcher = cv.BFMatcher()
matches = bruteForceMatcher.knnMatch(des1,des2, k=2)

# finding good matches
good_matches = []
good_matches_list = []
pts1 = []
pts2 = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good_matches.append([m])
        good_matches_list.append(m)
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)

# cv.drawMatchesKnn expects list of lists as matches.
img3 = cv.drawMatches(img_left,kp1,img_right,kp2,good_matches_list,None,flags=2)

cv.imwrite('task2_matches_knn.jpg',img3)

pts1 = np.int32(pts1)
pts2 = np.int32(pts2)
Fundamental_matrix, mask = cv.findFundamentalMat(pts1,pts2,cv.RANSAC)


# task 2.2 finding fundamental matrix
print(Fundamental_matrix)
```

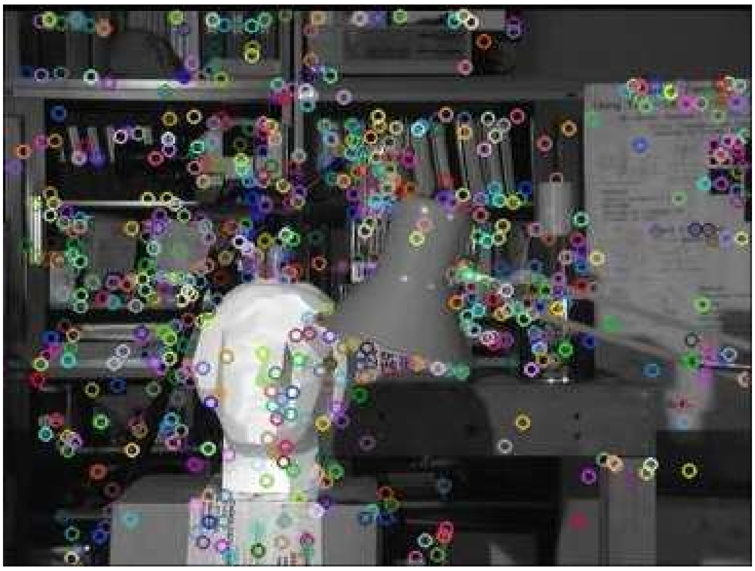
```
# selecting only inliners
pts1 = pts1[mask.ravel()==1]
pts2 = pts2[mask.ravel()==1]

np.random.seed(sum([ord(c) for c in UBIT]))
random_indices = np.random.randint(low=0, high=len(pts2), size=11)

# finding random indices for colors
clr = []
for i in range(11):
        clr.append(np.random.randint(0, 255, 3))
print("clr-",clr)

# defining new points
new_pts1 = []
new_pts2 = []
for i in random_indices:
        new_pts1.append(pts1[i])
        new_pts2.append(pts2[i])

def drawEpilines(img1, img2, lines, pts1, pts2):

   r,c = img1.shape[:2]
   for r, col in zip(lines, clr):
      color = tuple(col.tolist())
      x_0,y_0 = map(int, [0, -r[2]/r[1] ])
      x_1,y_1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
      img1 = cv.line(img1, (x_0,y_0), (x_1,y_1), color,1)

   return img1,img2
# finding epi polar line on right image
line_right = cv.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2, Fundamental_matrix)
line_right = line_right.reshape(-1, 3)
img4, img5 = drawEpilines(img_left, img_right, line_right, new_pts1, new_pts2)

# finding epipolar line on left image
line_left = cv.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1, Fundamental_matrix)
line_left = line_left.reshape(-1, 3)
img6, img7 = drawEpilines(img_right, img_left,line_left, new_pts2, new_pts1)

cv.imwrite('task2_epi_left.jpg', img4)
cv.imwrite('task2_epi_right.jpg', img6)


# creating stereo object
stereo_obj = cv.StereoSGBM_create(numDisparities=64, blockSize=12)
disparity_image = stereo_obj.compute(img_right,img_left)
cv.imwrite('task2_disparity.jpg',disparity_image)
```

Output

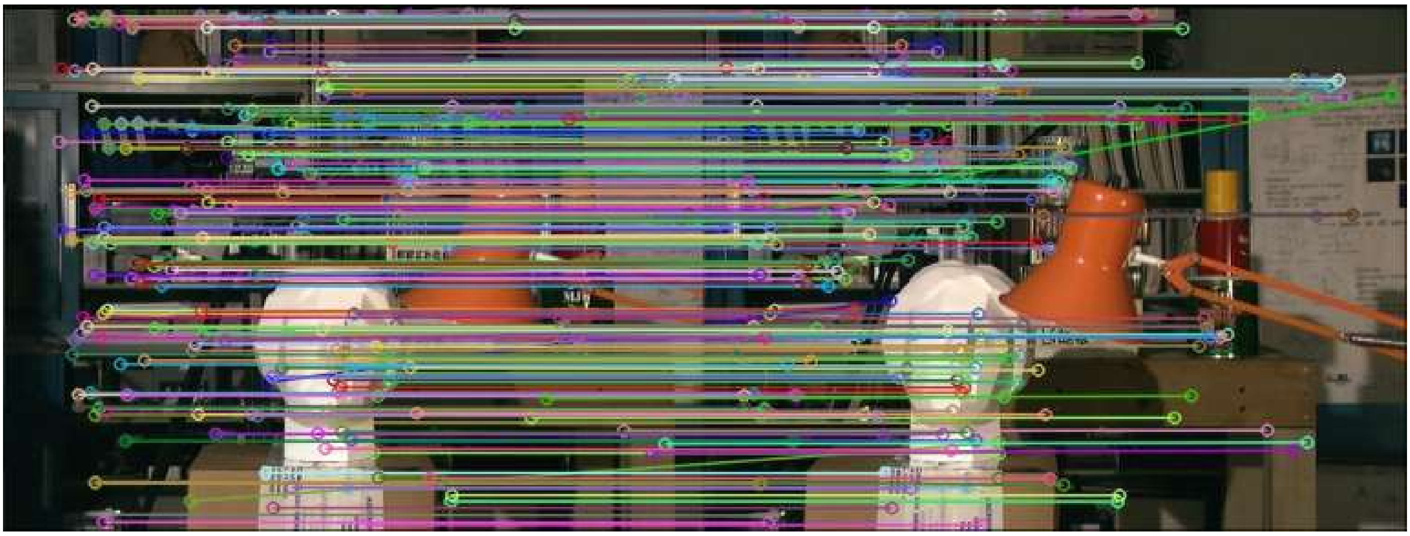
2.1 [ task2_sift1.jpg]



2.1 [task2_sift1.jpg]



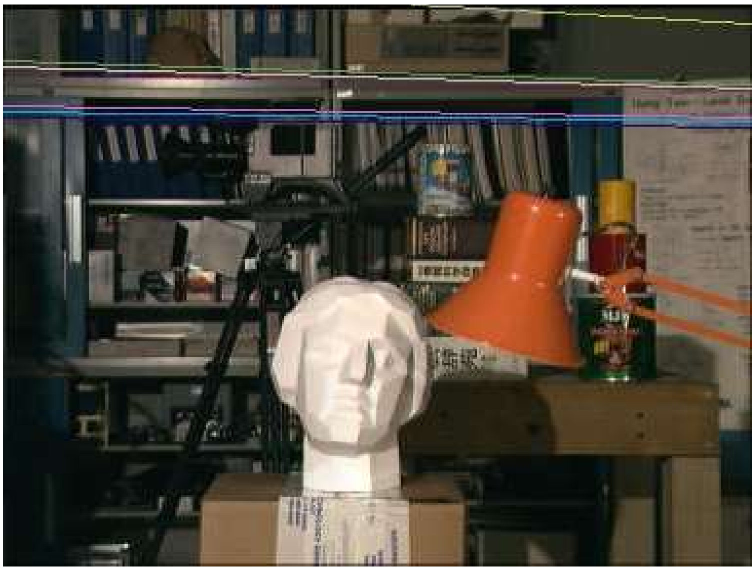
2.1 [ task2_matches_knn.jpg ]



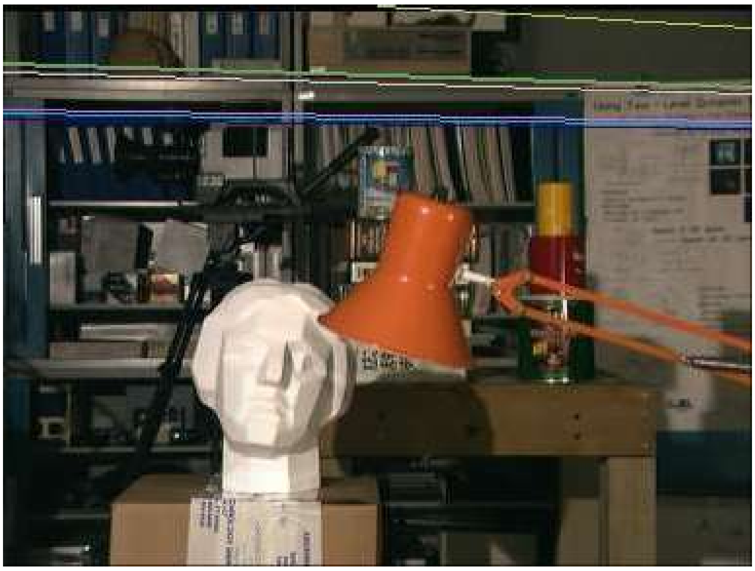
2.2 [ Fundamental Matrix ]

```
[[-1.07832959e-07   -7.11948899e-04   5.57647085e-02]
 [ 7.15827498e-04   -8.90557705e-05  -1.11432709e+00]
 [-5.58854787e-02    1.10821089e+00   1.00000000e+00]]
```

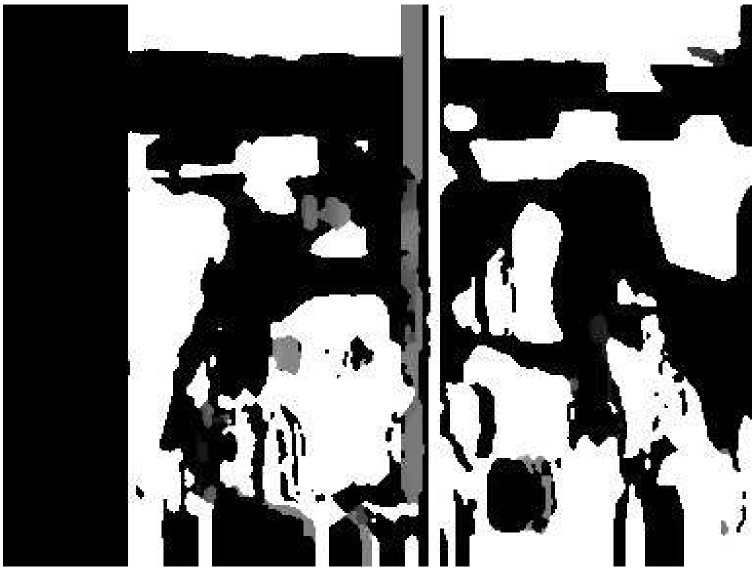
2.3 [task2_epi_left.jpg]



2.3 [task2_epi_right.jpg]



2.4 [Disparity Map]



Reference :

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_epipolar_geometry/py_epipolar_geometry.html

3 [K -means clustering]

Code:

```
import numpy.matlib
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from math import sqrt
import cv2 as cv

points = [(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)]
k = 3
N = 10

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

print(matplotlib.__version__)
# given centroids
redMu_coordinates = [6.2,3.2]
greenMu_coordinates = [6.6,3.7]
blueMu_coordinates = [6.5,3.0]

def compute_mean(clusterVector):
    clusterVector = np.array(clusterVector)
    x_array = clusterVector[:,0]
    y_array = clusterVector[:,1]
    x_mean = sum(x_array)/len(x_array)
    y_mean = sum(y_array)/len(y_array)
    return [x_mean,y_mean]


def plotAllPoints( clr, cluster):
    x_cord_array = []
    y_cord_array = []
    for pt in cluster:
        x_cord_array.append(pt[0])
        y_cord_array.append(pt[1])
        text = '(' + str(pt[0])+',' + str(pt[1]) + ')'
        plt.text(pt[0], pt[1],text, family="monospace")

    plt.scatter(x_cord_array, y_cord_array,   marker='^', facecolors="None", edgecolors=clr)
```

```
def plotCentroid(centroidCordinates, clr):
   Mu_X_cord = centroidCordinates[0]
   Mu_Y_cord = centroidCordinates[1]
   text = '(' + str(Mu_X_cord) + ','+ str(Mu_Y_cord) +')'
   plt.text(Mu_X_cord,Mu_Y_cord,text, family="monospace")
   plt.scatter(Mu_X_cord,Mu_Y_cord,  c=clr)

def getClusterVector(points, redCentroid, greenCentroid, blueCentroid):
   redClusterVector = []
   greenClusterVector= []
   blueClusterVector = []
   for pt in points:
      point = [pt[0], pt[1]]
      dist_from_red_centroid =  sqrt((pt[0] - redCentroid[0])**2 + (pt[1] - redCentroid[1])**2)
      dist_from_green_centroid =  sqrt((pt[0] - greenCentroid[0])**2 + (pt[1] - greenCentroid[1])**2)
      dist_from_blue_centroid =  sqrt((pt[0] - blueCentroid[0])**2 + (pt[1] - blueCentroid[1])**2)
      minimum_distance = min(dist_from_red_centroid, dist_from_green_centroid,
dist_from_blue_centroid)
      if(minimum_distance == dist_from_red_centroid):
         redClusterVector.append(point)
      elif(minimum_distance == dist_from_green_centroid):
         greenClusterVector.append(point)
      elif(minimum_distance == dist_from_blue_centroid):
         blueClusterVector.append(point)
   return redClusterVector, greenClusterVector, blueClusterVector


#part 1
redClusterVector_itr1, greenClusterVector_itr1, blueClusterVector_itr1 = getClusterVector(points,
redMu_coordinates, greenMu_coordinates, blueMu_coordinates)
print(' redClusterVector= ', redClusterVector_itr1)
print(' greenClusterVector = ', greenClusterVector_itr1)
print(' blueClusterVector = ', blueClusterVector_itr1)

# plot all points
plt.figure(1)
plotAllPoints("red", redClusterVector_itr1)
plotAllPoints("green", greenClusterVector_itr1)
plotAllPoints("blue", blueClusterVector_itr1)
plt.savefig('task3_iter1_a.jpg')


# part 2
# update centroids
newCentroid_red_itr1 = compute_mean(redClusterVector_itr1)
newCentroid_green_itr1 = compute_mean(greenClusterVector_itr1)
newCentroid_blue_itr1 = compute_mean(blueClusterVector_itr1)
print('newCentroid_red',newCentroid_red_itr1)
print('newCentroid_green',newCentroid_green_itr1)
print('newCentroid_blue',newCentroid_blue_itr1)

# plot newly calculated centroids
```

```
plt.figure(2)
plotCentroid(newCentroid_red_itr1,'red')
plotCentroid(newCentroid_green_itr1,'green')
plotCentroid(newCentroid_blue_itr1,'blue')
plt.savefig('task3_iter1_b.jpg')

# part 3
# classify points according to newly updated centroids (Mu)
redClusterVector_itr2, greenClusterVector_itr2, blueClusterVector_itr2 = getClusterVector(points,
newCentroid_red_itr1, newCentroid_green_itr1, newCentroid_blue_itr1)
print('new redClusterVector= ', redClusterVector_itr2)
print('new greenClusterVector = ', greenClusterVector_itr2)
print('new blueClusterVector = ', blueClusterVector_itr2)
plt.figure(3)
# plot all points
plotAllPoints("red", redClusterVector_itr2)
plotAllPoints("green", greenClusterVector_itr2)
plotAllPoints("blue", blueClusterVector_itr2)

plt.savefig('task3_iter2_a.jpg')

# part 4
# compute new centroids
newCentroid_red_itr2 = compute_mean(redClusterVector_itr2)
newCentroid_green_itr2 = compute_mean(greenClusterVector_itr2)
newCentroid_blue_itr2 = compute_mean(blueClusterVector_itr2)
print('newCentroid_red',newCentroid_red_itr2)
print('newCentroid_green',newCentroid_green_itr2)
print('newCentroid_blue',newCentroid_blue_itr2)

# plot newly computed centroids
plt.figure(4)
plotCentroid(newCentroid_red_itr2,'red')
plotCentroid(newCentroid_green_itr2,'green')
plotCentroid(newCentroid_blue_itr2,'blue')
plt.savefig('task3_iter2_b.jpg')

# part 4

# color quantization

def getRandomPoints():

    return (np.random.randint(low=0, high=255, size=1),np.random.randint(low=0, high=255, size=1),
np.random.randint(low=0, high=255, size=1))


def copyImage(img):
    h,w,c = img.shape
    temp = [[0 for i in range(w)] for i in range(h)]
    for i in range(h):
        for j in range(w):
          temp[i][j]=img[i][j]
    return temp

def findDistance(rgbCord, meanColorsCord):

    r = (rgbCord[0]-meanColorsCord[0]) ** 2
    g = (rgbCord[1]-meanColorsCord[1]) ** 2
    b = (rgbCord[2]-meanColorsCord[2]) ** 2
    return sqrt(r + g + b)

def findClusters(meanColors,clusters,red,green,blue,clustersArray):
    for i in range(height):
        for j in range(width):
            distance_array = np.zeros([clusters,],dtype ='uint8')
            # find minimum distance of each point with cluster mean
            for z in range(clusters):
                rgbCord = red[j][i], green[j][i], blue[j][i]
                distance_array[z] = findDistance(rgbCord,meanColors[z])
            min_index = np.argmin(distance_array)
            clustersArray[min_index].append((j,i))
    return clustersArray

def drawNewImage(temp_Img,k,clustersArray,meanColors,red,green,blue):
    for kIterator in range(k):
        a = meanColors[kIterator][0]
        b = meanColors[kIterator][1]
        rgb_color = red[a][b],green[a][b],blue[a][b]
        for z in range(len(clustersArray[kIterator])):
            x_cord = clustersArray[kIterator][z][0]
            y_cord = clustersArray[kIterator][z][1]
            temp_Img[x_cord][y_cord] = rgb_color
    temp_Img = np.asarray(temp_Img, dtype="float32")
    return temp_Img

def getClusteredImage(k,img):
    print("K value = ",k)
    itr = 0
    blue = img[:, :, 0]
    green = img[:, :, 1]
    red = img[:, :, 2]
    clustersArray = [[] for i in range(k)]
    temp_Img = copyImage(img)
    lastIterationMean = [0 for i in range(k)]
    matchingMean = False
    while (matchingMean == False):
        meanColors = [0 for i in range(k)]
        for i in range(k):
            if(itr == 0):
                meanColors[i] = getRandomPoints()

            if(itr > 0):
                meanColors[i] = calculateMean(clustersArray[i],img)
        if(meanColors == lastIterationMean):
            matchingMean = True
            break
        lastIterationMean = meanColors
        itr+= 1
        print("Iteration count = ",itr)
        clustersArray = findClusters(meanColors,k,red,green,blue,clustersArray)
    temp_Img = drawNewImage(temp_Img,k,clustersArray,meanColors,red,green,blue)
    print("Image created for K value - ",k)
    temp_Img = np.asarray(temp_Img, dtype="float32")
    img_name = "task3_baboon_"+str(k)+".jpg"
    cv.imwrite(img_name, temp_Img)


def calculateMean(cluster,img):
    blue = img[:, :, 0]
    green = img[:, :, 1]
    red = img[:, :, 2]
    sum_red = 0
    sum_blue = 0
    sum_green = 0
    # if cluster has no elements in it ; to avoid divide by zero condition
    if(len(cluster) == 0):
        return (10000,10000,10000)
    for i in range(len(cluster)):
        sum_red+= red[cluster[i][0]][cluster[i][1]]
        sum_blue+= blue[cluster[i][0]][cluster[i][1]]
        sum_green+= green[cluster[i][0]][cluster[i][1]]
    return sum_red//len(cluster), sum_green//len(cluster), sum_blue//len(cluster)


baboonOriginal_Img = cv.imread("baboon.jpg")
height, width, ch = baboonOriginal_Img.shape

k_array = [3,5,10,20]

for i in range(len(k_array)):
    getClusteredImage(k_array[i],baboonOriginal_Img)
```

Output :

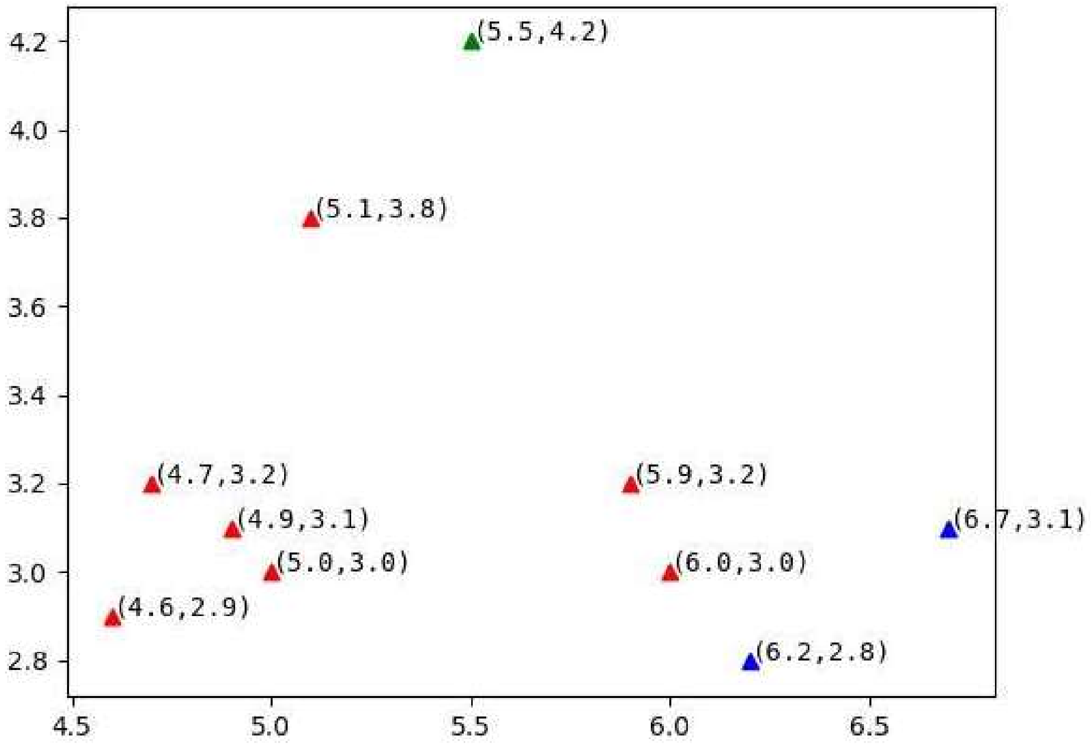
3.1 [Classification Vector]

Red  Cluster Vector=  [[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]]

Green Cluster Vector =  [[5.5, 4.2]]

Blue Cluster Vector =  [[6.2, 2.8], [6.7, 3.1]]

3.1 [task2_itr1_a.jpg]



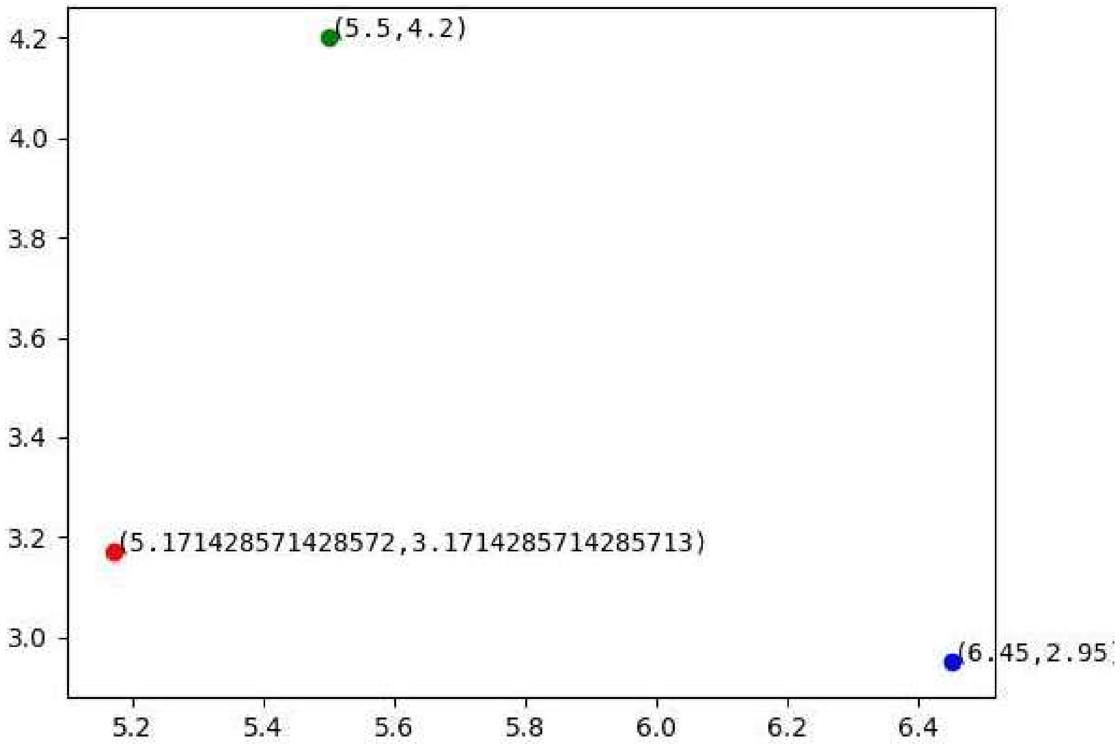
3.2 [Updated Mu]

New_Centroid_red = [5.171428571428572, 3.1714285714285713]

New_Centroid_green = [5.5, 4.2]

New_Centroid_blue = [6.45, 2.95]

3.2 [task3_iter1_b.jpg]



3.3 [ Updated centroids and Cluster Vector]

new redClusterVector= [[4.6, 2.9], [4.7, 3.2], [5.0, 3.0], [4.9, 3.1]]

new greenClusterVector = [[5.5, 4.2], [5.1, 3.8]]

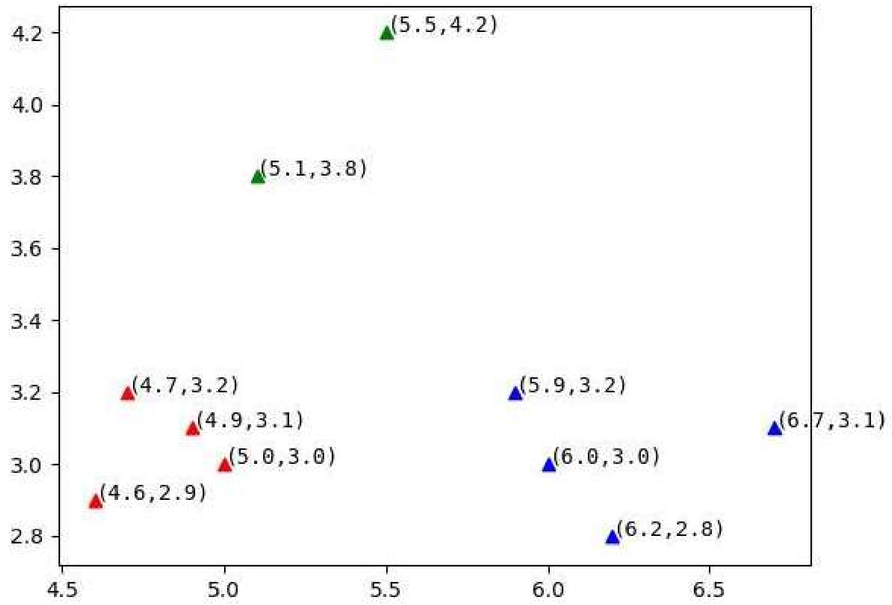
new blueClusterVector = [[5.9, 3.2], [6.2, 2.8], [6.7, 3.1], [6.0, 3.0]]

newCentroid_red = [4.800000000000001, 3.05]

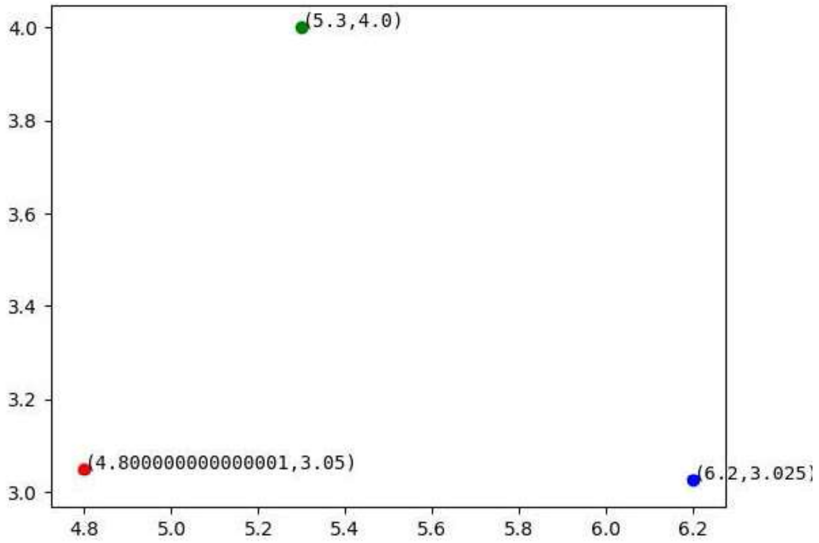
newCentroid_green = [5.3, 4.0]

newCentroid_blue = [6.2, 3.025]

3.3 [task3_iter2_b.jpg]

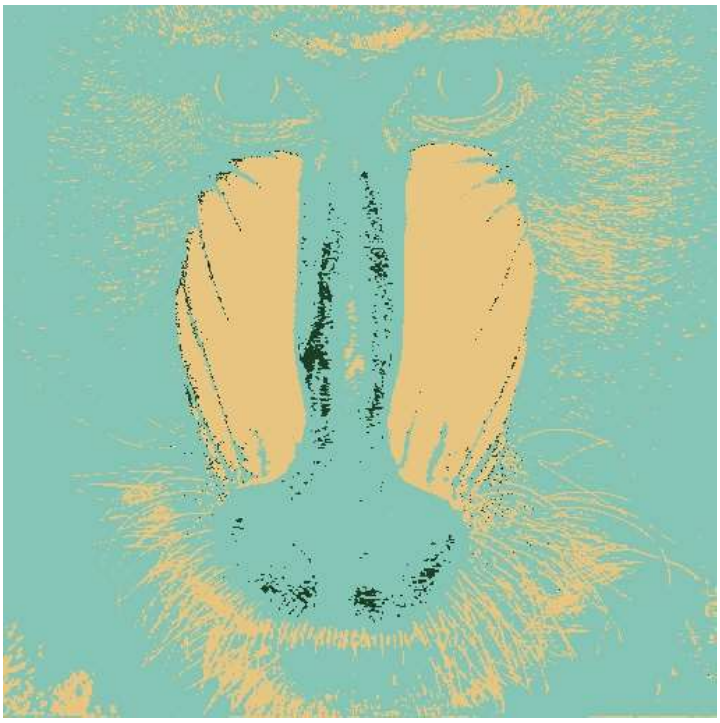


3.3 [task3_iter2_b.jpg]

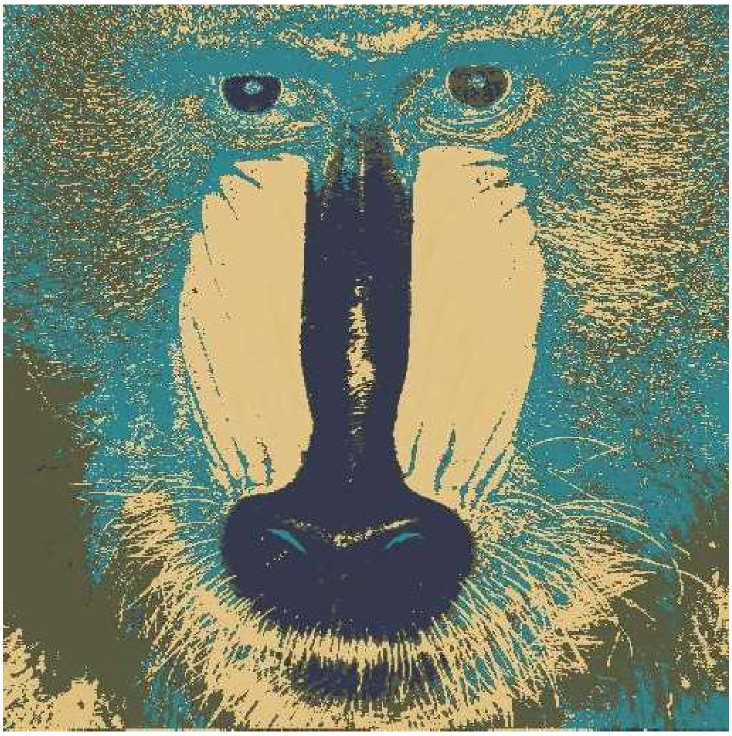


3.4 [Color Quantization]

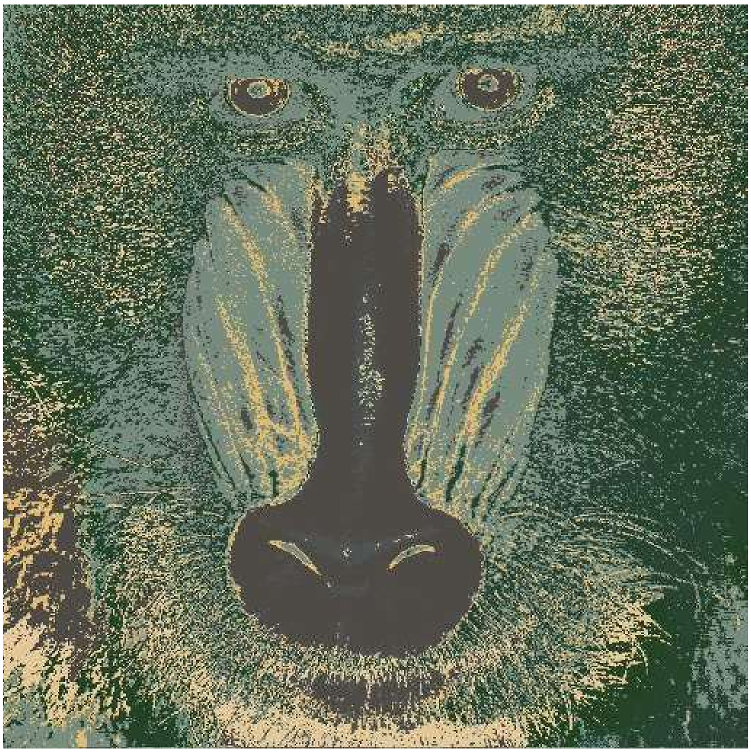
K = 3 [ task3_baboon_3.jpg ]



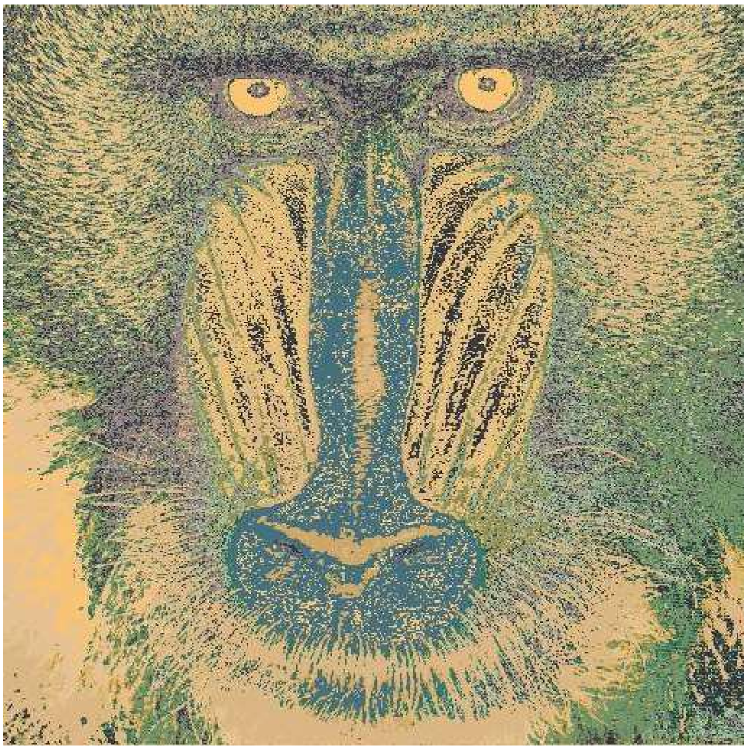
K = 5 [ task3_baboon_5.jpg ]



K = 10 [task3_baboon_10.jpg]



K = 20 [ task3_baboon_20.jpg ]



3 .5  [GMM]

Code:

```
import numpy as np
import math
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt

points =
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)])
k = 3
N = 10


Covariance_mat1 = Covariance_mat2 = Covariance_mat3 = np.array(([0.5,0],[0,0.5]))

def updateMu(Points, P_Mu1, P_Mu2, P_Mu3,Prior_1,Prior_2,Prior_3,Mu_old1,Mu_old2,Mu_old3):
   Mu_1 = []
   Mu_2 = []
   Mu_3 = []
   Posterior_1 = []
   Posterior_2 = []
   Posterior_3 = []
   Co_mat1 = []
   Co_mat2 = []
   Co_mat3 = []
   Cov1 = 0
   Cov2 = 0
   Cov3 = 0
   print("Point shape",Points.shape)
   for i in range(P_Mu1.shape[0]):
      #print(P_Mu1[i])
      print(i)
      C1 = (P_Mu1[i]*Prior_1)/(P_Mu1[i]*Prior_1 + P_Mu2[i]*Prior_2 + P_Mu3[i]*Prior_3)
      C2 = (P_Mu2[i]*Prior_2)/(P_Mu1[i]*Prior_1 + P_Mu2[i]*Prior_2 + P_Mu3[i]*Prior_3)
      C3 = (P_Mu3[i]*Prior_3)/(P_Mu1[i]*Prior_1 + P_Mu2[i]*Prior_2 + P_Mu3[i]*Prior_3)
      Posterior_1.append(C1)
      Posterior_2.append(C2)
      Posterior_3.append(C3)
      Cov1+= P_Mu1[i]*((np.subtract(Points[i],Mu_old1))*(np.subtract(Points[i],Mu_old1)).T)
      Cov2+= P_Mu2[i]*((np.subtract(Points[i],Mu_old2))*(np.subtract(Points[i],Mu_old2)).T)
      Cov3+= P_Mu3[i]*((np.subtract(Points[i],Mu_old3))*(np.subtract(Points[i],Mu_old3)).T)

   for i in range(len(Posterior_1)):
      Cov1+= Posterior_1[i]*((np.subtract(Points[i],Mu_old1))*(np.subtract(Points[i],Mu_old1)).T)
      Cov2+= Posterior_2[i]*((np.subtract(Points[i],Mu_old2))*(np.subtract(Points[i],Mu_old2)).T)
      Cov3+= Posterior_3[i]*((np.subtract(Points[i],Mu_old3))*(np.subtract(Points[i],Mu_old3)).T)
```

```
    print("Posterior_1 = ",Posterior_1)
    print("Posteroir 1 shape ",np.array(Posterior_1).shape)
    X_array = np.array(Points[:,0])
    print("X_array shape",X_array.shape)
    Y_array = np.array(Points[:,1])
    print("Y_array shape",Y_array.shape)
    Mu_1_x = np.dot((np.array(Posterior_1)), X_array)/sum(np.array(Posterior_1))
    Mu_1_y = np.dot((np.array(Posterior_1)), Y_array)/sum(np.array(Posterior_1))
    Co_mat1 = Cov1/sum(np.array(Posterior_1))
    Mu_1.append([Mu_1_x,Mu_1_y])

    Mu_2_x = (np.dot(np.array(Posterior_2), X_array))/sum(np.array(Posterior_2))
    Mu_2_y = (np.dot(np.array(Posterior_2), Y_array))/sum(np.array(Posterior_2))
    Co_mat2 = Cov2/sum(np.array(Posterior_2))
    Mu_2.append([Mu_2_x,Mu_2_y])

    Mu_3_x = (np.dot(np.array(Posterior_3), X_array))/sum(np.array(Posterior_3))
    Mu_3_y = (np.dot(np.array(Posterior_3), Y_array))/sum(np.array(Posterior_3))
    Co_mat3 = Cov3/sum(np.array(Posterior_3))
    Mu_3.append([Mu_3_x, Mu_3_y])


    Prior_1 = sum(np.array(Posterior_1))/len(Posterior_1)
    Prior_2 = sum(np.array(Posterior_2))/len(Posterior_2)
    Prior_3 = sum(np.array(Posterior_3))/len(Posterior_3)



    return Mu_1, Mu_2, Mu_3, Prior_1, Prior_2, Prior_3, Co_mat1, Co_mat2, Co_mat3

# initial mu
Mu_1 = np.array([6.2,3.2])
Mu_2 = np.array([6.6,3.7])
Mu_3 = np.array([6.5,3.0])
P_Mu1_1 = multivariate_normal.pdf(points, mean=Mu_1, cov= Covariance_mat1)
P_Mu2_1 = multivariate_normal.pdf(points, mean=Mu_2, cov= Covariance_mat2)
P_Mu3_1 = multivariate_normal.pdf(points, mean=Mu_3, cov= Covariance_mat3)

Prior_1 = Prior_2 = Prior_3 = 1/3

Mu_1_1, Mu2_1, Mu_3_1, Prior_1_1, Prior_2_1,
Prior_3_1,Covariance_mat1_1,Covariance_mat2_1,Covariance_mat3_1 = updateMu(points, P_Mu1_1,
P_Mu2_1, P_Mu3_1,Prior_1,Prior_2,Prior_3,Mu_1,Mu_2,Mu_3)
print("Mu_1_1 = ",Mu_1_1)
# Iteration 1
```

## 3.5 [Updated Mu after first iteration]

Updated Mu_1 =  [[5.316507899024571, 3.2152729183353053]]

Updated Mu_2 =  [[5.611297951076313, 3.385053105024623]]

Updated Mu_3 =  [[5.604435653845083, 3.144200610784218]]