1. Image Features and Homography

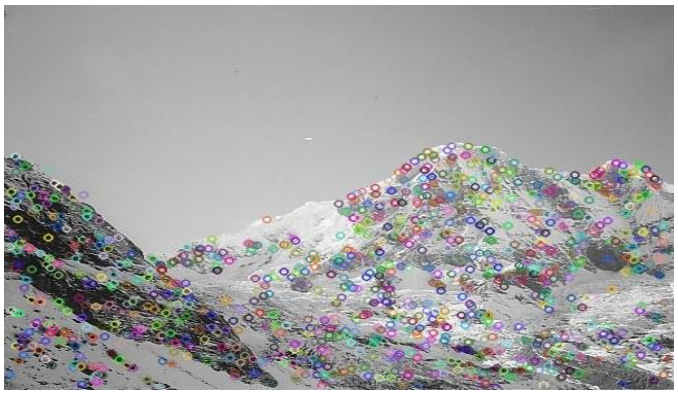
Source Code:

```
import numpy as np
import cv2
np.random.seed(sum([ord(c) for c in "123"]))
from random import*
import random
UBIT='srujanko'
image1= cv2.imread('D://mountain1.jpg')
image2= cv2.imread('D://mountain2.jpg')
img1 = cv2.imread('D://mountain1.jpg',0) # queryImage
img2 = cv2.imread('D://mountain2.jpg',0) # trainImage
# Initiate SIFT detector
sift = cv2.cv2.xfeatures2d.SIFT_create()
# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
cv2.imwrite("D://task_1sift1.jpg",cv2.drawKeypoints(img1, kp1,image1.copy()))
cv2.imwrite("D://task_1sift2.jpg",cv2.drawKeypoints(img2, kp2,image2.copy()))
# BFMatcher with default params
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)
best_match = []
x=[]
for m,n in matches:
    if m.distance < 0.75*n.distance:
        best_match.append([m])
        x.append(m)
cv2.imwrite("D://task1_matches_knn.jpg",cv2.drawMatchesKnn(image1,kp1,image2,kp2,best_match,None,flags=2))
src_pts = np.float32([ kp1[m.queryIdx].pt for m in x ]).reshape(-1,1,2)
dst_pts = np.float32([ kp2[m.trainIdx].pt for m in x ]).reshape(-1,1,2)
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
print(M)
matchesMaskAll = []
matchesMask = mask.ravel().tolist()
print(matchesMask)
masko = np.asarray(matchesMask)
dnd = np.where(masko==1)
dnd = np.asarray(dnd).ravel()
msk =[]
seed(sum([ord(c) for c in UBIT]))
dnd=np.random.choice(dnd, 10, replace=False)
msk =np.zeros(len(matchesMask))
for i in range(len(dnd)):
    msk[dnd[i]]=1
#print(msk)
h, w = img1.shape
pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
#dst = cv2.perspectiveTransform(pts, M)
#img2 = cv2.polylines(img2, [np.int32(dst)], True, 255, 3, cv2.LINE_AA)
#cv2.imshow("im",img2)
```

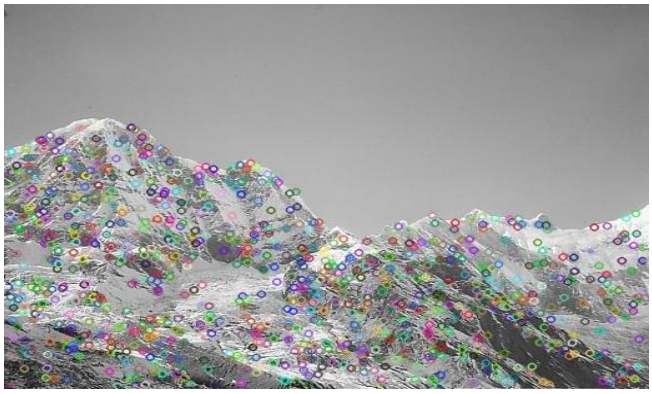
```
draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   matchesMask = msk, flags = 2)# draw only inliers

img10 = cv2.drawMatches(image1,kp1,image2,kp2,x,None,**draw_params)
cv2.imwrite("D://task1_matches.jpg",img10)
def warpTwoImages(img1, img2, H):
   h1,w1 = img1.shape[:2]
   h2,w2 = img2.shape[:2]
   pts1 =np.float64([[0,0],[0,h1],[w1,h1],[w1,0]]).reshape(-1,1,2)
   pts2 = np.float64([[0,0],[0,h2],[w2,h2],[w2,0]]).reshape(-1,1,2)
   pts2_ = cv2.perspectiveTransform(pts2, H)
   pts =np.concatenate((pts1, pts2_), axis=0)
   [xmin, ymin] = np.int32(pts.min(axis=0).ravel() - 0.5)
   [xmax, ymax] = np.int32(pts.max(axis=0).ravel() + 0.5)
   t = [-xmin,-ymin]
   Ht = np.array([[1,0,t[0]],[0,1,t[1]],[0,0,1]]) # translate
   result = cv2.warpPerspective(img2, Ht.dot(H), (xmax-xmin, ymax-ymin))
   result[t[1]:h1+t[1],t[0]:w1+t[0]] = img1
   return result
result = warpTwoImages(img2, img1, M)
cv2.imwrite("D://task1 pano.jpg",result)
```

1.1 The keypoints for two images mountain1.jpg and mountain2.jpg

first image- task1_sift1.jpg

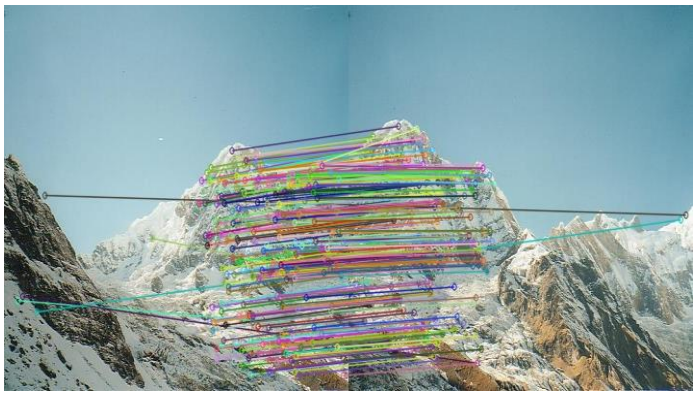


second image task1_sift2.jpg



1.2 the keypoints using k-nearest neighbour (k=2), i.e., for a keypoint in the left image, finding the best 2 matches in the right image.

task1_matches_knn.jpg



1.3 the homography matrix H (with RANSAC) from the first image to the second image

The Homography matrix is:

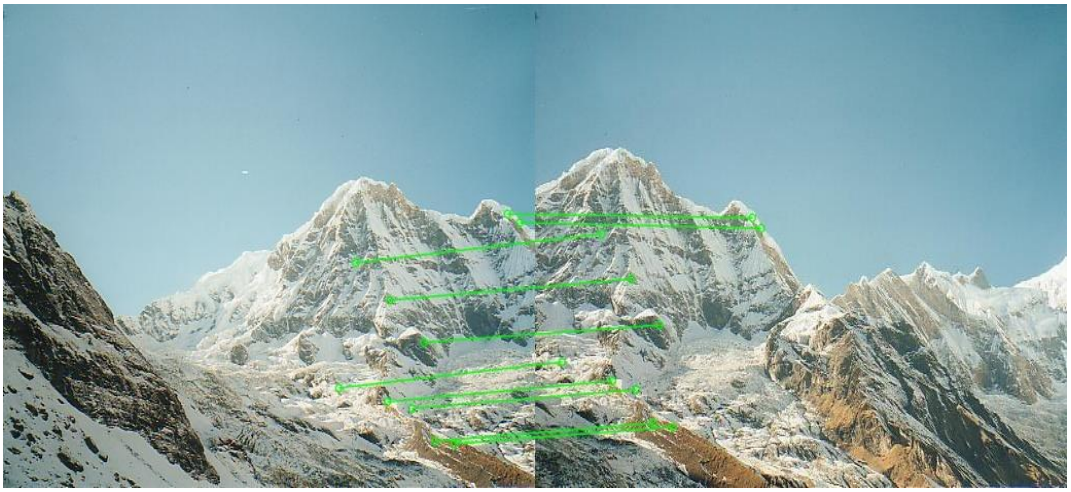
[[1.58930230e+00 -2.91559040e-01 -3.95969265e+02]

[4.49423930e-01 1.43110916e+00 -1.90613988e+02]

[1.21265043e-03 -6.28729364e-05 1.00000000e+00]]

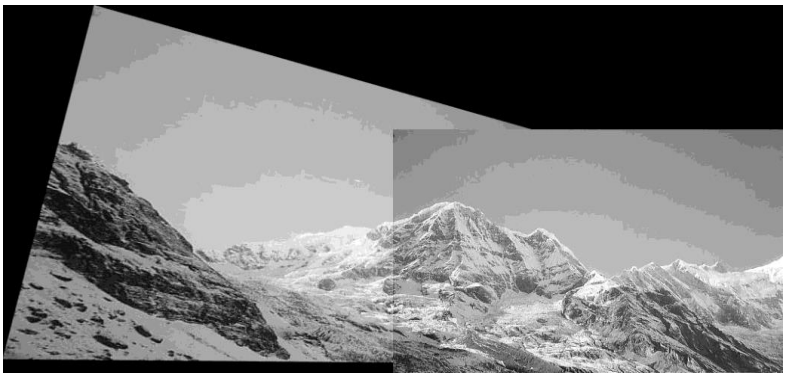
1.4 the match image for around 10 random matches using only inliers

task1_matches.jpg



1.5 Warpping the first image to the second image using H.

1.5-Task_pano.png



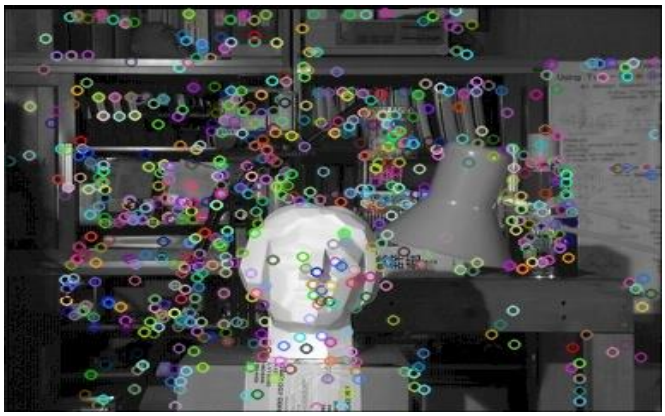
2 Epipolar Geometry

```
import numpy as np
import random
import cv2
np.random.seed(sum([ord(c) for c in "123"]))
from random import*
import random
UBIT='srujanko'
from matplotlib import pyplot as plt
def SIFTMATCH(img1,img2):
    img1=img1.copy()
    img2=img2.copy()
    sift = cv2.xfeatures2d.SIFT_create()
    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(img1,None)
    kp2, des2 = sift.detectAndCompute(img2,None)
    cv2.imwrite("D://task2_sift1.jpg",cv2.drawKeypoints(img1, kp1,img1.copy()))
    cv2.imwrite("D://task2_sift2.jpg",cv2.drawKeypoints(img2, kp2,img2.copy()))

    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
#    matches = sorted(matches, key = lambda x:x.distance)
    good = []
    pts1 = []
    pts2 = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append([m])
            pts2.append(kp2[m.trainIdx].pt)
```

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

cv2.imwrite("D://task2_matches_knn.jpg",cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2)
)

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


# We select only inlier points
    pts1 = pts1[mask.ravel()==1]
    pts2 = pts2[mask.ravel()==1]
    pts1en = list(enumerate(pts1, 1))
    pts2en = list(enumerate(pts2, 1))
    seed(sum([ord(c) for c in UBIT]))
    pts1enr = random.sample(pts1en, 10)

    pts3=[]
    pts4=[]
    ran = [i[0] for i in pts1enr]
    for x in ran:
        for y in pts1enr:
            if(x==y[0]):
                print("yes")
                pts3.append(y[1])
    for x in ran:
        for y in pts2en:
            if(x==y[0]):
                pts4.append(y[1])
    pts3 = np.int32(pts3)
    pts4 = np.int32(pts4)
    print(pts3)


    pts1 = pts3
    pts2 = pts4
    colors = []

    def drawlines(img1,img2,lines,pts1,pts2,colors):
        x=0
        ''' img1 - image on which we draw the epilines for the points in img2
            lines - corresponding epilines '''
        r,c = img1.shape
        img1 = cv2.cvtColor(img1,cv2.COLOR_GRAY2BGR)
        img2 = cv2.cvtColor(img2,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] ])
            img1 = cv2.line(img1, (x0,y0), (x1,y1), colors[x],1)
            img1 = cv2.circle(img1,tuple(pt1),5,colors[x],-1)
            img2 = cv2.circle(img2,tuple(pt2),5,colors[x],-1)
            x+=1
        return img1,img2
    lines1 = cv2.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2,F)
    lines1 = lines1.reshape(-1,3)
    for i in range(10):
        seed(sum([ord(c) for c in UBIT]))
        colors.append(tuple(np.random.randint(0,255,3).tolist()))

    img5,img6 = drawlines(img1,img2,lines1,pts1,pts2,colors)
    print(colors)
# Find epilines corresponding to points in left image (first image) and
# drawing its lines on right image
    lines2 = cv2.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1,F)
    lines2 = lines2.reshape(-1,3)
    img3,img4 = drawlines(img2,img1,lines2,pts2,pts1,colors)
    cv2.imwrite('D:\\task2 epi right.jpg',img3)
    cv2.imwrite('D:\\task2 epi left.jpg',img5)

    plt.show()

img2 = cv2.imread("D:\\tsucuba_left.PNG",0)
img1 = cv2.imread("D:\\tsucuba_right.PNG",0)
SIFTMATCH(img1,img2)
```

2.1 For the first image- task1_sift1.jpg task2_sift2.jpg



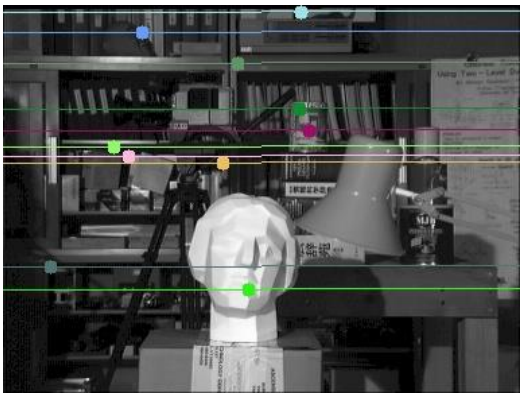
the keypoints using k-nearest neighbour (k=2), i.e., for a keypoint in the left image, finding the best 2 matches in the right image.

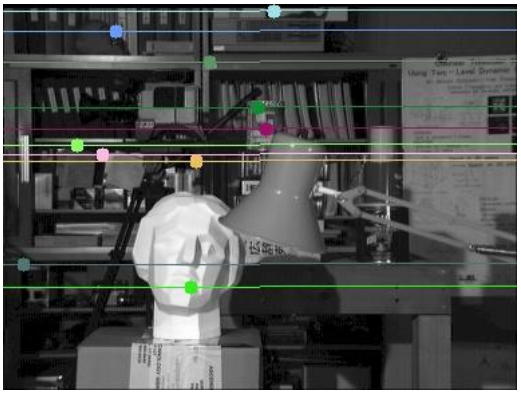
task1_matches_knn.jpg

2.2 Fundamental matrix:

[[0.00000000e+00 -8.05664062e-03 2.37500000e+00]
[7.08007812e-03 1.15966797e-03 3.79588115e+15]
[-2.28125000e+00 -3.79588115e+15 1.00000000e+00]].

2.3Randomly selecting 10 inlier match pairs. For each keypoint in the left image, computing the epiline on the right image. For each keypoint in the right image, computing the epiline on the left image





2.4 the disparity map for tsucuba left.png and tsucuba right.png

3.K-means Clustering

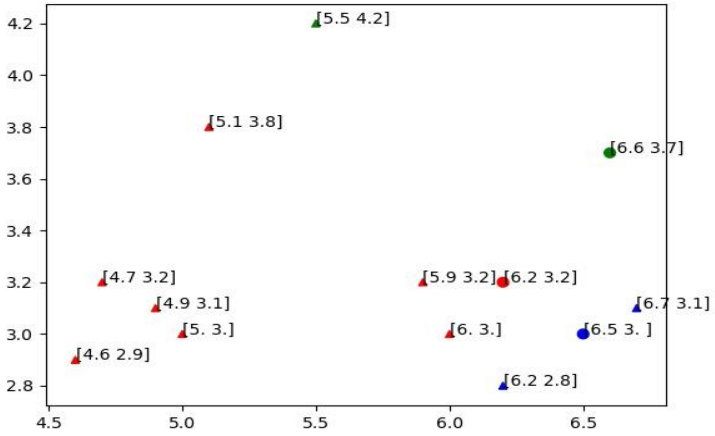
```
import numpy as np
from math import e,pi,sqrt
import matplotlib.pyplot as plt
u = np.array([[6.2,3.2],[6.6,3.7],[6.5,3.0]])
x = np.array([[5.9,3.2],[4.6,2.9],[6.2,2.8],[4.7,3.2],[5.5,4.2],[5.0,3.0],[4.9,3.1],[6.7,3.1],[5.1,3.8],[6.0,3.0]])

gar = []
dist= []
xa=[]
ya=[]
co=[]
u1=[]
u2=[]
u3=[]
tot=[]


def kmeans(x,uz):
   ux=np.array([[0.0 for j in range(2)] for i in range(3)])
   yes=0
#    mean.append(uz)
   u1[:]=[]
   u2[:]=[]
   u3[:]=[]
   tot[:]=[]
   dist[:] = []
   for i in range(len(x)):
      gar=[]
```

```
        for j in range(len(uz)):
            distance = sqrt((uz[j][0]-x[i][0])**2 + (uz[j][1]-x[i][1])**2)
            gar.append(distance)
        y= gar.index(min(gar))
        if(y==0):
            col='red'
        if(y==1):
            col='green'
        if(y==2):
            col='blue'
        gar.append(col)
        gar.append(y)
        dist.append(gar)
#       print(dist)

    for i in range(len(dist)):
        if(dist[i][4]==0):
            u1.append(x[i])
        if(dist[i][4]==1):
            u2.append(x[i])
        if(dist[i][4]==2):
            u3.append(x[i])


    tot.append(u1)
    tot.append(u2)
    tot.append(u3)
#    sto = uz

#    print(sto)
    for i in range(3):
        ux[i][0]=sum([x[0] for x in tot[i]])/len(tot[i])
        ux[i][1]=sum([x[1] for x in tot[i]])/len(tot[i])

    print(ux)
    for i in range(3):
        for j in range(2):
            if(ux[i][j]==uz[i][j]):
                yes +=1
    print(uz)
    return dist,uz,ux

di,yu,yuup = kmeans(x,u)

f = []
g = []
h = ['red','green','blue']
fu = []
```

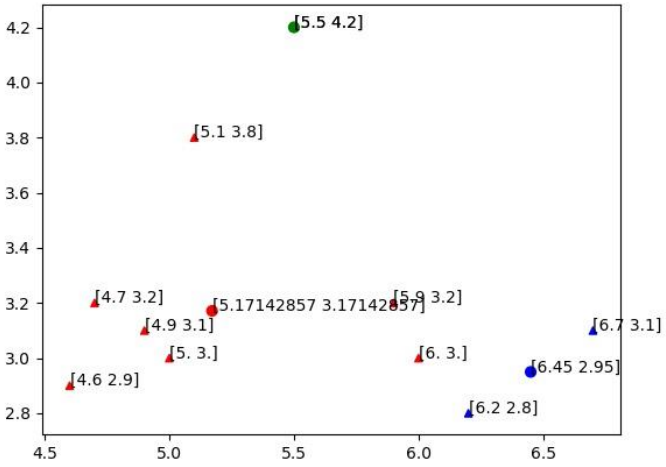
```
gu = []


for i in range(len(x)):
   xa.append(x[i][0])
   ya.append(x[i][1])
   co.append(di[i][3])
for i in range(3):
   f.append(yu[i][0])
   g.append(yu[i][1])

for i in range(3):
   fu.append(yuup[i][0])
   gu.append(yuup[i][1])
fig,ax = plt.subplots()

ax.scatter(xa, ya, s=20, c=co, marker='^')
ax.scatter(f,g, s=40, c=h,marker = 'o')
for i,txt in enumerate(yu):
   ax.annotate(txt, (f[i],g[i]))
for i, txt in enumerate(x):
   ax.annotate(txt, (xa[i], ya[i]))
fig.savefig('D:\\task3_iter1_a.jpg', dpi=fig.dpi)

figu,axu = plt.subplots()
axu.scatter(xa, ya, s=20, c=co, marker='^')
axu.scatter(fu,gu, s=40, c=h,marker = 'o')
for i,txt in enumerate(yuup):
   axu.annotate(txt, (fu[i],gu[i]))
for i, txt in enumerate(x):
   axu.annotate(txt, (xa[i], ya[i]))
figu.savefig('D:\\task3_iter1_b.jpg', dpi=figu.dpi)

di,yu,yuup = kmeans(x,yuup)

xa=[]
ya=[]
co=[]

f = []
g = []
h = ['red','green','blue']
fu = []
gu = []


for i in range(len(x)):
   xa.append(x[i][0])
```

```
    ya.append(x[i][1])
    co.append(di[i][3])
for i in range(3):
    f.append(yu[i][0])
    g.append(yu[i][1])
#    co.append(h[i])
for i in range(3):
    fu.append(yuup[i][0])
    gu.append(yuup[i][1])
#    co.append(h[i])

fig,ax = plt.subplots()

ax.scatter(xa, ya, s=20, c=co, marker='^')
ax.scatter(f,g, s=40, c=h,marker = 'o')
for i,txt in enumerate(yu):
    ax.annotate(txt, (f[i],g[i]))
for i, txt in enumerate(x):
    ax.annotate(txt, (xa[i], ya[i]))

fig.savefig('D:\\task3_iter2_a.jpg', dpi=fig.dpi)

figu,axu = plt.subplots()
axu.scatter(xa, ya, s=20, c=co, marker='^')
axu.scatter(fu,gu, s=40, c=h,marker = 'o')
for i,txt in enumerate(yuup):
    axu.annotate(txt, (fu[i],gu[i]))
for i, txt in enumerate(x):
    axu.annotate(txt, (xa[i], ya[i]))
figu.savefig('D:\\task3_iter2_b.jpg', dpi=figu.dpi)
```

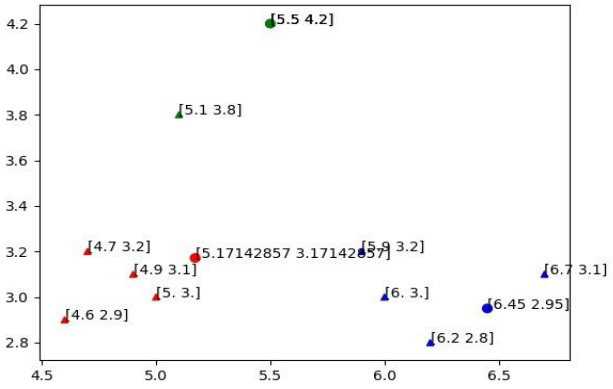
3.1.the classification vector and the classification plot. For the given means (task3_iter1a.jpg)
The circle marked points are cluster points in the below diagram.
The tringle marked points are the points to which cluster they belong to.



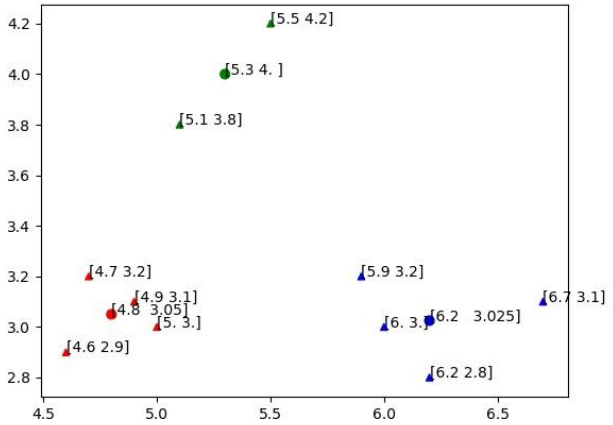
3.2 Classifying the datapoints for the updated u values.(Task3_iter1b.jpg)
The below centroids are updated for the data.



3.3 Task3_iter2a.jpg (below points are classified according to their centroids for above updated centroids)



Task3_iter2b.jpg (the updated centroids are represented below)



Source code for KMEANS FOR BABOON:

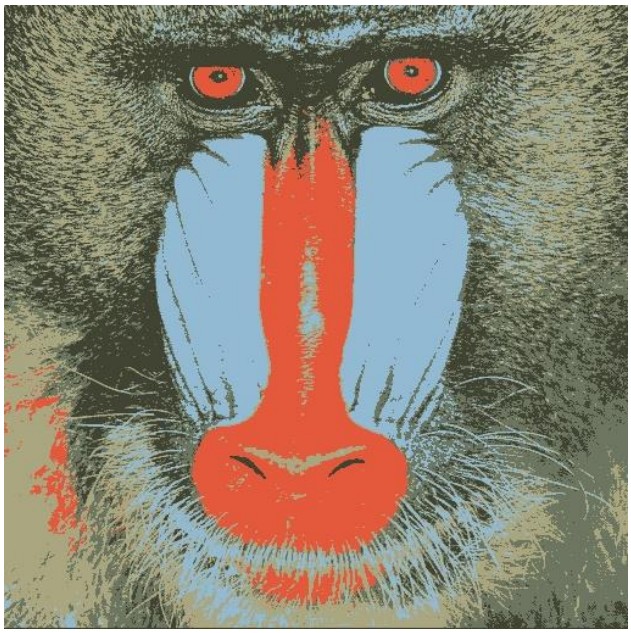
```
import numpy as np
import random
import cv2
np.random.seed(sum([ord(c) for c in "123"]))
from random import*
import random
UBIT='srujanko'
from matplotlib import pyplot as plt
def SIFTMATCH(img1,img2):
```

```
    img1=img1.copy()
    img2=img2.copy()
    sift = cv2.xfeatures2d.SIFT_create()
    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(img1,None)
    kp2, des2 = sift.detectAndCompute(img2,None)
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
#    matches = sorted(matches, key = lambda x:x.distance)
    good = []
    pts1 = []
    pts2 = []
    for m,n in 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)
    print(F)
# We select only inlier points
    pts1 = pts1[mask.ravel()==1]
    pts2 = pts2[mask.ravel()==1]
    pts1en = list(enumerate(pts1, 1))
    pts2en = list(enumerate(pts2, 1))
    seed(sum([ord(c) for c in UBIT]))
    pts1enr = random.sample(pts1en, 10)
    pts3=[]
    pts4=[]
    ran = [i[0] for i in pts1enr]
    for x in ran:
        for y in pts1enr:
            if(x==y[0]):
                print("yes")
                pts3.append(y[1])
    for x in ran:
        for y in pts2en:
            if(x==y[0]):
                pts4.append(y[1])
    pts3 = np.int32(pts3)
    pts4 = np.int32(pts4)
    print(pts3)
    pts1 = pts3
    pts2 = pts4
    colors = []

  def drawlines(img1,img2,lines,pts1,pts2,colors):
```

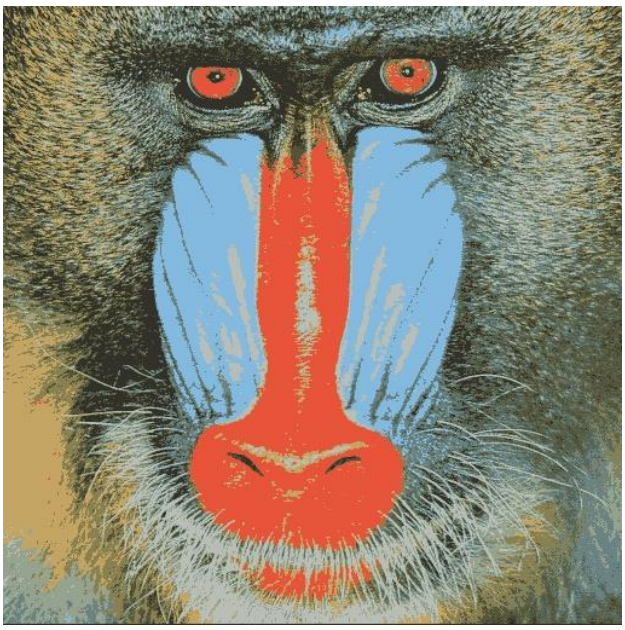
```
        x=0
        ''' img1 - image on which we draw the epilines for the points in img2
            lines - corresponding epilines '''
        r,c = img1.shape
        img1 = cv2.cvtColor(img1,cv2.COLOR_GRAY2BGR)
        img2 = cv2.cvtColor(img2,cv2.COLOR_GRAY2BGR)
        for r,pt1,pt2 in zip(lines,pts1,pts2):
#           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] ])
            img1 = cv2.line(img1, (x0,y0), (x1,y1), colors[x],1)
            img1 = cv2.circle(img1,tuple(pt1),5,colors[x],-1)
            img2 = cv2.circle(img2,tuple(pt2),5,colors[x],-1)
            x+=1
        return img1,img2
    lines1 = cv2.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2,F)
    lines1 = lines1.reshape(-1,3)
    for i in range(10):
        seed(sum([ord(c) for c in UBIT]))
        colors.append(tuple(np.random.randint(0,255,3).tolist()))
    img5,img6 = drawlines(img1,img2,lines1,pts1,pts2,colors)
    print(colors)
# Find epilines corresponding to points in left image (first image) and
# drawing its lines on right image
    lines2 = cv2.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1,F)
    lines2 = lines2.reshape(-1,3)
    img3,img4 = drawlines(img2,img1,lines2,pts2,pts1,colors)
    cv2.imwrite('D:\\task2 epi right.jpg',img3)
    cv2.imwrite('D:\\task2 epi left.jpg',img5)
    plt.show()
img2 = cv2.imread("D:\\tsucuba_left.PNG",0)
img1 = cv2.imread("D:\\tsucuba_right.PNG",0)
SIFTMATCH(img1,img2)
```

K = 3

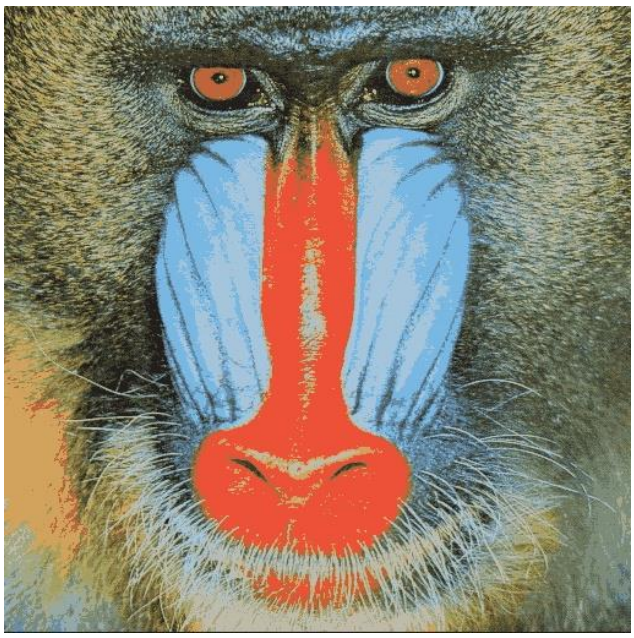
K= 5



k =10



K=20



3.(Bonus)

Source code for gmm.

```
 from scipy.stats import multivariate_normal as mvn
import numpy as np
#import cv2
arr = 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]], np.float32)
mu1 = [6.2, 3.2]
mu2 = [6.6, 3.7]
mu3 = [6.5, 3.0]

sig1 = [[0.5+0.1, 0],[0, 0.5+0.1]]

k=3
nrow,ncol = arr.shape
mean_arr = np.asmatrix([mu1,mu2,mu3])
sigma_arr = np.array([np.asmatrix(sig1) for i in range(k)])
prior = np.ones(k)/k
post = np.asmatrix(np.empty((nrow, k), dtype=float))
data = arr



def e_step(k,mean_arr,sigma_arr,prior,post,data):
   nrow,ncol = data.shape
   for i in range(nrow):

      den = 0
      for j in range(k):
           num = (mvn.pdf(data[i, :], mean_arr[j].A1, sigma_arr[j]))* prior[j]
           den += num
           post[i, j] = num
      post[i, :] /= den
#       assert post[i, :].sum() - 1 < 1e-4

   return post

def m_step(k,mean_arr,sigma_arr,prior,post,data):
   nrow, ncol = data.shape
   for j in range(k):
      const = post[:, j].sum()
      prior[j] = 1/nrow * const
      _mu_j = np.zeros(ncol)
      _sigma_j = np.zeros((ncol, ncol))
      for i in range(nrow):
         _mu_j += (data[i, :] * post[i, j])
         _sigma_j += post[i, j] * ((data[i, :] - mean_arr[j, :]).T * (data[i, :] - mean_arr[j, :]))
            #print((self.data[i, :] - self.mean_arr[j, :]).T * (self.data[i, :] - self.mean_arr[j, :]))
```

```
        mean_arr[j] = _mu_j / const
        sigma_arr[j] = _sigma_j / const
    print(mean_arr)
    return sigma_arr,mean_arr,prior

def fit(k,mean_arr,sigma_arr,prior,post,data):
        tol=1e-1
#       self._init()
        num_iters = 0
        ll = 1
        previous_ll = 0
        while(ll-previous_ll > tol):
            previous_ll = loglikelihood(k,mean_arr,sigma_arr,prior,post,data)
            post,sigma_arr,mean_arr,prior = _fit(k,mean_arr,sigma_arr,prior,post,data)
            num_iters += 1
            ll = loglikelihood(k,mean_arr,sigma_arr,prior,post,data)
            print('Iteration %d: log-likelihood is %.6f'%(num_iters, ll))
        print('Terminate at %d-th iteration:log-likelihood is %.6f'%(num_iters, ll))

def loglikelihood(k,mean_arr,sigma_arr,prior,post,data):

        ll = 0
        for i in range(nrow):
            tmp = 0
            for j in range(k):
                #print(self.sigma_arr[j])
                tmp += mvn.pdf(data[i, :],mean_arr[j, :].A1,sigma_arr[j, :])*prior[j]
            ll += np.log(tmp)
        return ll
def _fit(k,mean_arr,sigma_arr,prior,post,data):
        post = e_step(k,mean_arr,sigma_arr,prior,post,data)
        sigma_arr,mean_arr,prior = m_step(k,mean_arr,sigma_arr,prior,post,data)
        return post,sigma_arr,mean_arr,prior

i_sigma_arr = np.array([np.asmatrix(np.identity(ncol)) for i in range(k)])

i_mean_arr  =np.asmatrix(np.random.random((k, ncol)))

x=0

'''while(x<100):
    i_mean_arr = mean_arr.copy()
#    print(i_mean_arr)
    i_sigma_arr = sigma_arr.copy()
#    print(i_sigma_arr)
#    print(post)
    post = e_step(k,i_mean_arr,i_sigma_arr,prior,post,arr)
    sigma_arr,mean_arr,prior = m_step(arr,post,k,i_mean_arr,i_sigma_arr)
```

```
    #print(sigma_arr)
    #print("--------------")
    #print(mean_arr)
    x=x+1
    print(x)'''
fit(k,mean_arr,sigma_arr,prior,post,arr)

#print(sigma_arr)
#print(mean_arr)
```

```
[[5.33236135 3.21676678]
 [5.58541528 3.35614478]
 [5.5717395  3.15628924]]
Iteration 1: log-likelihood is -17.369597
[[5.41158144 3.19275212]
 [5.4144791  3.47601343]
 [5.57031893 3.11843471]]
Iteration 2: log-likelihood is -14.494245
[[5.39481119 3.15130199]
 [5.37782193 3.64253086]
 [5.60822157 3.05809845]]
Iteration 3: log-likelihood is -12.711925
[[5.38455977 3.09864916]
 [5.31169409 3.79246346]
 [5.62213155 3.0343824 ]]
Iteration 4: log-likelihood is -10.546288
[[5.40444366 3.04625639]
 [5.25963182 3.88579602]
 [5.60301663 3.03481947]]
Iteration 5: log-likelihood is -8.128958
[[5.42378402 3.03035507]
 [5.24587004 3.90865776]
 [5.59068936 3.03881579]]
Iteration 6: log-likelihood is -6.213318
[[5.44054286 3.01991653]
 [5.22618457 3.89925233]
 [5.59128315 3.04183845]]
Iteration 7: log-likelihood is -5.089052
[[5.45423578 3.00935514]
 [5.21459841 3.88613001]
 [5.59115896 3.04531258]]
Iteration 8: log-likelihood is -4.964973
[[5.46047138 3.00120975]
 [5.21751827 3.89002432]
 [5.58438396 3.0515515 ]]
```

The updated values of u after first iteration are:

U1=[5.33236135, 3.21676678]

U2=[5.58541528, 3.35614478]

U3=[5.5717395 , 3.15628924]