Project 2: Image stitching, Epipolar Geometry, K-means and GMM Clustering

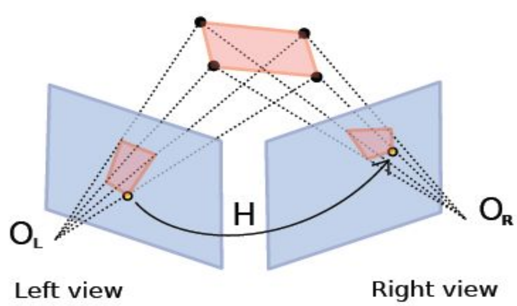
Hariprasath Parthasarathy
Department of Computer Science and Engineering
University at Buffalo.
hparthas@buffalo.edu

Abstract

Image stitching ,Epipolar geometry and Clustering algorithms provide a major gateway to various applications in computer vision. which include panorama stitching, Depth detection, and Color quantization. These below operations are the major building blocks for most computer vision applications. This report will help us understand the implementation of the concepts in detail

1 Image Feature and Homography

Features are points in images that are useful points for processing an image. they tend to be the points of importance in any image. while homography matrices are information regarding translation of image which would facilitate stitching of two images which are taken from slightly different angles . the application of stitching include maps creating long frame panorama creation etc



➢

1.1 The process in done in the following steps:

- ❖ keypoint detection is done using scalar invariant fourier transform on both the images
- ❖ the extracted keypoints are matched using k nearest neighbour algorithm to find 2 nearest neighbours
- ❖ Good matches are extracted based on the match distance of 0.75 between the match points
- ❖ Further homography matrix is computed using cv2.findHomography to obtain the homography matrix M and matches mask. -using Random sample consensus (RANSAC) Algorithm
  - ➢ the homography matrix consists of the information regarding to rotation translation and scaling of the images second image so that it can be warped with the first image
- ❖ This matches mask is used to used along with the key points to warp two images into a

single image.

- since there is a requirement to show all the pixels in the image warping cannot be done with normal CV2 warp images and preprocessing is required in the warp the image. the steps are as follows
    - Do a perspective transform on the image to compute the new points for the image
    - concatenate the new points from the perspective transform along with the other images key points and find the min and max of keypoints
    - update the homography matrix for the new crop values
    - The two images are warped using cv2.warpperspective

1.2 Output


Figure 1.1 and 1.2 - Input images for feature detection


Figure 1.3 and 1.4 - Keypoints detected using SIFT on input images

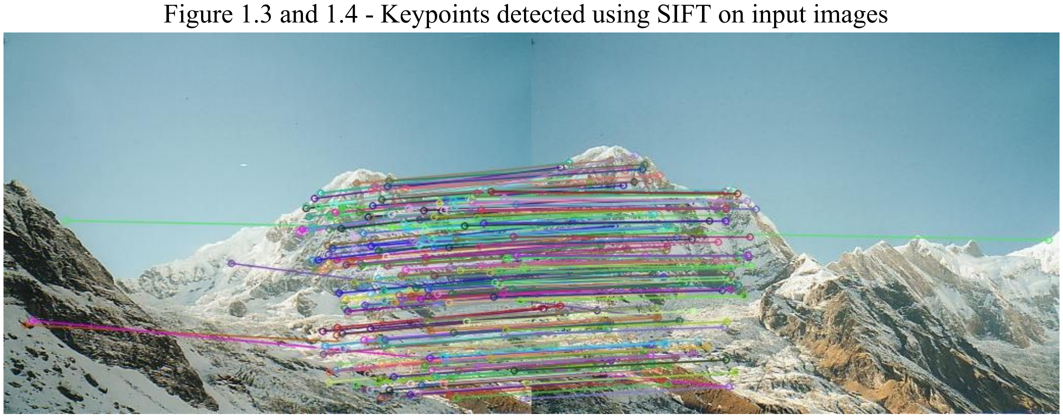


Figure 1.5 - Keypoints matches based on KNN algorithm

5.90420481e-01	1.32208920e-01	2.58541496e+02
-2.82721424e-01	8.60619704e-01	5.19165443e+01
-7.32806155e-04	-1.04842624e-04	1.00000000e+00

The output homography matrix calculated using ransac

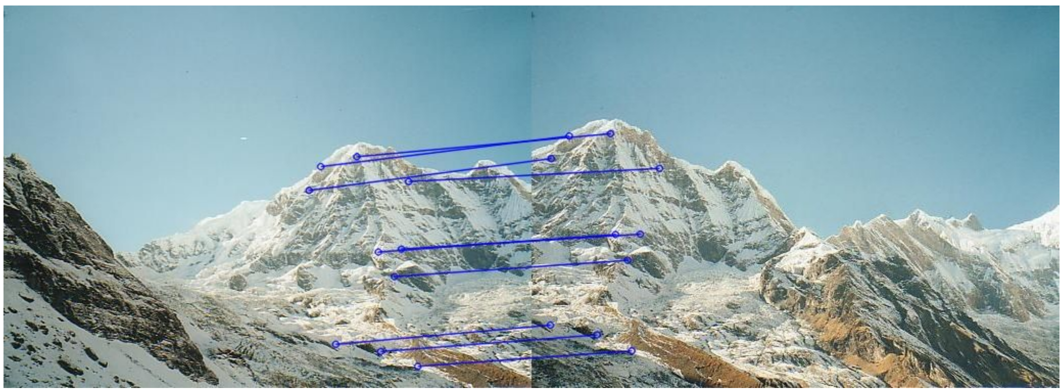


Figure 1.5 - The inliers detected by homography - RANSAC Algorithm

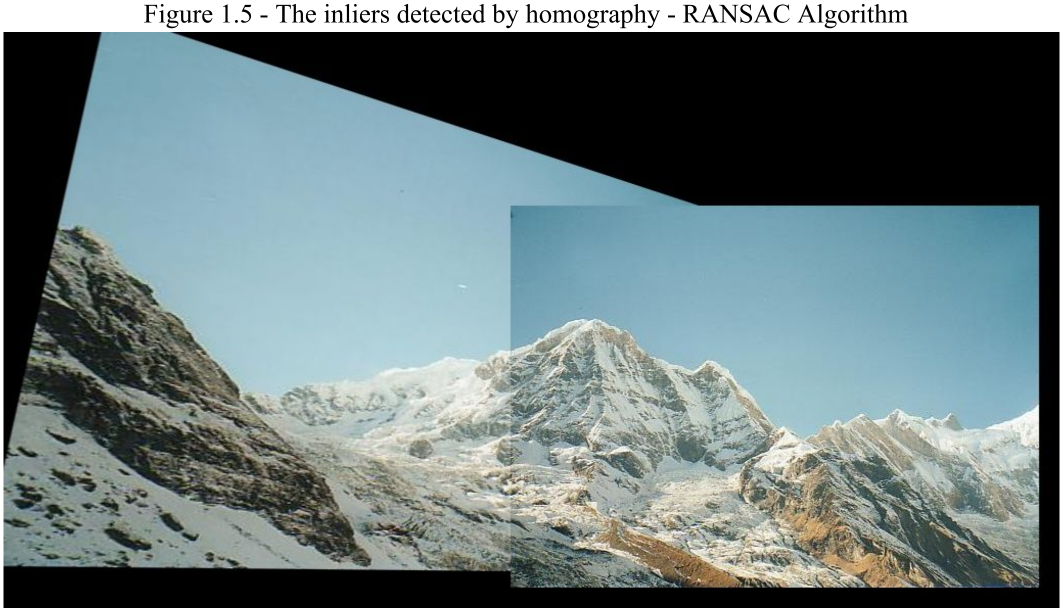


Figure 1.6 : The output image after wraping

Here Image 1 is transformed based on the homography matrix and is warped along with image 2 .

2 Epipolar Geometry

A single camera does not hold any depth information. But when two cameras view a 3D scene from two distinct positions, there are a number of geometric relations between the 3D points and their projections onto the 2D images that lead to constraints between the image points. with this knowledge a depth map can be calculated

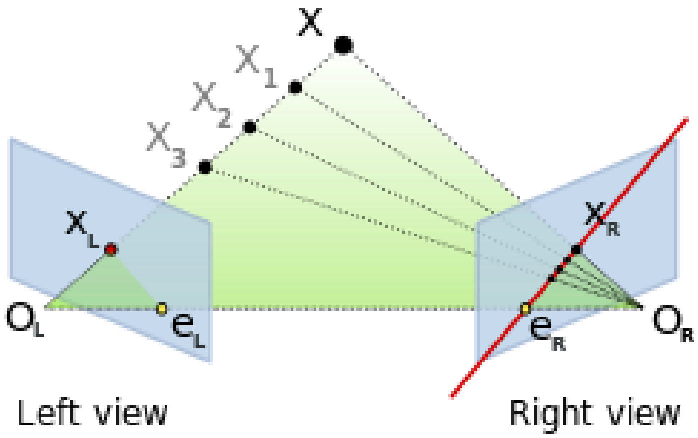


Figure 2.1 - A Visual representation of epipolar plane and depth features

This depth information can be further used to isolate subject from the background and can be very useful in computer vision tasks like feature matching futher A disparity map is computed using this information that can serve as a depth map for scene.

2.1 Steps to Draw Epilines and compute Disparity map

- Apply SIFT to detect keypoints on both the images
- Using the key points apply K-Nearest neighbour algorithm to compute at most 2 nearest neighbours
- Fundamental matrix is computed using cv2.findFundamentalMat Using the matches found
- Inlier pairs are selected and lines are drawn based on fundamental matrix and the key points

2.2 Output

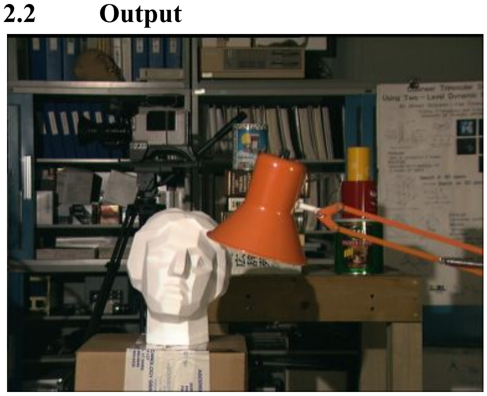


Figure 2.2 & 2.3 - Input images to compute epiline in images

Figure 2.4 & 2.5 - Sift keypoint detected using the input images

3.75314231e-06	-6.97790553e-04	6.33868746e-02
6.63138462e-04	-4.00041312e-05	-4.71158894e-01
-6.33533087e-02	4.68619370e-01	1.00000000e+00

The Fundamental matrix calculated from the detected keypoints

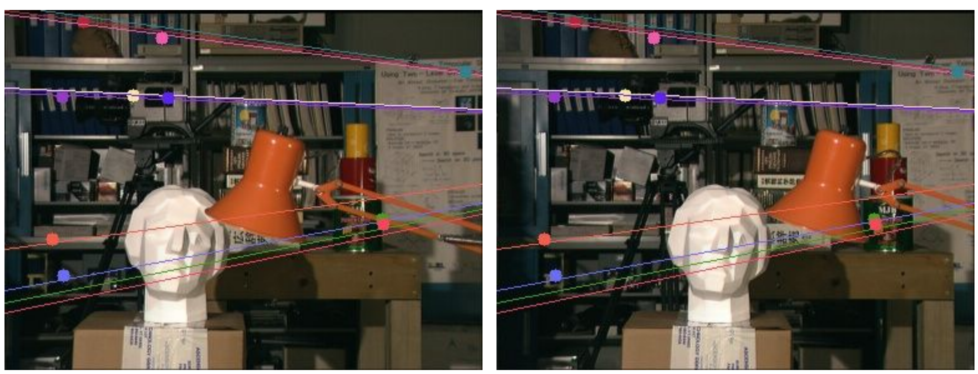


Figure 2.6 & 2.7 - 10 Random epipolar lines drawn for both the images

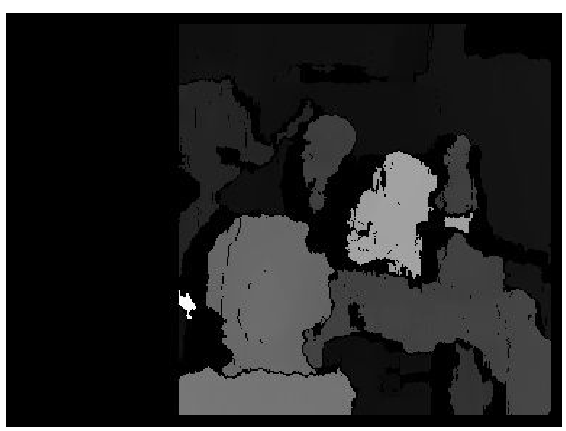
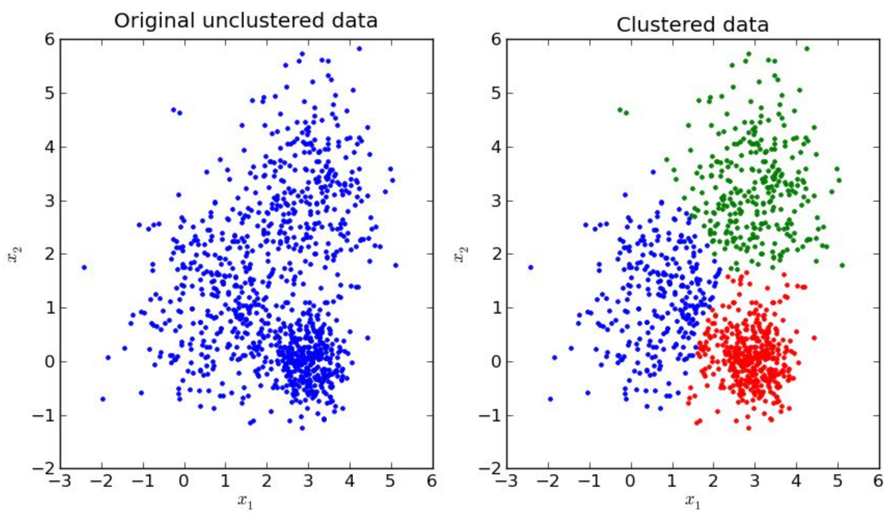


Fig 2.8 - The disparity map output for the above images

3 k-Means clustering

K means clustering is an expectation maximization problem used to cluster data points to groups with local optimum. It is done iteratively by computing the mean followed but the points belonging to the means. This K-means is used to partition the data based on a parameter into n groups



3.1 K-means clustering algorithm

Step1: K points are picked in random.

Step2: Considering them as centroids points are assigned to centroids based on least euclidean distance

step 3: The mean is chosen for the points and assigned as the new centroid

step 4: Step 2 and 3 are repeated until the centroids remain the same

3.2 Output

Output mask for the clustering is before first iteration
[1, 1, 3, 1, 2, 1, 1, 3, 1, 1]
Output mask for the clustering is after first iteration
[3, 1, 3, 1, 2, 1, 1, 3, 2, 3]

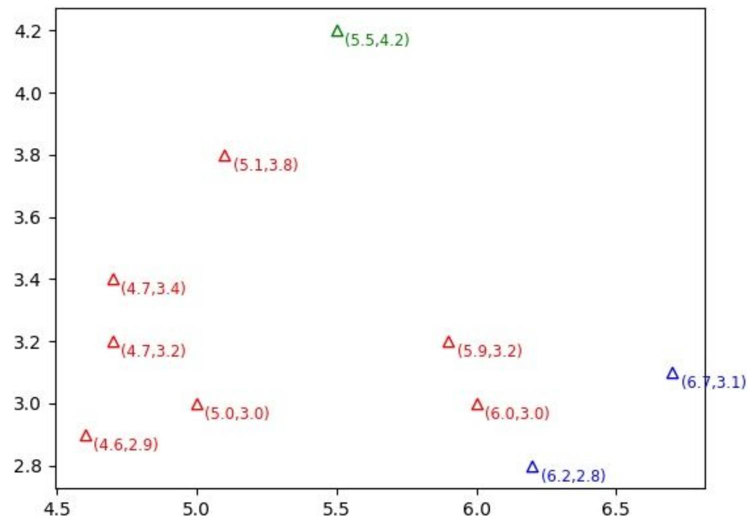


Image 3.2 Plotting before first iteration

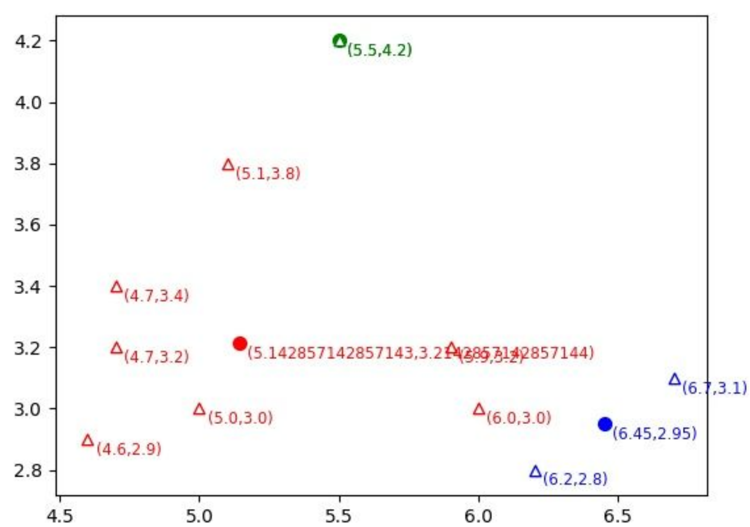


image 3.3 - Computed centroid after first iteration

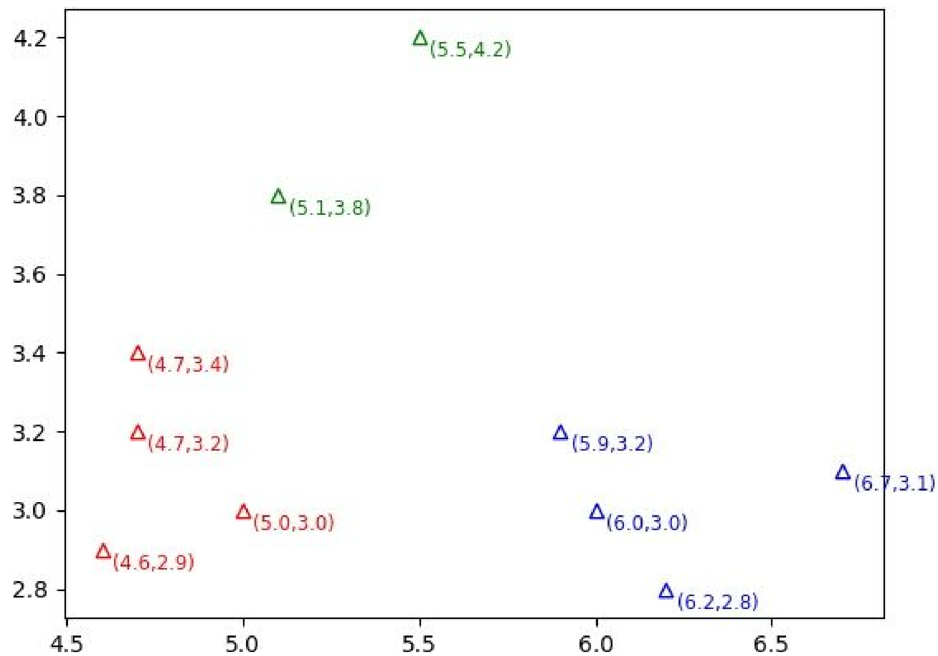


Image 3.4 - Plotting After first Iteration

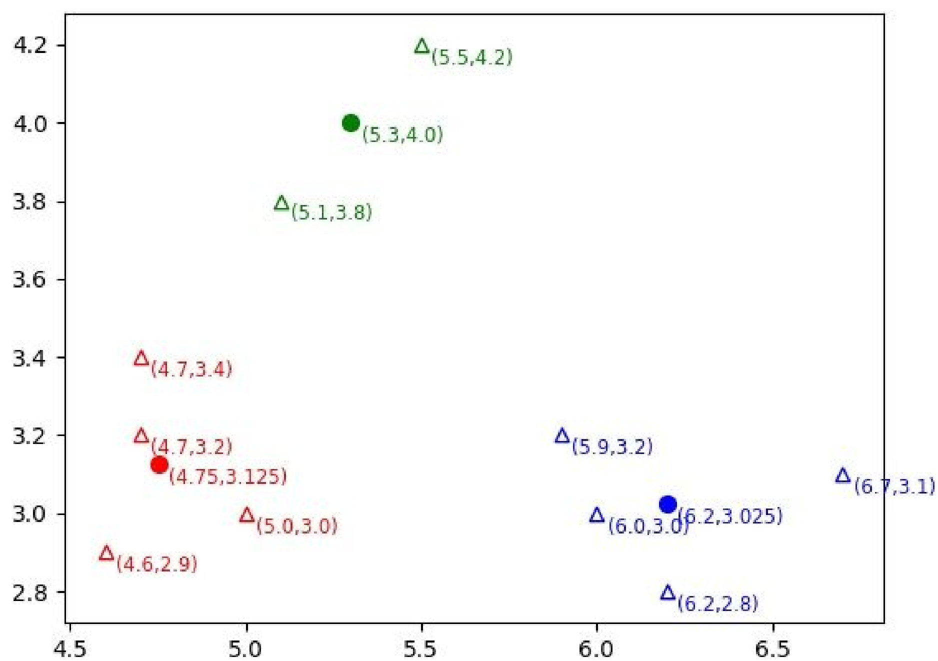


Image 3.5 - Computed centeroids after first Iteration

Output for Image 2 :

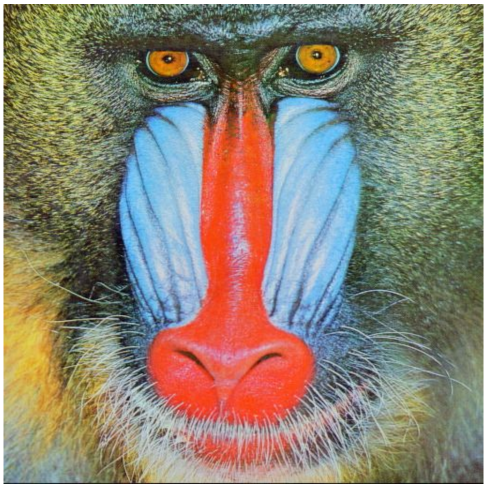


Figure 3.6 Input image

Figure 3.7 Output with k = 3

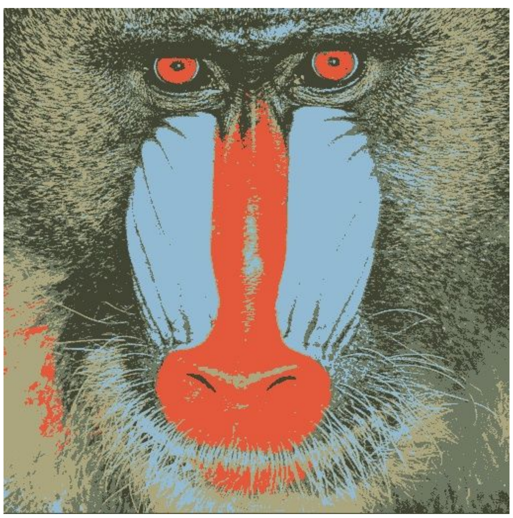


Figure 3.8 Output with k = 5

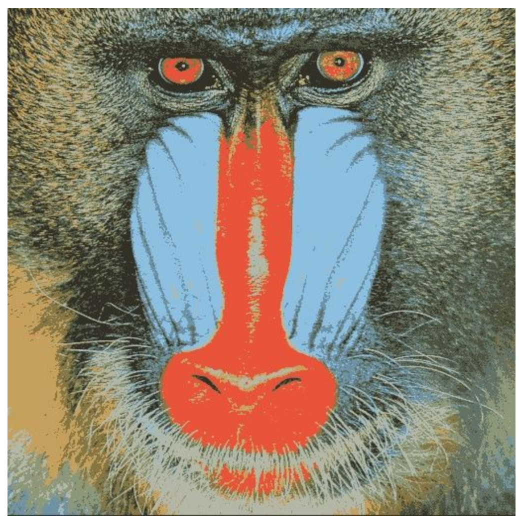


Figure 3.9 Output with k = 10

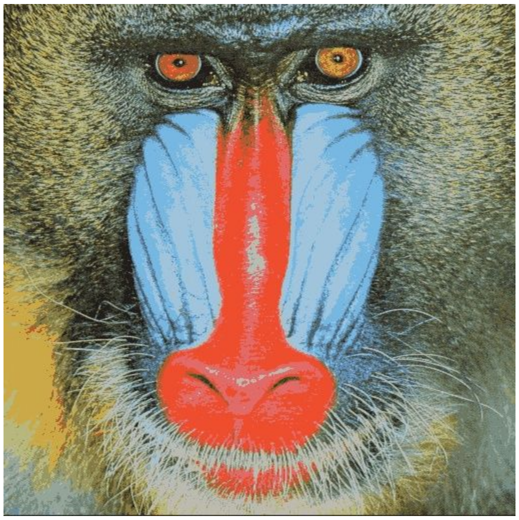


Figure 3.10 Output with k = 20

Gaussian Mixture model

Gaussian Mixture model is similar to k means, but here its a probabilistic model ie: A point might belong to two clusters at a time. The gaussian Curve is the famously known bell curve. the model again implements Expectation maximization problem. The task is to find out mean variance and weight for the model that would fit and the further select points to compute max likelihood

$$\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)},$$

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

the formula for computing mean and variance is as above

Output

For the first problem the mean value is as follows for the first iteration

[5.2896965 , 3.25223226],

[5.58796463, 3.4038864],

[5.60142834, 3.16722706]

for the second problem the outputs are as follows

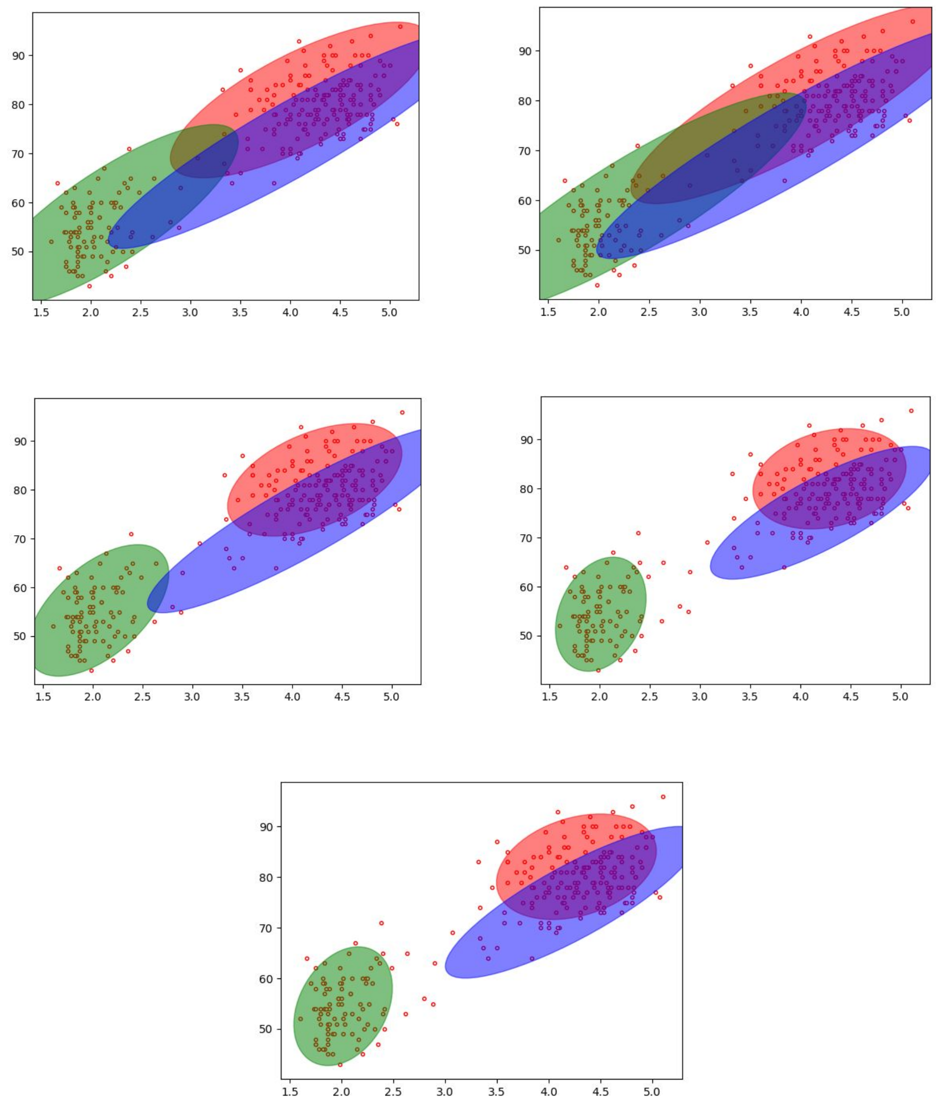


figure 3.11 - 3.15 - output of gaussian mixture model on old faithful dataset over 5 iterations

Annexure 1:

Task 1:

```
UBIT = 'hparthas'

import cv2

from matplotlib import pyplot as plt

import numpy as np

import random

np.random.seed(sum(ord(c) for c in UBIT))

#                                                                                     reference
https://stackoverflow.com/questions/13063201/how-to-show-the-whole-image-when-using-openc
v-warpperspective

def warpTwoImages(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_ = cv2.perspectiveTransform(pts2, H)

  pts = np.concatenate((pts1, pts2_), axis=0)

  [xmin, ymin] = np.int32(pts.min(axis=0).ravel() + 20)

  [xmax, ymax] = np.int32(pts.max(axis=0).ravel() + 20)

  Ht = np.array([[1,0,-xmin],[0,1,-ymin],[0,0,1]])

  Ht1 = Ht.dot(H)

  Ht1[0][2] = Ht1[0][2]

  Ht1[1][2] = Ht1[1][2]

  result = cv2.warpPerspective(img2, Ht1, (xmax-xmin, ymax-ymin))

  result[-ymin:h1+(-ymin),-xmin:w1+(-xmin)] = img1

  return result

# read image 1 and convert to BW

m1_clr = cv2.imread('data/mountain1.jpg')

mountain1= cv2.cvtColor(m1_clr,cv2.COLOR_BGR2GRAY)

# read image 2 and convert to BW

m2_clr = cv2.imread('data/mountain2.jpg')

mountain2 = cv2.cvtColor(m2_clr,cv2.COLOR_BGR2GRAY)

# Extract Sift features and compute Descriptors for image 1 and image 2

sift = cv2.xfeatures2d.SIFT_create()

keypoints_mountain1 ,m1_des= sift.detectAndCompute(mountain1,None)
```

```
image1_withkp = cv2.drawKeypoints(m1_clr,keypoints_mountain1,None)
cv2.imwrite('output/task1/task1_sift1.jpg',image1_withkp)

keypoints_mountain2,m2_des = sift.detectAndCompute(mountain2,None)
image2_withkp =  cv2.drawKeypoints(m2_clr,keypoints_mountain2,None)
cv2.imwrite('output/task1/task1_sift2.jpg',image2_withkp)

bf = cv2.BFMatcher()
matches = bf.knnMatch(m1_des,m2_des, k=2)

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

img3 = cv2.drawMatchesKnn(m1_clr,keypoints_mountain1,m2_clr,keypoints_mountain2,good_knnmatch,None,flags=2)
cv2.imwrite("output/task1/task1_matches_knn.jpg",img3)

src_pts = np.float32([ keypoints_mountain1[m[0].queryIdx].pt for m in good_knnmatch ]).reshape(-1,1,2)
dst_pts = np.float32([ keypoints_mountain2[m[0].trainIdx].pt for m in good_knnmatch ]).reshape(-1,1,2)

M, mask = cv2.findHomography(dst_pts,src_pts , cv2.RANSAC,3.0)
print(M)

matchesMask = mask.ravel().tolist()
h,w,d = m1_clr.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)

selected_matches = [0] * len(matchesMask)
matches_count = 0
```

```
while  matches_count < 10 :
    select_index = np.random.randint(0,len(matchesMask))
    if selected_matches[select_index] == 0 and matchesMask[select_index] == 1:
        selected_matches[select_index] = 1
        matches_count =matches_count+1

draw_params = dict(matchColor = (255,0,0),
                   singlePointColor = None,
                   matchesMask = selected_matches,
                   flags = 2)

img3                                                                                    =
cv2.drawMatches(m1_clr,keypoints_mountain1,m2_clr,keypoints_mountain2,good_filter,None,**
draw_params)

cv2.imwrite("output/task1/task1_matches.jpg",img3)
im_out =warpTwoImages(m2_clr,m1_clr,np.linalg.inv(M))

cv2.imwrite("output/task1/task1_pano.jpg",im_out)
```

Task 2:

```
UBIT = 'hparthas'
import cv2
import numpy as np
from matplotlib import pyplot as plt
import random
np.random.seed(sum([ord(c) for c in UBIT]))
# read image 1 and convert to BW
m1_clr = cv2.imread('data/tsucuba_left.png')
image1_bw= cv2.cvtColor(m1_clr,cv2.COLOR_BGR2GRAY)

# read image 2 and convert to BW
m2_clr = cv2.imread('data/tsucuba_right.png')
image2_bw = cv2.cvtColor(m2_clr,cv2.COLOR_BGR2GRAY)

# Extract Sift features and compute Descriptors for image 1 and image 2
```

```
sift = cv2.xfeatures2d.SIFT_create()
keypoints_mountain1 ,m1_des= sift.detectAndCompute(image1_bw,None)
image1_withkp = cv2.drawKeypoints(m1_clr,keypoints_mountain1,None)
cv2.imwrite('output/task2/task2_sift1.jpg',image1_withkp)

keypoints_mountain2,m2_des = sift.detectAndCompute(image2_bw,None)
image2_withkp =  cv2.drawKeypoints(m2_clr,keypoints_mountain2,None)
cv2.imwrite('output/task2/task2_sift2.jpg',image2_withkp)

def drawlines(img1,img2,lines,pts1,pts2,color):
  r,c = (cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)).shape
  i = 0
  for r,pt1,pt2 in zip(lines,pts1,pts2):
    x0,y0 = map(int, [0, -r[2]/r[1] ])
    x1,y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
    img1 = cv2.line(img1, (x0,y0), (x1,y1), color[i],1)
    img1 = cv2.circle(img2,tuple(pt1),5,color[i],-1)
    i = i+1
  return img1


pts1 = []
pts2 = []

bf = cv2.BFMatcher()
matches = bf.knnMatch(m1_des,m2_des, k=2)

for i,(m,n) in enumerate(matches):
    pts2.append(keypoints_mountain2[m.trainIdx].pt)
    pts1.append(keypoints_mountain1[m.queryIdx].pt)

fundamentalmat,                                   mask                                   =
cv2.findFundamentalMat(np.array(pts1),np.array(pts2),cv2.FM_RANSAC)
print(fundamentalmat)
```

```
pts1 = np.array(pts1)[mask.ravel() == 1]
pts2 = np.array(pts2)[mask.ravel() == 1]

random_points = np.random.randint(0, len(pts1), 10)

selected_point1,selected_point2 = list(), list()
for i, (p1, p2) in enumerate(zip(pts1, pts1)):
    if i in random_points:
        selected_point1.append(p1)
        selected_point2.append(p2)
selected_point1 = np.float32(selected_point1)
selected_point2 = np.float32(selected_point2)
colors = []
for i in range(0,10):
    colors.append(tuple(np.random.randint(0,255,3).tolist()))

img1_lines = cv2.computeCorrespondEpilines(selected_point1.reshape(-1, 1, 2), 2, fundamentalmat)
img1_lines = img1_lines.reshape(-1, 3)
img1_lines1 = drawlines(m1_clr,m2_clr,img1_lines,selected_point1,selected_point2,colors)

img2_lines = cv2.computeCorrespondEpilines(selected_point2.reshape(-1, 1, 2), 2, fundamentalmat)
img2_lines = img1_lines.reshape(-1, 3)
img2_lines1 = drawlines(m2_clr,m1_clr,img2_lines,selected_point2,selected_point1,colors)


stereo = cv2.StereoBM_create(96, blockSize=17)
stereo.setMinDisparity(16)
stereo.setDisp12MaxDiff(0)
stereo.setUniquenessRatio(10)
stereo.setSpeckleRange(32)
stereo.setSpeckleWindowSize(100)
disparity_map = stereo.compute(image1_bw, image2_bw).astype(np.float32) / 16.0
disp_map = (disparity_map - 16)/96
```

```
# printing out all the output
plt.imsave('output/task2/task2_disparity.jpg', disp_map, cmap=plt.cm.gray)
cv2.imwrite('output/task2/task2_epi_right.jpg', img2_lines1)
cv2.imwrite('output/task2/task2_epi_left.jpg', img1_lines1)
cv2.imwrite("output/task2/merged.jpg", np.hstack([img2_lines1, img1_lines1]))
```

Task 3:(k means)

```
import cv2
import numpy as np
import math
import random
import sys
import matplotlib.pyplot as plt
from tqdm import tqdm
np.random.seed(sum([ord(c) for c in UBIT]))
COLORS = ["red","green","blue"]
def plot_points(points,clusters,marker):
    for i,x in enumerate(points):
        plt.scatter(x[0], x[1], edgecolor=COLORS[clusters[i]-1], facecolor='white', linewidth='1', marker=marker)
        plt.text(x[0]+0.03, x[1]-0.05, "("+str(x[0])+","+str(x[1])+")", color=COLORS[clusters[i]-1], fontsize='small')


def plot_centroids(centroids,colors,marker):
    for c,color in zip(centroids,colors):
        plt.scatter(c[0], c[1], marker=marker, s=200, c=color)
        plt.text(c[0]+0.03, c[1]-0.05, "("+str(c[0])+","+str(c[1])+")", color=color, fontsize='small')


def dist(points,dims):
    dist = 0
    if(dims == 2):
        dist = math.sqrt((points[0][0]-points[1][0])**2 + (points[0][1]-points[1][1])**2)
    elif(dims == 3):
        dist = math.sqrt((points[0][0]-points[1][0])**2 + (points[0][1]-points[1][1])**2 + (points[0][2]-points[1][2])**2)
    return dist


def make_random_points(dims,num_clusters,type):
```

```
    result = []
    for i in range(0,num_clusters):
        point = []
        for i in range(0,dims):
            point.append( random.randint(1,255))
        result.append(point)
    return result

def computer_kmeans(points,num_clusters,iterns,centeroids = None):
    dims = len(points[0])
    output = []
    mask = []
    temp_output = []
    temp_mask = []
    flag = True
    for i in range(0, num_clusters):
        output.append([])
    if centeroids is None:
        centeroids = make_random_points(dims,num_clusters,np.uint8)

    i = 0
    while((iterns > 0 and i < iterns) or(iterns == 0 and flag is True)):
        temp_output = []
        temp_mask = []
        flag = False
        i = i+1
        for j in range(0, num_clusters):
            temp_output.append([])
        for j,point in tqdm(enumerate(points)):
            old_dist = sys.float_info.max
            index = 0
            for k,centeroid in enumerate(centeroids):
                curr_dist = dist((point,centeroid),dims)

                if curr_dist < old_dist:
                    index = k
```

```
                old_dist = curr_dist
            temp_output[index].append(points[j])
            temp_mask.append(index+1)
        print(temp_mask)
        plot_points(points,temp_mask,'^')
        plt.savefig("output/task3/task_3_iter"+ str(i)+"_a"".jpg")
        plt.clf()



        for j in range(0,num_clusters):
            avg = [float(sum(col))/len(col) for col in zip(*temp_output[j])]
            print(centeroids[j],"avg = ",avg)
            if centeroids[j] != avg:
                flag = True
                centeroids[j] = avg
        plot_centroids(centeroids,COLORS,'.')
        plot_points(points,temp_mask,'^')
        plt.savefig("output/task3/task_3_iter"+ str(i)+"_b"".jpg")
        plt.clf()
    output = temp_output
    mask = temp_mask

    return output,centeroids,mask

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],[4.7,3.4],[6.7,3.1],[5.1,3.8],[6.0,3.0]]
centeroids = [[6.2,3.2],[6.6,3.7],[6.5,3.0]]
output,centeroids,mask = computer_kmeans(points,3,2,centeroids)


print(output)
print(centeroids)
print(mask)
```

Task 3:(K means vectorized)

```
UBIT = 'hparthas'
import cv2
import numpy as np
```

```
import math
import random
import sys
import matplotlib.pyplot as plt
from tqdm import tqdm
np.random.seed(sum([ord(c) for c in UBIT]))
COLORS = ["red","green","blue"]
def plot_points(points,clusters,marker,text=True):
  for i,x in enumerate(points):
        plt.scatter(x[0], x[1], edgecolor=COLORS[clusters[i]], facecolor='white', linewidth='1', marker=marker)
    if text:
        plt.text(x[0]+0.03, x[1]-0.05, "("+str(x[0])+","+str(x[1])+")", color=COLORS[clusters[i]], fontsize='small')

#Plot the centroids and its values if text is true
def plot_centroids(centroids,colors,marker,text=True):
  for i,c in enumerate(centroids):
    plt.scatter(c[0], c[1], marker=marker, s=200, c=COLORS[i])
    if text:
            plt.text(c[0]+0.03, c[1]-0.05, "("+str(c[0])+","+str(c[1])+")", color=COLORS[i], fontsize='small')
def make_image(kmeans_input,mask,centeroids,h,w):
  for i in range(0,len(mask)):
    kmeans_input[i] = centeroids[mask[i]]
  return kmeans_input.reshape((h, w, 3))

def make_random_points(points,num_clusters):
  centeroids = points.copy()
  np.random.shuffle(centeroids)
  return centeroids[:num_clusters]

def computer_kmeans(points,num_clusters,iterns,centeroids = None):
  flag = True
  if centeroids is None:
    centeroids = make_random_points(points,num_clusters)
```

```
    i = 0
    while((iterns > 0 and i < iterns) or(iterns == 0 and flag is True)):
        flag = False
        i = i+1
        # reference https://flothesof.github.io/k-means-numpy.html
        distances = np.sqrt(((np.array(points) - np.array(centeroids)[:, np.newaxis])**2).sum(axis=2))
        min_values = np.argmin(distances, axis=0)
        new_centeroids = np.array([points[min_values==k].mean(axis=0) for k in range(np.array(centeroids).shape[0])],dtype = int)

        if np.array_equiv(new_centeroids,np.array(centeroids)):
            print("reached_convergence after", i,"iterations")
            break
        else:
            flag = True
        centeroids = new_centeroids
    return centeroids, min_values

baboon_image = cv2.imread("data/baboon.jpg")
(h,w) = baboon_image.shape[:2]

kmeans_input = baboon_image.reshape((baboon_image.shape[0]* baboon_image.shape[1],3))

centeroids,mask = computer_kmeans(kmeans_input.copy(),3,0)
image = make_image(kmeans_input.copy(),mask,centeroids,h,w)
cv2.imwrite("output/task3/task3_baboon_3.jpg",image)

centeroids,mask = computer_kmeans(kmeans_input.copy(),5,0)
image = make_image(kmeans_input.copy(),mask,centeroids,h,w)
cv2.imwrite("output/task3/task3_baboon_5.jpg",image)

centeroids,mask = computer_kmeans(kmeans_input.copy(),10,0)
image = make_image(kmeans_input.copy(),mask,centeroids,h,w)
cv2.imwrite("output/task3/task3_baboon_10.jpg",image)

centeroids,mask = computer_kmeans(kmeans_input.copy(),20,0)
```

```
image = make_image(kmeans_input.copy(),mask,centeroids,h,w)
cv2.imwrite("output/task3/task3_baboon_20.jpg",image)
```

Task 3:(GMM)

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
COLORS = ["red","green","blue"]
def plot_cov_ellipse(cov, pos, nstd=2, ax=None, **kwargs):
    def eigsorted(cov):
        vals, vecs = np.linalg.eigh(cov)
        order = vals.argsort()[::-1]
        return vals[order], vecs[:,order]

    if ax is None:
        ax = plt.gca()

    vals, vecs = eigsorted(cov)
    theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))

    # Width and height are "full" widths, not radius
    width, height = 2 * nstd * np.sqrt(vals)
    ellip = Ellipse(xy=pos, width=width, height=height, angle=theta, **kwargs)

    ax.add_artist(ellip)
    return ellip

def prob_density_function(b,mean,cov):
        return np.exp(-0.5*b.shape[0]*np.log(2*np.pi)) * (np.power(np.linalg.det(cov),-0.5)) *
(np.exp(-0.5*np.dot(np.dot((b-mean).transpose(),np.linalg.inv(cov)),(b-mean))))


def compute_cluster_probablity(p,m,s):
    cluster_probablity = np.zeros((len(input_array),K))
    for i in range(len(input_array)):
```

```
        for j in range(K):
            cluster_probablity[i,j] = p[j]*prob_density_function(input_array[i],m[j],s[j])
        cluster_probablity[i] /= (np.sum(cluster_probablity[i]))
    return cluster_probablity
def compute_gmm(K,pi_value,mean,co_variance,num_iterns,showplot,points):
    for i in range(num_iterns):
        Q = compute_cluster_probablity(pi_value, mean, co_variance)
        new_pi_value = np.sum(Q,axis=0)/len(Q)
        numerator = zip(np.sum(Q*input_array[:,0][:,np.newaxis],axis=0),np.sum(Q*input_array[:,1][:,np.newaxis],axis=0))
        new_mu_value = numerator/(np.sum(Q,axis=0)[:,np.newaxis])
        computed_probablities = np.zeros([len(Q),K,2,2])
        for t in range(len(Q)):
            for j in range(K):
                dot_val1 = (input_array[t]-new_mu_value[j])[:,np.newaxis]
                dot_val2 = (input_array[t]-new_mu_value[j])[np.newaxis,:]
                computed_probablities[t,j] = Q[t,j] * np.dot(dot_val1,dot_val2)
        new_cov_value = np.sum(computed_probablities,axis=0)/np.sum(Q,axis=0)[:,np.newaxis,np.newaxis]
        pi_value = new_pi_value
        mean = new_mu_value
        co_variance = new_cov_value
        print("iteration",i+1)
        if(showplot):
            plt.clf()
            for j,x in enumerate(input_array):
                plt.scatter(x[0], x[1], edgecolor="red", facecolor='white', linewidth='1', marker=".")
            for j in range(3):
                plot_cov_ellipse(co_variance[j], mean[j], nstd=2,ax=None, alpha = 0.5,color = COLORS[j])
            plt.savefig("output/task3/task_3_gmm_iter"+str(i+1)+".png")
            plt.close()
    print("mean",mean)
# for task 1
input_array = 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],[4.7,3.4],[6.7,3.1],[5.1,3.8],[6.0,3.0]])
```

```
K = 3

pi_value = np.array([1.0/3,1.0/3,1.0/3])

mean = np.array([[6.2,3.2],[6.6,3.7], [6.5,3.0]])

co_variance = np.array([[[0.5,0],[0,0.5]], [[0.5,0],[0,0.5]], [[0.5,0],[0,0.5]]])

compute_gmm(K,pi_value,mean,co_variance,1,False,None)

# for task 2

input_array = np.array([[3.600, 79],[1.800, 54],[3.333, 74],[2.283, 62],[4.533, 85],[2.883, 55],[4.700, 88],[3.600, 85],[1.950, 51],[4.350, 85],[1.833, 54],[3.917, 84],[4.200, 78],[1.750, 47],[4.700, 83],[2.167, 52],[1.750, 62],[4.800, 84],[1.600, 52],[4.250, 79],[1.800, 51],[1.750, 47],[3.450, 78],[3.067, 69],[4.533, 74],[3.600, 83],[1.967, 55],[4.083, 76],[3.850, 78],[4.433, 79],[4.300, 73],[4.467, 77],[3.367, 66],[4.033, 80],[3.833, 74],[2.017, 52],[1.867, 48],[4.833, 80],[1.833, 59],[4.783, 90],[4.350, 80],[1.883, 58],[4.567, 84],[1.750, 58],[4.533, 73],[3.317, 83],[3.833, 64],[2.100, 53],[4.633, 82],[2.000, 59],[4.800, 75],[4.716, 90],[1.833, 54],[4.833, 80],[1.733, 54],[4.883, 83],[3.717, 71],[1.667, 64],[4.567, 77],[4.317, 81],[2.233, 59],[4.500, 84],[1.750, 48],[4.800, 82],[1.817, 60],[4.400, 92],[4.167, 78],[4.700, 78],[2.067, 65],[4.700, 73],[4.033, 82],[1.967, 56],[4.500, 79],[4.000, 71],[1.983, 62],[5.067, 76],[2.017, 60],[4.567, 78],[3.883, 76],[3.600, 83],[4.133, 75],[4.333, 82],[4.100, 70],[2.633, 65],[4.067, 73],[4.933, 88],[3.950, 76],[4.517, 80],[2.167, 48],[4.000, 86],[2.200, 60],[4.333, 90],[1.867, 50],[4.817, 78],[1.833, 63],[4.300, 72],[4.667, 84],[3.750, 75],[1.867, 51],[4.900, 82],[2.483, 62],[4.367, 88],[2.100, 49],[4.500, 83],[4.050, 81],[1.867, 47],[4.700, 84],[1.783, 52],[4.850, 86],[3.683, 81],[4.733, 75],[2.300, 59],[4.900, 89],[4.417, 79],[1.700, 59],[4.633, 81],[2.317, 50],[4.600, 85],[1.817, 59],[4.417, 87],[2.617, 53],[4.067, 69],[4.250, 77],[1.967, 56],[4.600, 88],[3.767, 81],[1.917, 45],[4.500, 82],[2.267, 55],[4.650, 90],[1.867, 45],[4.167, 83],[2.800, 56],[4.333, 89],[1.833, 46],[4.383, 82],[1.883, 51],[4.933, 86],[2.033, 53],[3.733, 79],[4.233, 81],[2.233, 60],[4.533, 82],[4.817, 77],[4.333, 76],[1.983, 59],[4.633, 80],[2.017, 49],[5.100, 96],[1.800, 53],[5.033, 77],[4.000, 77],[2.400, 65],[4.600, 81],[3.567, 71],[4.000, 70],[4.500, 81],[4.083, 93],[1.800, 53],[3.967, 89],[2.200, 45],[4.150, 86],[2.000, 58],[3.833, 78],[3.500, 66],[4.583, 76],[2.367, 63],[5.000, 88],[1.933, 52],[4.617, 93],[1.917, 49],[2.083, 57],[4.583, 77],[3.333, 68],[4.167, 81],[4.333, 81],[4.500, 73],[2.417, 50],[4.000, 85],[4.167, 74],[1.883, 55],[4.583, 77],[4.250, 83],[3.767, 83],[2.033, 51],[4.433, 78],[4.083, 84],[1.833, 46],[4.417, 83],[2.183, 55],[4.800, 81],[1.833, 57],[4.800, 76],[4.100, 84],[3.966, 77],[4.233, 81],[3.500, 87],[4.366, 77],[2.250, 51],[4.667, 78],[2.100, 60],[4.350, 82],[4.133, 91],[1.867, 53],[4.600, 78],[1.783, 46],[4.367, 77],[3.850, 84],[1.933, 49],[4.500, 83],[2.383, 71],[4.700, 80],[1.867, 49],[3.833, 75],[3.417, 64],[4.233, 76],[2.400, 53],[4.800, 94],[2.000, 55],[4.150, 76],[1.867, 50],[4.267, 82],[1.750, 54],[4.483, 75],[4.000, 78],[4.117, 79],[4.083, 78],[4.267, 78],[3.917, 70],[4.550, 79],[4.083, 70],[2.417, 54],[4.183, 86],[2.217, 50],[4.450, 90],[1.883, 54],[1.850, 54],[4.283, 77],[3.950, 79],[2.333, 64],[4.150, 75],[2.350, 47],[4.933, 86],[2.900, 63],[4.583, 85],[3.833, 82],[2.083, 57],[4.367, 82],[2.133, 67],[4.350, 74],[2.200, 54],[4.450, 83],[3.567, 73],[4.500, 73],[4.150, 88],[3.817, 80],[3.917, 71],[4.450, 83],[2.000, 56],[4.283, 79],[4.767, 78],[4.533, 84],[1.850, 58],[4.250, 83],[1.983, 43],[2.250, 60],[4.750, 75],[4.117, 81],[2.150, 46],[4.417, 90],[1.817, 46],[4.467, 74]])

K = 3

pi_value = np.array([1.0/3,1.0/3,1.0/3])

mean = np.array([[4.0,81],[2.0,57], [4.0,71]])

co_variance = np.array([[[[1.30,13.98],[13.98,184.82]], [[1.30,13.98],[13.98,184.82]], [[1.30,13.98],[13.98,184.82]]])

compute_gmm(K,pi_value,mean,co_variance,5,True,input_array)
```