CSE 573: Introduction to Computer Vision and Image Processing

Instructor: Dr. Junsong Yuan

University at Buffalo

Project 2 Report

Submitted By:

Shreyas Narasimha (UB Person Number: 50289736)

Task 1 – Image Features and Homography

Objective: Given two stereo images, i.e two images of the same scene taken by different camaeras; the objectives are to :-

- To extract features from a given image using Scale Invariant Feature Transforms.
- To match the features represented by keypoints using k-nearest neighbour.
- Compute Homography matrix from the first image to the second image and warp the first image to the second image.

Developed code:

```
#Shreyas N
#UBIT Name: sn58
#Person No. - 50289736
import cv2
import numpy as np
import math
UBID = '50289736'
np.random.seed(sum([ord(c) for c in UBID]))
def display(name,img):
    cv2.namedWindow(name, cv2.WINDOW_NORMAL)
    cv2.imshow(name, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
imc1 = cv2.imread('mountain1.jpg')
imc2 = cv2.imread('mountain2.jpg')
img1 = cv2.imread('mountain1.jpg',0)
img2 = cv2.imread('mountain2.jpg',0)
#display('',img1)
#display('',img2)

# Creating a SIFT Object
sift = cv2.xfeatures2d.SIFT_create()
#Gettig keypoints
k1,d1 = sift.detectAndCompute(img1, None)
k2,d2 = sift.detectAndCompute(img2, None)
#Overlay keypoints
i1 = imc1
i2 = imc2
imgk1 = cv2.drawKeypoints(i1, k1, None)
imgk2 = cv2.drawKeypoints(i2, k2, None)
#Save image to a file
imgmatcher = cv2.BFMatcher()
matches = imgmatcher.knnMatch(d1,d2,k=2)
g = []
for m,n in matches:
    if m.distance<0.75*n.distance:
        g.append(m)
nomask = cv2.drawMatches(imc1,k1,imc2,k2,g,None,flags = 2)

#Creating a 3x3 mask for finding inliers using RANSAC
source = np.float32([ k1[m.queryIdx].pt for m in g ]).reshape(-1,1,2)
destination = np.float32([ k2[m.trainIdx].pt for m in g ]).reshape(-1,1,2)
homo, mask = cv2.findHomography(source, destination, cv2.RANSAC,5.0)
#print(homo)
#Turn it into a form we can pass as a parameter by ravelling
matchesMask = mask.ravel().tolist()

#Compute the panaramic image after warping the first image
height1,width1 = img1.shape
height2,width2 = img2.shape
```

```
pointssrc = np.float32([ [0,0],[0,height1],[width1,height1],[width1,0] ]).reshape(-1,1,2)
pointsdest = np.float32([ [0,0],[0,height2],[width2,height2],[width2,0] ]).reshape(-1,1,2)
pointssrc1 = cv2.perspectiveTransform(pointssrc, homo)

points = np.concatenate((pointssrc1, pointsdest), axis=0)
[xmin, ymin] = np.int32(points.min(axis=0).ravel() - 0.6)
[xmax, ymax] = np.int32(points.max(axis=0).ravel() + 0.6)

t = [-xmin,-ymin]
translate = np.array([[1,0,t[0]],[0,1,t[1]],[0,0,1]]) # translate

res = cv2.warpPerspective(imc1, translate.dot(homo), (xmax-xmin, ymax-ymin))
res[t[1]:height2+t[1],t[0]:width2+t[0]] = imc2


indices = []
for i in range(0,len(matchesMask)):
    if(matchesMask[i] == 1):
        indices.append(i)
np.random.shuffle(indices)
rand = indices[:10]

for i in range(0,len(matchesMask)):
    matchesMask[i] = 0

for i in rand:
    matchesMask[i] = 1
imgmatch = cv2.drawMatches(imc1,k1,imc2,k2,g,None,flags = 2, matchesMask =
matchesMask,matchColor = (255,0,0))

#Warping the images
#print(img2.shape)
imc2 = cv2.warpPerspective(imc1, homo, (img1.shape[1],img1.shape[0]))
'''
display('',imgk1)
display('',imgk2)
display('',imgmatch)
display('',nomask)
display('',res)
'''
print('The homography image H is:\n '+str(homo))
cv2.imwrite('task1_sift1.jpg',imgk1)
cv2.imwrite('task1_sift2.jpg',imgk2)
cv2.imwrite('task1_matches.jpg',imgmatch)
cv2.imwrite('task1_matches_knn.jpg',nomask)
cv2.imwrite('task1_pano.jpg',res)
```

Input:

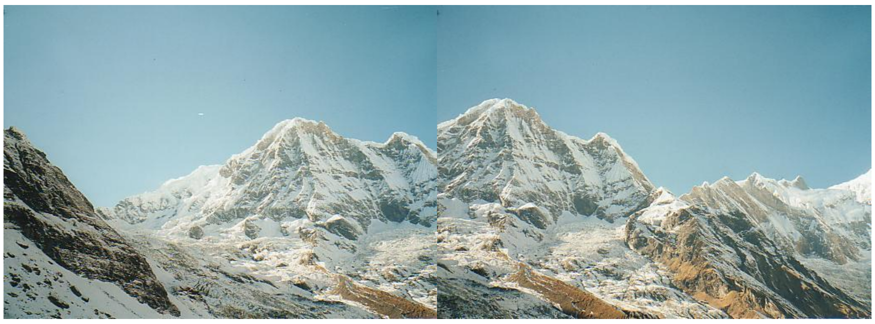


Fig. 1 The input images

Implementation and Output:

1) Extracting SIFT features:
    - Read the two images using `cv2.imread.`
    - Create a SIFT object using `cv2.xfeatures2d.SIFT_create()` and extract the keypoints using `.detectAndCompute.`

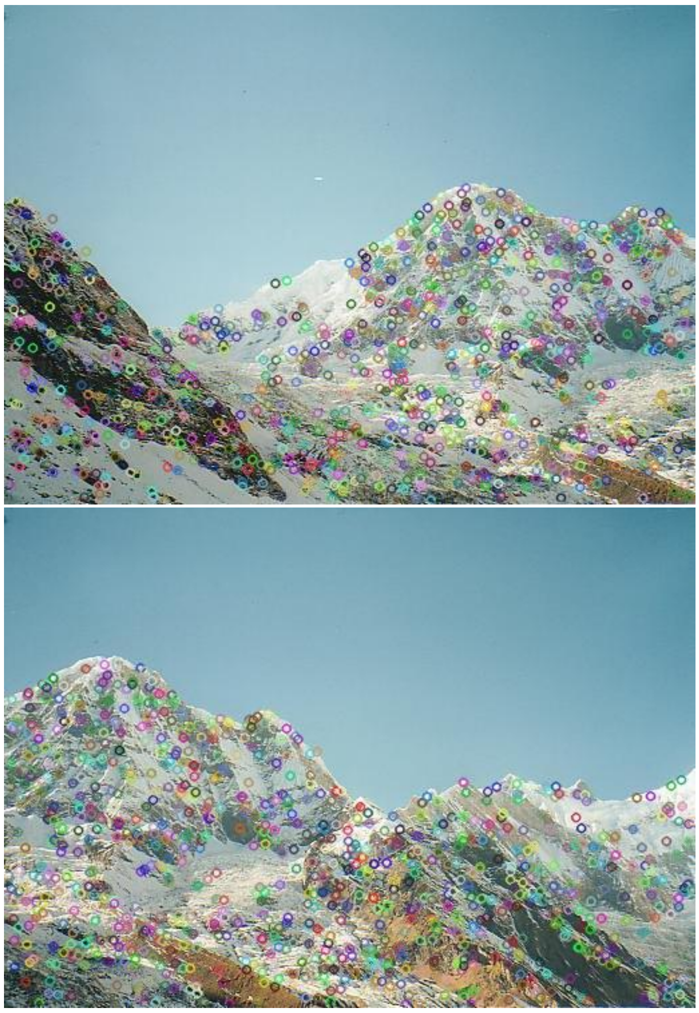


Fig. 2 Keypoints detected

2) Match keypoints using k-nearest neighbour:
    - Create a matcher object using `cv2.BFMatcher()` and find the matching keypoints using `.knnmatch` with k=2. The matching is done with a threshold of 0.75 for distance. The matches are drawn using `cv2.drawMatches.`

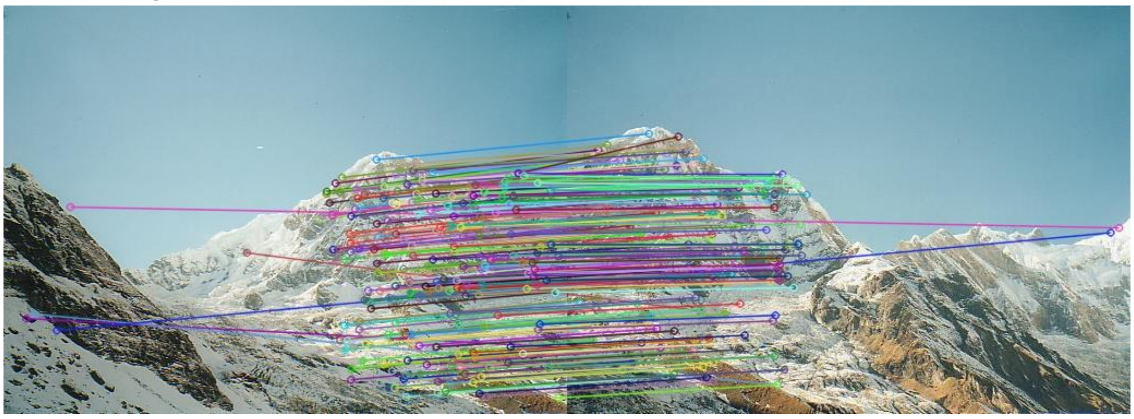
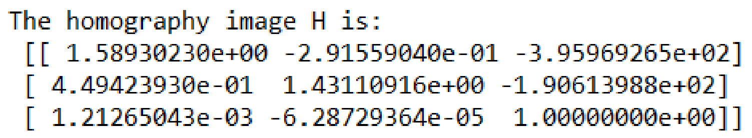


Fig. 3 Image matches having both inliers and outliers

3) Compute Homography matrix :

- Use the `findHomography` and find H and the mask using RANSAC with threshold as 5.



```
The homography image H 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]]
```

Fig. 4 The homography matrix as printed by the program

4) Draw 10 random matches:

- Use `np.random.shuffle` to shuffle the indices in the mask which correspond do inliers and select 10 random points.(We choose the first 10 since the indices have been shuffled.)
- Modify the mask using the 10 random inliers selected and apply it on the images while using `cv2.drawMatches.`

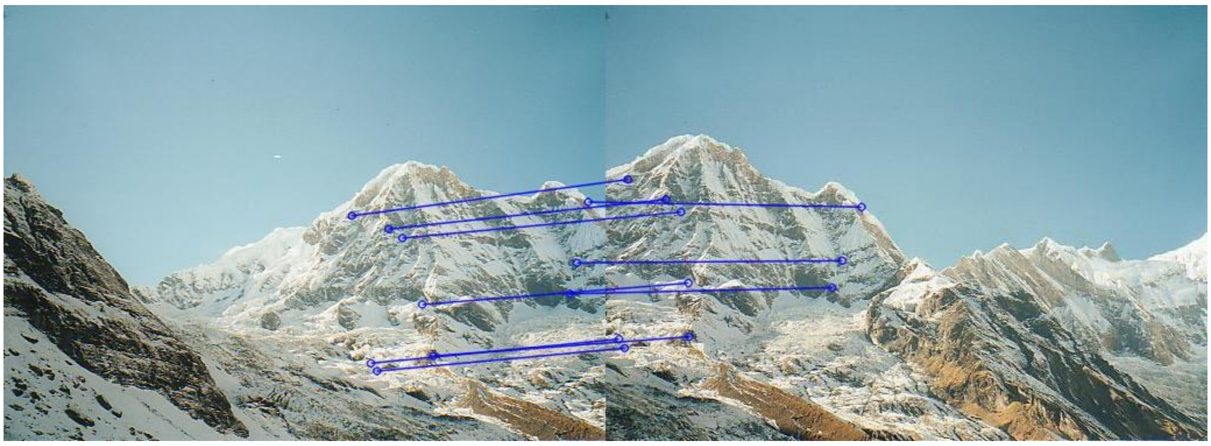


Fig. 5 The matches for 10 random inliers

5) Warping the image:

- The homography matrix calculated is to warp the first image to the second. But, it contains some negative values which cannot be plotted, thus leading to loss of pixels.
- We process the image transformed using `perspectiveTransform` and then translate the image by offsets which we calculate using the concatenation of both the images.
- Use the `warpPerspective` function to warp the first image to the second.

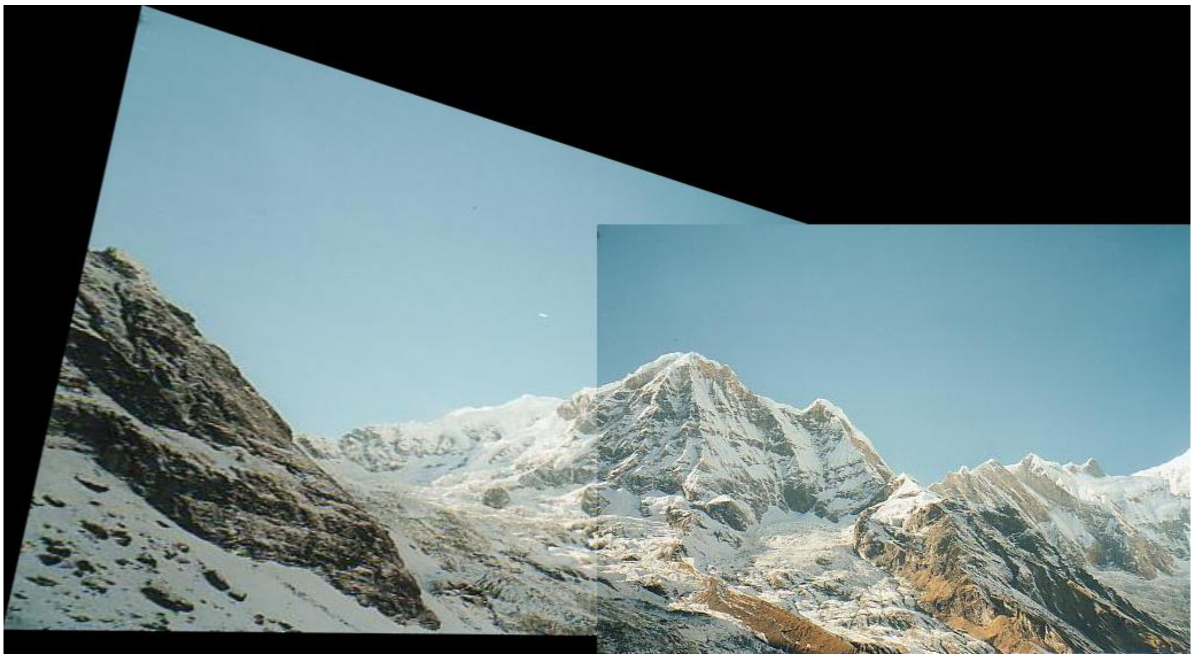


Fig. 6 Result after warping first image to the second

Task 2 – Epipolar Geometry

Objective: Given two stereo images, i.e two images of the same scene taken by different camaeras; the objectives are to :-

- Extract features using SIFT and find matches using k-nearest neighbour.
- Compute the Fundamental Matrix F.
- Select 10 random inliers and compute the epilines for both images w.r.t each other.
- Compute the disparity map using the two images.

Developed code:

```
#Shreyas N
#UBIT Name: sn58
#Person No. - 50289736
def task2():
    import cv2
    import numpy as np
    import math
    UBID = '50289736'
    np.random.seed(sum([ord(c) for c in UBID]))

    def draw_epi(p1,p2,line,i1,i2,ic1,ic2,colo):
        h = i1.shape[0]
        w = i1.shape[1]
        for c,pp1,pp2,l in zip(colo,p1,p2,line):
            #print(l)
            x0,y0 = map(int, [0, -l[2]/l[1]])
            x1,y1 = map(int, [w, -(l[2]+l[0]*w)/l[1]])
            ic1 = cv2.line(ic1, (x0,y0), (x1,y1),
(np.int(c[0]),np.int(c[1]),np.int(c[2])),1)
            ic1 = cv2.circle(ic1,tuple(pp1),3,(np.int(c[0]),np.int(c[1]),np.int(c[2])),-1)
        return ic1

    def display(name,img):
        cv2.namedWindow(name, cv2.WINDOW_NORMAL)
        cv2.imshow(name, img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    imc1 = cv2.imread('tsucuba_left.png')
    imc2 = cv2.imread('tsucuba_right.png')
    img1 = cv2.imread('tsucuba_left.png',0)
    img2 = cv2.imread('tsucuba_right.png',0)

    #Compute the panaramic image after warping the first image
    height1,width1 = img1.shape
    height2,width2 = img2.shape
    # Creating a SIFT Object
    sift = cv2.xfeatures2d.SIFT_create()
    #Gettig keypoints
    k1,d1 = sift.detectAndCompute(img1, None)
    k2,d2 = sift.detectAndCompute(img2, None)
    #Overlay keypoints
    i1 = imc1
    i2 = imc2
    imgk21 = cv2.drawKeypoints(i1, k1, None)
    imgk22 = cv2.drawKeypoints(i2, k2, None)
    #Save image to a file
    imgmatcher = cv2.BFMatcher()
    matches = imgmatcher.knnMatch(d1,d2,k=2)
    g = []
    for m,n in matches:
        if m.distance<0.75*n.distance:
            g.append(m)
    nomask = cv2.drawMatches(imc1,k1,imc2,k2,g,None,flags = 2)

    source = np.int32([ k1[m.queryIdx].pt for m in g ]).reshape(-1,1,2)
    destination = np.int32([ k2[m.trainIdx].pt for m in g ]).reshape(-1,1,2)

    F, mask = cv2.findFundamentalMat(source,destination,cv2.RANSAC,50.0)
    #print(F)
    matchesMask = mask.ravel().tolist()

    indices = []
    #imgmatch = cv2.drawMatches(imc1,k1,imc2,k2,g,None,flags = 2, matchesMask =
matchesMask,matchColor = (255,0,0))
    for i in range(0,len(matchesMask)):
        if(matchesMask[i] == 1):
            indices.append(i)
    np.random.shuffle(indices)
    rand = indices[:10]

    for i in range(0,len(matchesMask)):
        matchesMask[i] = 0

    for i in rand:
        matchesMask[i] = 1
    imgmatch = cv2.drawMatches(imc1,k1,imc2,k2,g,None,flags = 2, matchesMask =
matchesMask,matchColor = (255,0,0))
    #print(indices)
    #computing epilines
    color = []
    #color = tuple(np.random.randint(0,255,3).tolist())
    #for i in range(10):
        #color.append(np.random.randint(0,200,3))
    color.append([255,0,0])
    color.append([0,255,0])
    color.append([0,0,255])
    color.append([255,0,255])
    color.append([0,255,255])
    color.append([255,255,0])
    color.append([4,120,255])
    color.append([150,25,60])
    color.append([200,200,255])
    color.append([255,255,255])
    color.append([0,0,255])
    #color = np.int32(color)
    g1 = []
    for i in rand:
        g1.append(g[i])

    source = np.int32([ k1[m.queryIdx].pt for m in g1]).reshape(-1,1,2)
    destination = np.int32([ k2[m.trainIdx].pt for m in g1]).reshape(-1,1,2)
    #print(source1)
    #print(destination1)
    l2 = cv2.computeCorrespondEpilines(source.reshape(-1,1,2),1,F).reshape(-1,3)
    l1 = cv2.computeCorrespondEpilines(destination.reshape(-1,1,2),2,F).reshape(-1,3)
    '''
    for c,i,p1,p2 in zip(color,l1,source.reshape(-1,2),destination.reshape(-1,2)):
        x,y = map(int, [0,-i[2]/i[1]])
        x1,y1 = map(int,[width1, -(i[2]+i[0]*width1)/i[1]])
        img_1 = cv2.line(imc1, (x,y), (x1,y1),(np.int(c[0]),np.int(c[1]),np.int(c[2])),1)
        img_1 = cv2.circle(imc1,tuple(p1),2,(np.int(c[0]),np.int(c[1]),np.int(c[2])),-1)
        #img_2 = cv2.circle(imc2,tuple(p2),2,(np.int(c[0]),np.int(c[1]),np.int(c[2])),-1)
    for c,i,p1,p2 in zip(color,l1,destination.reshape(-1,2),source.reshape(-1,2)):
        x,y = map(int, [0,-i[2]/i[1]])
        x1,y1 = map(int,[width2, -(i[2]+i[0]*width2)/i[1]])
        img_3 = cv2.line(imc2, (x,y), (x1,y1),(np.int(c[0]),np.int(c[1]),np.int(c[2])),1)
        img_3 = cv2.circle(imc2,tuple(p1),2,(np.int(c[0]),np.int(c[1]),np.int(c[2])),-1)
        img_4 = cv2.circle(imc1,tuple(p2),2,(np.int(c[0]),np.int(c[1]),np.int(c[2])),-1)
    '''
    img_1 = draw_epi(source.reshape(-1,2),destination.reshape(-
1,2),l1,img1,img2,imc1,imc2,color)
    img_2 = draw_epi(destination.reshape(-1,2),source.reshape(-
1,2),l2,img2,img1,imc2,imc1,color)
    #mix = cv2.StereoBM_create(numDisparities=96, blockSize = 5)
    mix =  cv2.StereoBM_create(numDisparities = 64,blockSize = 15)
    #disparity = mix.compute(img1,img2)
    disparity = mix.compute(img1, img2).astype(np.float32) / 3.0
    #disparity = (disparity-32)/8
    disparity = disparity[:,60:]
    print('The Fundamental Matrix F is: \n'+str(F))
    cv2.imwrite('task2_sift1.jpg',imgk21)
    cv2.imwrite('task2_sift2.jpg',imgk22)
    cv2.imwrite('task2_matches_knn.jpg',nomask)
    #cv2.imwrite('task2_matches_knn.jpg',imgmatch)
    cv2.imwrite('task2_epi_left.jpg',img_1)
    cv2.imwrite('task2_epi_right.jpg',img_2)
    cv2.imwrite('task2_disparity.jpg',disparity)

task2()
```

Input:

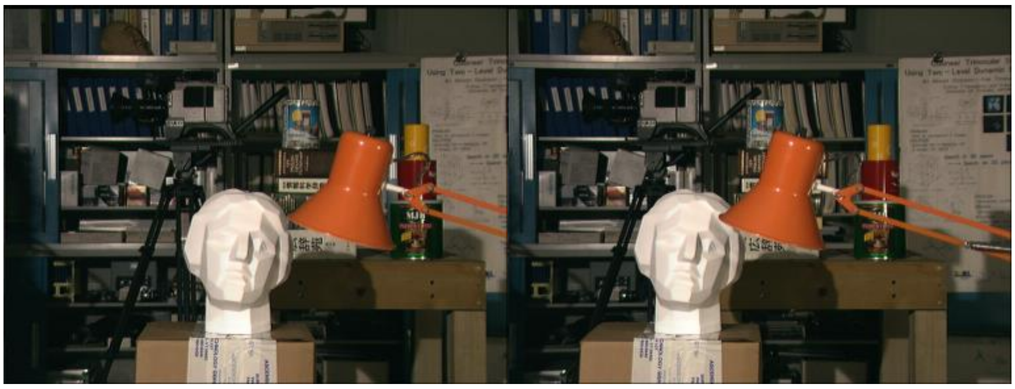


Fig. 7 The input stereo images

Output and Implementation:

1) Extracting SIFT features:
    - Read the two images using `cv2.imread.`
    - Create a SIFT object using `cv2.xfeatures2d.SIFT_create()` and extract the keypoints using `.detectAndCompute.`

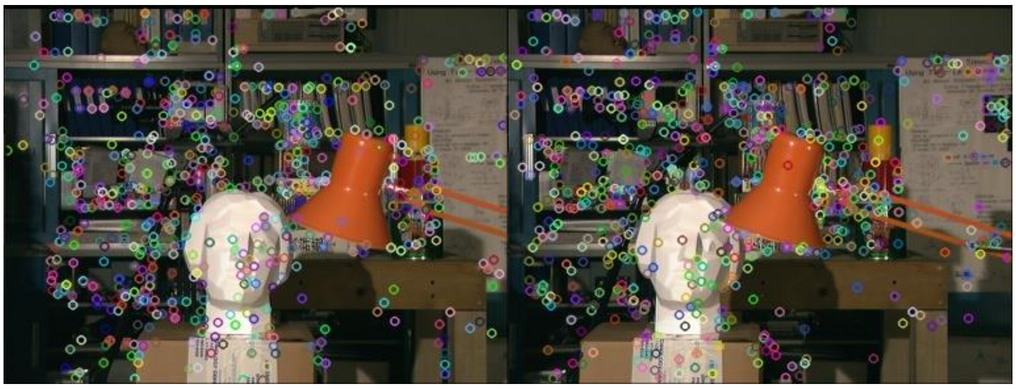


Fig. 8 The SIFT keypoints

2) Match keypoints using k-nearest neighbour:
    - Create a matcher object using `cv2.BFMatcher()` and find the matching keypoints using `.knnmatch` with k=2. The matching is done with a threshold of 0.75 for distance. The matches are drawn using `cv2.drawMatches`.

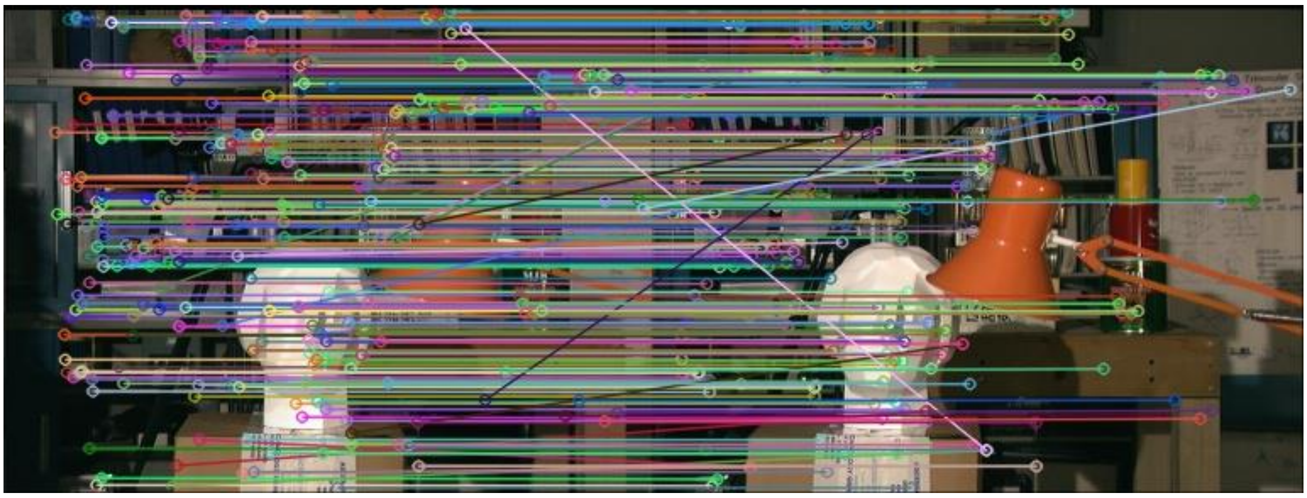
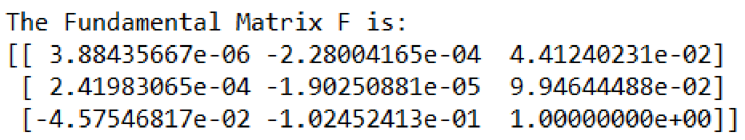


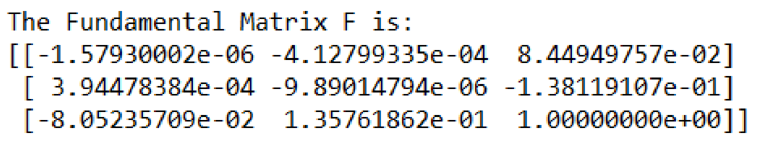
Fig. 9 The pairs generated after k-nearest neighbour.

3) Compute Fundamental Matrix:
    - Compute the Fundamental Matrix and its mask using `findFundamentlMat.`



```
The Fundamental Matrix F is:
[[ 3.88435667e-06 -2.28004165e-04  4.41240231e-02]
 [ 2.41983065e-04 -1.90250881e-05  9.94644488e-02]
 [-4.57546817e-02 -1.02452413e-01  1.00000000e+00]]
```

Fig. 10 The Fundamental matrix as printed by the program for a poor fit(RANSAC param = 5)



```
The Fundamental Matrix F is:
[[-1.57930002e-06 -4.12799335e-04  8.44949757e-02]
 [ 3.94478384e-04 -9.89014794e-06 -1.38119107e-01]
 [-8.05235709e-02  1.35761862e-01  1.00000000e+00]]
```

Fig. 11 The fundamental matrix as printed by the program for a good fit(RANSAC param = 50)

4) Compute epilines for 10 random inliers:
   - 10 random inliers are selected using the same method as in **Task2.**
   - Compute the epilines for these points using `cv2.computeCorrespondEpilines` by passing the left and then the right image as parameters.
   - Draw the points and lines using `cv2.circle` and `cv2.line`.
   - Repeat the last two steps to find the epilines for the right image with respect to the left one, by passing the right image first and then the left one into the `cv2.computeCorrespondEpilines` function.

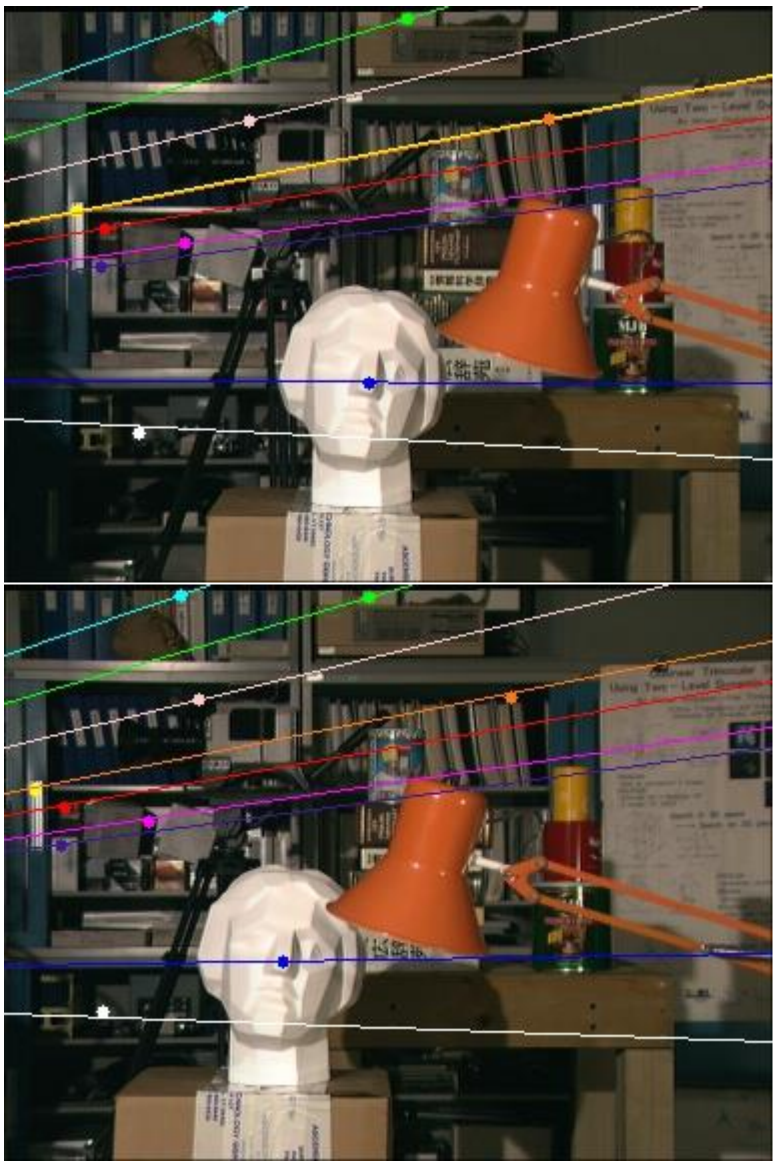


Fig. 12 Epilines for the two images(poor fit)

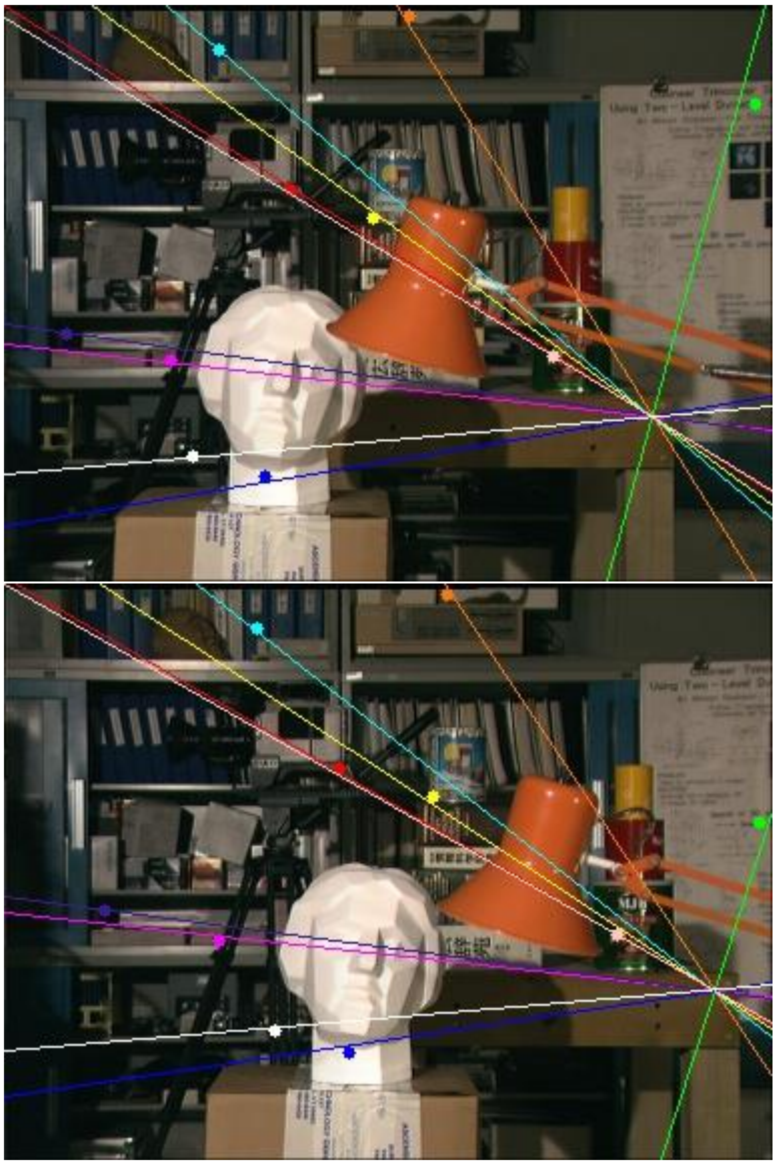


Fig. 13 Epilines for the two images(good fit)

5) Compute disparity:
    - Create a stereoBM object using `cv2.StereoBM_create` function passing suitable parameters(We pass 64 and 15).
    - Find the disparity matrix by using `.compute` on the object.
    - We then clip the disparity matrix to get rid of the displacement.

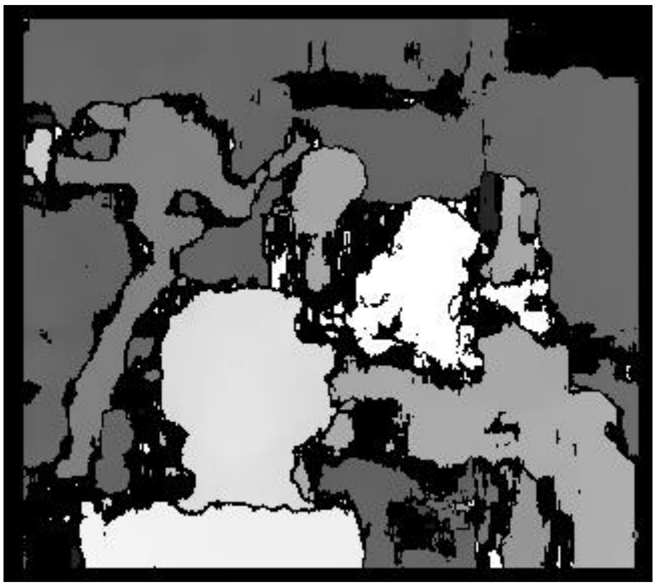


Fig. 14 The disparity map.(Lighter colors are closer)

Task 3 – K-Means Clustering

Objective: To achieve the following:

1) Given a dataset of 10 points and 3 initial values for cluster centres, perform K-means clustering on the points using the given cluster centres.
2) Update the initial cluster centers.
3) Achieve Image quantization using k-means clustering for colors of a given image.

Developed code:

The code was split into 4 different files/functions and all were called from one .py file(task3.py)

task3.py

```
import task3_GMM
import task3_GMMb
import task3_kmeans
import task3_baboon
```

task3_kmeans.py

```
#Shreyas N
#UBIT Name: sn58
#Person No. - 50289736
def task3_kmeans():
    import cv2
    import numpy as np
    import math
    from matplotlib import pyplot as plt
    UBID = '50289736'
    np.random.seed(sum([ord(c) for c in UBID]))

    #The datapoints matrix is
    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]]
    X = np.asarray(X)
    MU = [[6.2,3.2],[6.6,3.7],[6.5,3.0]]

    def iterations(mu,i):
        cva = classify_and_plot(mu[0],mu[1],mu[2])
        plt.savefig('task3_iter'+str(i+1)+'_a.jpg')
        plt.clf()
        mu = recompute(cva,mu)
        plot_mu(mu[0],mu[1],mu[2])
        plt.savefig('task3_iter'+str(i+1)+'_b.jpg')
        plt.clf()
        print('The recomputed MU are:')
        for i in mu:
            print(i)
        return mu
    def recompute(class_vector,mu):
        x1 = 0
        x2 = 0
        x3 = 0
        y1 = 0
        y2 = 0
        y3 = 0
        count1 = 0
        count2 = 0
        count3 = 0
        print(class_vector)
        for x,i,class_v in zip(X,range(0,len(X)),class_vector):
            if(class_v == mu[0]):
                x1 = x1 + x[0]
                y1 = y1 + x[1]
                count1 = count1+1
            elif(class_v == mu[1]):
                x2 = x2 + x[0]
                y2 = y2 + x[1]
                count2 = count2+1
            elif(class_v == mu[2]):
                x3 = x3 + x[0]
                y3 = y3 + x[1]
                count3 = count3+1
        if(count1!=0):
            x1 = round(x1/count1, 2)
            y1 = round(y1/count1, 2)
        if(count2!=0):
            x2 = round(x2/count2, 2)
            y2 = round(y2/count2, 2)
        if(count3!=0):
            x3 = round(x3/count3, 2)
            y3 = round(y3/count3, 2)
        mu[0] = [x1,y1]
        mu[1] = [x2,y2]
        mu[2] = [x3,y3]
        return mu
    def plot_mu(mu0,mu1,mu2):
        plt.text(mu0[0]+0.03,mu0[1], s = '('+str(mu0[0])+','+str(mu0[1])+')',fontsize = 8)
        plt.text(mu1[0]+0.03,mu1[1], s = '('+str(mu1[0])+','+str(mu1[1])+')',fontsize = 8)
        plt.text(mu2[0]+0.03,mu2[1], s = '('+str(mu2[0])+','+str(mu2[1])+')',fontsize = 8)

        plt.scatter(mu0[0],mu0[1],s = 50,c = 'r',marker = 'o', linewidths = 1,edgecolor = 
'r' )
        plt.scatter(mu1[0],mu1[1],s = 50,c = 'g',marker = 'o', linewidths = 1,edgecolor = 
'g' )
        plt.scatter(mu2[0],mu2[1],s = 50,c = 'b',marker = 'o', linewidths = 1,edgecolor = 
'b' )

    def classify_and_plot(mu0,mu1,mu2):
        #Initializing Centres
        plot_mu(mu0,mu1,mu2)
        for i in X:
            plt.text(i[0]+0.03,i[1], s = '('+str(i[0])+','+str(i[1])+')',fontsize = 8)
        class_vec = []
        for x in X:
            euc1 = np.sqrt((mu0[0]-x[0])**2 + (mu0[1]-x[1])**2)
            euc2 = np.sqrt((mu1[0]-x[0])**2 + (mu1[1]-x[1])**2)
            euc3 = np.sqrt((mu2[0]-x[0])**2 + (mu2[1]-x[1])**2)
            if(min(euc1,euc2,euc3) == euc1):
                class_vec.append(mu0)
            elif(min(euc1,euc2,euc3) == euc2):
                class_vec.append(mu1)
            elif(min(euc1,euc2,euc3) == euc3):
                class_vec.append(mu2)

        for i,j in zip(class_vec,X):
            if(i == mu0):
                plt.scatter(j[0],j[1],s = 50,c = 'w',marker = '^', linewidths = 
1,edgecolor = 'r' )
            if(i == mu1):
                plt.scatter(j[0],j[1],s = 50,c = 'w',marker = '^', linewidths = 
1,edgecolor = 'g' )
            if(i == mu2):
                plt.scatter(j[0],j[1],s = 50,c = 'w',marker = '^', linewidths = 
1,edgecolor = 'b' )
        print('The classification vector is: ')
        for i in class_vec:
            print(i)
        return class_vec

    for i in range(2):
        MU = iterations(MU,i)
    ########baboooooon#########
```

task3_baboon.py

```
#Shreyas N
#UBIT Name: sn58
#Person No. - 50289736
def task3_baboon():
    import cv2
    import numpy as np
    import math
    from matplotlib import pyplot as plt
    import time

    UBID = '50289736'
    np.random.seed(sum([ord(c) for c in UBID]))

    def display(name,img):
        cv2.namedWindow(name, cv2.WINDOW_NORMAL)
        cv2.imshow(name, img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    s = time.time()
    baboon = cv2.imread('baboon.jpg')
    h,w,c = baboon.shape
    MU = []
    baboon = baboon.reshape((baboon.shape[0]*baboon.shape[1]),3)
    temp = baboon.copy()
    np.random.shuffle(temp)

    def draw_baboon(clusters):

        MU = []
        for i,j in zip(range(clusters),temp):
            MU.append(temp[i])
        MU = np.asarray(MU)

        X = baboon
        def iterations(mu):
            cva = []
            cva,kc = classify_and_plot(mu)
            mu = recompute(cva,mu,kc)
            return mu,cva
        def recompute(class_vector,mu,kc):

            nmu = []
            for i in range(len(mu)):
                clus = np.asarray(kc[i])
                meaney = np.mean(clus, axis = 0)
                nmu.append(meaney.tolist())
            return np.asarray(nmu)

        def classify_and_plot(mu):
            class_vec = []
            kc = []
            for i in range(clusters):
                kc.append([])
            for x in X:
                euc = []
                for i in range(len(mu)):
                    euc.append(np.sqrt(sum(np.square(np.subtract(mu[i],x)))))
                t = euc.index(min(euc))
                class_vec.append(mu[t])
                #print(t)
                kc[t].append(x)
            return np.asarray(class_vec),kc

        for i in range(20):
            MU,image = iterations(np.asarray(MU))
        image = image.reshape(h,w,3)
        cv2.imwrite('task3_baboon_'+str(clusters)+'.jpg',image)
        print('Time taken for k = '+ str(clusters))
        taime = time.time() - s
        print(taime)
    a = [3,5,10,20]
    #a = [3]
    for i in a:
        draw_baboon(i)

#call the main function
task3_baboon()
```

task3_GMM.py and task3_GMMb.py

def task3_GMMb():

```
    from scipy.stats import multivariate_normal
    import matplotlib.pyplot as plt
    import numpy as np
    import cv2
    import numpy as np
    import math
    from matplotlib import pyplot as plt
    import time
    import csv
    from matplotlib.patches import Ellipse
    UBID = '50289736'
    np.random.seed(sum([ord(c) for c in UBID]))
    X = []

    t = []
    with open('old_faithful.csv','r') as f:
        reader = csv.reader(f)
        for row in reader:
            dataRow = []
            for column in row:
                dataRow.append(float(column))
            t.append(dataRow)
    X = t
    #print(X)
    MU = [[4.0,81],[2.0,57],[4.0,71]]
    COV =
np.array([[[1.3,13.92],[13.98,184.82]],[[1.3,13.92],[13.98,184.82]],[[1.3,13.92],[13.98,18
4.82]]])
    clusters = 3
```

```
def plot_point_cov(points, nstd=2, ax=None, **kwargs):
    pos = points.mean(axis=0)
    cov = np.cov(points, rowvar=False)
    return plot_cov_ellipse(cov, pos, nstd, ax, **kwargs)

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 iterations(mu,cov,i):
    cva,kc = classify_and_plot(mu,cov)
    plt.savefig('task3_gmm_iter'+str(i+1)+'_a.jpg')
    plt.clf()
    mu,cov = recompute(cva,mu,kc,cov)
    #plot_mu(mu[0],mu[1],mu[2])
    #plt.savefig('task3_gmm_iter'+str(i+1)+'_b.jpg')
    #plt.clf()
    print('The recomputed MU are:')
    for i in mu:
        print(i)
        print('\n')
    print('The recomputed Covariances are:')
    for i in cov:
        print(i)
        print('\n')
    return mu,cov

def recompute(class_vector,mu,kc,cov):

        nmu = []
        for i in range(len(mu)):
            clus = np.asarray(kc[i])
            meaney = np.mean(clus, axis = 0)
            nmu.append(meaney.tolist())
        nmu = np.asarray(nmu)
        for i in range(len(mu)):
            cov[i] = np.cov(kc[i],rowvar = False)
        #print(cov)
        return nmu,cov

def plot_mu(mu0,mu1,mu2):
    plt.text(mu0[0]+0.03,mu0[1], s = '('+str(mu0[0])+','+str(mu0[1])+')',fontsize = 8)
    plt.text(mu1[0]+0.03,mu1[1], s = '('+str(mu1[0])+','+str(mu1[1])+')',fontsize = 8)
    plt.text(mu2[0]+0.03,mu2[1], s = '('+str(mu2[0])+','+str(mu2[1])+')',fontsize = 8)

    plt.scatter(mu0[0],mu0[1],s = 50,c = 'r',marker = 'o', linewidths = 1,edgecolor =
'r' )
    plt.scatter(mu1[0],mu1[1],s = 50,c = 'g',marker = 'o', linewidths = 1,edgecolor =
'g' )
    plt.scatter(mu2[0],mu2[1],s = 50,c = 'b',marker = 'o', linewidths = 1,edgecolor =
'b' )
```

```
    def classify_and_plot(mu,cov):
        class_vec = []

        pdfs = []
        for i in range(len(mu)):
            pdfs.append([])
        for i in range(len(mu)):
            pdfs[i].append(multivariate_normal.pdf(X, mu[i], cov[i]))
        kc = []
        for i in range(clusters):
            kc.append([])

        for i in range(len(X)):
            mac = max(pdfs[0][0][i],pdfs[1][0][i],pdfs[2][0][i])
            if( mac == pdfs[0][0][i]):
                class_vec.append(mu[0])
                kc[0].append(X[i])
            elif(mac == pdfs[1][0][i]):
                class_vec.append(mu[1])
                kc[1].append(X[i])
            elif(mac == pdfs[2][0][i]):
                class_vec.append(mu[2])
                kc[2].append(X[i])

        kc1 = np.asarray(kc[0])
        kc2 = np.asarray(kc[1])
        kc3 = np.asarray(kc[2])

        plt.scatter(kc1[:,0].ravel(),kc1[:,1].ravel(),s = 30,c = 'w',marker = '^',
linewidths = 1,edgecolor = 'r' )
        plt.scatter(kc2[:,0].ravel(),kc2[:,1].ravel(),s = 30,c = 'w',marker = '^',
linewidths = 1,edgecolor = 'g' )
        plt.scatter(kc3[:,0].ravel(),kc3[:,1].ravel(),s = 30,c = 'w',marker = '^',
linewidths = 1,edgecolor = 'b' )

        #cov[0] = np.cov(kc1,rowvar = False)
        pos1 = kc1.mean(axis = 0)

        #cov[1] = np.cov(kc2,rowvar = False)
        pos2 = kc2.mean(axis = 0)

        #cov[2] = np.cov(kc3,rowvar = False)
        pos3 = kc3.mean(axis = 0)


        plot_cov_ellipse(cov[0],pos1,nstd = 3, alpha = 0.5, color = 'red')
        plot_cov_ellipse(cov[1],pos2,nstd = 3, alpha = 0.5, color = 'green')
        plot_cov_ellipse(cov[2],pos3,nstd = 3, alpha = 0.5, color = 'blue')

        return class_vec,kc

    for i in range(5):
        MU,COV = iterations(MU,COV,i)
#call the function
task3_GMMb()
```

Input:

$$X = \begin{bmatrix} 5.9 & 3.2 \\ 4.6 & 2.9 \\ 6.2 & 2.8 \\ 4.7 & 3.2 \\ 5.5 & 4.2 \\ 5.0 & 3.0 \\ 4.9 & 3.1 \\ 6.7 & 3.1 \\ 5.1 & 3.8 \\ 6.0 & 3.0 \end{bmatrix} \quad \begin{array}{l} \mu_3 = (6.5, 3.0) \\ \mu_1 = (6.2, 3.2) \\ \mu_2 = (6.6, 3.7) \end{array}$$

Fig. 15 Input points and initial cluster centres

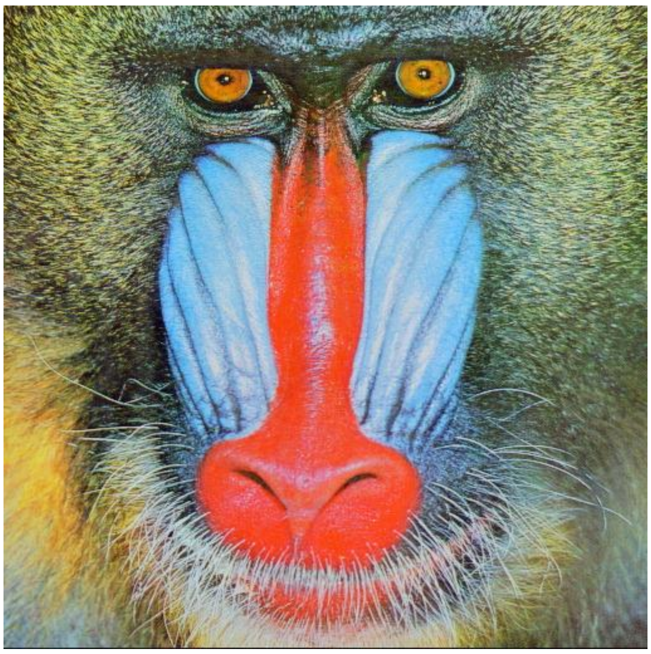
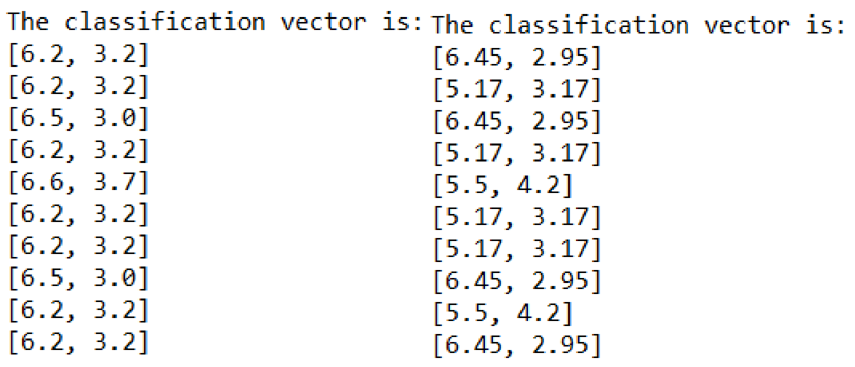


Fig. 16 Image on which k-means clustering is to be applied

Implementation and output:

1) Classify samples:
    - Classify the 10 given points into 3 clusters by measuring the Euclidean distance between each cluster centre and all the points and categorizing the points with the minimum distance to its respective cluster centre.
    - The result is a classification vector which is of the same dimensions as the input points, with its corresponding index to the input vector X containing the cluster centre to which the point belongs.

- Plot the cluster centres and their points using `pyplot.scatter` with a different color for each cluster. Use `plt.text` to print the position of each point on the plot.



```
The classification vector is:
[6.2, 3.2]
[6.2, 3.2]
[6.5, 3.0]
[6.2, 3.2]
[6.6, 3.7]
[6.2, 3.2]
[6.2, 3.2]
[6.5, 3.0]
[6.2, 3.2]
[6.2, 3.2]
```

```
The classification vector is:
[6.45, 2.95]
[5.17, 3.17]
[6.45, 2.95]
[5.17, 3.17]
[5.5, 4.2]
[5.17, 3.17]
[5.17, 3.17]
[6.45, 2.95]
[5.5, 4.2]
[6.45, 2.95]
```

Fig. 17 The classification vector as output by the program for the first(left) and second(right) iterations

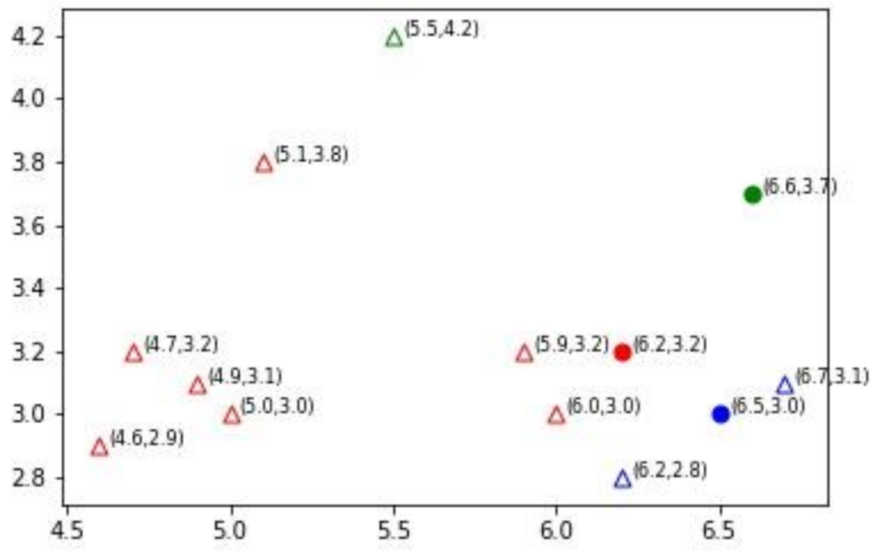


Fig. 18 Plot of classification using initial cluster centres

2) Recompute cluster centres:
   - The classvector is used and the mean of each of the [x,y] position pairs in the is calculated and this mean becomes the new cluster centre.

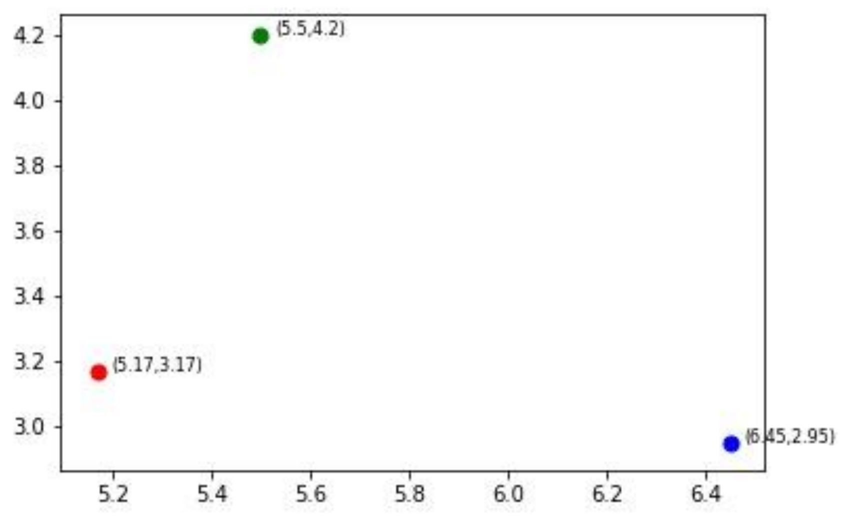


Fig. 19 Plot of the recomputed cluster centres

3) Iterate:
   - Perform classification again using the new cluster centres. Plot the cluster centres as well as the points in each cluster with a different color.
   - Recompute the clusters again and plot the centers using different colors.

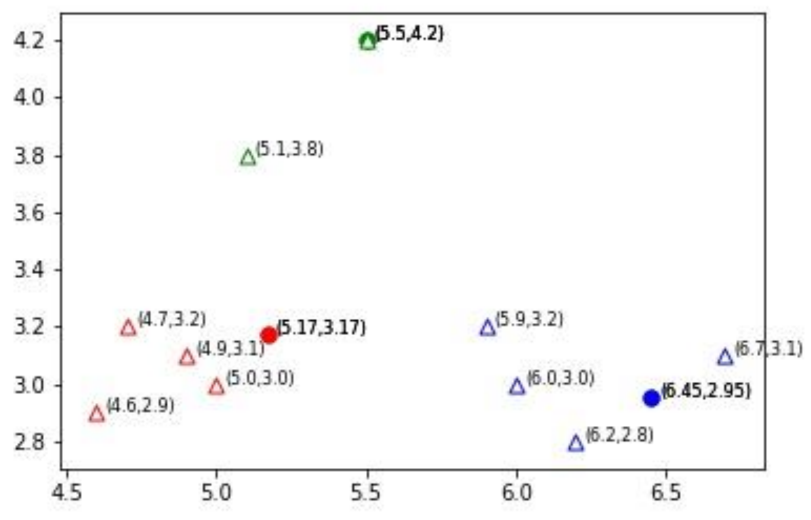


Fig. 20 Plot of classification after first recomputation

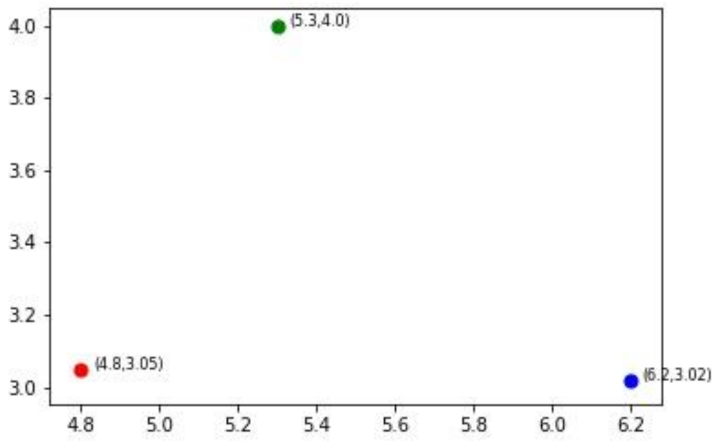


Fig. 21 Recomputer cluster centres after second iteration

4) Color Quantization:
   - Given an image, cluster all the colors in the image using k-means clustering.
   - The image is read first and copied to a temporary variable which is then randomly shuffled.
   - First k points are selected from this randomized array after reshaping it using `reshape` so that it is iterated over as a vector.
   - These k points are the initial cluster centers.
   - k-means clustering is applied to the input vector which is nothing but the reshaped image itself.
   - A classification vector is created in a similar way as **Step 3** and is used to find the mean of the clusters and update the cluster centres.
   - The classification and recomputation of cluster centres is repeated multiple times. (10 in the program)
   - The classification vector is the same dimensions as the image vector and it contains the k clusters. So, the classification is reshaped to the size of the original image.

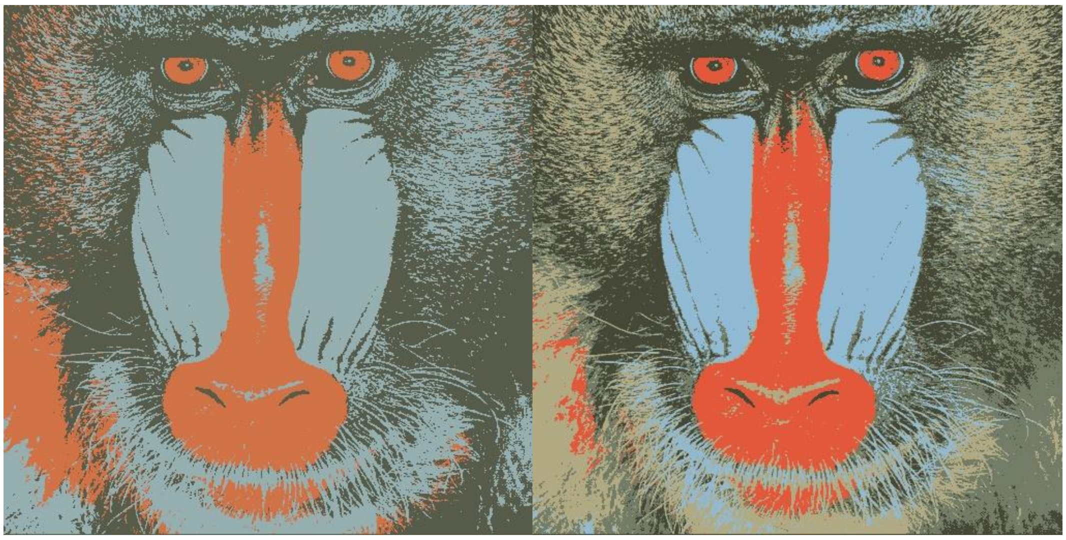


Fig. 22 Output images for k=3(left) and k=5(right)

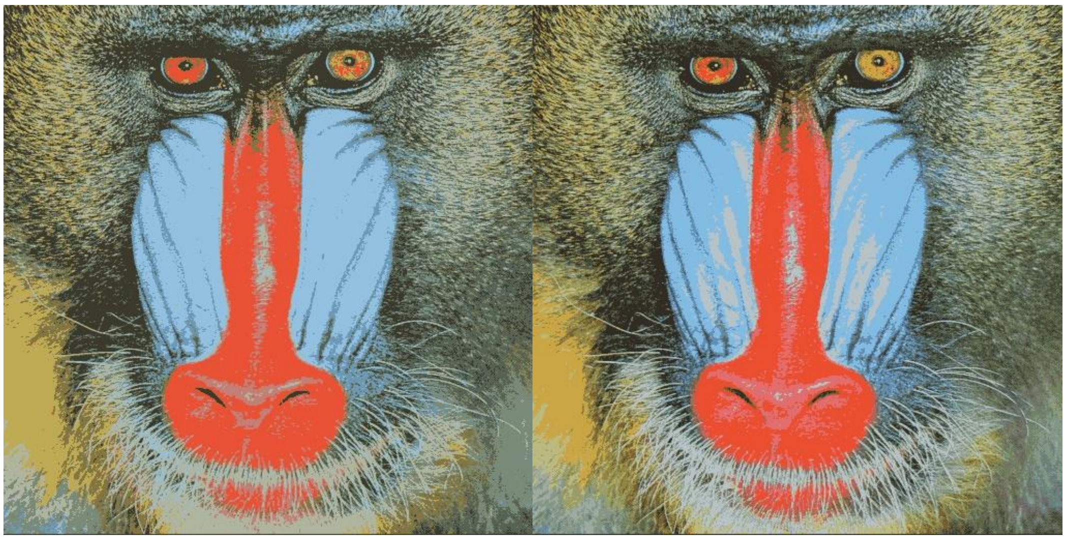
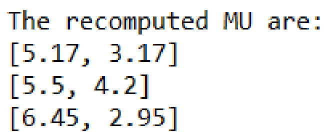


Fig. 23 Output images for k=10(left) and k=20(right)

5) Gaussian Mixture Model(GMM):
   - Perform the same operations as in **Step 1** ,except instead of using minimizing of Euclidean distance, maximize Gaussian likelihood, i.e, the probabilities found using `multivariate_normal.pdf(X).`
   - Plot the first iteration along with the mu values.
   - Apply GMM to the `old-faithful` dataset using the `csv` library.
   - Plot the result for 5 iterations using `plot_cov_ellipse.`



```
The recomputed MU are:
[5.17, 3.17]
[5.5, 4.2]
[6.45, 2.95]
```

Fig. 24 Recomputed MU for GMM Clustering

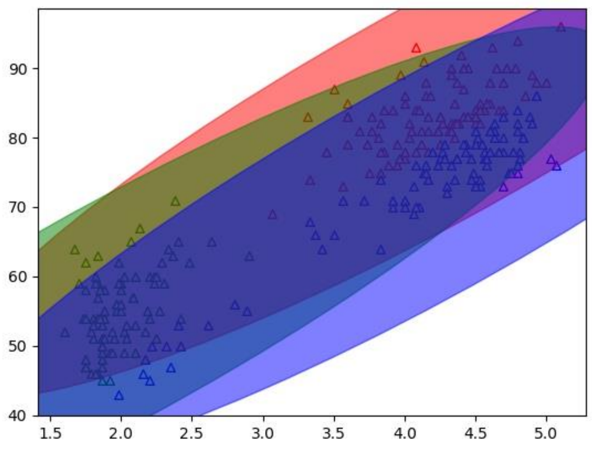


Fig. 25 Iteration 1 Output

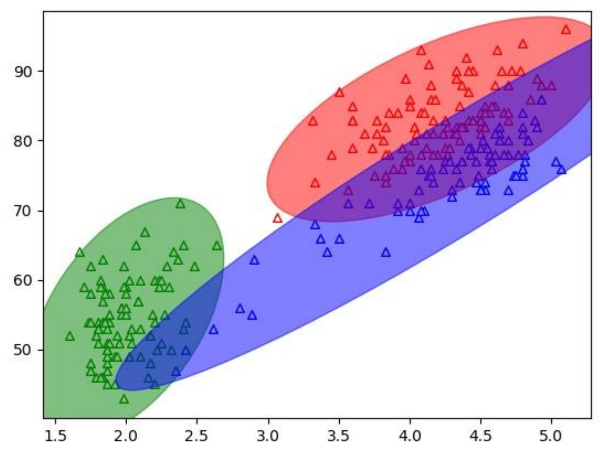


Fig. 26 Iteration 2 Output

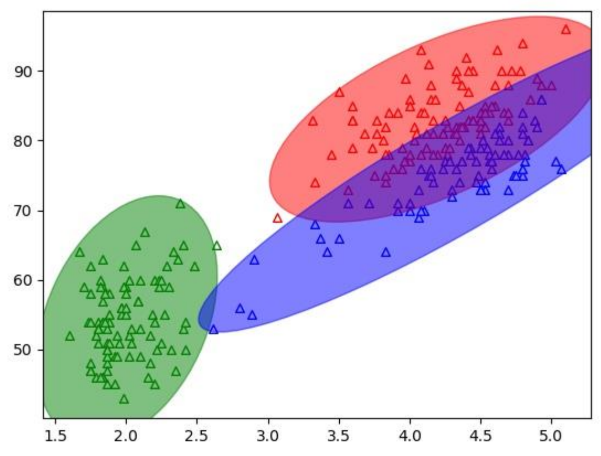


Fig. 27 Iteration 3 Output

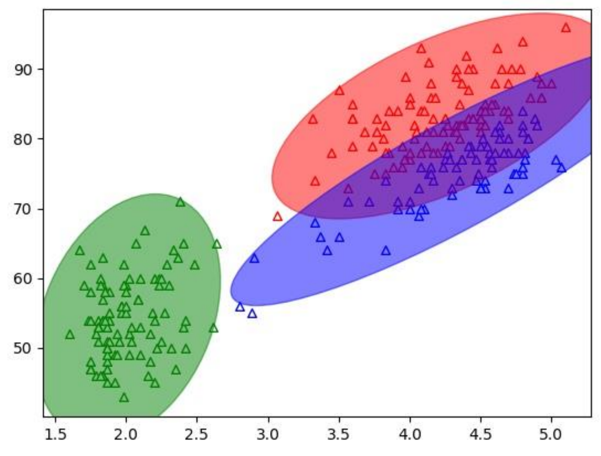


Fig. 28 Iteration 4 Output

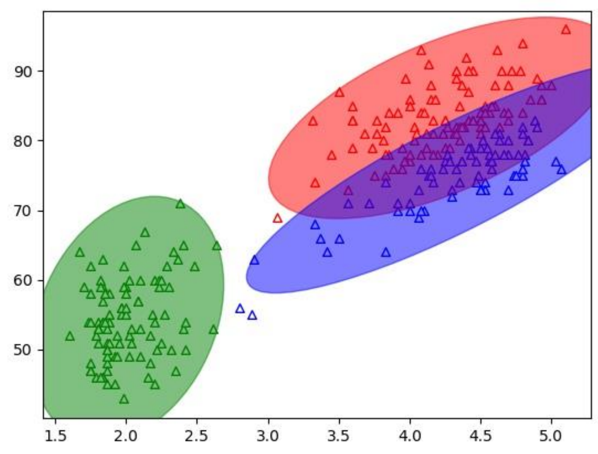


Fig. 29 Iteration 5 Output

Conclusion:

Task 1 – We learned how to perform k-nearest neighbour to understand how to warp stereo images using the Homography matrix

Task2 – We learned how to find and plot epilines of 2 stereo images.

Task3 – We learned how to perform k-means clustering on colors of a given image, i.e, color quantization. We also learned clustering using the Gaussian Mixture Model.

References

- http://stamfordresearch.com/basic-sift-in-python/
- https://www.programcreek.com/python/example/89309/cv2.drawKeypoints
- https://stackoverflow.com/questions/46607647/sift-feature-matching-point-coordinates
- https://stackoverflow.com/questions/48063525/error-with-matches1to2-with-opencv-sift
- https://www.youtube.com/watch?v=MlaIWymLCD8
- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html
- https://stackoverflow.com/questions/33695580/selecting-random-elements-in-a-list-conditional-on-attribute
- https://www.learnopencv.com/homography-examples-using-opencv-python-c/
- https://stackoverflow.com/questions/13063201/how-to-show-the-whole-image-when-using-opencv-warpperspective/20355545#20355545
- https://docs.opencv.org/3.2.0/da/de9/tutorial_py_epipolar_geometry.html
- https://programtalk.com/python-examples/cv2.computeCorrespondEpilines/
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html
- https://rdmilligan.wordpress.com/2016/05/23/disparity-of-stereo-images-with-python-and-opencv/
- https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
- https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
- https://www.pyimagesearch.com/2014/07/07/color-quantization-opencv-using-k-means-clustering/
- https://github.com/joferkington/oost paper code/blob/master/error_ellipse.py
- https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat
- https://www.youtube.com/watch?v=0NMC2NfJGqo - Jordan Boyd Graber, GMM