Submitted By: Anupriya Goyal

UB Person No.: 50287108

Date: November 5th ,2018

Computer Vision and Image Processing (CSE473/573)
Project2 Report

Task1 : Image Features and Homography

1. **Code snippet for extracting SIFT features and Keypoints drawn on both images is as follows:**

```
#Task1 part1 to extract SIFT and detect keypoints in both the given images of mountains
import cv2
import numpy as np
#read the two images of mountains and convert it to grayscale images
#specify the particular path where image is stored, to read it
image1 = cv2.imread("path/mountain1.jpg",0)
image2 = cv2.imread("path/mountain2.jpg",0)
sift = cv2.xfeatures2d.SIFT_create()
#detecting keypoints
keypoint1 = sift.detect(image1,None)
keypoint2 = sift.detect(image2,None)
#drawing keypoints on both images
img_key1=cv2.drawKeypoints(image1,keypoint1,None)
img_key2=cv2.drawKeypoints(image2,keypoint2,None)

#saving images at particular location

cv2.imwrite('path/task1_sift1.jpg',img_key1)
cv2.imwrite('path/task1_sift2.jpg',img_key2)
```

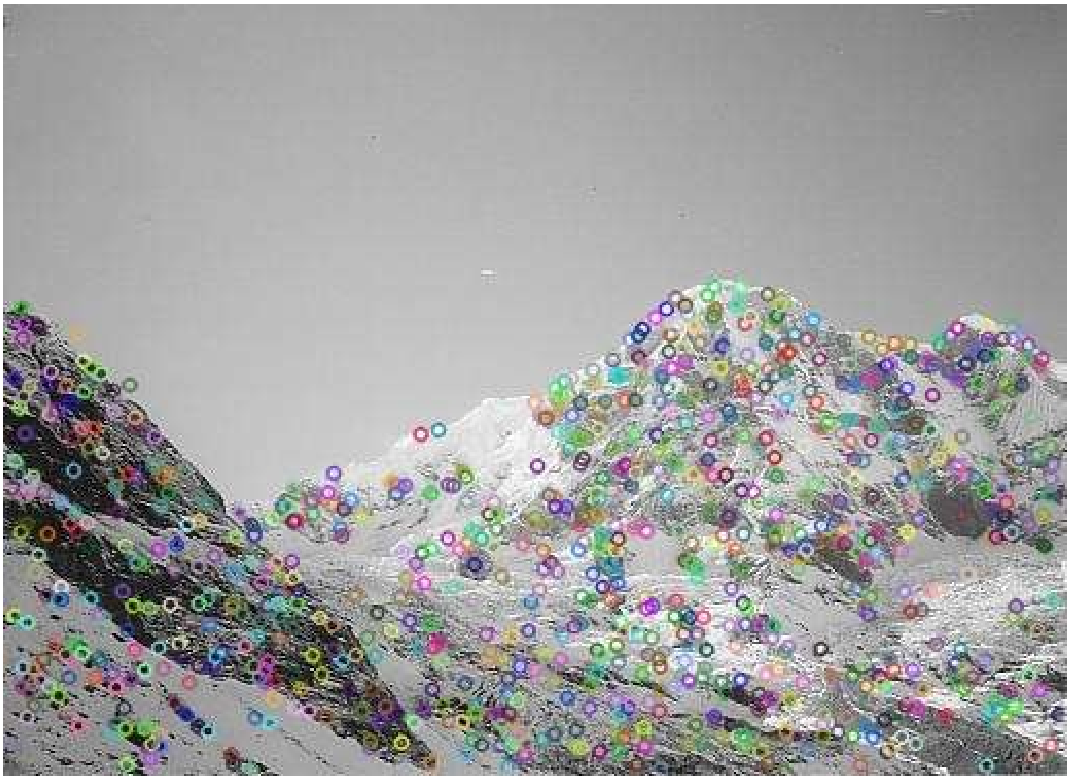


Fig1. Task1 part1 Keypoints detected for image1 (task1_sift1.jpg)

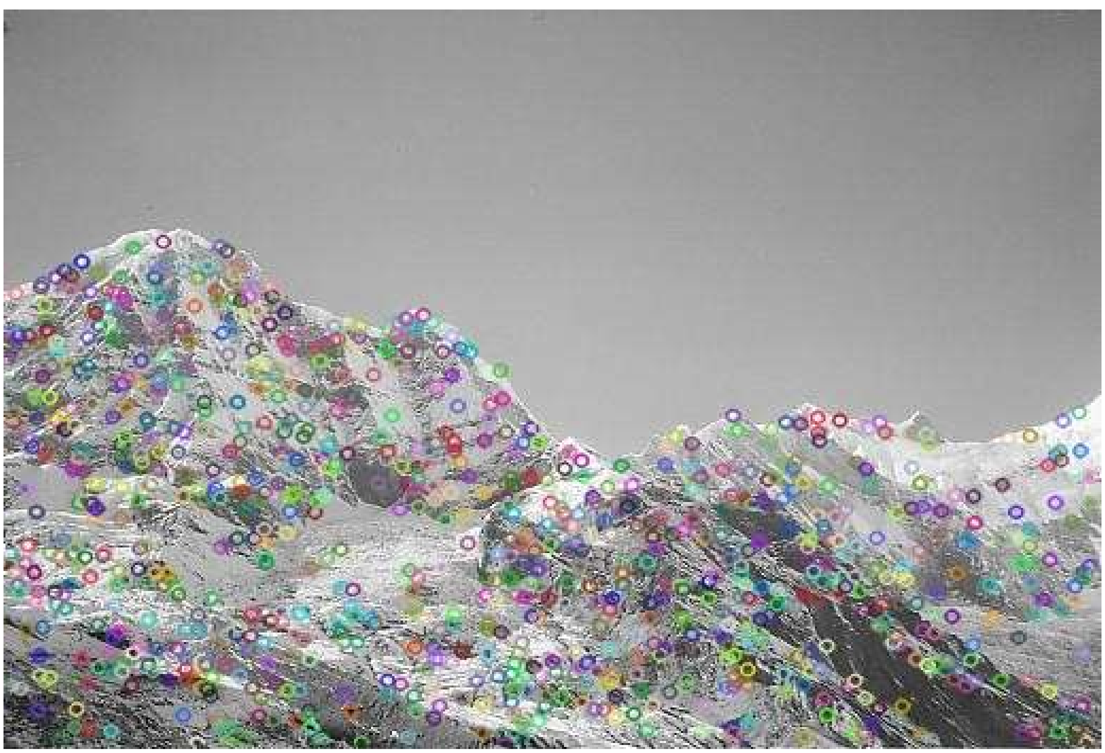


Fig2. Task1 part1 Keypoints detected for image2 (task1_sift2.jpg)

References for Task1 part1:

1. https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html

2. **Code snippet for Task1 part2 : matching keypoints using k=2 and filtering good matches m.distance<0.75 n.distance (m is first match and n is second match)**

```
sift = cv2.xfeatures2d.SIFT_create()

#detecting keypoints

keypoint1, d1 = sift.detectAndCompute(image1,None)
keypoint2, d2 = sift.detectAndCompute(image2,None)

#drawing keypoints on both images

img_key1=cv2.drawKeypoints(image1,keypoint1,None)
img_key2=cv2.drawKeypoints(image2,keypoint2,None)

# BFMatcher with default params

burteforce_matcher = cv2.BFMatcher()
total_matches = burteforce_matcher.knnMatch(d1,d2, k=2)

# Apply ratio test
good_points = []

for m,n in total_matches:
   if m.distance < 0.75*n.distance:
      good_points.append([m])

# cv2.drawMatchesKnn expects list of lists as matches.

image_knn =
cv2.drawMatchesKnn(image1,keypoint1,image2,keypoint2,good_points,None,flags=2)

#saving images at particular location

cv2.imwrite("C:/Users/Anupriya/Documents/Project2_final_images/task1_matches_knn.j
pg",image_knn)
```

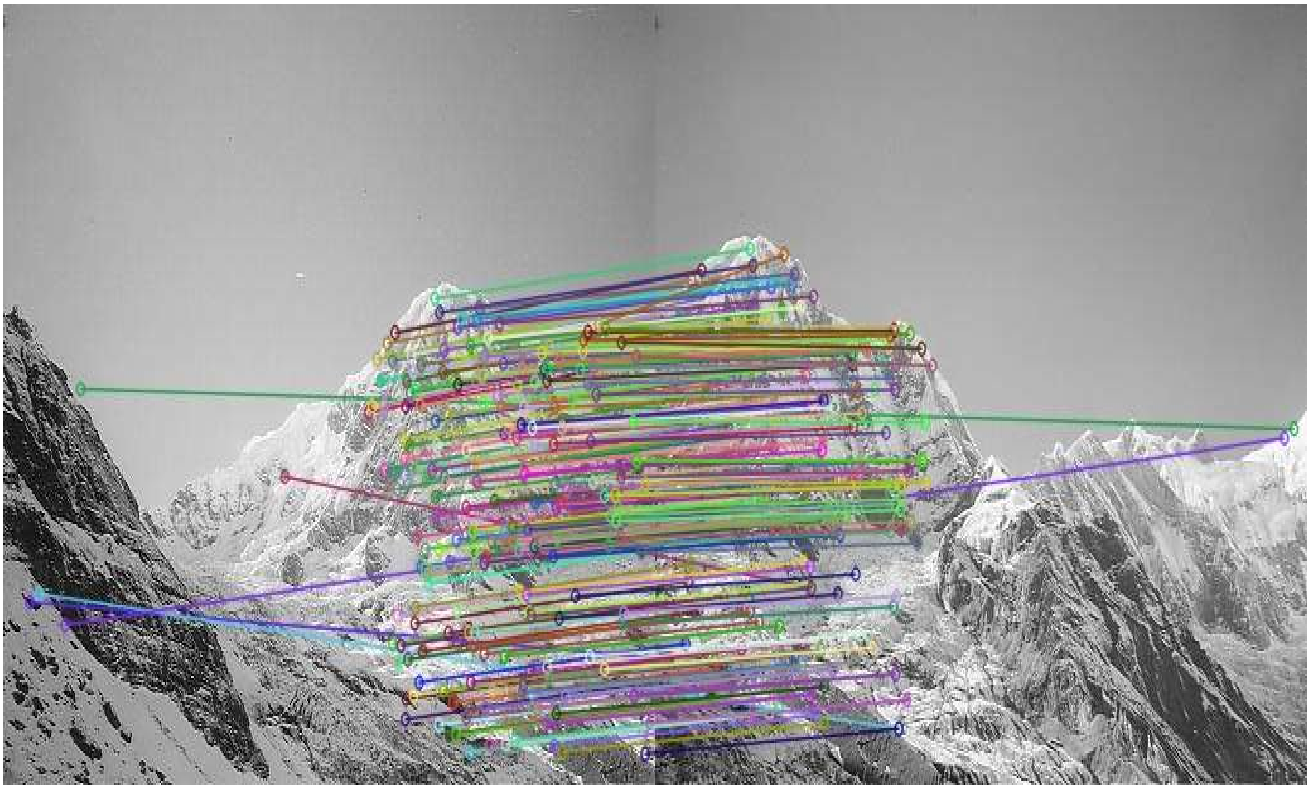


Fig3. Task1 part2 matching keypoints using k=2 (task1_matches_knn.jpg)

References for Task1 part2

1. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html

3. **Code snippet for task1 part3 : computing homography matrix H (with RANSC)**

```
#task1 part3 computing homography matrix H(with RANSC) using FLANN method

FLANN_INDEX_KDTREE = 0
inx_parameter = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
srch_parameter = dict(checks = 50)

dect_flann = cv2.FlannBasedMatcher(inx_parameter, srch_parameter )

matches_by_flann = dect_flann.knnMatch(des1,des2,k=2)

# store all the good matches as per Lowe's ratio test.
matched_good_points = []
```

```
for m,n in matches_by_flann:
   if m.distance < 0.75*n.distance:
      matched_good_points.append(m)

RANDOM_MATCHES_COUNT= 10
if len(matched_good_points)>RANDOM_MATCHES_COUNT:
   source_points = np.float32([ keypoint1[m.queryIdx].pt for m in matched_good_points
]).reshape(-1,1,2)
   destination_points = np.float32([ keypoint2[m.trainIdx].pt for m in
matched_good_points ]).reshape(-1,1,2)

  #finding homography matrix H

   H, mask = cv2.findHomography(source_points ,destination_points, cv2.RANSAC,5.0)
   #printing homography matrix H
   print(H)
```

HOMOGRAPHY MATRIX H :

```
[[ 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]]
```

References for task1 part3:

1. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html

4. **Code snippet for task1 part4 : match image for 10 random matches (using only inliers) :**
   The part of code here is for match image for 10 random matches (using only inliers) which is extended after code in part3

```
RANDOM_MATCHES_COUNT= 10
if len(matched_good_points)>RANDOM_MATCHES_COUNT:
   source_points = np.float32([ keypoint1[m.queryIdx].pt for m in matched_good_points
]).reshape(-1,1,2)
   destination_points = np.float32([ keypoint2[m.trainIdx].pt for m in matched_good_points
]).reshape(-1,1,2)

   #finding homography matrix H
   H, mask = cv2.findHomography(source_points ,destination_points, cv2.RANSAC,5.0)
   #printing homography matrix H
   print(H)
```

```
    matchesMask = mask.ravel().tolist()
    h,w = image1.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    dst = cv2.perspectiveTransform(pts,H)
   # h = cv2.getPerspectiveTransform(src_pts,dst_pts)
#drawing 10 random matches using only inliers
draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   matchesMask = matchesMask, # draw only inliers
                   flags = 2)
image3 =
cv2.drawMatches(image1,keypoint1,image2,keypoint2,matched_good_points,None,**draw_par
ams)
cv2.imwrite("path/task1_matches.jpg",image3)
```

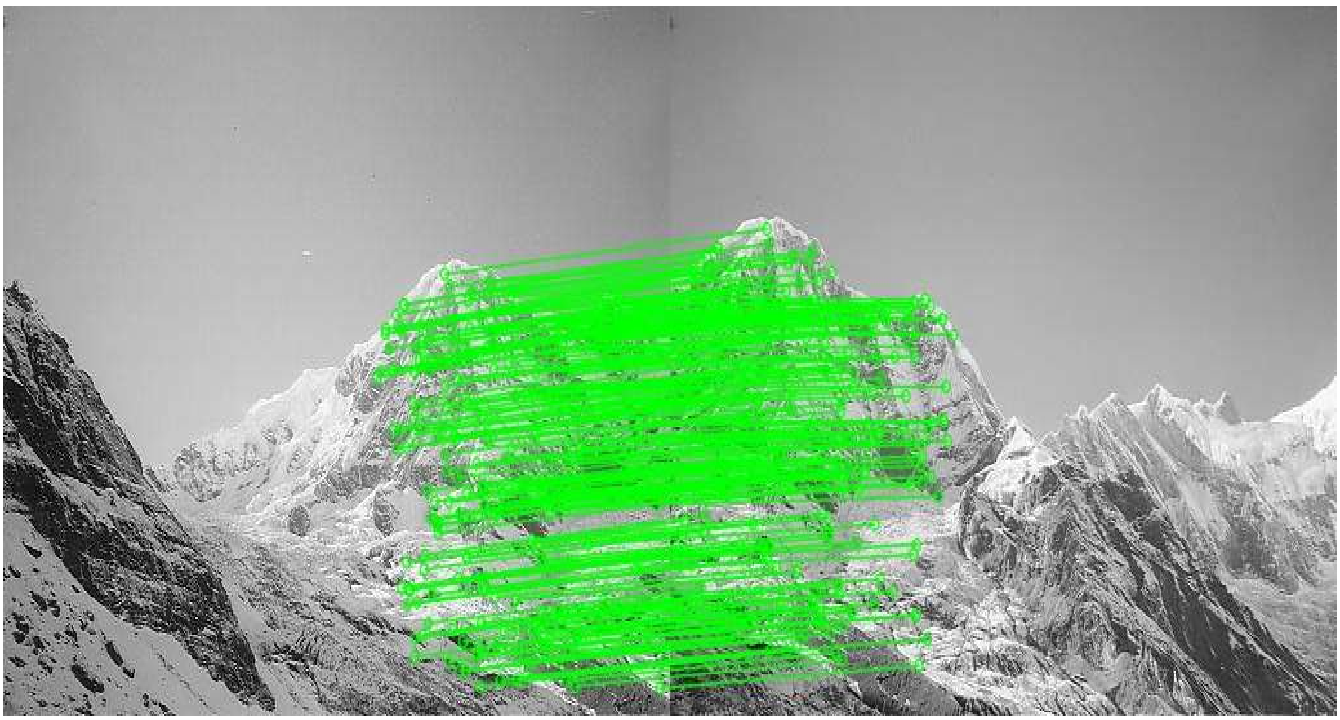


Fig4: Task1 part4 drawn match image for 10 random matches using only inliers (task1_matches.jpg)

References for task1 part4:

1. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html

5. Code snippet for warp image: first image to second image using H is as follows:

The part of code here is for warping the first image to the second image using H which is extended after code in part4

```
 r1, c1 = image1.shape[:2]
r2, c2 = image4.shape[:2]
points_from_1 = np.float32([[0,0], [0,r1], [c1, r1], [c1,0]]).reshape(-1,1,2)
pts = np.float32([[0,0], [0,r2], [c2, r2], [c2,0]]).reshape(-1,1,2)
points_from_2 = cv2.perspectiveTransform(pts, H)
pts_list = np.concatenate((points_from_1, points_from_2), axis=0)
[minimum_x, minimum_y] = np.int32(pts_list.min(axis=0).ravel() - 0.5)
[maximum_x, maximum_y] = np.int32(pts_list.max(axis=0).ravel() + 0.5)

d_trans = [-minimum_x, -minimum_y]
trans_Homo = np.array([[1, 0, d_trans[0]], [0, 1, d_trans[1]], [0,0,1]])
result_image = cv2.warpPerspective(image1, trans_Homo.dot(H), (maximum_x - 
minimum_x, maximum_y - minimum_y))
result_image[d_trans[1]:r1+d_trans[1],d_trans[0]:c1+d_trans[0]] = image4
cv2.imwrite("path/task1_pano.jpg",result_image)
```

Fig5: Task1 part5 warping the first image to second image using H (task1_pano.jpg)

References for task1 part5:

1. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html
2. https://www.kaggle.com/asymptote/homography-estimate-stitching-two-imag/code

Task2 : Epipolar Geometry

1. **Code snippet for extracting SIFT features and draw keypoints for both images and matching keypoints for k=2:**

```
sift =cv2.xfeatures2d.SIFT_create()

keypoint1, d1 = sift.detectAndCompute(image1,None)
keypoint2, d2 = sift.detectAndCompute(image2,None)

image1_keypoint = cv2.drawKeypoints(image1, keypoint1, None)
image2_keypoint = cv2.drawKeypoints(image2, keypoint2, None)
# BFMatcher with default params
burteforce_matcher= cv2.BFMatcher()
total_matches  = burteforce_matcher.knnMatch(d1,d2, k=2)

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

# cv2.drawMatchesKnn expects list of lists as matches.
image_knn = cv2.drawMatchesKnn(image1,
keypoint1,image2,keypoint2,good_points,None,flags=2)
cv2.imwrite("path/task2_sift1.png", image1_keypoint)
cv2.imwrite("path/task2_sift2.png", image2_keypoint)
cv2.imwrite("path/task2_matches_knn.jpg",image_knn)
```

Fig1. Task2 part1 Keypoints detected for image1 (task1_sift1.jpg)

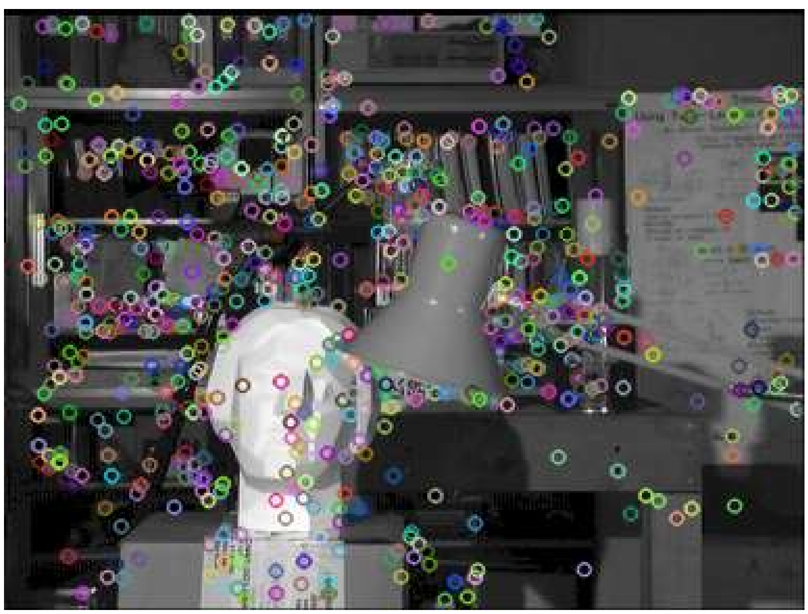


Fig2. Task2 part1 Keypoints detected for image1 (task2_sift2.jpg)

Fig3. Task2 part1 matching keypoints using k=2 (task2_matches_knn.jpg)

Part2 : Code for computing fundamental matrix F

```
flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(d1,d2,k=2)

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.75*n.distance:
        good.append(m)
        pts2.append(keypoint2[m.trainIdx].pt)
        pts1.append(keypoint1[m.queryIdx].pt)
pts1 = random.sample(pts1, 10)
pts2 = random.sample(pts2, 10)
pts1 = np.int32(pts1)
pts2 = np.int32(pts2)
#print(pts1)
#print(pts2)
F, mask = cv2.findFundamentalMat(pts1,pts2,cv2.FM_LMEDS)
print(F)
def drawlines(img1,img2,lines,pts1,pts2):
    ''' 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)
```

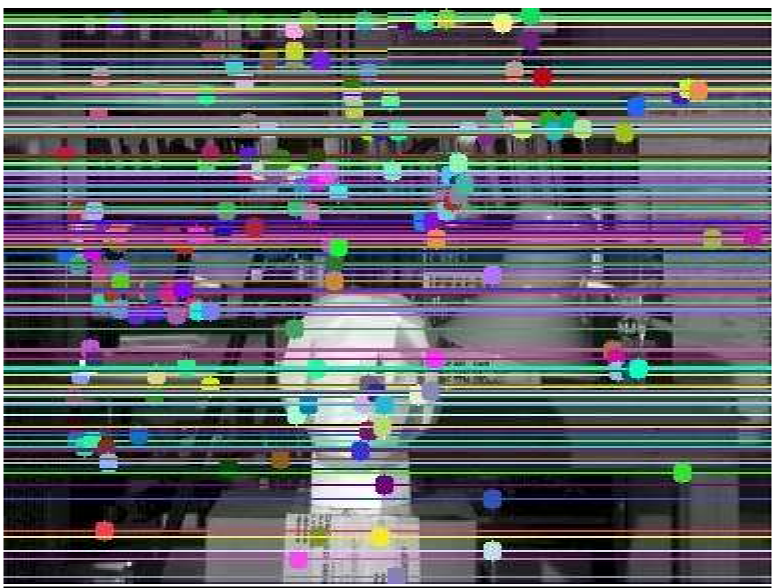
```
    img2 = cv2.cvtColor(img2,cv2.COLOR_GRAY2BGR)

    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), color,1)
        img1 = cv2.circle(img1,tuple(pt1),5,color,-1)
        img2 = cv2.circle(img2,tuple(pt2),5,color,-1)
    return img1,img2
```

Value of fundamental matrix vary according to random points choosen

Fundamental matrix is:

```
[[ 1.69144624e-04  1.28930872e-03 -1.40339093e-01]
 [ 8.18913461e-05 -1.52314967e-03  7.16449108e-02]
 [-5.08342076e-03  1.07507678e-02  1.00000000e+00]]
```



<u>Fig4: epilines on left image when all points are considered</u>

<u>Fig5: epilines on left image when all points are considered</u>

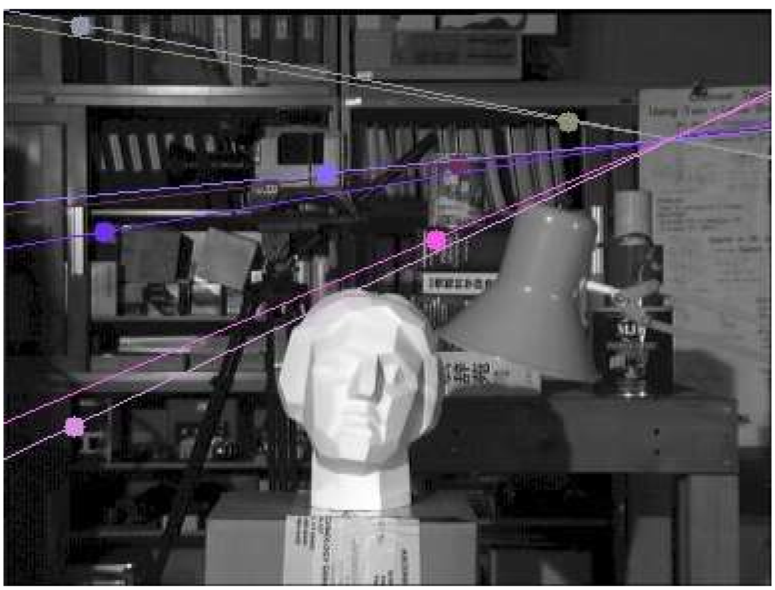


Fig6: task2_epi_left.jpg

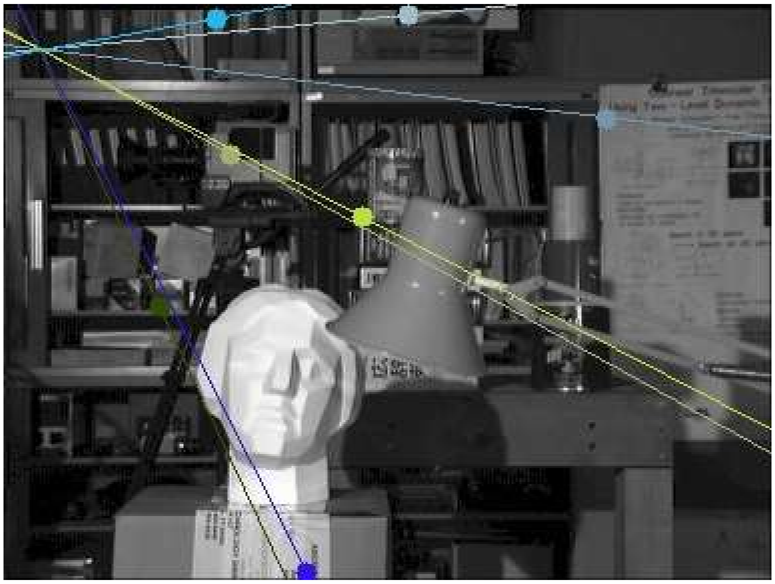


Fig7: : task2_epi_right.jpg

References for task2 part3 :

1. https://docs.opencv.org/3.4.3/da/de9/tutorial_py_epipolar_geometry.html

5. **Disparity task2 part5 : code snippet for disparity image and disparity map computation:**

```
min_disp=16
num_disp=16*7
stereo=cv2.StereoBM_create(numDisparities=16*7, blockSize=21)

disparity = stereo.compute(img_left, img_right).astype(np.float32) / 16.0
disp_map = (disparity - min_disp)/num_disp

cv2.imwrite("path/task2_disparity2.jpg",disparity)
```

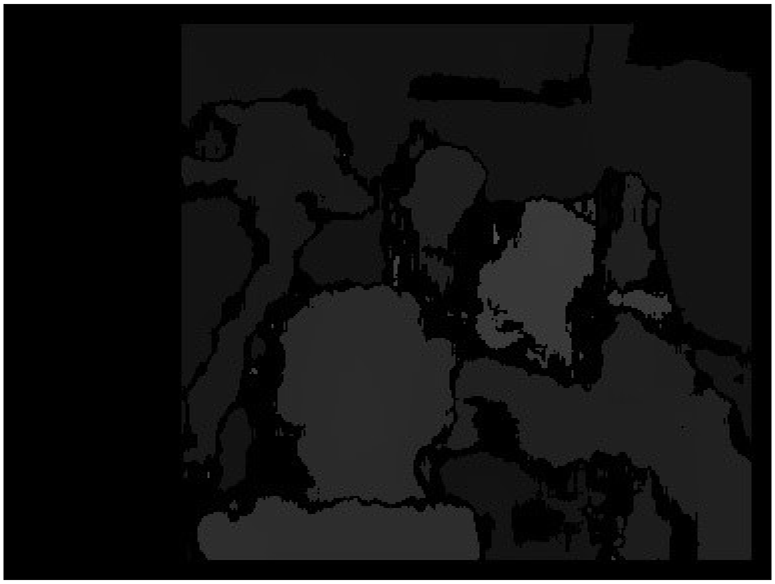


Fig6. Task2 part4 disparity image

References for Task2 part4 :

1. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html

Task3: K means Clustering:

Part1: Classification vector , classification plot and Code snippet to classify N=10 samples to nearest three cluster centers [u_1 =(6.2,3.2)(red), u_2=(6.6,3.7) (green) , u_3=(6.5,3.0)(blue)] :

Code Snippet: (not complete code)

```
data = 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]])
centroids = np.array([[6.2,3.2],[6.6,3.7],[6.5,3.0]])

while count<=9:


   for i in range(3):

      distance.append(math.sqrt(((data[m][n]-centroids[i][j])**2)+((data[m][n+1]-centroids[i][j+1])**2)))

   a=min(distance[k],distance[k+1],distance[k+2])

   for p in range(k,k+3):
```

```
        if a==distance[p]:
            index= p


    if index == k:

        cluster1.append(data[m])
        classification_vector1.append(data[m])
        classification_vector1.append("red (cluster1)")

    if index==k+1:

        cluster2.append(data[m])
        classification_vector1.append(data[m])
        classification_vector1.append("green (cluster2)")
    if index==k+2:

        cluster3.append(data[m])
        classification_vector1.append(data[m])
        classification_vector1.append("blue (cluster3)")



    m=m+1
    count=count+1
    k=k+3
classification_vector1=np.array(classification_vector1)

plt.savefig('path/task3_iter1_a.jpg')
```

Classification Vector is : Here **'(cluster1)'** represents point belongs to red cluster 1 **'(cluster 2)'** represents point belong to green cluster 2 and **'(cluster 3)'** represents point belong to blue cluster 3

classification vector with respect to given means initially(u1,u2,u3):

```
[array([5.9, 3.2]) 'red (cluster1)' array([4.6, 2.9]) 'red (cluster1)'
 array([6.2, 2.8]) 'blue (cluster3)' array([4.7, 3.2]) 'red (cluster1)'
 array([5.5, 4.2]) 'green (cluster2)' array([5., 3.]) 'red (cluster1)'
 array([4.9, 3.1]) 'red (cluster1)' array([6.7, 3.1]) 'blue (cluster3)'
 array([5.1, 3.8]) 'red (cluster1)' array([6., 3.]) 'red (cluster1)']
```

```
classification vector with respect to given means initially(u1,u2,u3):
[array([5.9, 3.2]) 'red (cluster1)' array([4.6, 2.9]) 'red (cluster1)'
 array([6.2, 2.8]) 'blue (cluster3)' array([4.7, 3.2]) 'red (cluster1)'
 array([5.5, 4.2]) 'green (cluster2)' array([5., 3.]) 'red (cluster1)'
 array([4.9, 3.1]) 'red (cluster1)' array([6.7, 3.1]) 'blue (cluster3)'
 array([5.1, 3.8]) 'red (cluster1)' array([6., 3.]) 'red (cluster1)']
```

Fig1. Task3 part1 Classification vector screenshot

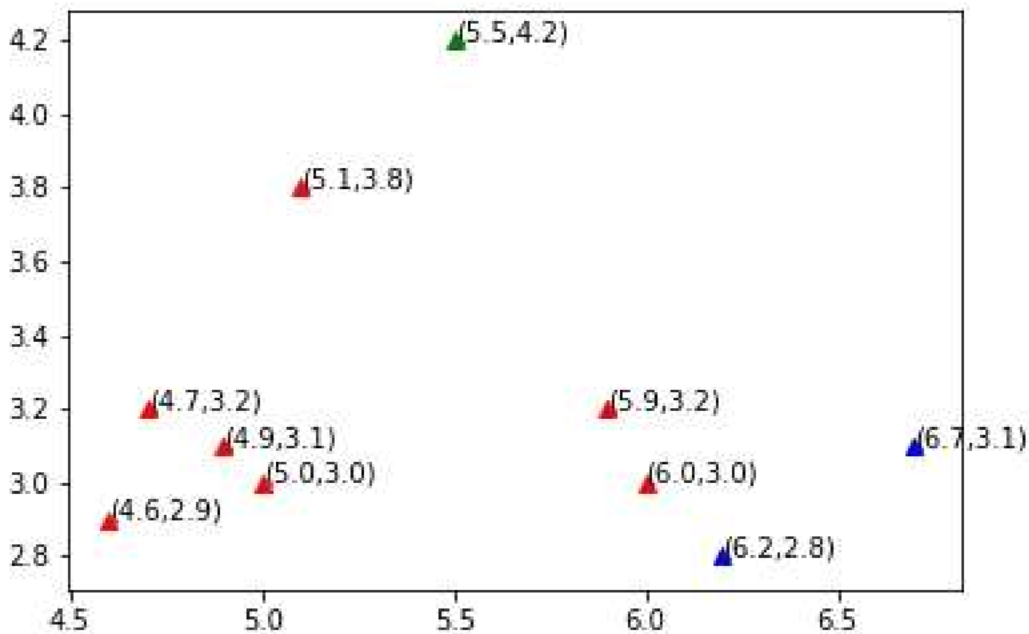


Fig2: Task3 part1 (task3_iter1_a.jpg) without the original three centers [u_1 =(6.2,3.2)(red), u_2=(6.6,3.7) (green) , u_3=(6.5,3.0)(blue)]

Part2 : Recomputed centers of 3 clusters are as follows:

Updated mean values u1,u2,u3 first iteration:

```
[[5.3        3.62857143]
 [6.05       3.95      ]
 [6.46666667 2.96666667]]]
```

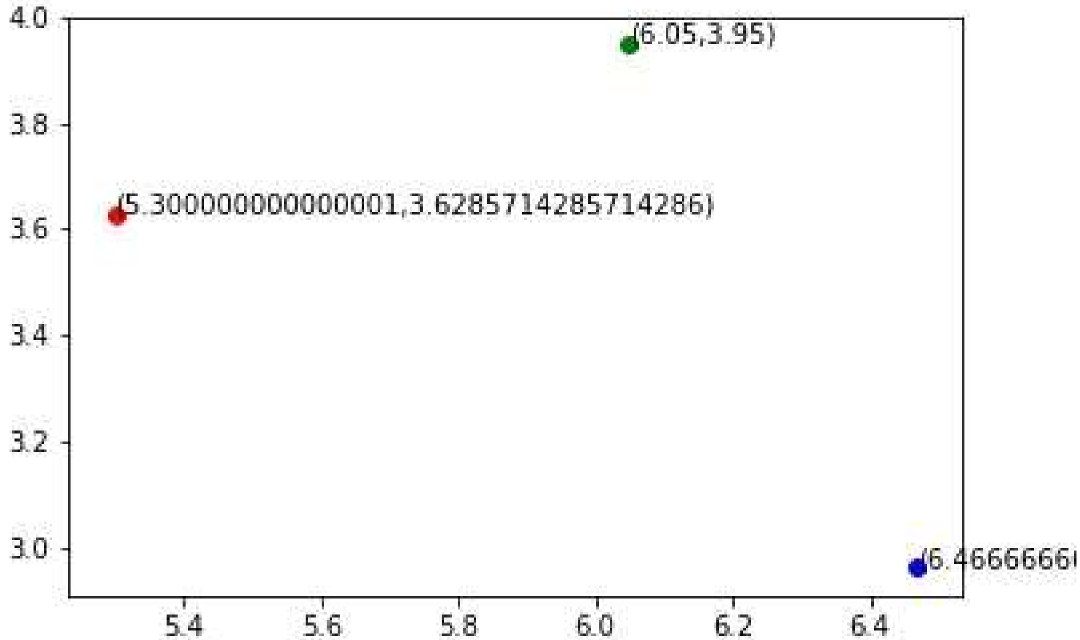


Fig5: Task3 part2 : Recomputed centers [u1,u2,u3] represented by red, green and blue colors respectively (task3_iter1_b.jpg)

Part3: Classification plot, updated mean values(u1,u2,u3) and classification vector for second iteration

updated mean values u1,u2,u3: Second Iteration
[[4.8   3.05]
 [5.3   4.   ]
 [6.2   3.025]]

updated mean values u1,u2,u3: Second Iteration
[[4.8   3.05]
 [5.3   4.   ]
 [6.2   3.025]]

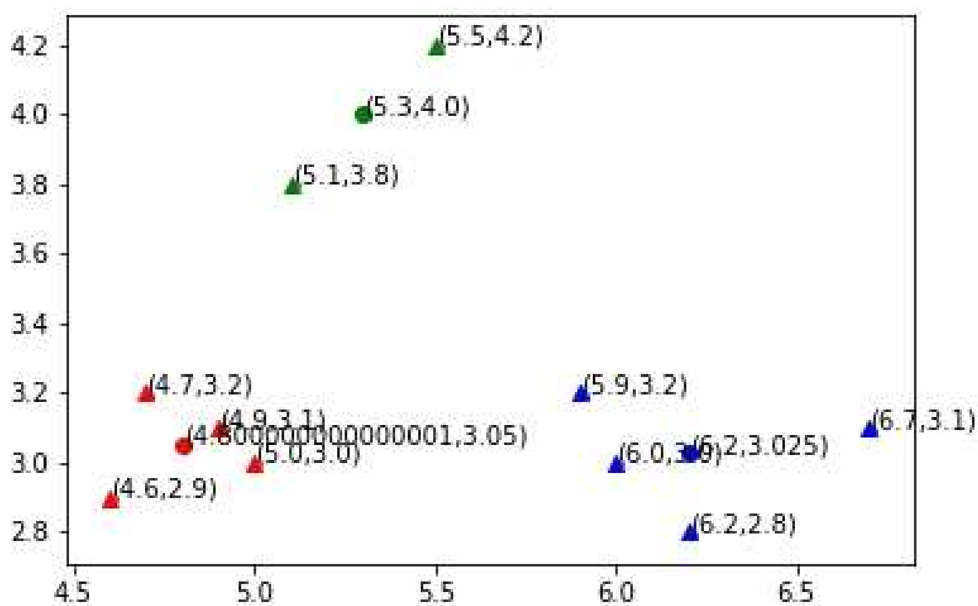
Fig5. Task3 part3 updated u1,u2,u3 (mean values) screenshot

Classification Vector for Second Iteration:

[array([5.9, 3.2]) 'blue (cluster3)' array([4.6, 2.9]) 'red (cluster1)'
 array([6.2, 2.8]) 'blue (cluster3)' array([4.7, 3.2]) 'red (cluster1)'
 array([5.5, 4.2]) 'green (cluster2)' array([5., 3.]) 'red (cluster1)'
 array([4.9, 3.1]) 'red (cluster1)' array([6.7, 3.1]) 'blue (cluster3)'
 array([5.1, 3.8]) 'green (cluster2)' array([6., 3.]) 'blue (cluster3)']

```
classification vector: Second iteration:
[array([5.9, 3.2]) 'blue (cluster3)' array([4.6, 2.9]) 'red (cluster1)'
 array([6.2, 2.8]) 'blue (cluster3)' array([4.7, 3.2]) 'red (cluster1)'
 array([5.5, 4.2]) 'green (cluster2)' array([5., 3.]) 'red (cluster1)'
 array([4.9, 3.1]) 'red (cluster1)' array([6.7, 3.1]) 'blue (cluster3)'
 array([5.1, 3.8]) 'green (cluster2)' array([6., 3.]) 'blue (cluster3)']
```

<u>Fig7. Task3 part3 Classification vector for second iteration screenshot</u>



<u>Fig8: Task3 part3 : Classification plot with recomputed centers [u1,u2,u3] represented by red, green and blue colors respectively (task3_iter2_a.jpg)</u>

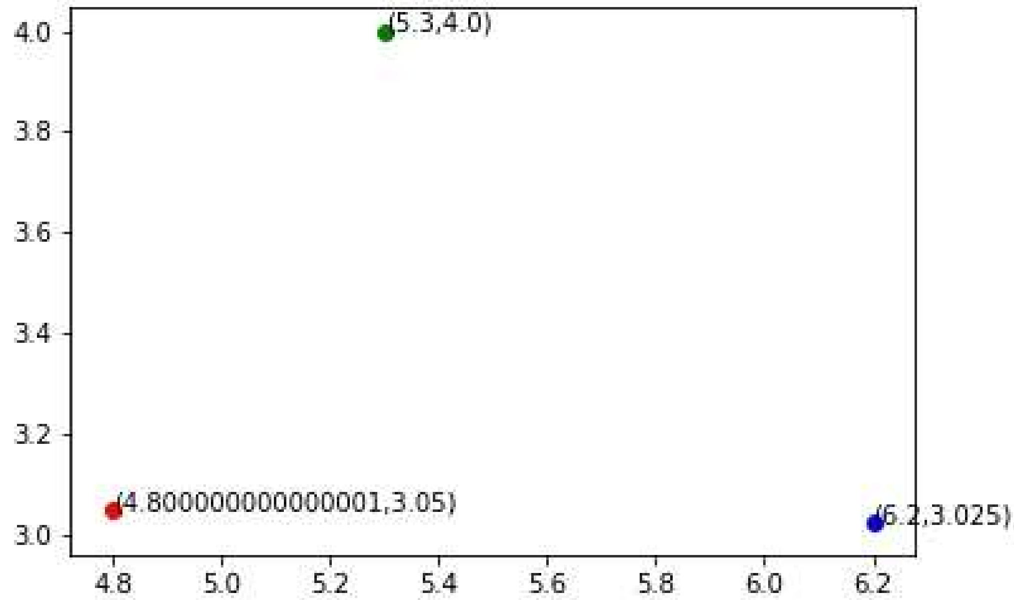


Fig5: Task3 part3 : Recomputed centers [u1,u2,u3] for second iteration represented by red, green and blue colors respectively (task3_iter2_b.jpg)

Part4: color Quantization: Images for k=3,5,10,20 and code snippet for color quantization

Steps followed for Color Quantization:

1. Considered initially the first three pixels as mean values.
2. Applied K means algorithm for each pixel in image to get pixels in same clusters (for k=3 there will be three clusters and so on for rest of k values)
3. In k means algorithm computed distance of each pixel every time with the new computed mean and took the min distance value and assigned that particular pixel to that cluster.
4. The no. of iteration are performed till convergence.

Code Snippet for Color Quantization:

```
distance.append(math.sqrt(((int(img[m][n][r])-int(centroid[i][j][h]))**2)+((int(img[m][n][r+1])-int(centroid[i][j][h+1]))**2)+((int(img[m][n][r+2])-int(centroid[i][j][h+2]))**2)))

    a=min(distance[index],distance[index+1],distance[index+2])
    for p in range(index,index+3):
        if a==distance[p]:
            ind_match=p

    if ind_match ==index:
        cluster1.append([img[m][n][r],img[m][n][r+1],img[m][n][r+2]])
```

```
            cluster1_xy_values.append([m , n])
        if ind_match==index+1:
            cluster2.append([img[m][n][r],img[m][n][r+1],img[m][n][r+2]])
            cluster2_xy_values.append([m, n])
        if ind_match==index+2:
            cluster3.append([img[m][n][r],img[m][n][r+1],img[m][n][r+2]])
            cluster3_xy_values.append([m , n])

        #print(cluster1)
        #print(cluster2)
        #print(cluster3)
        n=n+1
        #print("value of n is",n)
        test=test+1
        index=index+3
    m=m+1
    test1=test1+1

#new_centers_find
#print(len(cluster1))
new_center1 =[]
j=0
sum_x=0
for i in range(len(cluster1)):
    sum_x=cluster1[i][j]+sum_x

new_center1.append((sum_x)/(len(cluster1)))
#print(new_center1)
j=0
sum_y=0
for i in range(len(cluster1)):
    sum_y=cluster1[i][j+1]+sum_y

a=((sum_y)/(len(cluster1)))
new_center1.append(a)
#print(new_center1)

z=0
sum_z=0
for i in range(len(cluster1)):
    sum_z=cluster1[i][z+2]+sum_z

a=((sum_z)/(len(cluster1)))
new_center1.append(a)
#print(new_center1)

center_array = [[new_center1,new_center2 ,new_center3 ]]
```

```
    np.array(center_array)
    centroid = center_array
    #print(centroid)

    iterate=iterate+1

final_cluster1=[cluster1]
final_cluster1_array=np.array(final_cluster1)
#print(final_cluster1)
final_cluster1_xy_values=[cluster1_xy_values]
final_cluster1_xy_values=np.array(final_cluster1_xy_values)
#print(final_cluster1_xy_values)
#print(len(final_cluster1_xy_values[0]))
for i in range(len(final_cluster1_xy_values[0])):
    a=final_cluster1_xy_values[0][i][0]
    b=final_cluster1_xy_values[0][i][1]
    img[a][b]=center_array[0][0]
cv2.imwrite("path/to store image/task3_part4.jpg",img)
```

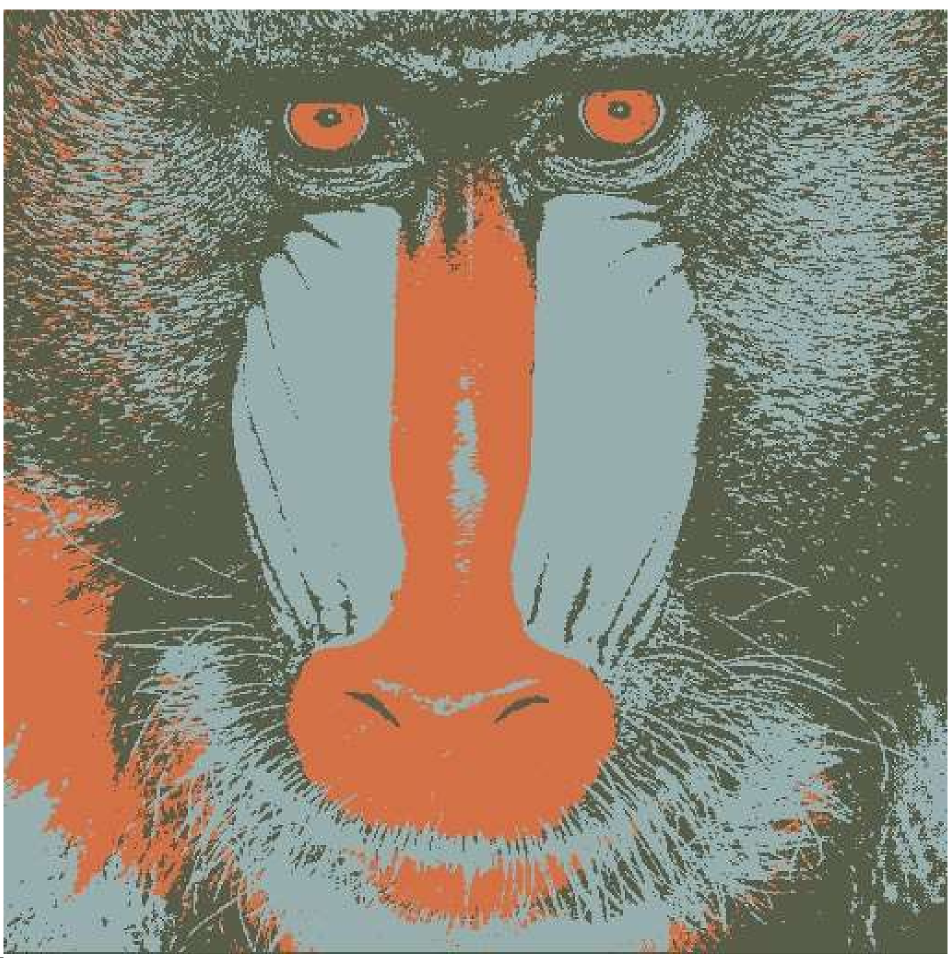


Fig 8: Task3 part4 : color quantization image when k=3

<u>Fig 9: Task3 part4 : color quantization image when k=5</u>

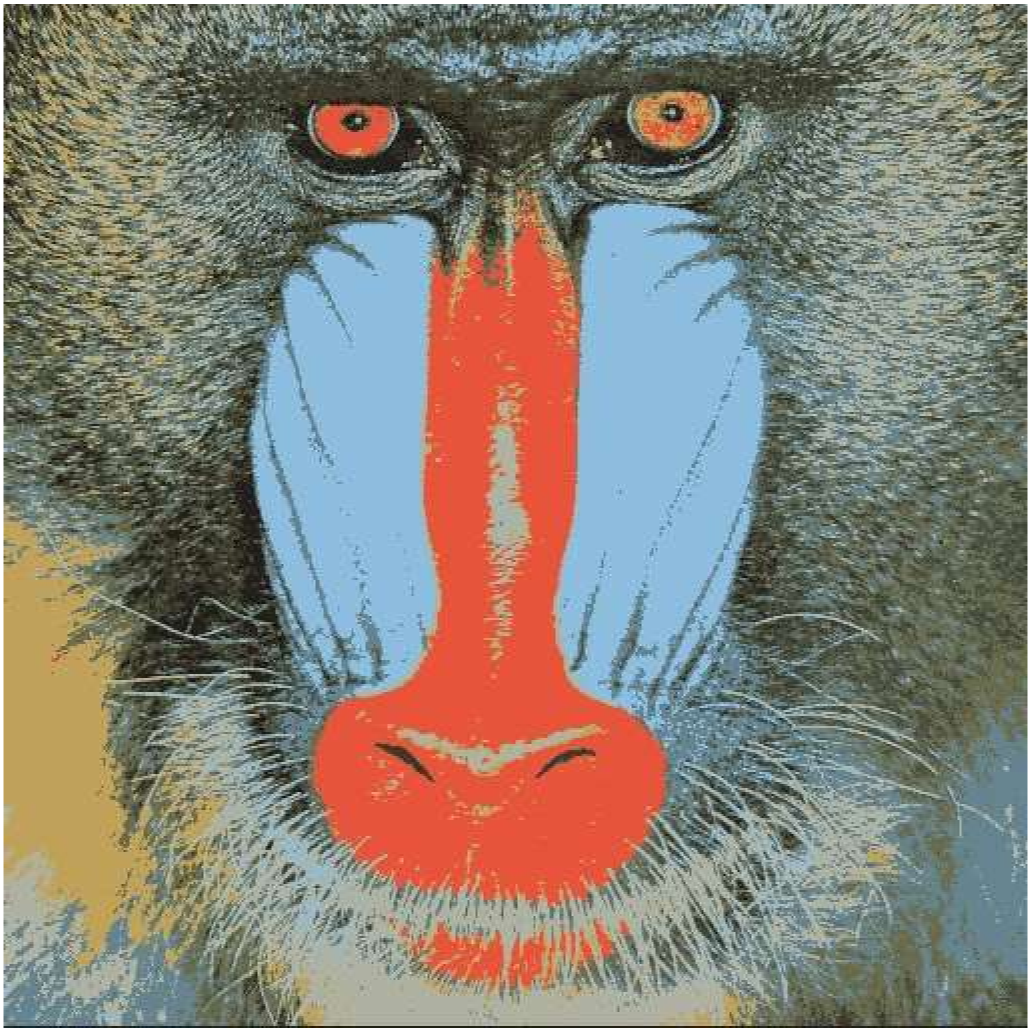


Fig 10: Task3 part4 : color quantization image when k=10

<u>Fig 11: Task3 part4 : color quantization image when k=20</u>