Project 2 : Computer Vision and Image Processing

KRISHNA SEHGAL

UB ID: 50291124

1. **Software Used**

- Anaconda - Navigator
- Jupyter Notebook
- Python 3.6

2. **Libraries Used**

- Num-py
- Open CV
- Matplot

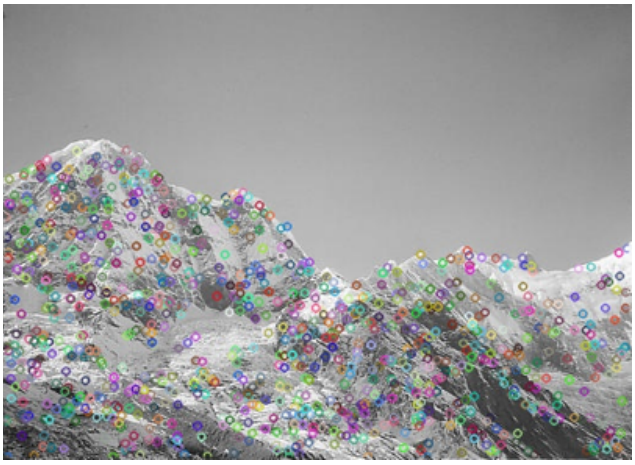
3. **Task**

3.1 Image Features and Homography

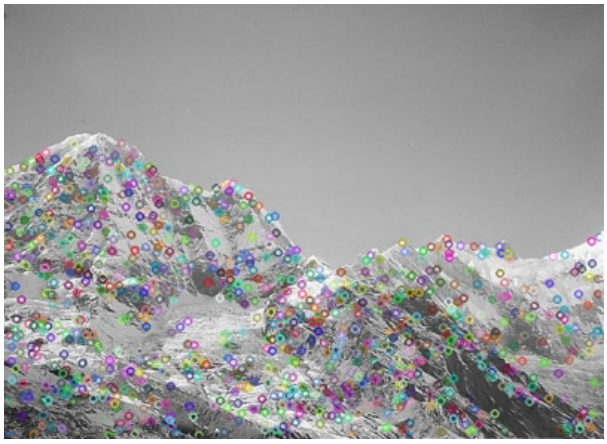
Reference : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html

3.1.1 Extract SIFT Features

Task1_sift1.png

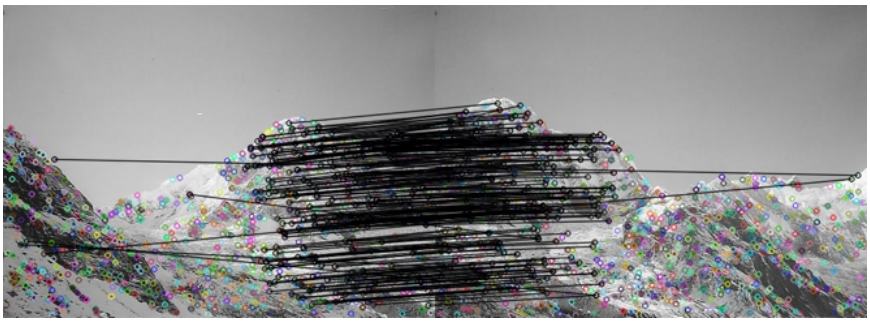


Task1_sift2.png



3.1.2 Match the key-points using k-nearest neighbors

Task1_matches_knn.png



3.1.3 Homography matrix H

```
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
matchesMask = mask.ravel().tolist()
print(M)
```

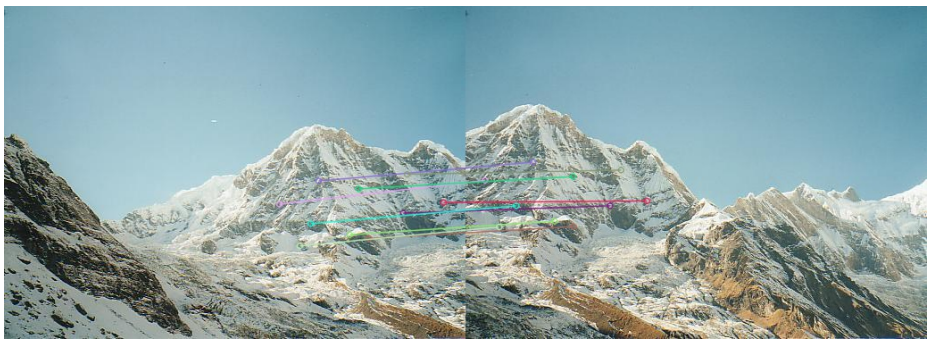
Ransac Values

```
[[ 1.58930391e+00 -2.91559485e-01 -3.95969641e+02]
 [ 4.49424740e-01  1.43111047e+00 -1.90614333e+02]
 [ 1.21265321e-03 -6.28722024e-05  1.00000000e+00]]
```

3.1.4 Match Image for 10 random matches

```
  points = []
for i in range(len(matchesMask)):
    if matchesMask[i] == 1:
        points.append(i)
points = points[:10]

img3 = cv2.drawMatches(img1,kp1,img2, kp2,[good[point] for point in points],None,flags=2)
plt.imsave(img3, 'task1_matches.jpg')
print(M)
```

Task1_matches.png



3.1.5 Warping Image

```
rows1, cols1 = temp1.shape[:2]
rows2, cols2 = temp2.shape[:2]

lp1 = np.float32([[0,0], [0,rows1], [cols1, rows1], [cols1,0]]).reshape(-1,1,2)
temp = np.float32([[0,0], [0,rows2], [cols2, rows2], [cols2,0]]).reshape(-1,1,2)

lp2 = cv2.perspectiveTransform(temp, M)
lp = np.concatenate((lp1, lp2), axis=0)

[x_min, y_min] = np.int32(lp.min(axis=0).ravel() - 0.5)
[x_max, y_max] = np.int32(lp.max(axis=0).ravel() + 0.5)

translation_dist = [-x_min, -y_min]
H_translation = np.array([[1, 0, translation_dist[0]], [0, 1, translation_dist[1]], [0,0,1]])

result = cv2.warpPerspective(temp1, H_translation.dot(M), (x_max - x_min, y_max - y_min))
result[translation_dist[1]:rows1+translation_dist[1],translation_dist[0]:cols1+translation_dist[0]] = temp2
```

Task1_pano.png

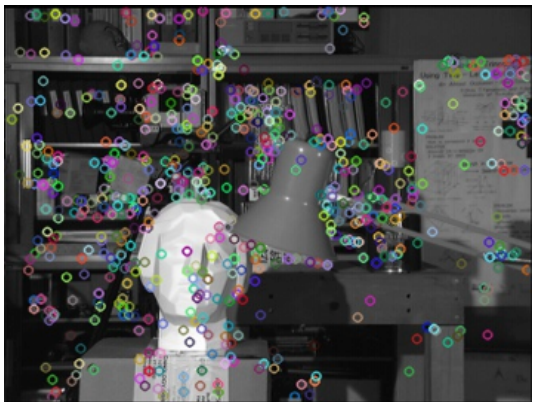
3.2 Epi-polar Geometry

Reference: https://docs.opencv.org/3.2.0/da/de9/tutorial_py_epipolar_geometry.html

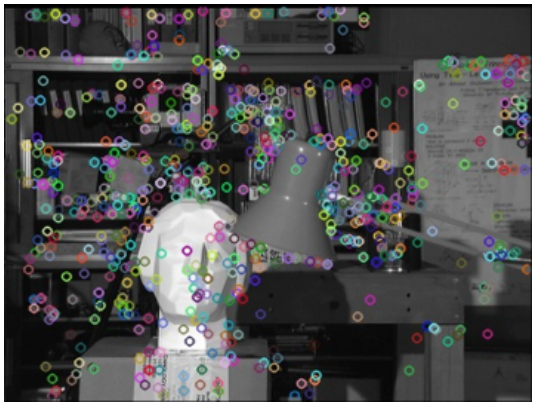
3.2.1 Extract SIFT Features and Match the key-points using k-nearest neighbors

```
import cv2
import numpy as np

img = cv2.imread('/Users/kamallala/Downloads/data/tsucuba_left.png', 0)

sift = cv2.xfeatures2d.SIFT_create()
kp = sift.detect(img,None)

cv2.drawKeypoints(img,kp)

cv2.imwrite('task2_sift1.jpg',img)
```

Task2_sift1.png



Task2_sift2.png



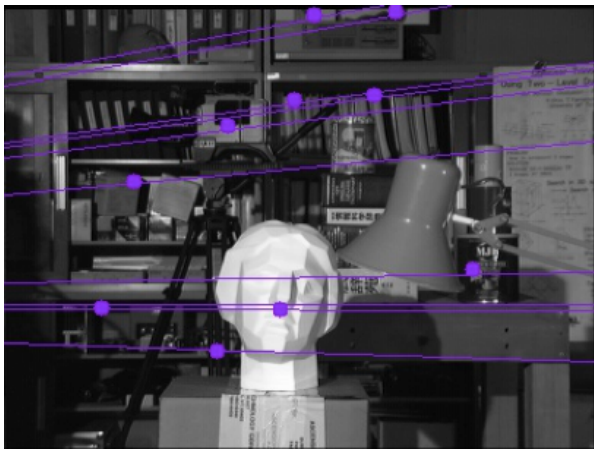
Task2_matches_knn.png

3.2.2 Fundamental Matrix F (with RANSAC)

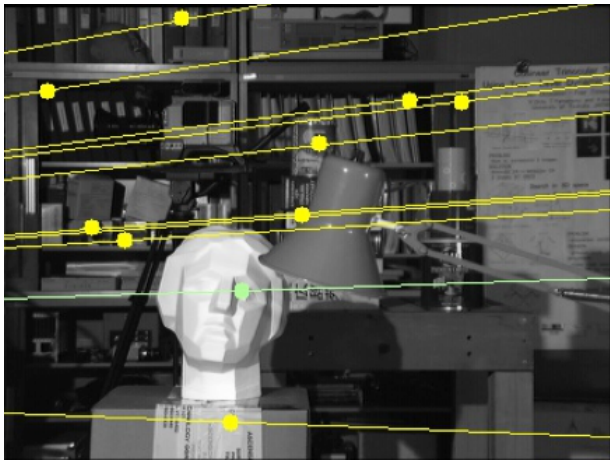
```
[[ 7.27595761e-11 -5.99488616e-04  5.31139374e-02]
 [ 5.99607825e-04  1.90566061e-06  7.94587572e+11]
 [-5.29851913e-02 -7.94587572e+11  1.00000000e+00]]
```

3.2.3 Computing Epiline

Task2_epi_right.png



Task2_epi_left.png



3.2.4 Disparity Map

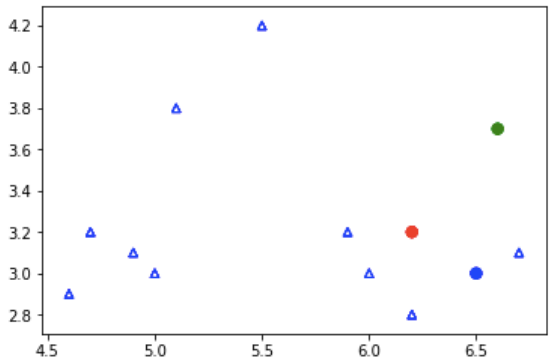
Task2_disparity.png

3.3 K-Means Clustering

3.3.1 Classifying Samples

```
======== Iteration : 0 ========
Clusters [[6.2 3.2]
 [6.6 3.7]
 [6.5 3. ]]
Assignments [0, 0, 2, 0, 1, 0, 0, 2, 0, 0]


Assignments [0, 0, 2, 0, 1, 0, 0, 2, 0, 0]
['red', 'red', 'blue', 'red', 'green', 'red', 'red', 'blue', 'red', 'red'
]
```

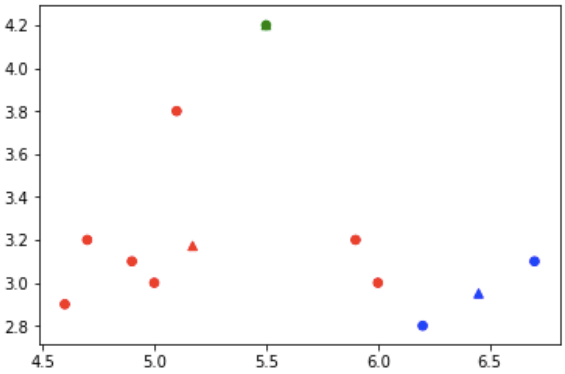
task3 iter1 a.png



3.3.2 Compute Clusters for 1st Iteration

```
-
======== Iteration : 1 ========
Clusters [[5.17142857 3.17142857]
 [5.5        4.2       ]
 [6.45       2.95      ]]
Assignments [2, 0, 2, 0, 1, 0, 0, 2, 1, 2]


Assignments [2, 0, 2, 0, 1, 0, 0, 2, 1, 2]
['blue', 'red', 'blue', 'red', 'green', 'red', 'red', 'blue', 'green', 'b
lue']
```

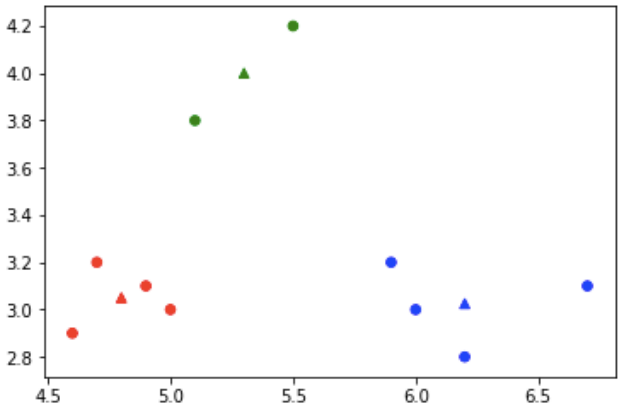
task3 iter1 b.png



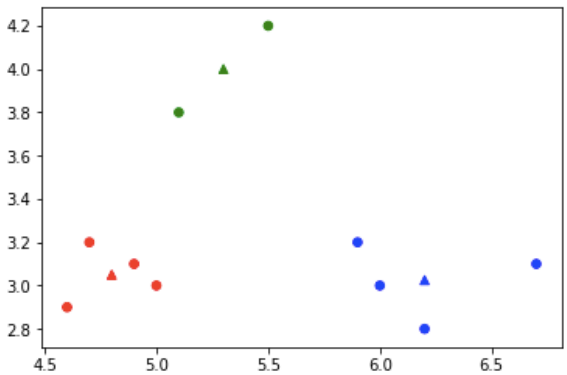
3.3.3 Compute Clusters for 2nd Iteration

```
======== Iteration : 2 ========
Clusters [[4.8   3.05 ]
 [5.3   4.   ]
 [6.2   3.025]]
Assignments [2, 0, 2, 0, 1, 0, 0, 2, 1, 2]


Assignments [2, 0, 2, 0, 1, 0, 0, 2, 1, 2]
['blue', 'red', 'blue', 'red', 'green', 'red', 'red', 'blue', 'green', 'b
lue']
```

task3 iter2 a.png



task3 iter2 b.png



Source Code

```
UBIT='ksehgal';
import random
import math
import numpy as np
np.random.seed(sum([ord(c) for c in UBIT]))
%matplotlib inline
from copy import deepcopy

import pandas as pd
from matplotlib import pyplot as plt
import warnings



#Euclidian Distance between two d-dimensional points
def eucldist(p0,p1):
    dist = 0.0
    for i in range(0,len(p0)):
        dist += (p0[i] - p1[i])**2
    return math.sqrt(dist)
```

```
    # d - Dimensionality of Datapoints
    d = len(datapoints[0])

    #Limit our iterations
    Max_Iterations = 1000
    i = 0
    cluster = [0] * len(datapoints)
    prev_cluster = [-1] * len(datapoints)
    #Choose Centers for the Clusters
    cluster_centers = []
    for i in range(0,k):
        cluster_centers=  np.array([[6.2,3.2],[6.6,3.7],[6.5,3.0]])
        c1_x=cluster_centers[:,:1]
        c1_y=cluster_centers[:,1:]
        f1=datapoints[:,:1]
        f2=datapoints[:,1:]
        plt.scatter(f1, f2,marker="^",facecolors='none',edgecolors='b', s=30)
        colors=["red","green","blue"]
        plt.scatter(c1_x, c1_y, marker=".", s=200, c=colors)

    while (cluster != prev_cluster) or (i > Max_Iterations):
        prev_cluster = list(cluster)
        i += 1
        #Update Point's Cluster
        for p in range(0,len(datapoints)):
            min_dist = float("inf")
            #Check min_distance against all centers
            for c in range(0,len(cluster_centers)):
                dist = eucldist(datapoints[p],cluster_centers[c])
                if (dist < min_dist):
                    min_dist = dist
                    cluster[p] = c   # Reassign Point to new Cluster
        print("======== Iteration :",i-3,"========")
        print("Clusters",cluster_centers)
        print("Assignments",cluster)

```

```
        for k in range(0,len(cluster_centers)):
            new_center = [0] * d
            members = 0
            for p in range(0,len(datapoints)):
                if (cluster[p] == k): #If this point belongs to the cluster
                    for j in range(0,d):
                        new_center[j] += datapoints[p][j]
                    members += 1

            for j in range(0,d):
                if members != 0:
                    new_center[j] = new_center[j] / float(members)


            cluster_centers[k] = new_center
```

Reference : https://gist.github.com/pmsosa/5454ade527adbee105dd51066ef30c5f

3.3.4 Image Quantization

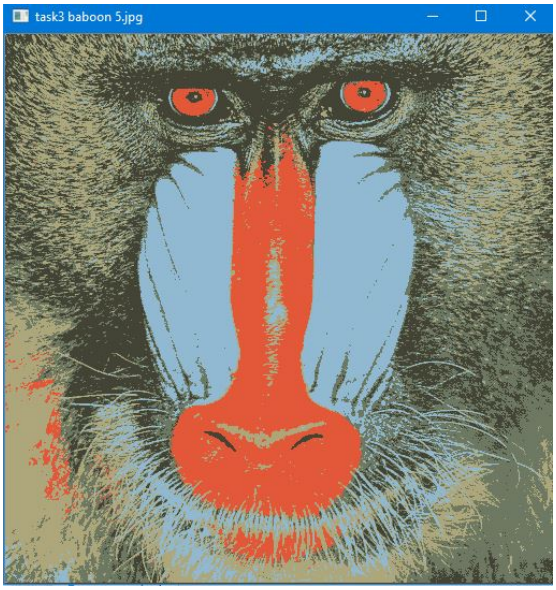
K - Number of Clusters

K=3

task3_baboon_3.png

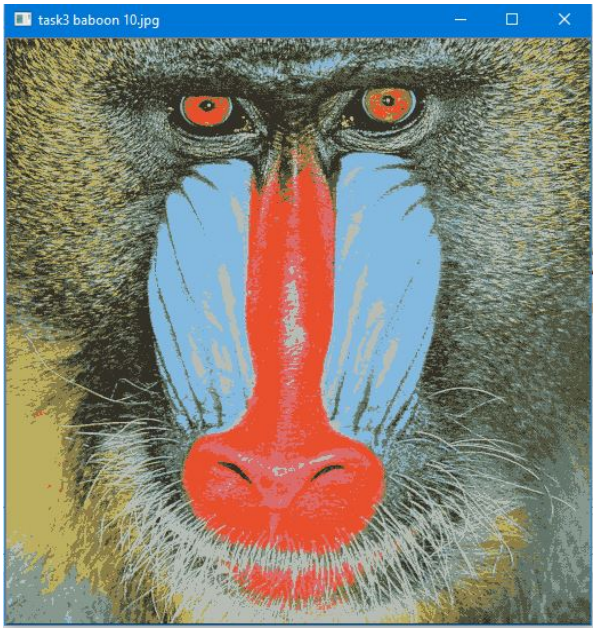
K=5

task3_baboon_5.png



K=10

task3_baboon_10.jpg



K=20

task3_baboon_20.jpg