

COMPUTER VISION AND IMAGE PROCESSING

PROJECT 2



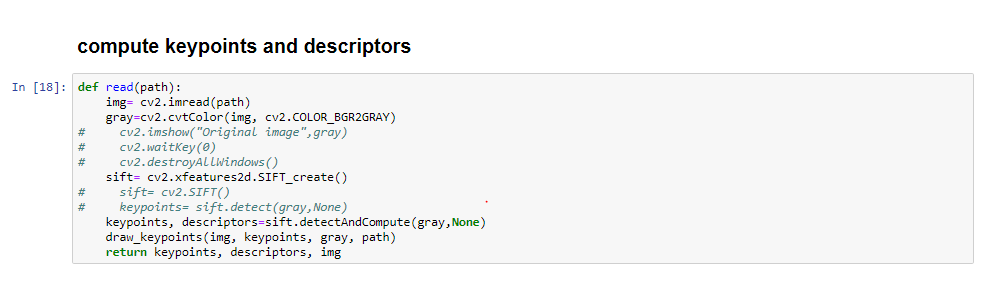
November 5, 2018

University at Buffalo

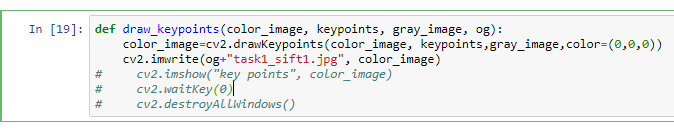
MAYANK KULSHRESTHA PERSON NUMBER: 50291406

TASK 1

1. For task 1.1 we need to extract SIFT features and draw the key points for both the input images, namely mountain1.jpg and mountain2.jpg. In order to draw key points we first extract key points using **xfeatures2d.SIFT\_create()**  provided by opencv.



**1.1**In order to draw key points for both images I used the method drawKeypoints(). I passed 3 parameters to this method which are original image, grayscale version of the same image and the **key points** obtained using the above mentioned method.

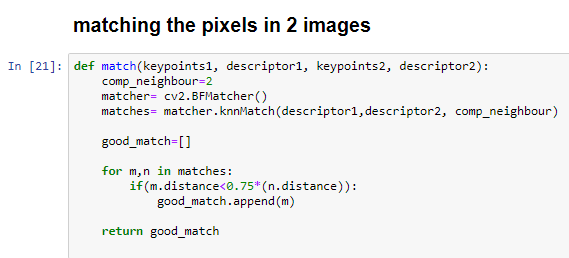


The output was following two images:

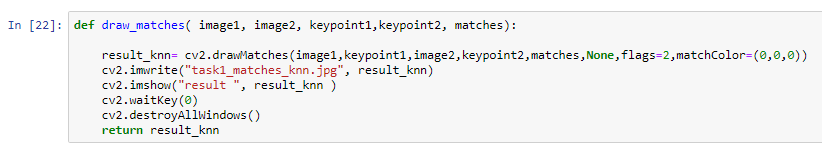
1.) 

2.) 

**1.2** In order to compute good matches in the 2 images I created method named match() in which I passed the key points and descriptors obtained from the two images. This function used descriptors to return matches obtained using **K Nearest Neighbor** matching in which each point is compared to the nearest K pixels around it(neighbor) and assign it to the class which is in majority among the K neighbors. In this project the k is specified as 2 hence each pixel is assigned the class which is in majority among its 2 nearest neighbors. Following is how I used the method and implemented.



After obtaining the good\_match list from the above method I passed good\_match, both the imges i.e. image1 and image2 and their respective keypoints to the method name Draw\_matches(). This method invoked cv2.drawMatches to draw the match image.

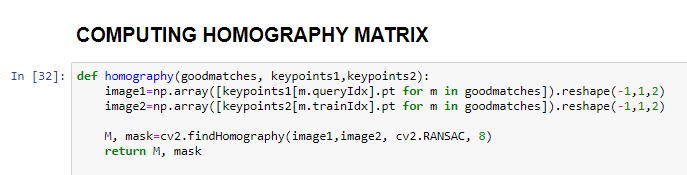


The result obtained by invoking the above mentioned method was as follows. The image contained bith inliers and the outliers (or noise).

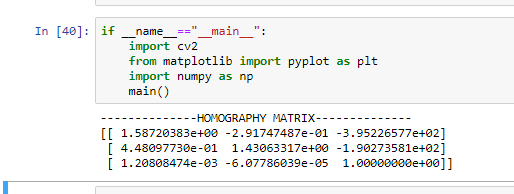


**1.3 Homography matrix**

In order to calculate homography matrix I wrote a method homography() which took goodmatches, keypoints of first image and keypoints of second image and invoked cv2.findHomography() method to return Homography matrix H and a mask. Mask contains the information about which keypoints are inliers (value of that index is 1) and outliers (value at that index is 0).



The homography matrix obtained from this method was as follows



**1.4**

In order to find the inliers, the mask obtained using the above mentioned method was used along with the good\_match, keypoints of both the images and both the images itself. We used the mask to find out the indexes at which the value is 1 because it represents the inlier. And then we use the goodmatch to find all the inliers using mask as a dictionary.

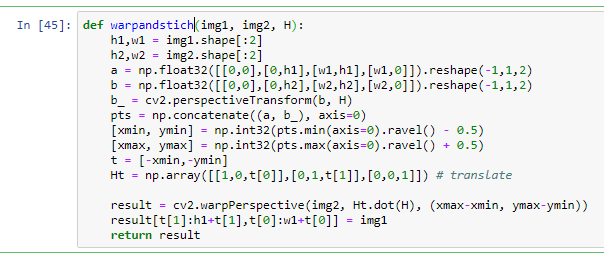


The image obtained using this method contained 10 randomly selected inliers.



1.5

The result after using the following warp method:



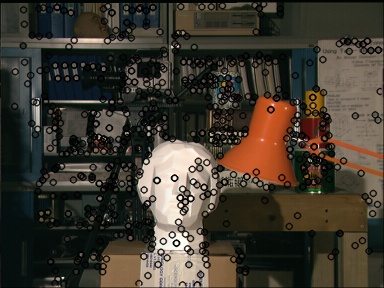
The result was as follows:



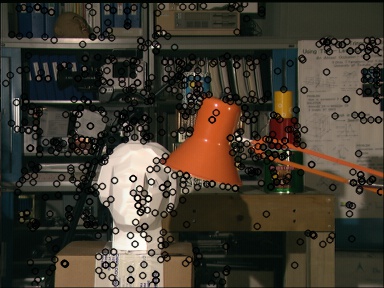
Task 2

**2.1** This goal was achieved the same way as mentioned in the above task. Following are the output images obtained for this task.

**Left image with keypoints detected:**

****

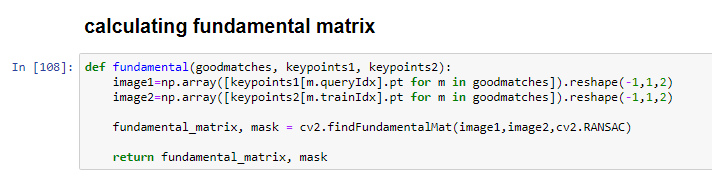
**Right image with keypoints detected:**

****

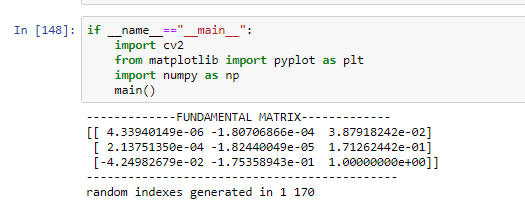
2.2 The fundamental matrix was calculated using the following method that invoked the



Where fundamental() method implementation is as follows:



Following fundamental matrix was obtained:

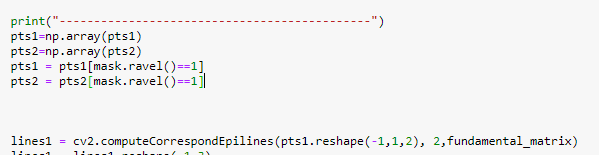


2.3

In order to calculate the 10 random epilnes firstly the we have to calculate the inliers and in order to find the inliers we again retrieve the mask using the following method ():

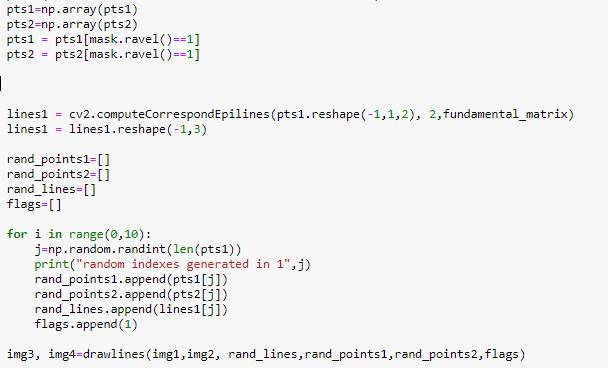


Using this mask all the inliers were found out. Then the epilines were found out using the following method for both the images. before calling the method to find the epilines the mask was converted into a 1-dimensional vector, since before that it was a vector of vectors (mask= [ [ ], [ ], [ ], [ ]…………]). This was done using ravel() method where the value of mask was 1 (that is where there are inliers).

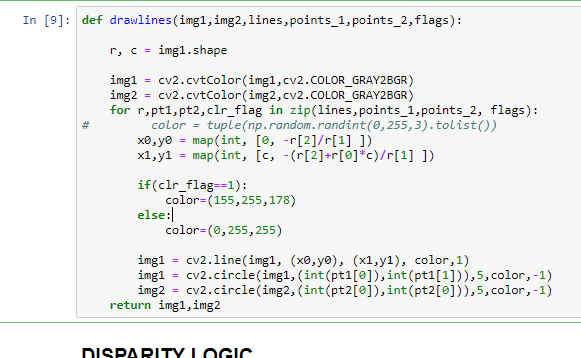


Now in order to find the 10 random epilines and keypoints. I generated 10 random numbers between the range of number of indices present in the both training and query points obtained after matching.

Pts1 and pts2 contains only those x-y coordinate points where mask is 1. Following is the code that I wrote.

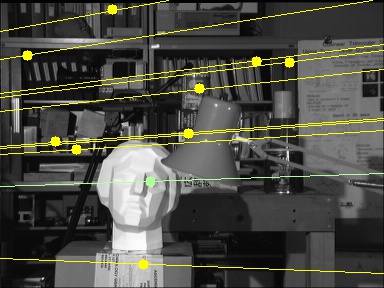
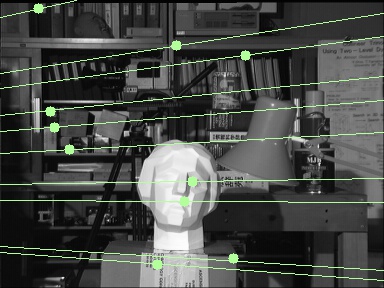


After this the randomly generated 10 lines and the corresponding points for the two images were send to the method drawlines whose implementation is as follows :



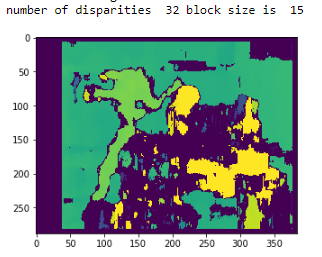
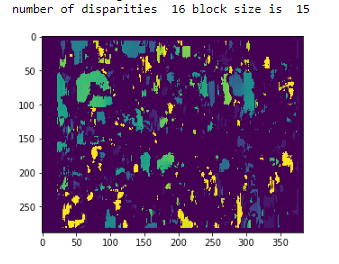
The flag used in this method basically keeps track if the corresponding points in the 10 randomly generated points on image 1 and two are same. If they are same then we have to the line of different color. Hence the color RGB value changes.

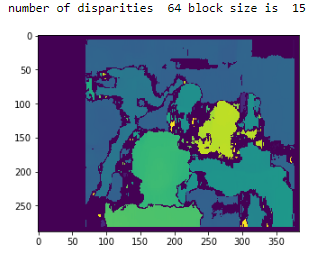
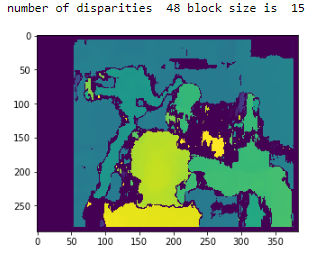
Now one of the images where the two keypoints out of 10 randomly generated inliers in different images match is attached (after executing the program 24 times). The two images are -



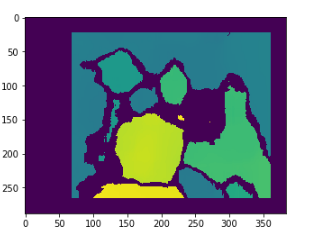
1.4

I coded method which computes an object of stereo bitmap of specific window size and block size. After that to compute the disparity of two images we simply invoke a method using the stereo object to compute the disparity. The compute () method was given two parameters namely- num of diaparities and blocksize. The number of disparities can only be a multiple of 16. Blocksize can be any odd value between 0 and 255. The results that I got for different values of window size and block size are as follows:





* Number of disparities =48 block size is 45

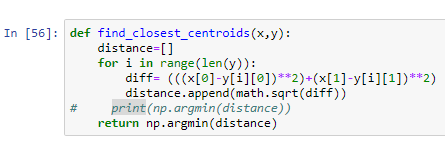


**TASK 3**

3.1

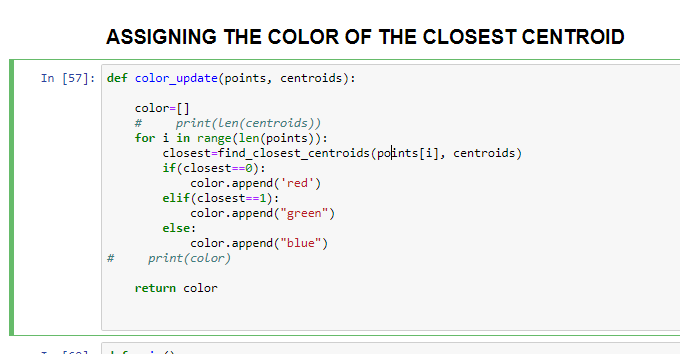
In order to classify 10 2 dimensional coordinates into 3 cluster with initial cluster centers as

µ1= (6.2, 3.2), µ2= (6.6, 3.7) and µ3 = (6.5, 3.0). I created a method called color\_update (), which takes array of points and the array of initial means or centroids. In this method I calculate the minimum distance by calling another method named find\_closest\_centroid (). This method calculates the distance between the coordinate received and the current set of centroids and returns the index of the centroid that is closest to that point.

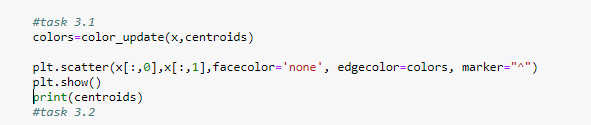


Color\_update keeps on building a list which contains the color of the centroid that the ithelement is closest to. This list will later help us plot the points on a graph.

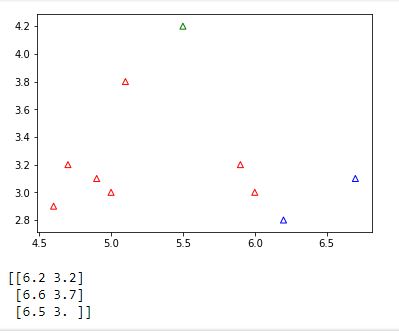
Here is the code that I wrote for color update:



Thus clustering the points based on the initial centroids provided we only need to pass the points and the centroids to the color\_update () method which will return a list of colors of respective points as follows:

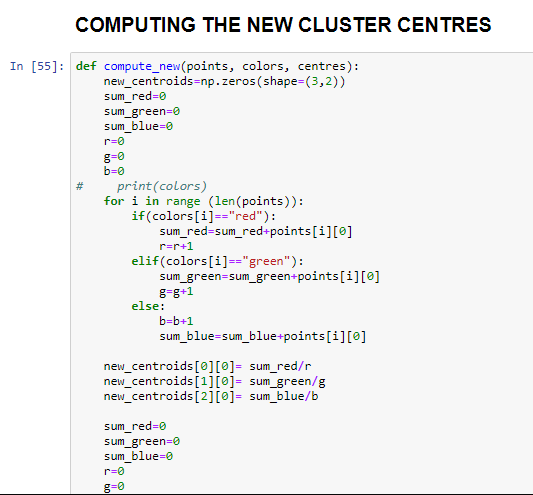


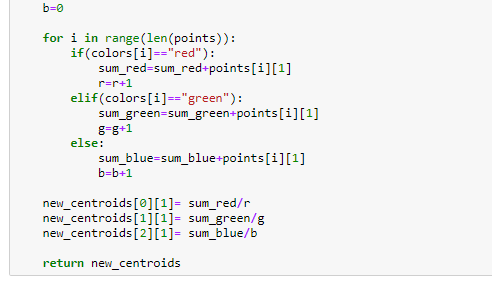
The resulting graph is as follows:



3.2

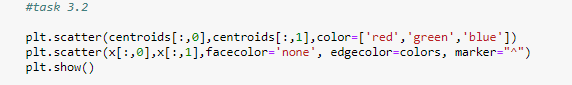
in order to compute the new centroids, I coded a method which is pretty self-explanatory- compute\_new ( ) which takes in points, centroids and colors and calculate the mean of points of same color and returns a set of new centroids( number of centroids same as number of initial centroids).



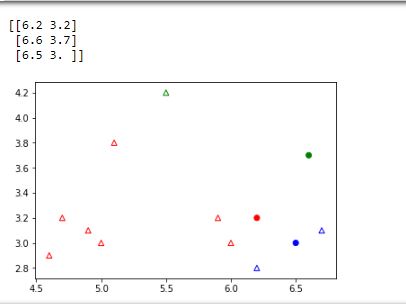


The above mentioned function is called in order to calculate the new cluster centers as follows:

But firstly we need to plot the points along with the initial centroids on the graph which is done as follows :

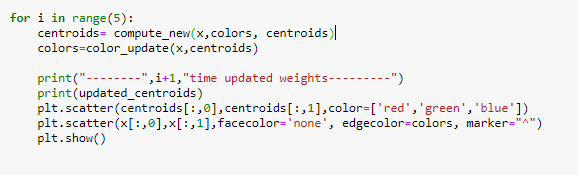


The result obtained was as follows:

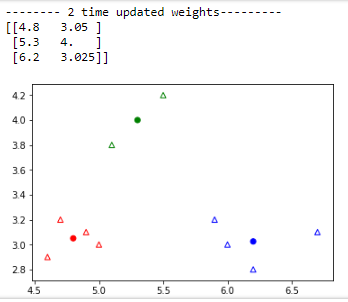
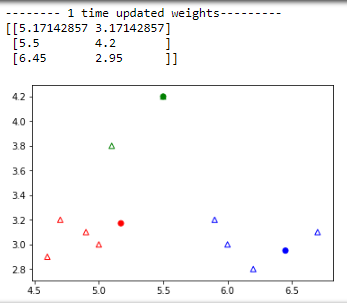


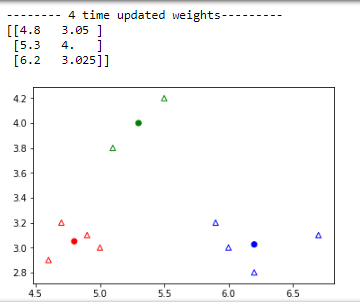
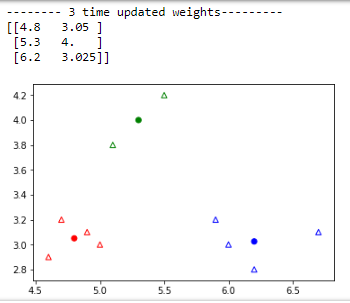
3.2

Now in order to update weight x times all we need to do is call compute\_new () and update\_color () methods in a loop that runs for x times and plot the centroids and points each time with their respective colors. Here is an example:



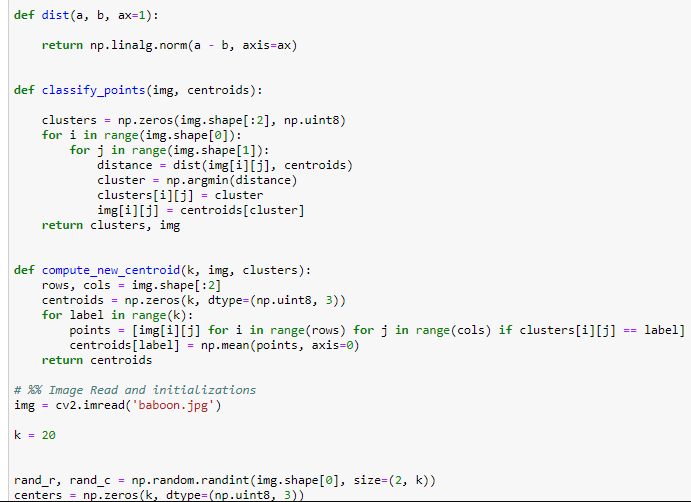
The result for second iteration weight update and the following times is attached below:

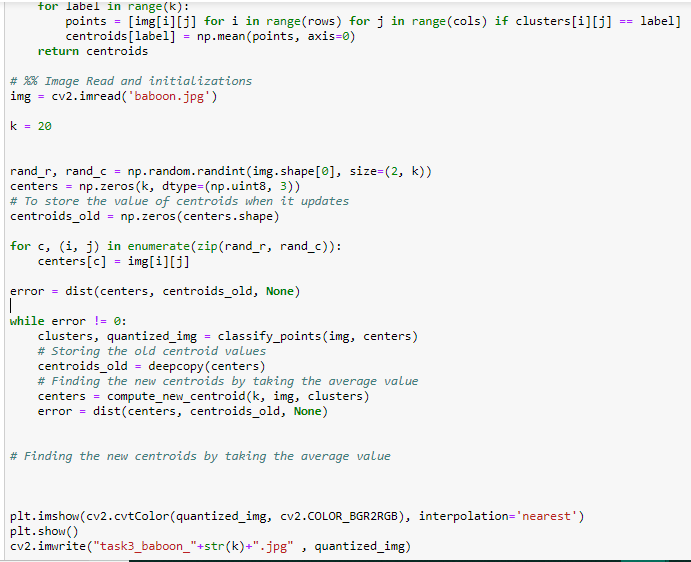




We can clearly see that after 3rd iteration the weights stopped changing hence our algorithm converged to the optimum centers for the given data points.

3.4





K=5

K=3

K=10 K=20

**SOURCES/BIBLIOGRAPHY**

* <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html>
* <https://www.youtube.com/watch?v=9991JlKnFmk>
* <https://medium.com/consonance/k-means-and-image-quantization-part-2-be0a62c50c11>