CS682 : Computer Vision Submissions

Brinston Gonsalves

Homework 3

```
How to use :
1. Download the python script.
2. Create a images folder in the same directory as the script.
3. Copy all the images to the newly created folder on which the operations are required to be performed.

** Please Use Open Image In new Tab to look at larger images.
```

Question 1 :

Here, I have displayed the output images from the SIFT and SURF feature detectors. I have also attached a copy of the console output which displays all the keypoints along with its size and orientation. The output images also have the keypoints displayed in random colors along with the size equal to the radius of the circle and orientation as the line connecting the centre to the boundary of the circle.

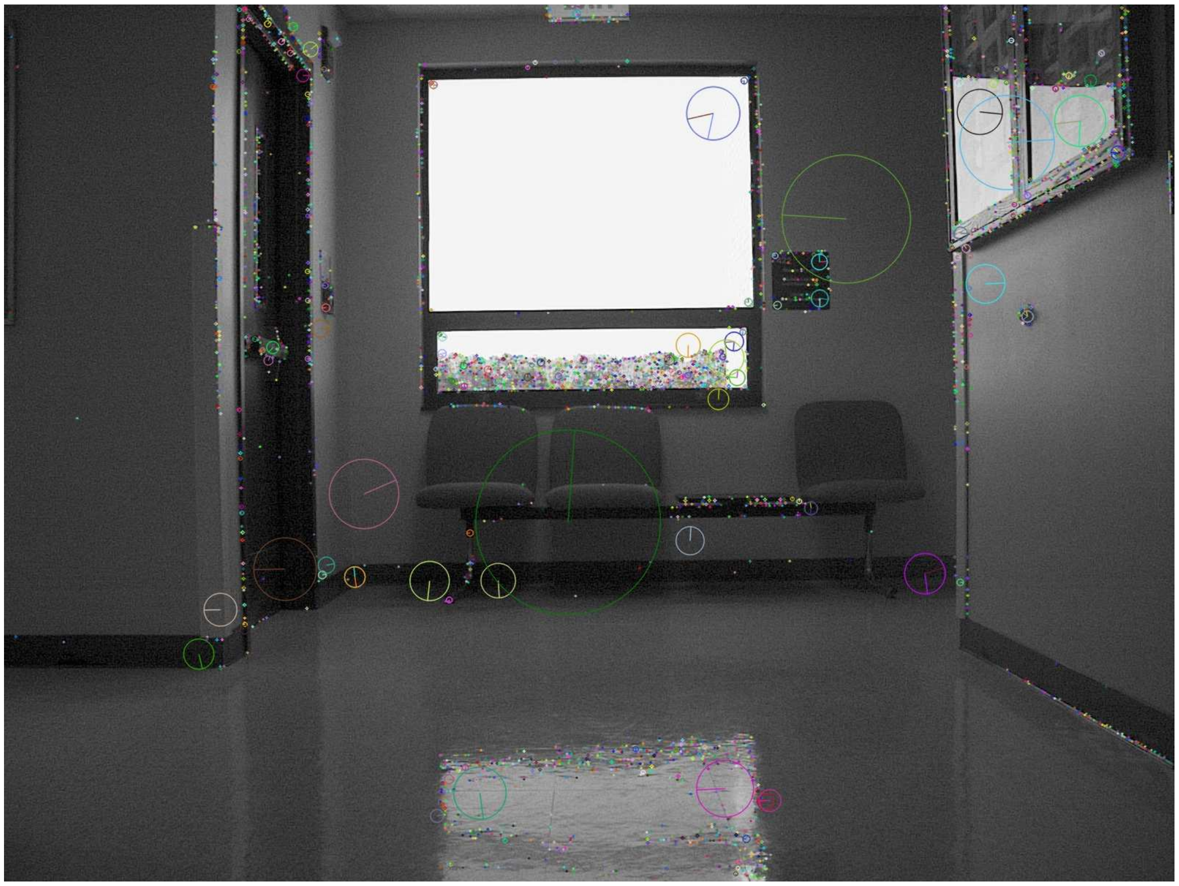
As you can see in the code, I ran both the SIFT and SURF feature detectors on the same image, and as we can see in the console output, SIFT found out 3089 keypoints whereas SURF found out 1448 keypoints. Though, generally SURF has a advantage of time over SIFT , SIFT found out more features and thus I found it to be more appropriate for the assignment.

Python Script : Download

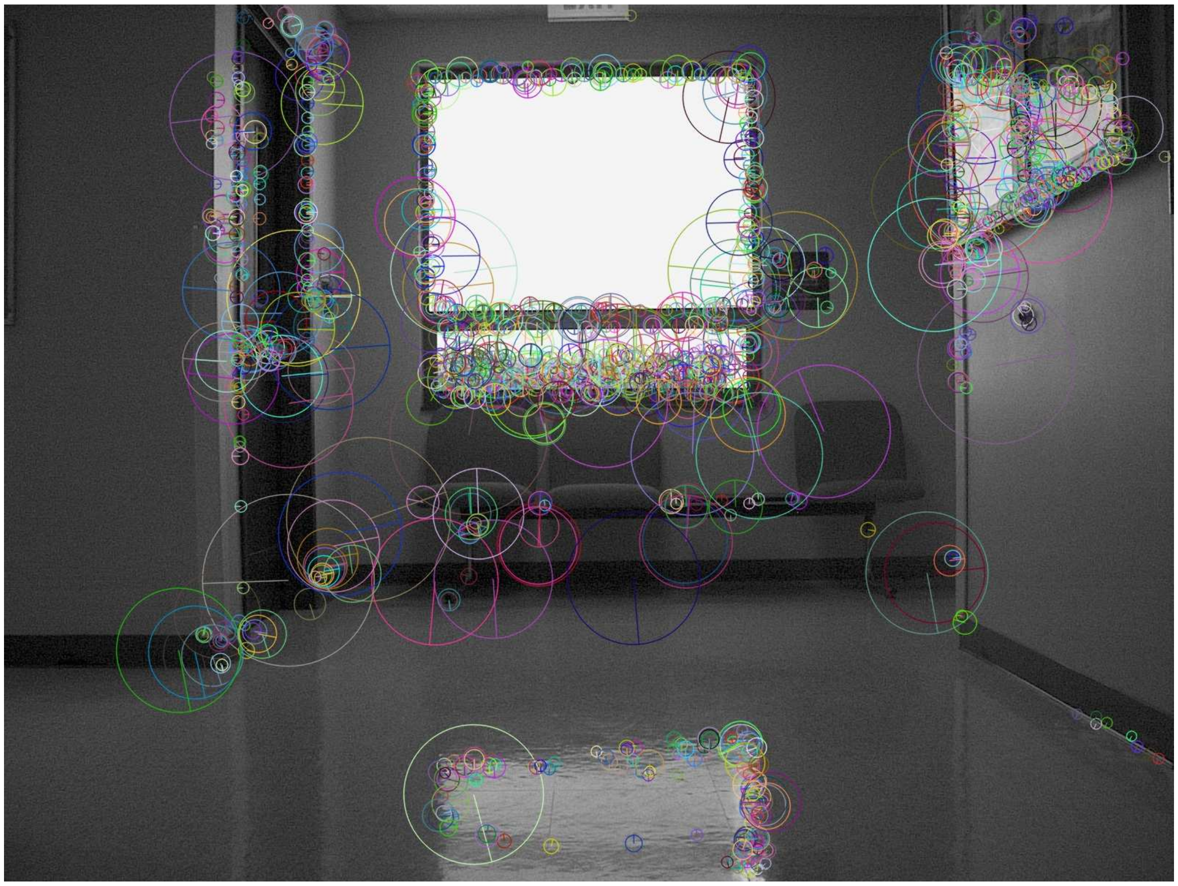
```
import cv2
import numpy as np
import scipy as sp


img = cv2.imread("images/ST2MainHall4019.jpg")
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

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

print "------------SIFT Keypoints--------------"
print("Number of Keypoints : {0}".format(len(kp)))

for i in range(0,len(kp)) :
    print("Keypoint Coordinates : {0}".format(kp[i].pt))
    print("Keyoint Size : {0}".format(kp[i].size))
    print("Keypoint Angle : {0}".format(kp[i].angle))

sift_output = img.copy()
cv2.drawKeypoints(gray,kp,sift_output,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imwrite("sift.jpg", sift_output)


surf_output = img.copy()
surf = cv2.xfeatures2d.SURF_create(400)
kp = None
kp = surf.detect(gray,None)

print "------------SURF Keypoints--------------"
print("Number of Keypoints : {0}".format(len(kp)))

for i in range(0,len(kp)) :
    print("Keypoint Coordinates : {0}".format(kp[i].pt))
    print("Keyoint Size : {0}".format(kp[i].size))
    print("Keypoint Angle : {0}".format(kp[i].angle))

cv2.drawKeypoints(gray,kp,surf_output,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imwrite("surf.jpg",surf_output)
```

Download Console Output : Includes Size and Orientation of KeyPoints for both SIFT and SURF

SIFT output Image :



SURF output Image :



Question 2

Here, in the python script I have provided the names of the files as links to what pairs of Images I have considered for matching. After having calculated the features, I have used a bruteforce matcher to identify the pairs of features in both the images that correspondingly match to each other.

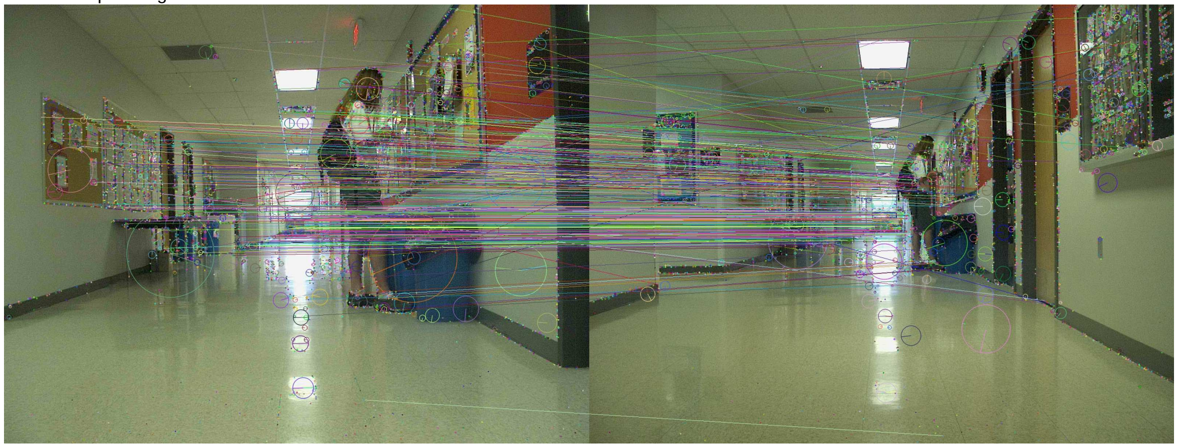
For discarding the bad correspondences, I am using the Lowe's method that states that a factor of 75% is a good balance to discard the false possitives. The way selection of this threshold affects the output is that, large value will introduce noise whereas smaller value will reduce the number of keypoints generated. I have kept ht e threshold same across all the images, since the Lowe's method seems to be working fine across all the images and in real time, always changing the threshold is never feasible.

Python Script : Download

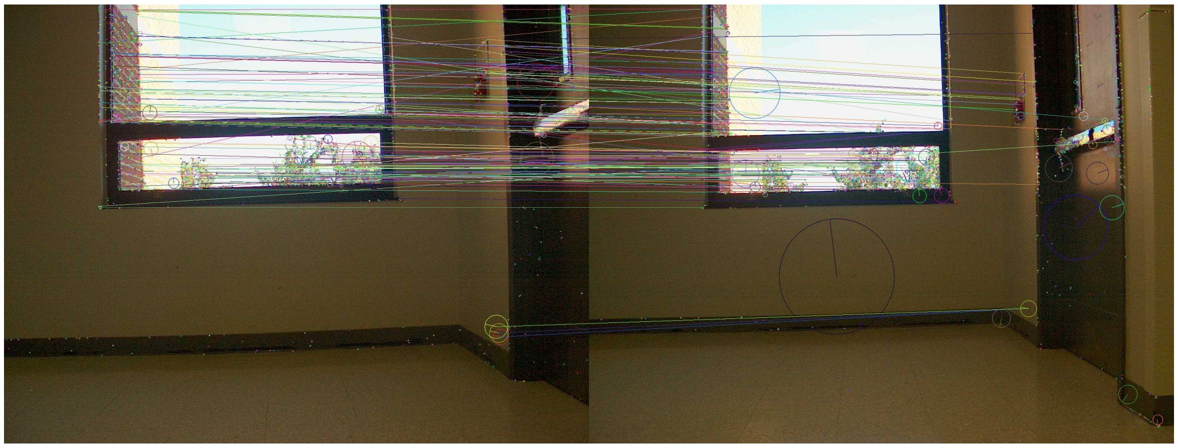
```
import cv2
import sys
import scipy as sp
import random


file1 = "images/ST2MainHall4099.jpg"
file2 = "images/ST2MainHall4093.jpg"

file3 = "images/ST2MainHall4075.jpg"
file4 = "images/ST2MainHall4073.jpg"

file5 = "images/ST2MainHall4041.jpg"
file6 = "images/ST2MainHall4046.jpg"

file7 = "images/ST2MainHall4079.jpg"
file8 = "images/ST2MainHall4039.jpg"

file9 = "images/ST2MainHall4022.jpg"
file10 = "images/ST2MainHall4098.jpg"
```

```
'''
# Use this block instead if you want to take input images from arguments
if len(sys.argv) < 3:
    print 'usage: %s img1 img2' % sys.argv[0]
    sys.exit(1)

img1_path = sys.argv[1]
img2_path = sys.argv[2]

img1 = cv2.imread(img1_path)
img2 = cv2.imread(img2_path)
'''

img1 = cv2.imread(file9)
img2 = cv2.imread(file10)

gray1 = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

sift = cv2.xfeatures2d.SIFT_create()

kp1, d1 = sift.detectAndCompute(gray1,None)
kp2, d2 = sift.detectAndCompute(gray2,None)

matches = cv2.BFMatcher().knnMatch(d1,d2,2)

selected_matches = [m for m,n in matches if m.distance < 0.75*n.distance]

view1 = img1.copy()
view2 = img2.copy()

cv2.drawKeypoints(img1, kp1, view1, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS);
cv2.drawKeypoints(img2, kp2, view2, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS);

h1, w1 = img1.shape[:2]
h2, w2 = img2.shape[:2]
view = sp.zeros((max(h1, h2), w1 + w2, 3), sp.uint8)
view[:h1, :w1] = view1
view[:h2, w1:] = view2

for m in selected_matches:
    color = tuple([sp.random.randint(0, 255) for _ in xrange(3)])
    cv2.line(view, (int(kp1[m.queryIdx].pt[0]), int(kp1[m.queryIdx].pt[1])),
    (int(kp2[m.trainIdx].pt[0] + w1), int(kp2[m.trainIdx].pt[1])),
    (random.randint(0,255),random.randint(0,255),random.randint(0,255)),
    lineType=cv2.LINE_8)

cv2.namedWindow("Matches", cv2.WINDOW_NORMAL)
cv2.imshow("Matches", view)
cv2.waitKey()

cv2.imwrite("match5.jpg",view)
```

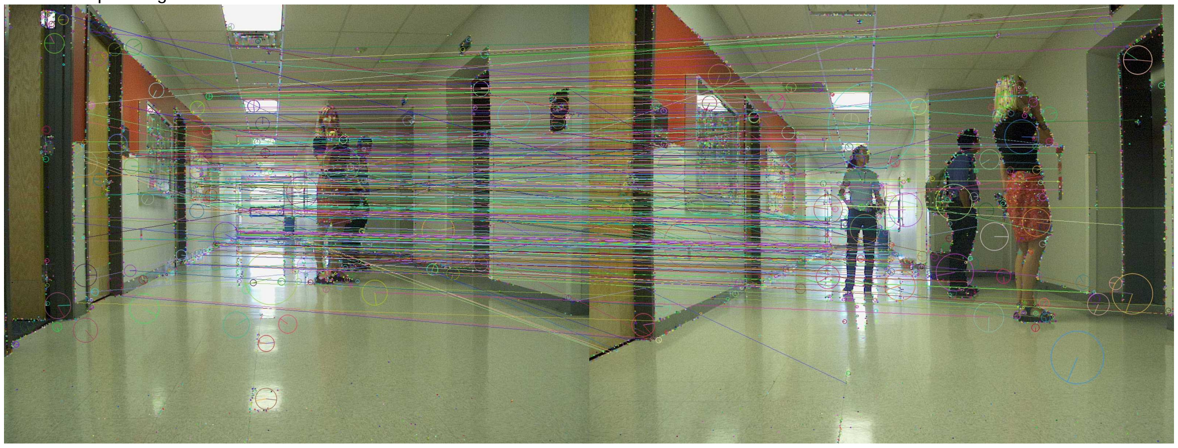
Match 1 output Image :



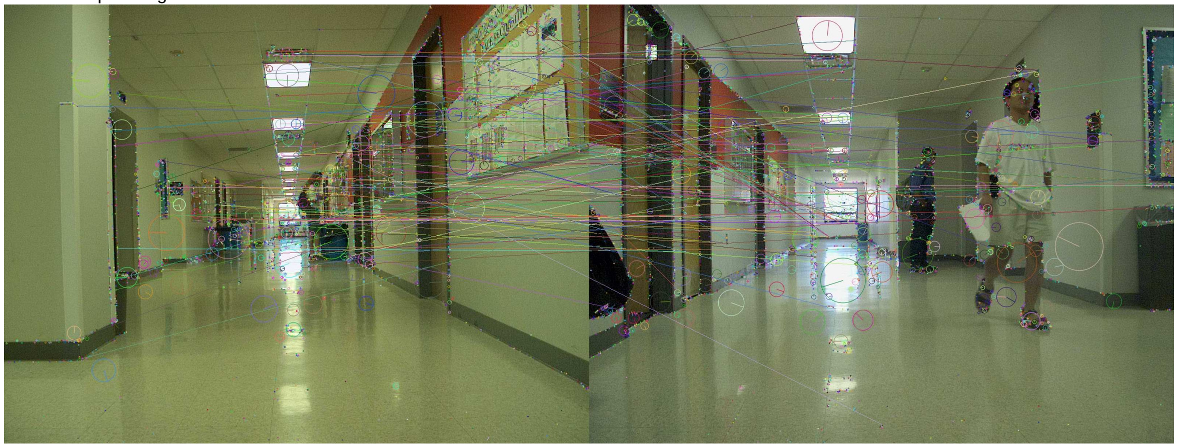
Match 2 output Image :



Match 3 output Image :



Match 4 output Image :



Match 5 output Image :

Question 3

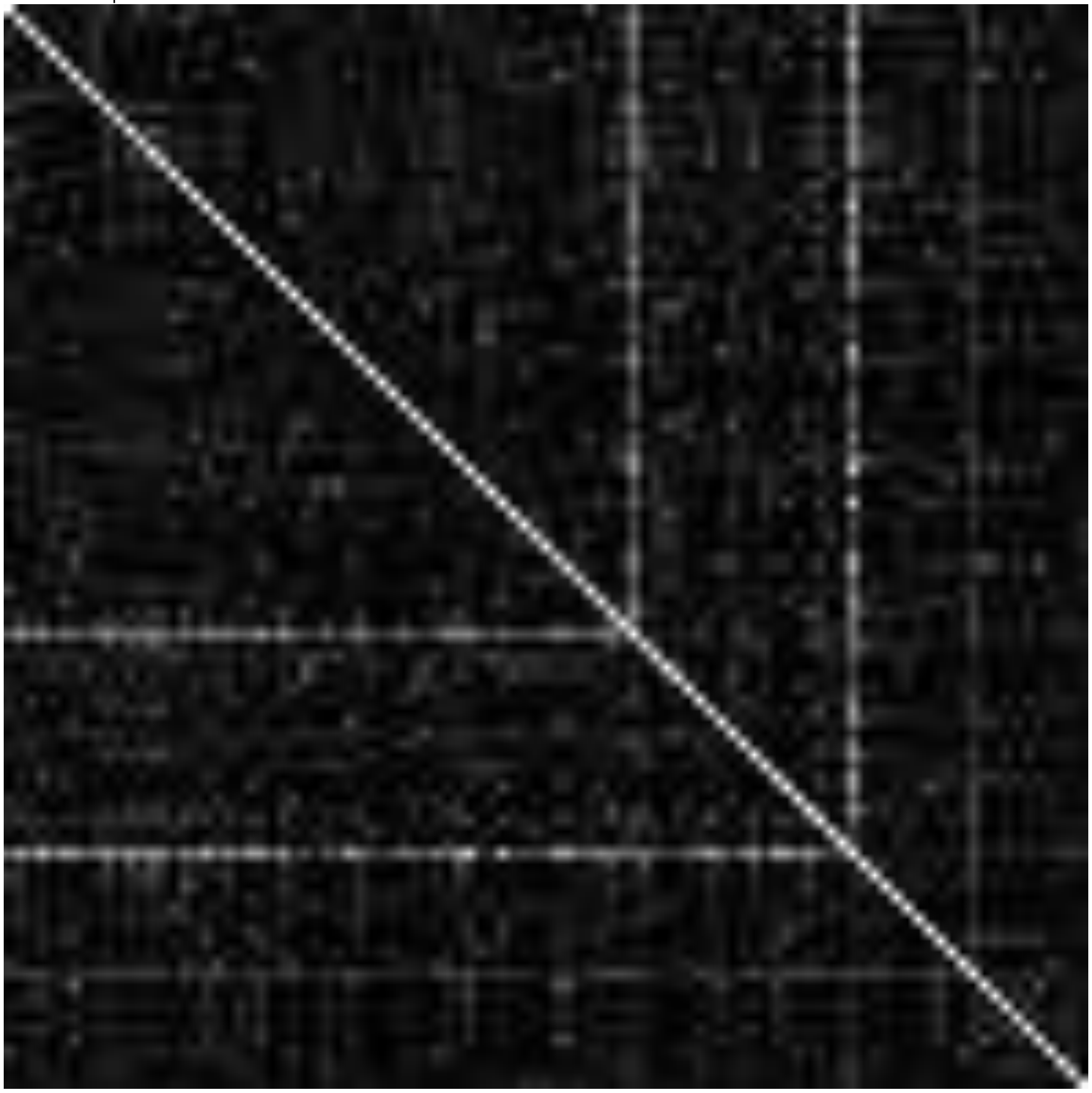
Here we have deviced a formulae that generates value proportional to the distance between the images. We use the same code that we have written for the part 2. For filtering the best results, we utilize the knn function in the Brute Force Matcher with the value of k equal to 2. Here, for just keeping the good results, we again use the Lowe's method and later calculate a proportional value. We then normalize the matrix and force the diagonal values to be one as specified in the question. Later, we scale the values on a 0-255 scale so that different colors are generated corresponding to the values. In the final Image, we note that, the size of the image is 99x99 which corresponds to 99 images in the given folder.

Python Script : Download

```
import cv2
import scipy as sp
import numpy as np
import sys
import glob
import csv
import os

def match(img1_path, img2_path):
    img1_c = cv2.imread(img1_path)
    img2_c = cv2.imread(img2_path)

    img1 = cv2.cvtColor(img1_c, cv2.COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(img2_c, cv2.COLOR_BGR2GRAY)

    sift = cv2.xfeatures2d.SIFT_create()

    k1, d1 = sift.detectAndCompute(img1, None)
    k2, d2 = sift.detectAndCompute(img2, None)

    matches = cv2.BFMatcher().knnMatch(d1, d2, 2)

    sel_matches = [m for m, n in matches if m.distance < 0.75 * n.distance]

    return float(len(sel_matches)) / min(len(k1), len(k2))


#output = match("images/ST2MainHall4099.jpg","images/ST2MainHall4099.jpg")
#print output

def main(argv):

    '''
    if len(argv) < 2:
        print 'usage: %s images_folder' % sys.argv[0]
        sys.exit(1)
    '''
    imgs = glob.glob("/home/brinstongonsalves/Documents/PyCharm/CV/images/" + "*.jpg")

    count = len(imgs)

    mat = np.zeros((count, count), np.float32);
    for i in range(0, count):
        for j in range(i + 1, count):
            m = match(imgs[i], imgs[j])
            print "%s %s %.3f \n" % (os.path.basename(imgs[i]), os.path.basename(imgs[j]),m),
            mat[i, j] = mat[j, i] = m

    mat = mat / mat.max()
    for i in range(0, count): mat[i, i] = 1.0
    np.savetxt('mat.txt',mat)

    table = np.loadtxt('/home/brinstongonsalves/Documents/PyCharm/CV/mat.txt')
    cv2.imwrite("allMatches1.png", table * 255)

if __name__ == "__main__":
    sys.exit(main(sys.argv));

# main("/home/brinstongonsalves/Documents/PyCharm/CV/images")
```

Download the txt file containing the distance matrix here

AllMatches output :



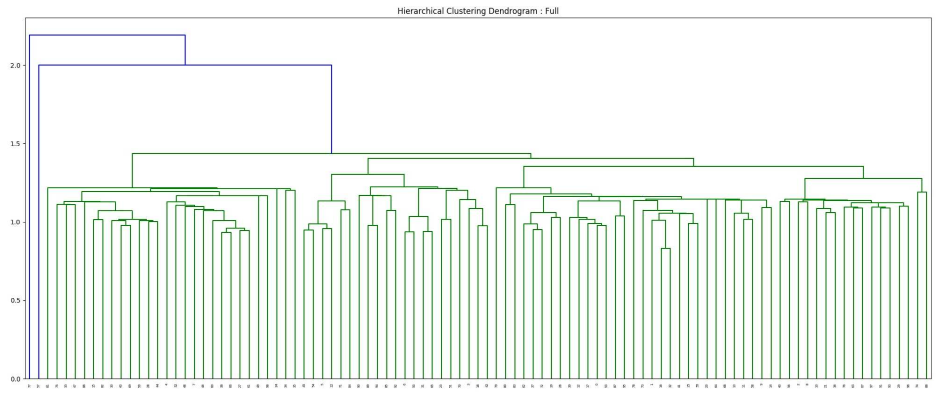
Question 4

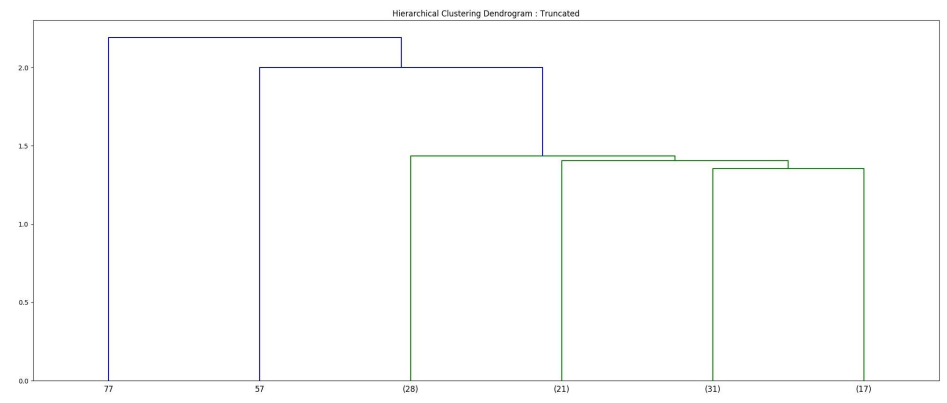
Here we use Hierarchical Clustering from the Scipy library. For the example I have specified the Number of Clusters to be equal to 6. The first image displays the entire dendrogram. The second image is of the same dendrogram truncated for the first merges to form 6 clusters. Here we note for the clusters at any point of time sum to the number of images. i.e 99. The number specified in the round brackets in the second image are the number of elements in that particular cluster.

Here, in the previous question we generated a distance matrix and stored it to a file. Now, for convinience, we are regenrating the distance matrix from this stored file.

Python Script : Download

```
import pylab
from matplotlib import pyplot as plt
import scipy.spatial.distance as dist
import scipy.cluster.hierarchy as hier
import cv2
import numpy as np

NUM_CLUST = 6

distSqMat = np.loadtxt('/home/brinstongonsalves/Documents/PyCharm/CV/mat.txt')
link_mat = hier.linkage(distSqMat,'single')
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram : Full')
hier.dendrogram(link_mat)
plt.savefig("dendogram.jpg")

plt.clf()
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram : Truncated')
hier.dendrogram(link_mat,truncate_mode='lastp',p = NUM_CLUST,)
plt.savefig("dendogram1.jpg")
```




Hierarchical Clustering Dendrogram : Truncated
2.0
1.5
1.0
0.5
0.0
77
57
(28)
(21)
(31)
(17)