CS217 : Assignment 2 : Structure From Motion

Please edit the cell below to include your name and student ID #

name:

SID:

1. Feature Matching

1.1 SIFT matching library

Install an implementation of SIFT-based feature matching; I recommend using the one integrated into opencv-python. However, getting it installed is a little bit tricky because the algorithm is patented so it isn't included with some distributions. I used the recipe below to install on my system.

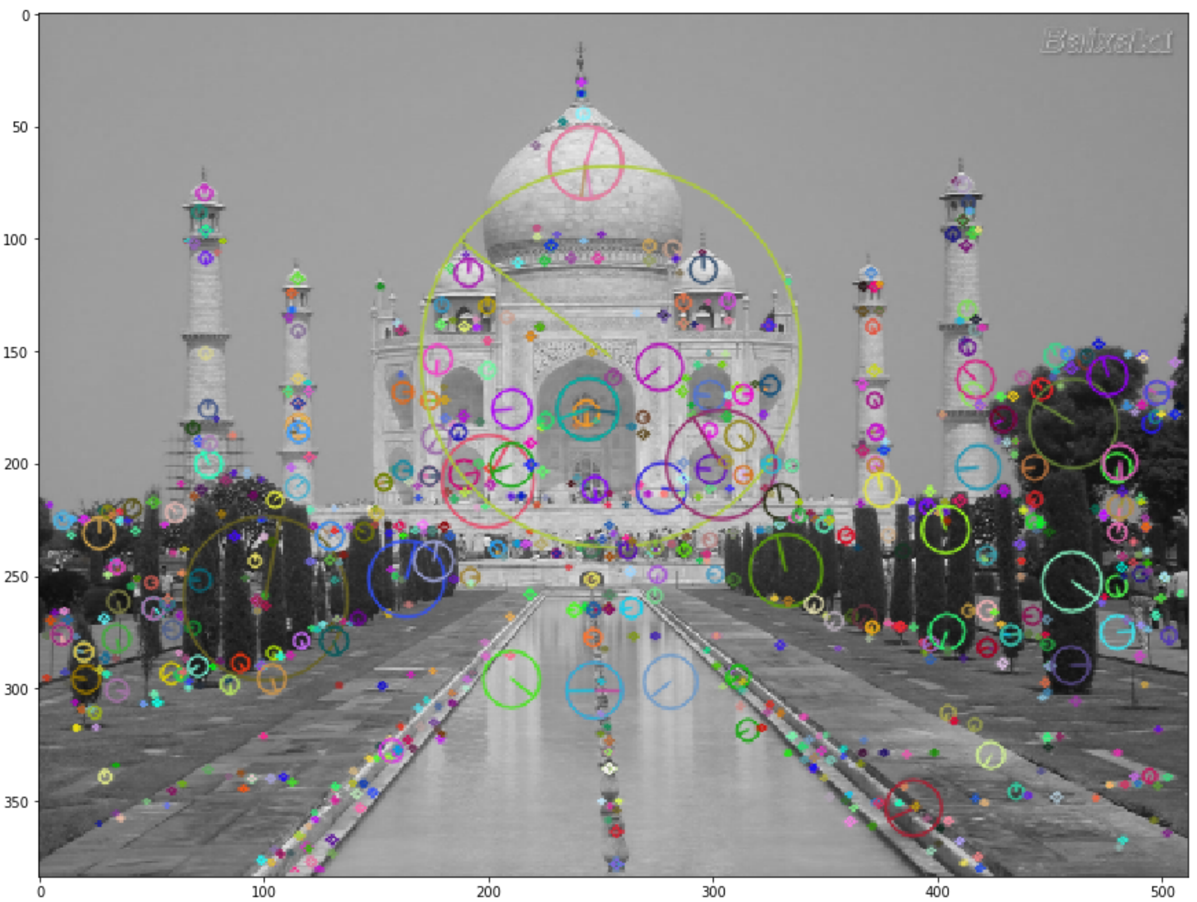
conda install opencv

pip install opencv-contrib-python==3.4.2.16

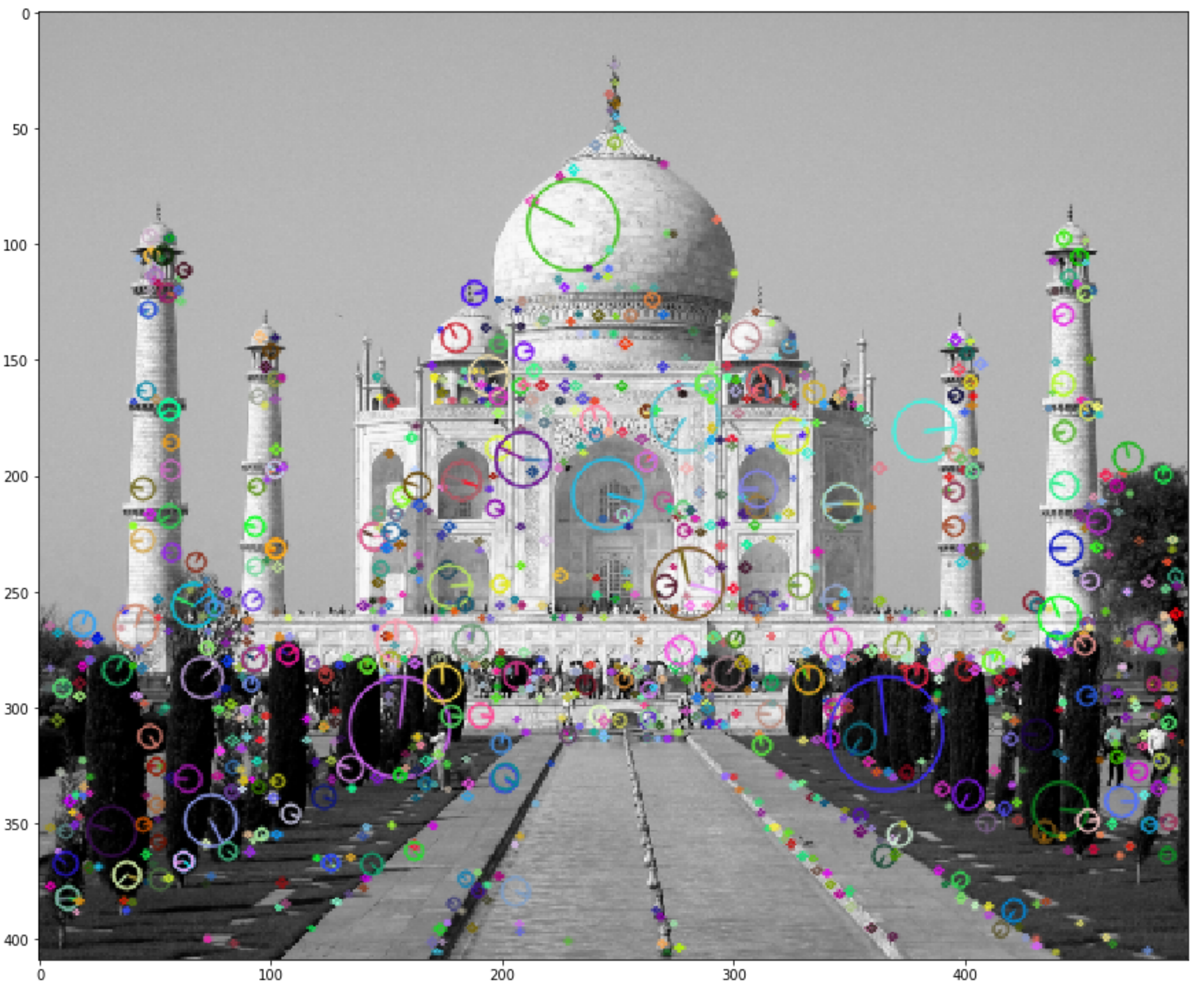
I have provided some example code below exercising the OpenCV SIFT matching functions. Once you are comfortable with this, please implement a function which takes as input the two images and returns the coordinates for the candidate matching points for use in the remainder of the assignment. If you decide not to use OpenCV then you should implement this function to wrap whatever version of SIFT you use.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
import cv2

# you can use matplotlib notebook during development but
# when you print out your notebook as pdf there may be
# problems with figures displaying so you may want to switch
# to inline
%matplotlib inline
```

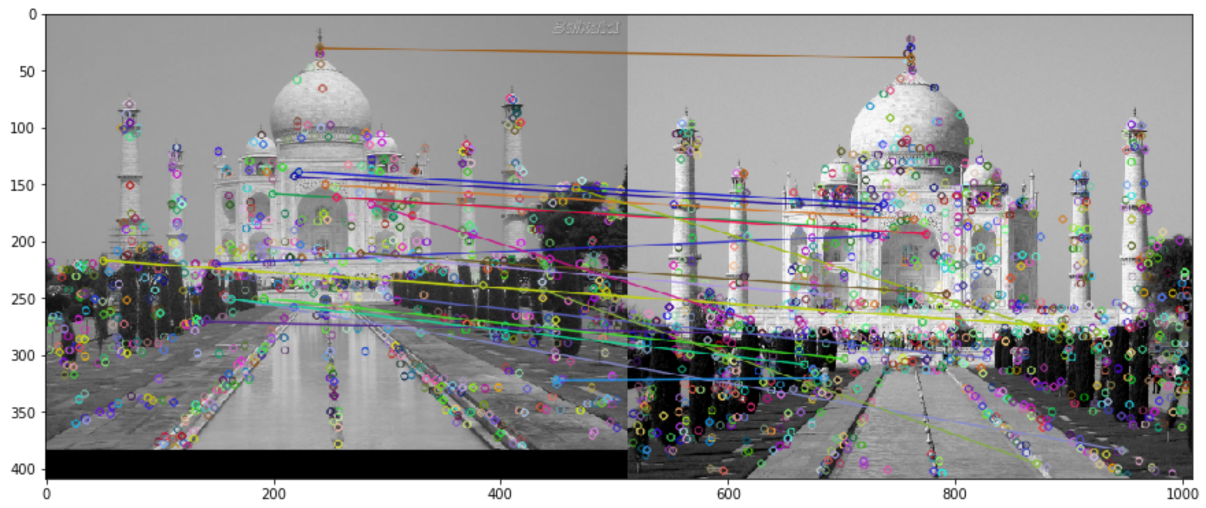
In [38]:
```python
#
# Example code exercising the OpenCV library matching functions
#

# load in images, convert to grayscale, resize
img1 = cv2.imread('data/G.jpg')
img1 = cv2.cvtColor(img1,cv2.COLOR_RGB2BGR)
gray1= cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
dim = (0.4*np.asarray(gray1.shape)).astype(int)
gray1 = cv2.resize(gray1,(dim[1],dim[0]))

img2 = cv2.imread('data/H.jpg')
img2 = cv2.cvtColor(img2,cv2.COLOR_RGB2BGR)
gray2= cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
dim = (0.4*np.asarray(gray2.shape)).astype(int)
gray2 = cv2.resize(gray2,(dim[1],dim[0]))


# extract SIFT feature descriptors and keypoints from each image
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray1,None)
kp2, des2 = sift.detectAndCompute(gray2,None)

# visualize SIFT features
vis1=cv2.drawKeypoints(gray1,kp1,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_
vis2=cv2.drawKeypoints(gray2,kp2,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_

plt.rcParams['figure.figsize'] = [15,15]
plt.imshow(vis1)
plt.show()
plt.imshow(vis2)
plt.show()

# perform matching to find the 2 nearest neighbors for each descriptor
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

# only keep a match if the distance to the 1st match is significantly
# less than the distance to the second match
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])


# Visualize the resulting matches. You may wish to write out the image t
# a file or open in a separate window to view at higher resolution.
img3 = cv2.drawMatchesKnn(gray1,kp1,gray2,kp2,good[:20],None)
plt.imshow(img3)
plt.show()
```



0
50
100
150
200
250
300
350
0
100
200
300
400
500



0
50
100
150
200
250
300
350
400
0
100
200
300
400



In [39]:
```python
def getMatches(img1,img2):
    """
    Given two grayscale images, return a set of candidate matches using 

    Parameters
    ----------
    img1 : 2D numpy.array
    img2 : 2D numpy.array
            Grayscale images to match

    Returns
    -------
    pts1 : 2D numpy.array (dtype=float)
    pts2 : 2D numpy.array (dtype=float)
        Candidate pairs of matching points from img1 and img2 stored in 


    """

    return (pts1,pts2)
```

1.2 Visualize SIFT candidate matches

Write code below to exercise your **getMatches** function. Choose a pair of images you want to work with. I've provided a few examples but you are encouraged to take your own photos or use some images off the web. Visualize the resulting set of candidate matches. You can either modify your **getMatches** function to also return the **cv2** keypoint data structures and use **cv2.drawMatchesKnn**, or alternately come up with your own visualization technique to show correpsondences.

In [ ]:
```

```

2. Estimation of the Fundamental Matrix

As you can see from the matching results, SIFT finds matches which are not consistent with a single overall camera motion. In order to find a consistent set of matches, implement RANSAC to estimate the fundamental matrix F. For each random sample you should estimate F using the normalized 8 point algorithm (as described in Hartley's paper "In defense of ..." linked from the class website and discussed in section 4.4 of Hartley and Zisserman). You will need to experiment with the appropriate error threshold for determining inliers and outliers

In [ ]:
```
def fitFudamental(pts2L,pts2R):
    """
    Estimate fundamental matrix from point correspondences in two views

    Parameters
    ----------
    pts2L : 2D numpy.array (dtype=float)
    pts2R : 2D numpy.array (dtype=float)
        Coordinates of N points in the left and right view stored in arr


    Returns
    -------
    F : 2D numpy.array (dtype=float)
        Fundamental matrix of shape (3,3)

    fiterror : float
        The quality of the fit measured as the average of (x^T*F*x')^2 o

    """




    return (F,fiterror)
```

In [ ]:
```
#
#  I recommend debugging / testing your fitting function with some synth
#  where you know what the "true" F matrix is. You can do this by genera
#  from two views using the project function from Assignment #1.
#
```

In [ ]:
```
def fitFundamentalRANSAC(pts2L,pts2R,thresh,niter):
    """
    Robust estimation of fundamental matrix from point correspondences i

    Parameters
    ----------
    pts2L : 2D numpy.array (dtype=float)
    pts2R : 2D numpy.array (dtype=float)
            Coordinates of N points in the left and right view stored in

    thresh : float
        Error threshold to decide whether a point is an inlier or outlie

    niter : int
        Number of RANSAC iterations to run

    Returns
    -------
    F : 2D numpy.array (dtype=float)
        Fundamental matrix of shape (3,3)

    inliers : numpy.array (dtype=int)
        Indices of the points which were determined to be inliers

    fiterror : float
        The quality of the fit measured as the average of (x^T*F*x')^2 o

    """

    return (F,inliers,fiterror)
```

In [ ]:
```
#
#  Again, it is probably good to test your RANSAC function using some to
#  You can add in some random point matches as outliers and see that RAN
#  correctly eliminates them.
#
```

2.1 Visualize Correspondences

Using the same image pairs as in Problem 1, show the best set of correspondences chosen by RANSAC as inliers. If things are working correctly, the correspondences should be more consistent than in Problem 1 since they all now satisfy the same epipolar constraint. Experiment with the inlier threshold and number of iterations in order to get the best result

In [ ]:

2.2 Epipolar Lines

Visualize the epipolar geometry for a subset of 20 feature points in one of your image pairs. For both image you should plot ~10 feature points along with the epipolar lines on which they should lie given your estimate of F and their corresponding point in the other image. What can you say about the relative pose of the two camera views based on the arrangement of epipolar lines? Can you tell which is the "left" and "right" camera from the lines alone.

In [ ]:

```

```

2.3 Stability

How stable is your estimate of F? If you rerun RANSAC multiple times with random samplings, does it always return the same F matrix? Please write a bit of code below to carry out this experiment and compute some quantitative measure of the variability in estimated F. How could you get a more robust estimate?

In [ ]:

```

```

3. Calibrated Structure From Motion

Download the calibration images provided (sfm.zip). Modify the provided camera calibration script **calibrate.py** as needed and use it to estimate the intrinsic parameter matrix K and enter the results below.

In [ ]:

```
K = np.array(....)
```

3.1 Estimate the Essential Matrix

Write code which loads in the pair of test images, use your SIFT matching routine to return a set of matching points. Normalize these point coordinates using K^{-1} and then modify your code from Problem 2 to recover the essential matrix E. Remember that you can still use the 8 point algorithm but now you will need to set the two singular values to be equal instead of just zeroing out the smallest singular value.

In [ ]:

```

```

3.2 Camera Pose from the Esential Matrix

Using the SVD technique we described in class, recover the two possible rotation matrices R and translation vectors t associated with E. Discuss your results. Is the recovered translation reasonable based on what you see in the images?

In [ ]:

```

```