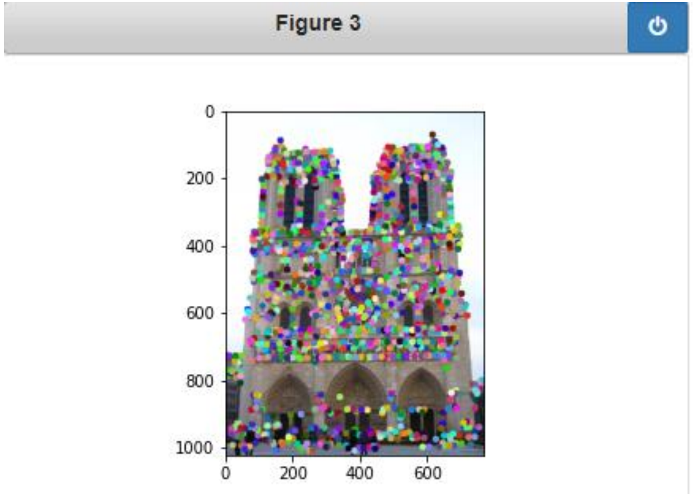
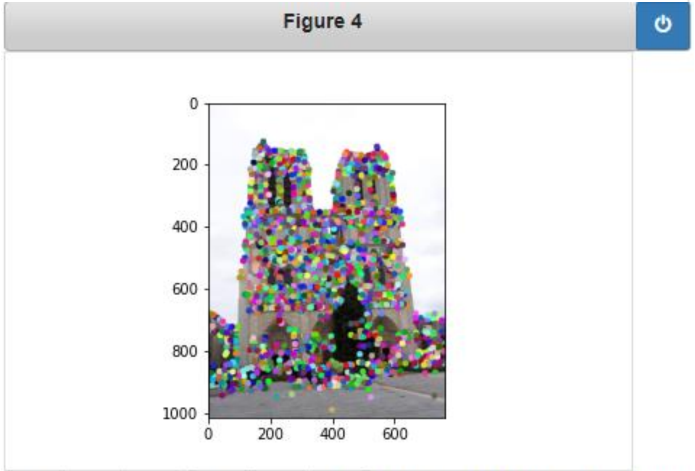
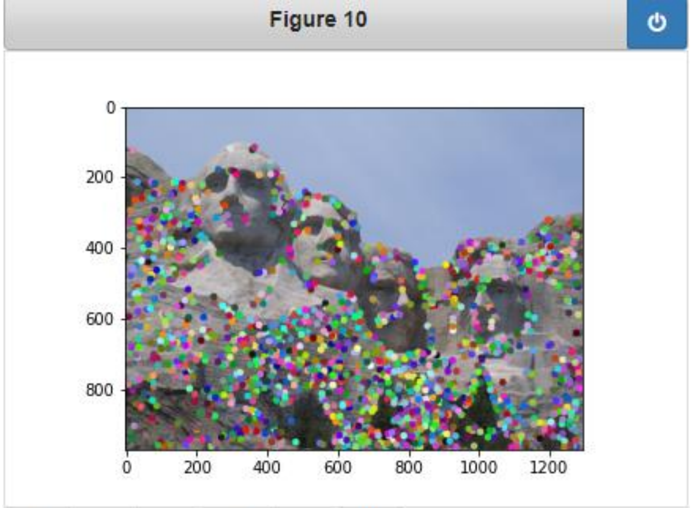
CS 4476 PS2

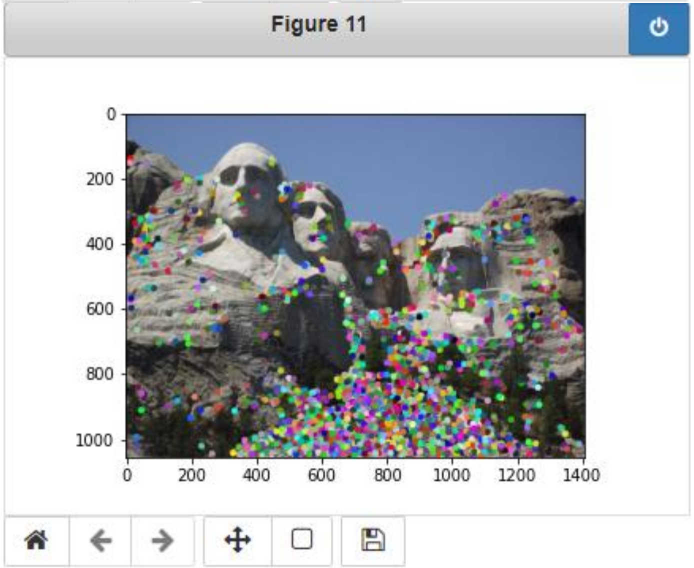
<Younes Djemmal>
<ydjemmal3@gatech.edu>
<ydjemmal3>
<903794219>

1.1: Harris Corner Detector

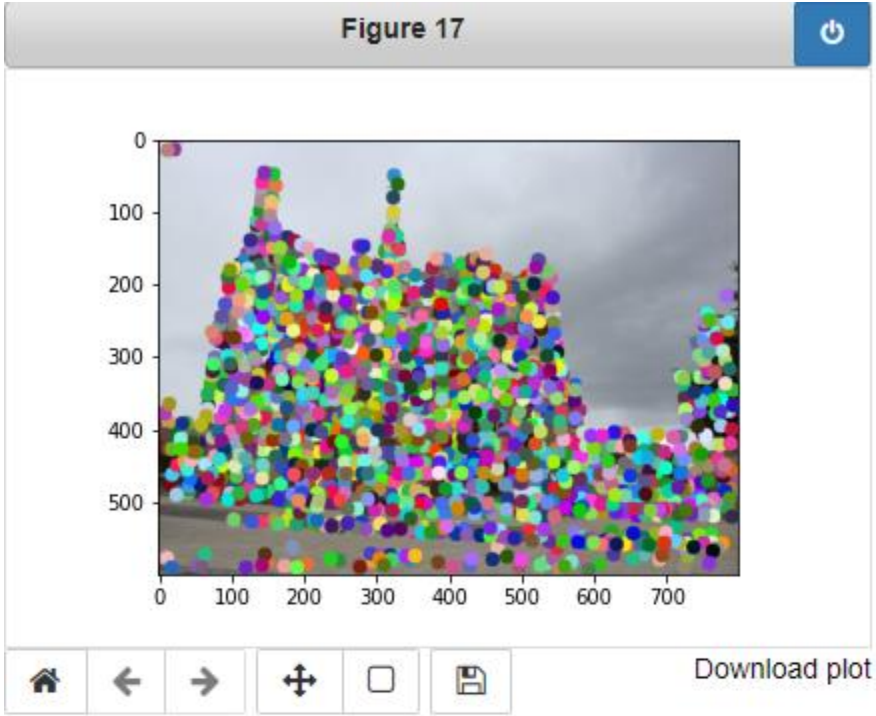


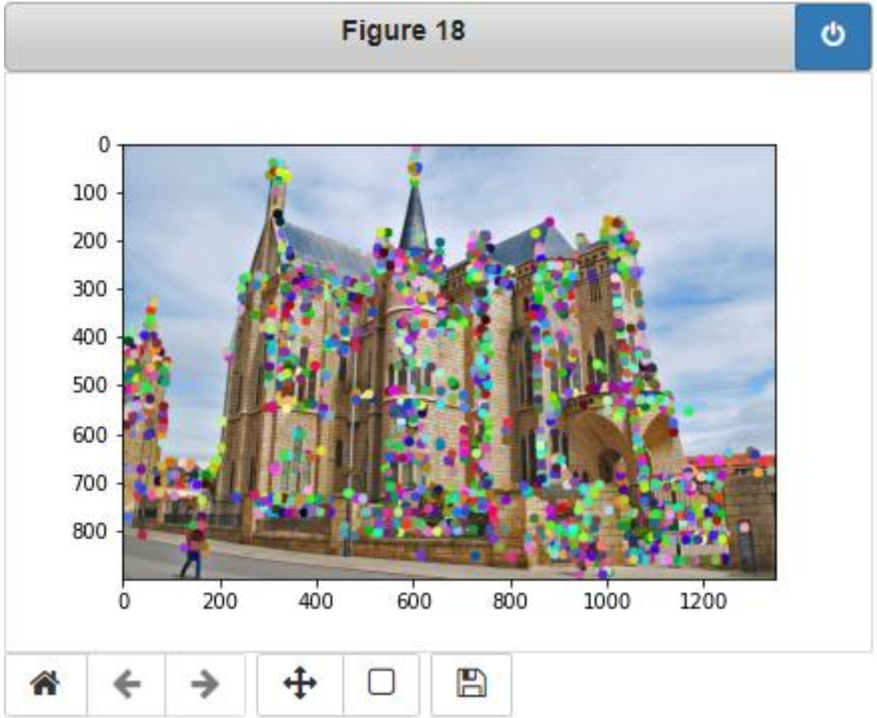






1.1: Harris Corner Detector





1.1: Harris Corner Detector

- Briefly describe how the Harris corner detector works. [1 pt]

The harris corner detector identifies interest points in images with the goal of detecting at least some of the same points in both images (or more images). We first get the first derivatives of our image in both direction x and y. Then we convolve them with a Gaussian for smoothing to get the second moments of our image IxIx, IyIy and IxIy=IyIx. We then use these second moments to calculate the corner response of each point in our image, the score will determine whether or not a point is a distinctive corner or not by taking only the local maxima per window size. We also drop all points that are below a certain threshold, in this case the median, to avoid having too small local maxima.

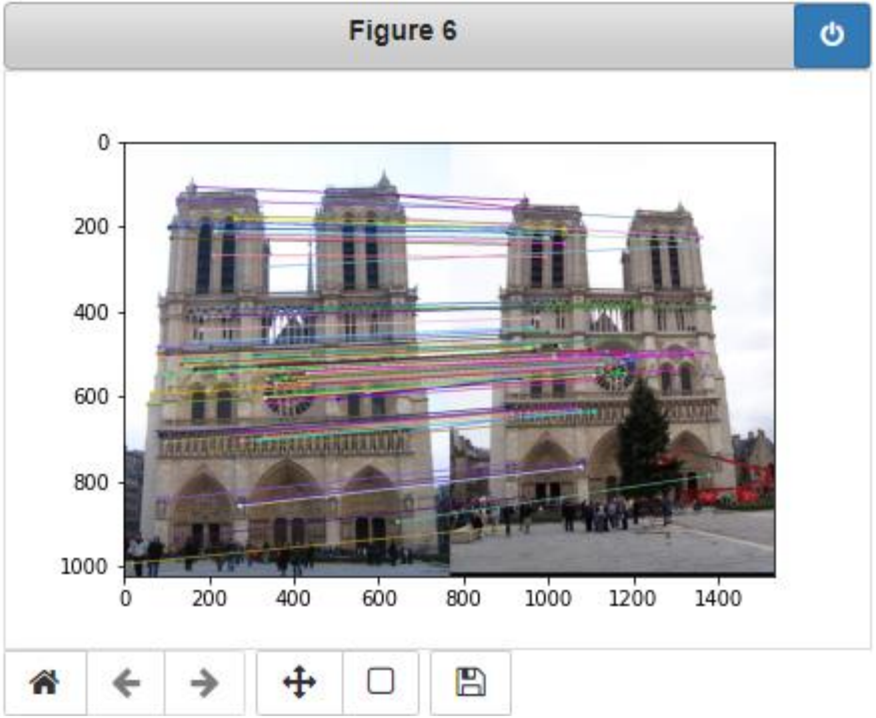
- What does the second_moments() helper function do? [1 pt]

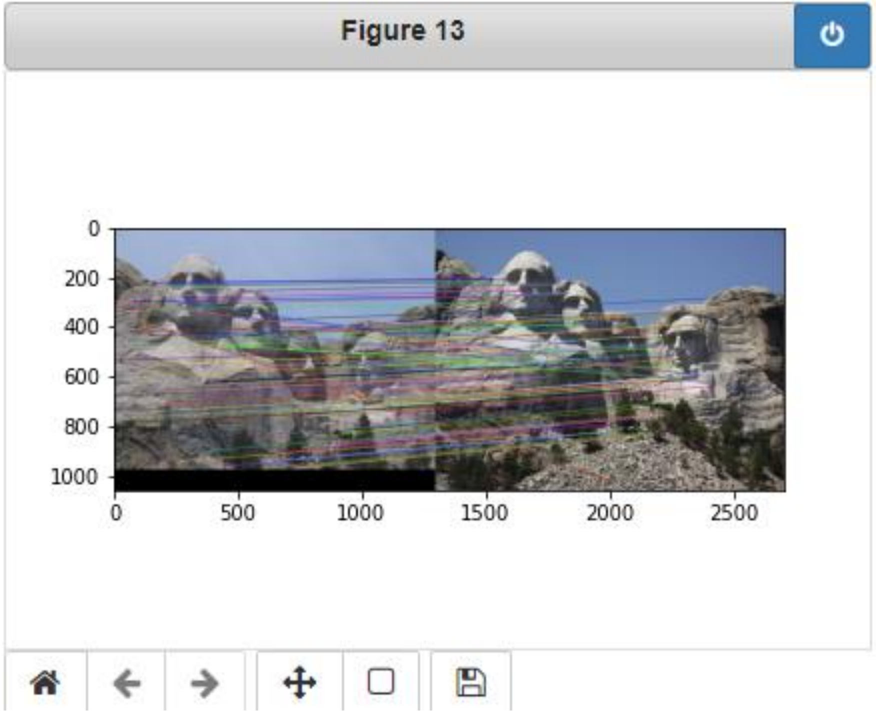
This helper function convolves a smoothing Gaussian filter with the second derivatives of our image which we will use to build our M matrix and calculate the corner response.

- What does the corner_response() helper function do? [1 pt]

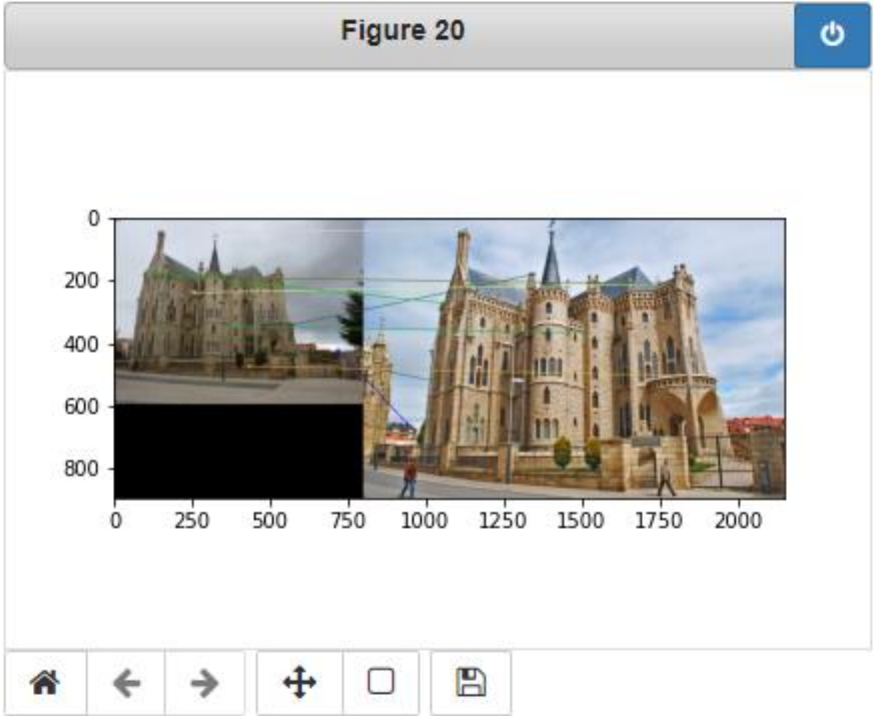
This helper function calculates the corner response for each pixel, this score will determine how likely a point is to be a corner or not. This score is calculated with the eigenvalues of our matrix M : R = det(M) - alpha(trace(M)^2)

1.3: Feature Matching





1.3: Feature Matching

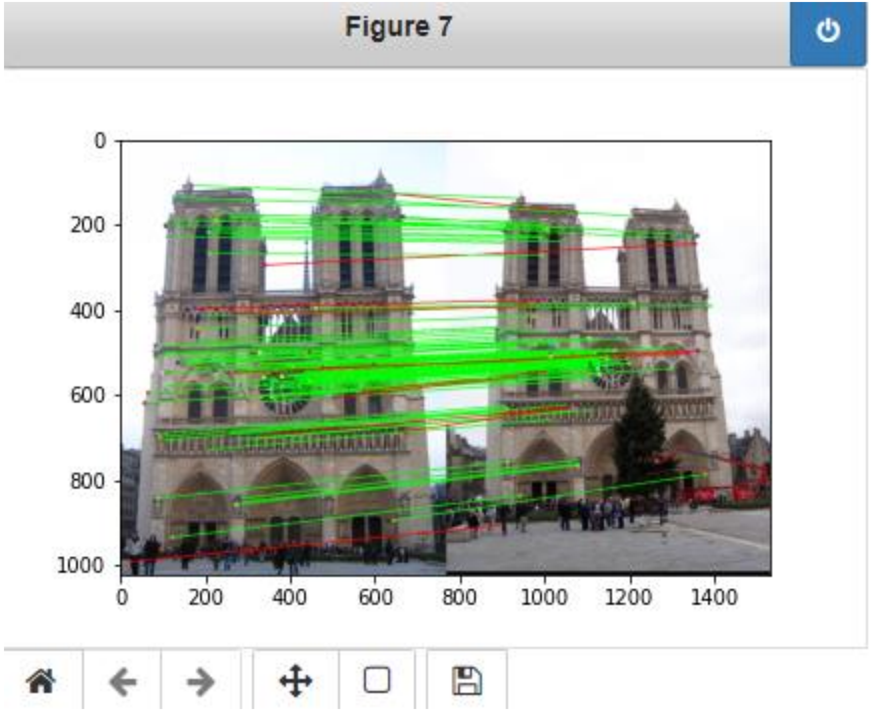


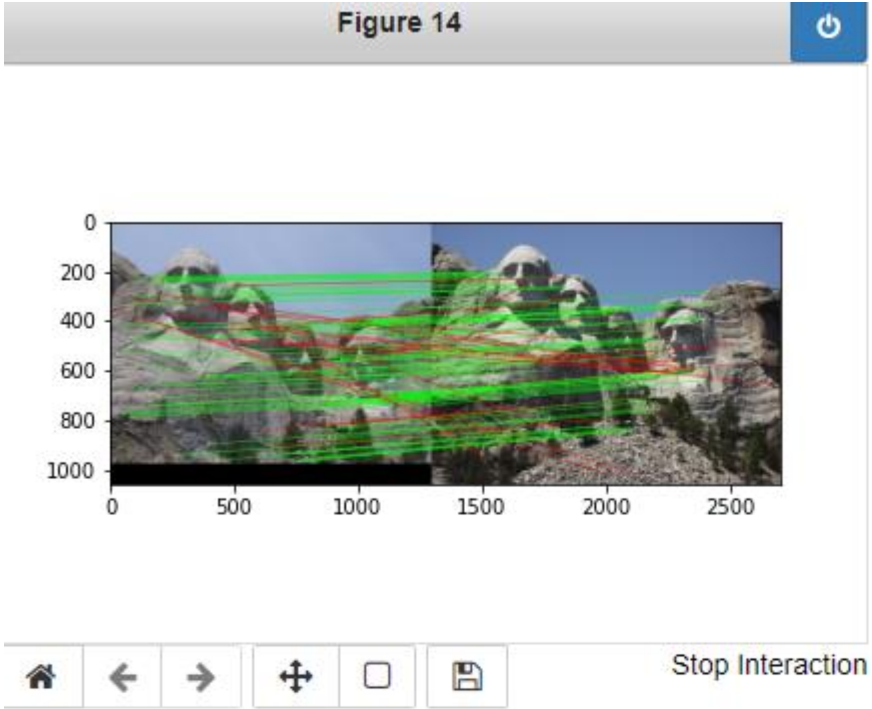
<Describe your implementation of feature matching.> [1.5 pts]

First I computed a matrix dists containing the distances between each feature vector from image 1 and all feature vectors in image 2. Then for each fv in im1 I sorted the distances to get the closest 2 neighbors and calculated the NND score. If the NND score is less than 0,75 we have a match and I added the match to the matches matrix with its corresponding distance in the confidence vector. Otherwise there is an ambiguity and the point is skipped.

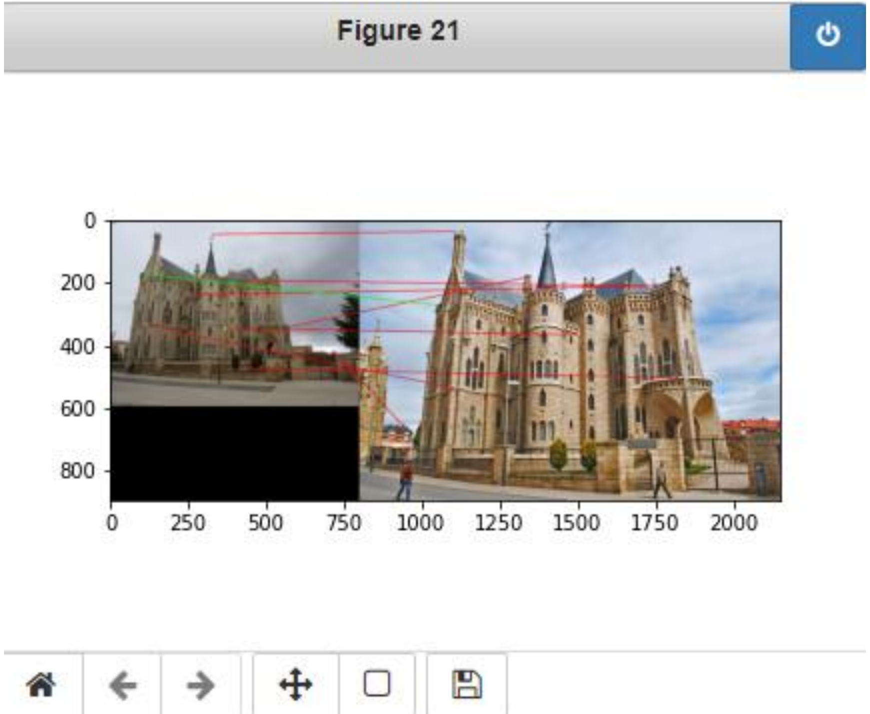
After doing this for all feature vectors, I sorted my matches by confidence to be able to use my 100 best match later on.

Results: Ground Truth Comparison





Results: Ground Truth Comparison



<Insert numerical performances on each image pair here. Also discuss what happens when you change the 4x4 subgrid to 2x2, 5x5, 7x7, 15x15 etc?> [2.5 pts]

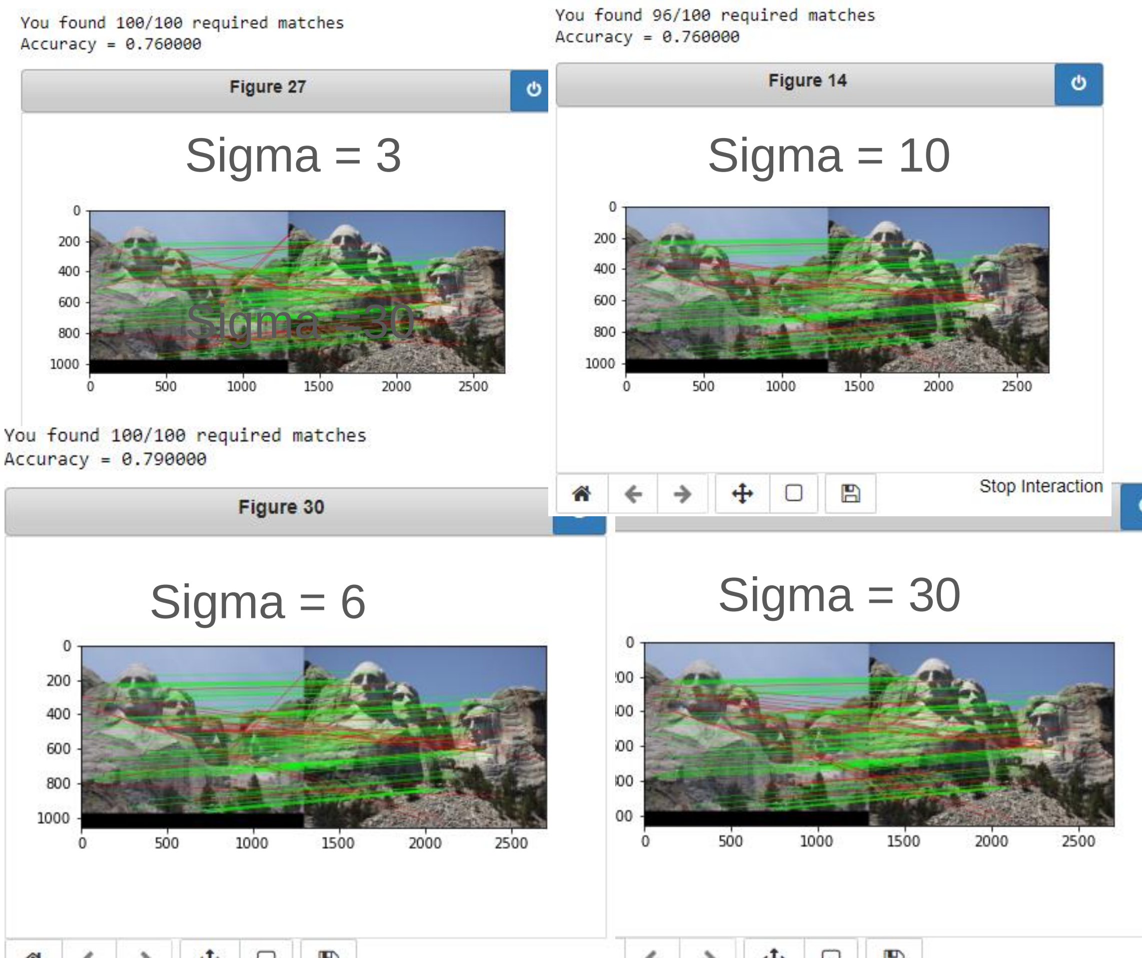
Notre dame: 100/100 required matches, Accuracy= 0,88

Mount Rushmore: 96/100 required matches, Accuracy = 0,76

Episcopal Gaudi: 9/100 required matches. Accuracy = 0,01

Changing the subgrid will change the amount of details you capture in a window. Having a 15x15 subgrid would not be precise will smaller features, and a 2x2 subgrid would miss the bigger ones. It all depends on the images but practically 4x4 works well in most of the case.

1.4(a): Hyperparameter Tuning part 1 [Extra credit]



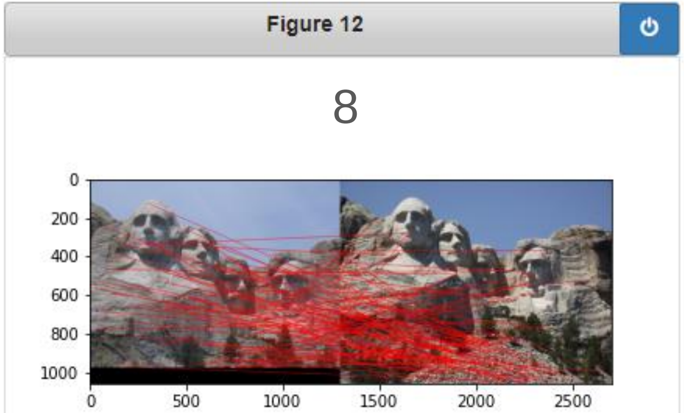
When changing the values for large sigma (>20), why are the accuracies generally the same? [0.5 pts]

The choice of sigma affects the size of the corners we will detect, if we want to detect small corners we would go for smaller sigmas, if we want bigger corners we would go for bigger sigmas. The choice of sigma should go with the choice of the size of the kernel as well, usually 1/6 of our kernel size is the most accurate choice.

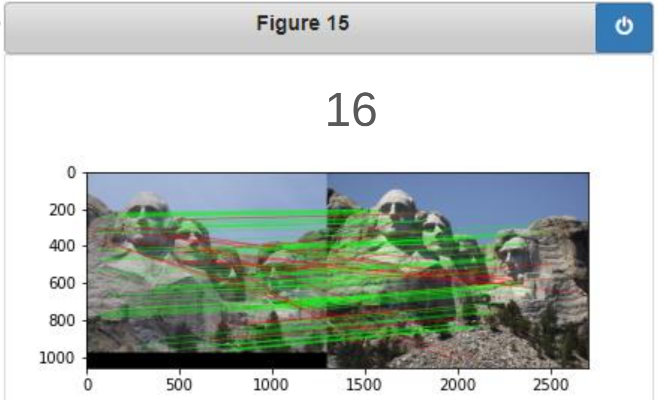
For bigger sigmas, the accuracy does not change much because all of them would detect the bug corners and there are no big differences in big corners. But for smaller sigmas we have much more fluctuations on the corners detected because a small change will change the set of corners detected.

1.4(a): Hyperparameter Tuning part 2 [Extra credit]

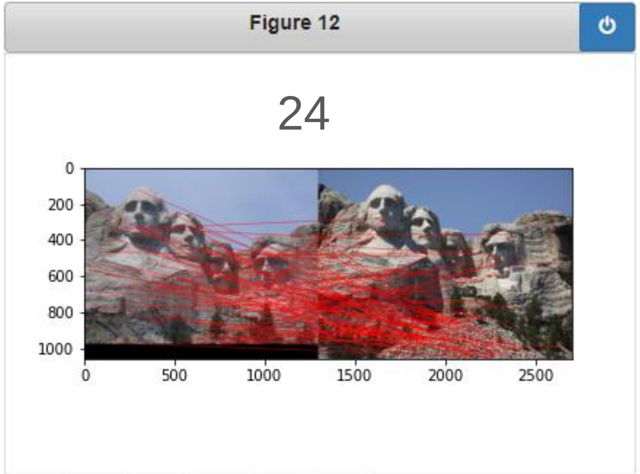
```
You found 100/100 required matches
Accuracy = 0.000000
```

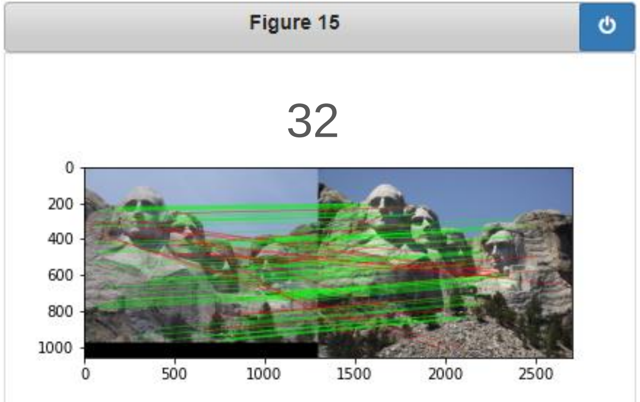



```
You found 96/100 required matches
Accuracy = 0.760000
```




```
You found 100/100 required matches
Accuracy = 0.000000
```





What is the significance of changing the feature width in SIFT? [0.5 pts]

The feature width determines the precision of your matching depending on the types of objects you want to match. If you want to match smaller features then you would go for a smaller feature width to be more precise, but you will lose any big details in the matching. And the other way around works as well.

1.4(c): Accelerated Matching [Extra credit]

<Insert Runtime/Accuracy of your faster matching implementation. What did you try and why is it faster?> [2 pts]

Runtime: `0.4491946697235107 4` seconds and accuracy was `0.760000`.

For my implementation I used the sklearn.neighbors.NearestNeighbors package with the algorithm ‘kd_tree’.

It’s faster because the Kd_tree algorithm it runs in O[N log(N)] instead of my previous implementation that ran in O[NxM].