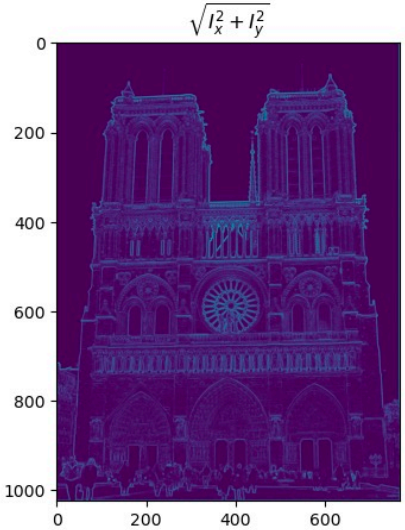
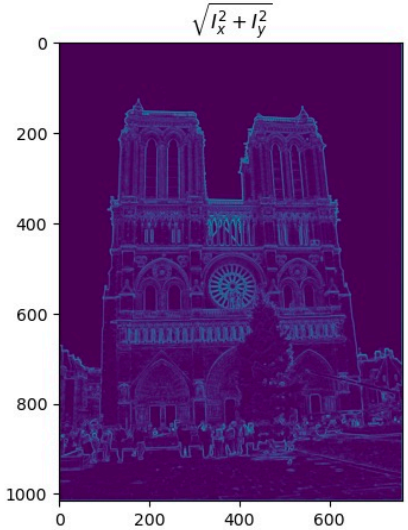
CS 4476 Project 2

Nakul Kuttua
nkuttua3@gatech.edu
nkuttua3
903520821

Part 1: Harris corner detector

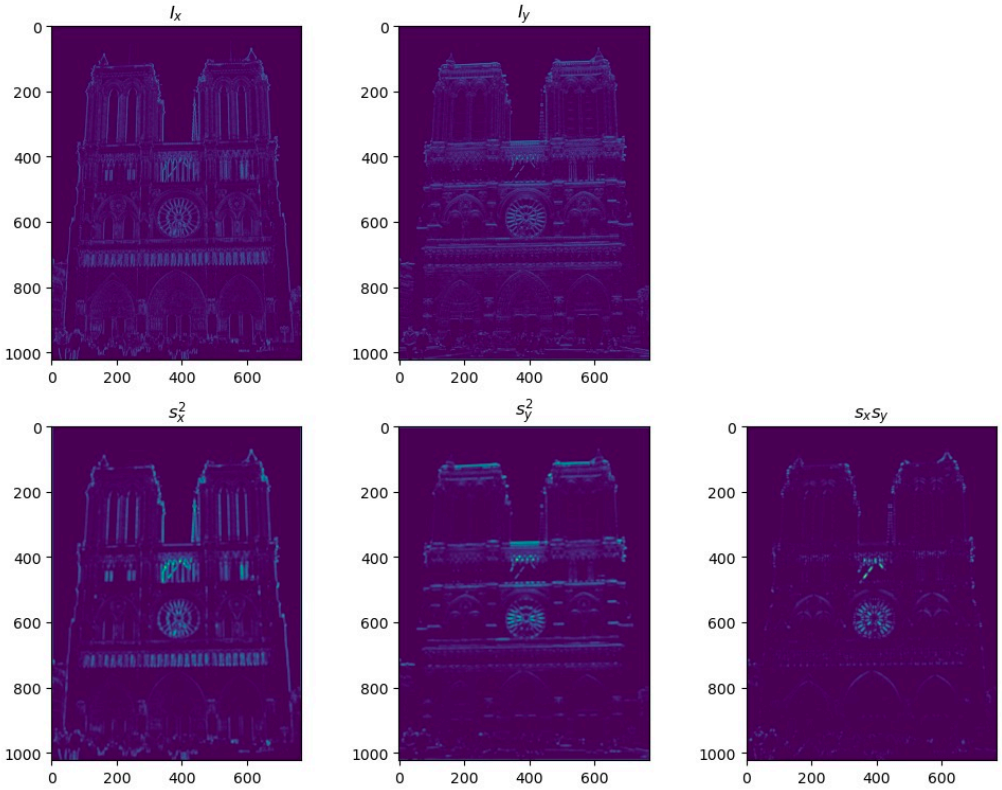




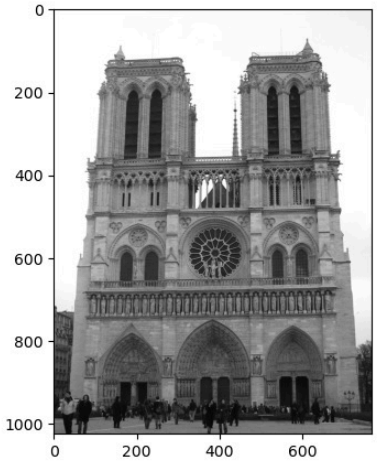
[Which areas have highest magnitude? Why?

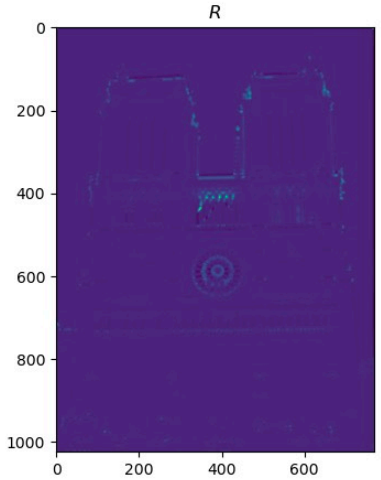
The circular window in the center, the area right above it, and the row of statues below it. At those areas, the gradient is relatively higher and so the magnitude is higher as well.

Part 1: Harris corner detector



Part 1: Harris corner detector

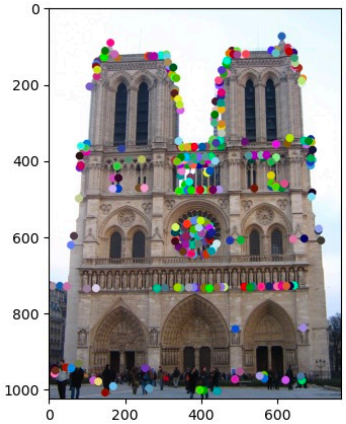


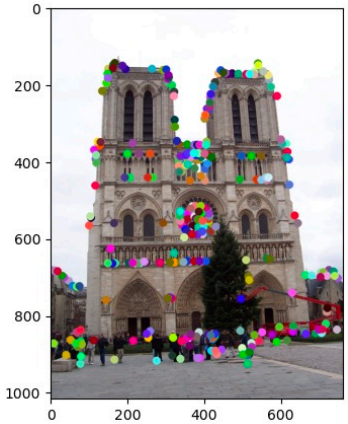


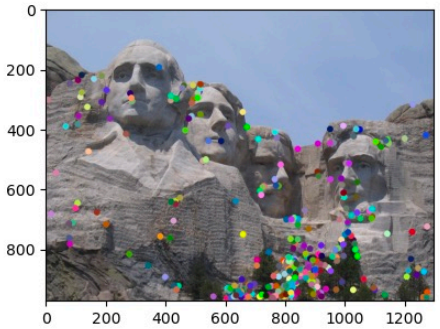
[Are gradient features invariant to both additive shifts (brightness) and multiplicative gain (contrast)? Why or why not? See Szeliski Figure 3.2]

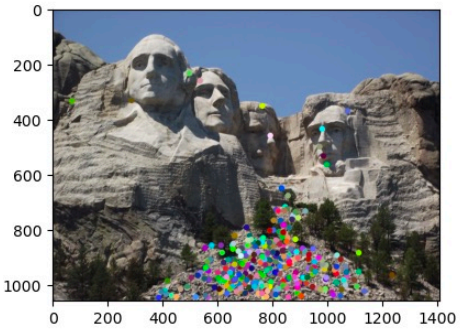
Yes, gradient features are invariant to additive shifts (brightness) but they are not invariant to multiplicative gain (contrast). The reason why for additive shifts is that if we add an equivalent shift to ALL the intensities, it doesn't change anything since gradients are based on pixel intensity differences. The reason why not for multiplicative gain is multiplying by a constant changes all the intensities.

Part 1: Harris corner detector

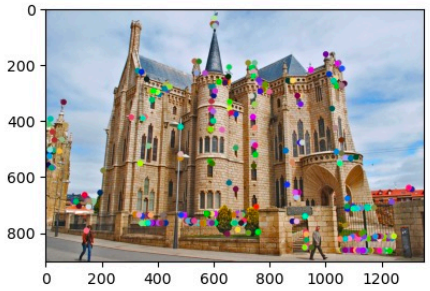


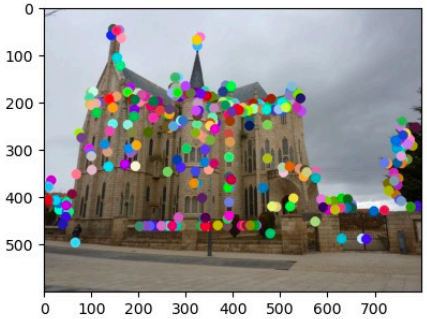






Part 1: Harris corner detector





[What are the advantages and disadvantages of using maxpooling for non-maximum suppression (NMS)?]

Advantages: It is very efficient and can calculate the local maxima quickly. It's also relatively simple.

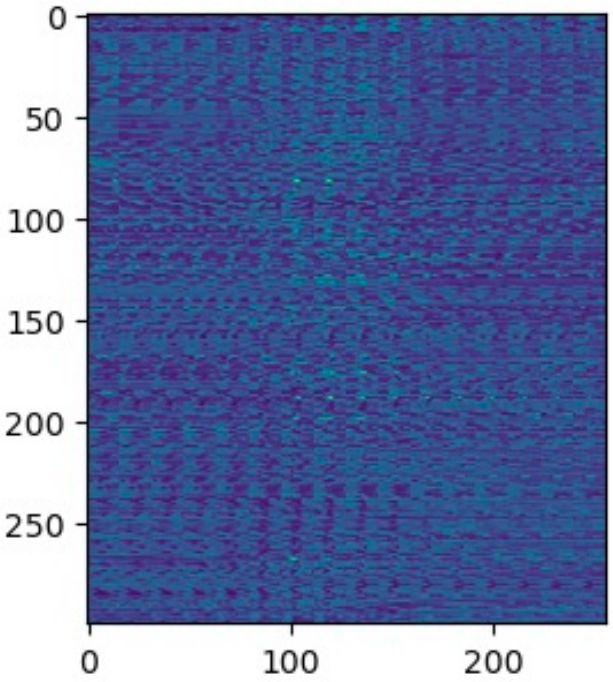
Disadvantages: Because we are using a limited window size, it doesn't capture all the local maxima outside, potentially losing valuable maxima data. Also max pooling is not very fine tuned for specific control, so it's not always easy to detect small objects.

Part 1: Harris corner detector

[What is your intuition behind what makes the Harris corner detector effective?]

My intuition is that the Harris Corner Detector uses repeat image features to detect strong gradient points (namely corners). It does this by scanning for high gradients in an image and comparing those points to what's around it to determine whether or not it's a corner.

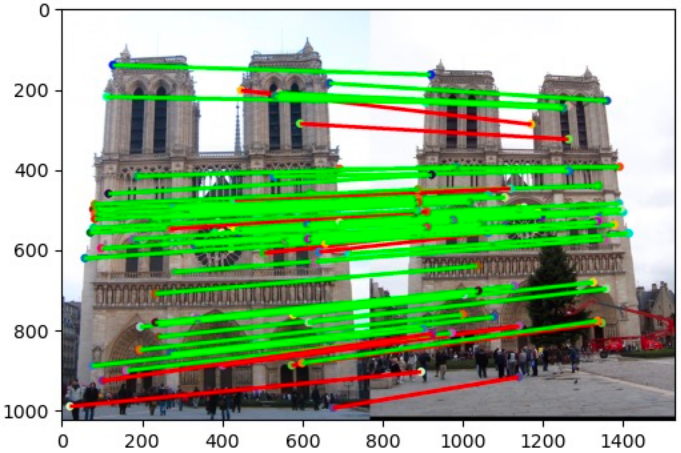
Part 2: Normalized patch feature descriptor



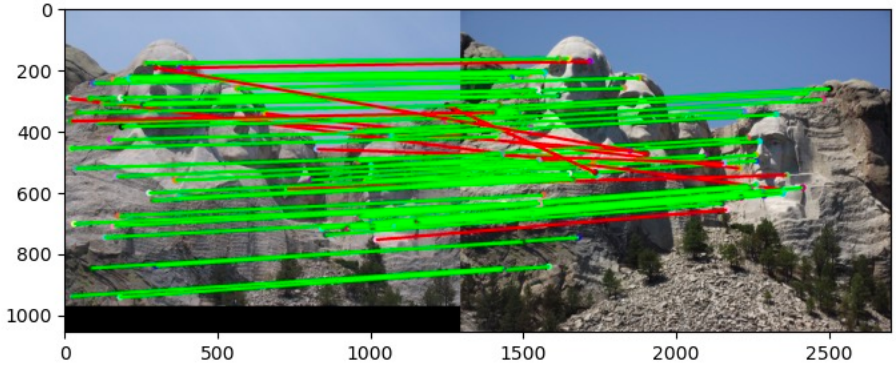
[Why aren't normalized patches a very good descriptor?]

Normalized patches don’t really make a good descriptor because they contain normalized pixels rather than the magnitude and orientations of the image itself. It just wouldn’t give conclusive results since everything is too closely similar.

Part 3: Feature matching

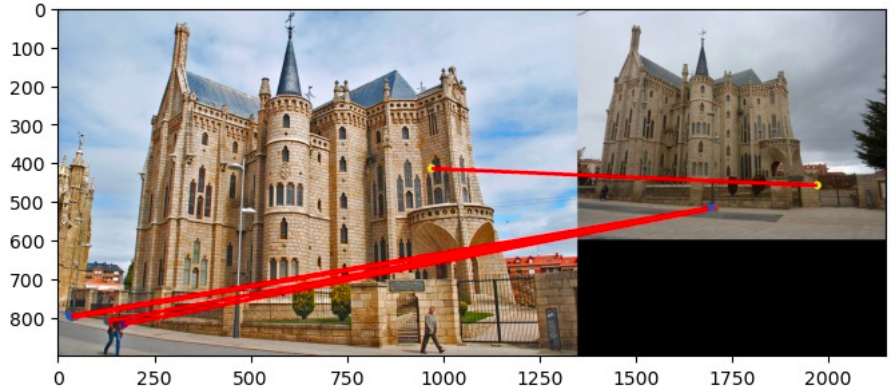


matches (out of 100): [65]
Accuracy: [54%]



matches: [63]
Accuracy: [53%]

Part 3: Feature matching

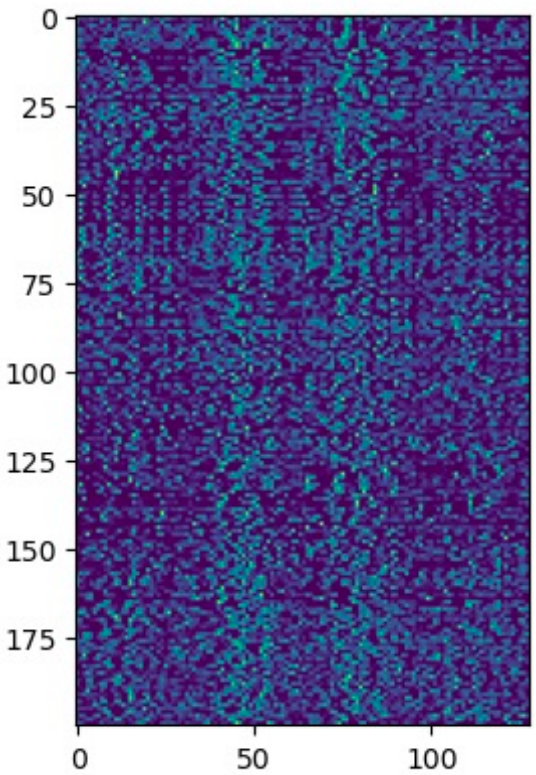


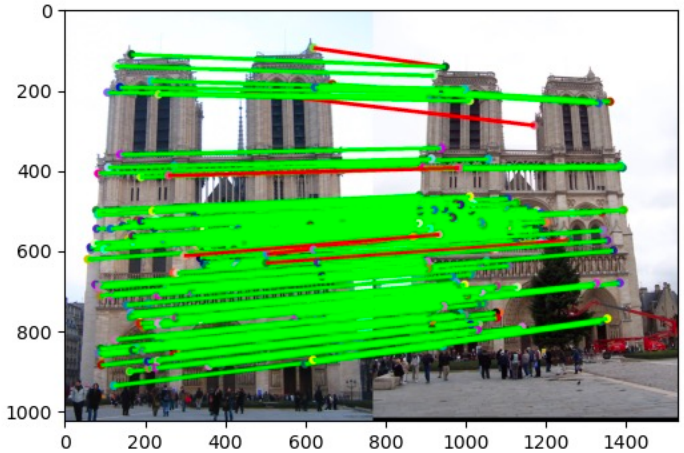
matches: [5]
Accuracy: [0%]

[Describe your implementation of feature matching here]

First, we get the distances between the two given feature sets. Also define a ratio threshold for comparing. Then, loop through the size of a feature set (doesn't matter which since both are same size) and sorted the distances and keep track of the indices at that index. Calculate the ratio of the distances, and if it's less than the ratio threshold we specified earlier, add it to the matches and confidences

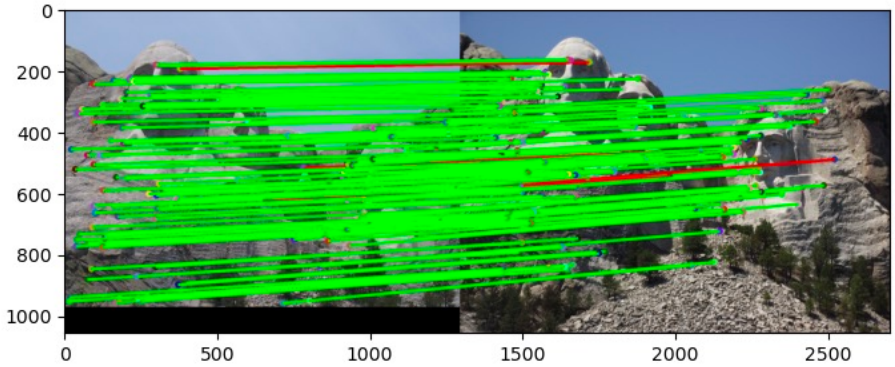
Part 4: SIFT feature descriptor





matches (out of 100): 124
Accuracy: 95.1613%

Part 4: SIFT feature descriptor



[insert visualization of matches for Gaudiimage pair from proj2.ipynb here]

matches: 113
Accuracy: 96.4602%

matches: [insert # matches here]
Accuracy: [insert accuracy here]

Part 4: SIFT feature descriptor

[Describe your implementation of SIFT feature descriptors here]
First define our fvs with the appropriate dimensions (k, 128). Then, compute the image gradients of the given black and white image, as well as the magnitudes and orientations. Then for each point in k, get the feature vector and ravel it so it can be stored into fvs.

[Why are SIFT features better descriptors than the normalized patches?]

The reason why they are better is simply because they use the image gradients as part of the calculations instead of the pixel values themselves.

Part 4: SIFT feature descriptor

[Why does our SIFT implementation perform worse on the given Gaudi image pair than the Notre Dame image and Mt. Rushmore pairs?]

The reason why it performs worse is because the images are noticeably different in color, as opposed to the other two which are much more similar. The colors are starkly different, and the difference in angles at which the pictures are taken is just slightly greater than the first two images.

Part 5: SIFT Descriptor Exploration

Describe the effects of changing window size around features. Did different values have better performance?

Part 5: SIFT Descriptor Exploration

Describe the effects of changing the number of local cells in a window around a feature? Did different values have better performance?

Part 5: SIFT Descriptor Exploration

Describe the effects of changing number of orientations (bins) per histogram. Did different values have better performance?

Part 5: SIFT Descriptor Exploration

[insert visualization of matches for your image pair from proj2.ipynb here]

Part 5: SIFT Descriptor Exploration

[Discuss why you think your SIFT pipeline worked well or poorly for the given building. Are there any characteristics that make it difficult to correctly match features]?

Conclusion

[Why aren't our version of SIFT features rotation- or scale-invariant? What would you have to do to make them so?]

The reason why is because that we do adjust for local maxima in the histograms for rotation invariancy and we did not use any Gauss applications to mark constant feature points as they get scaled. In order to make our SIFT features rotaion and scale invariant, we would have to add local maxima points into our histograms and use Gaussians to keep track of noticeable feature points as the image is scaled.