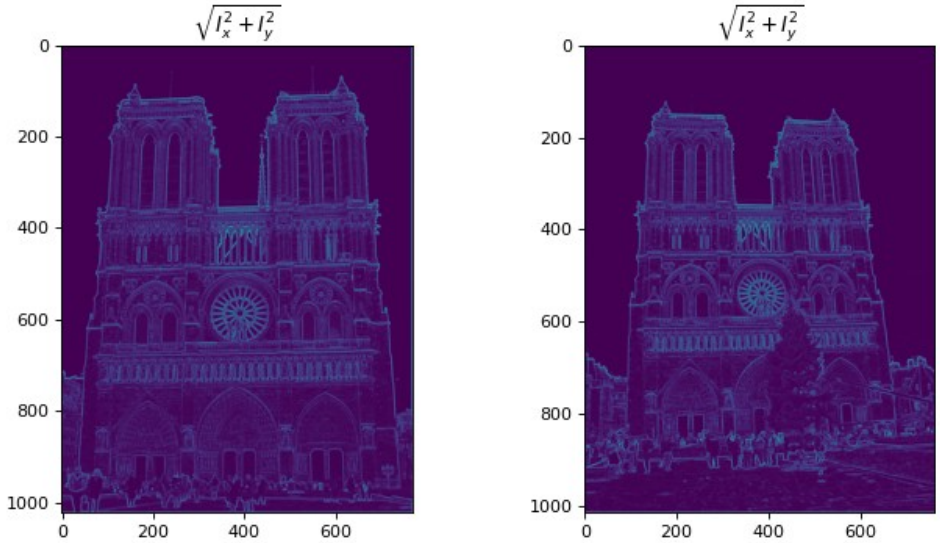
CS 5330 Programming Assignment 2

[Aditya Appana]
[appana.a@northeastern.edu]
[appana.a]
[001006587]

Part 1: Harris corner detector

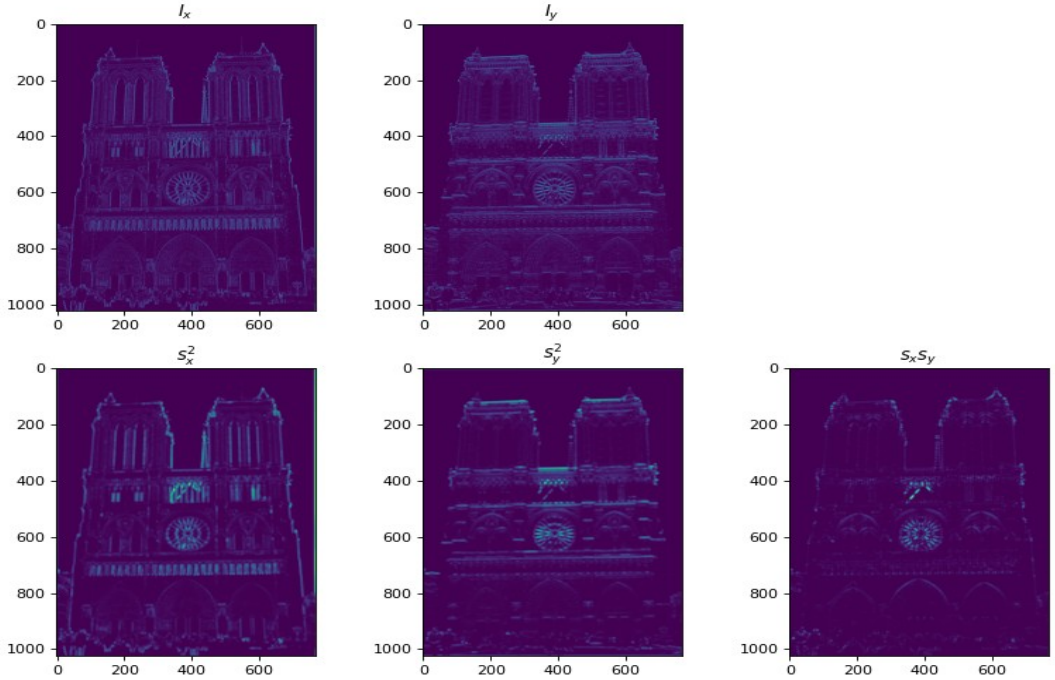
[insert visualization of \sqrt(I_x^2 + I_y^2) for Notre Dame image pair from pa2.ipynb here]



[Which areas have highest magnitude? Why? The brightest areas in the image have highest magnitude. White/bright regions means a high change in the initial image while black region means no change at all.]

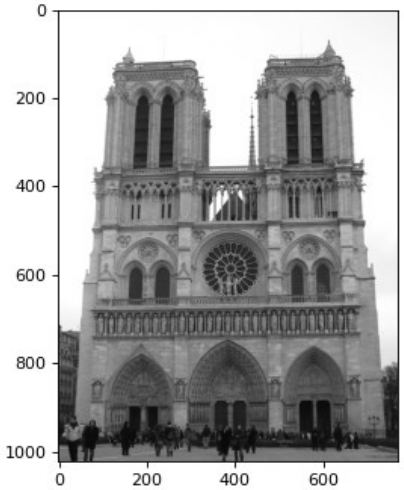
Part 1: Harris corner detector

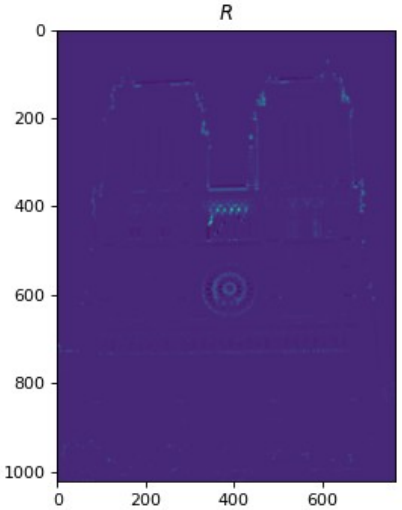
[insert visualization of I_x, I_y, s_x^2, s_y^2, $s_x s_y$ for Notre Dame image pair from pa2.ipynb here]



Part 1: Harris corner detector

[insert visualization of corner response map of Notre Dame image from pa2.ipynb here]



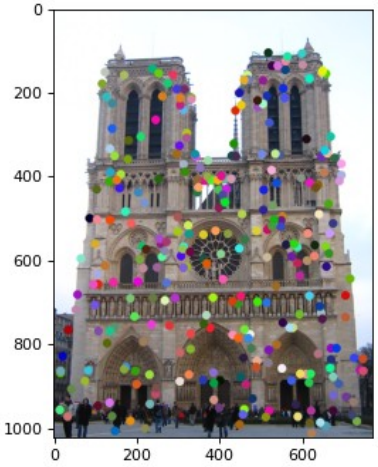


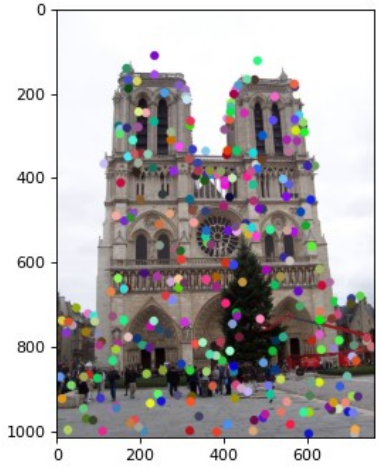
[Are gradient features invariant to both additive shifts (brightness) and multiplicative gain (contrast)?
Why or why not? See Szeliski Figure 3.2

Generally true, the gradiant features are invariant to brightness and contrast up to an extent. A gradiant at a pixel is a rate of change of a pixel intensity in that image. So the rate of change of pixels intensity is not affected by brightness or contrast when they are uniformly varied across the image.
]

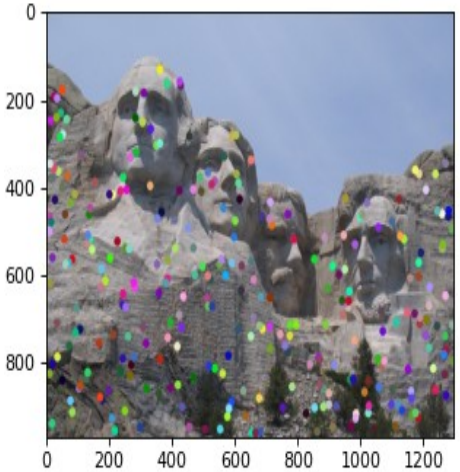
Part 1: Harris corner detector

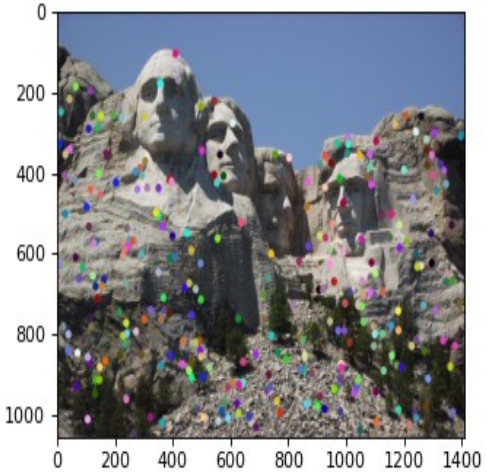
[insert visualization of Notre Dame interest points from pa2.ipynb here]





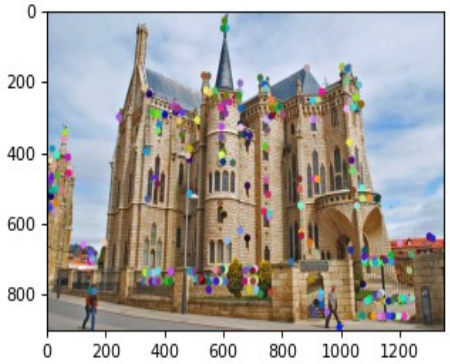
[insert visualization of Mt. Rushmore interest points from pa2.ipynb here]

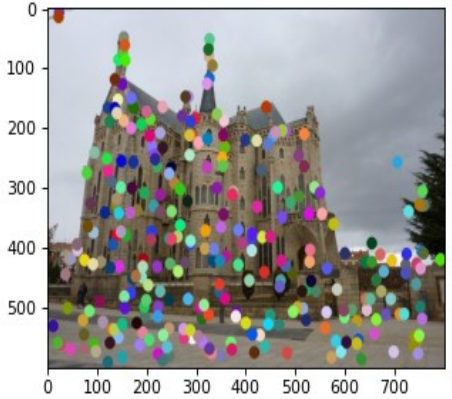




Part 1: Harris corner detector

[insert visualization of Gaudi interest points from pa2.ipynb here]





[What are the advantages and disadvantages of using maxpooling for non-maximum suppression (NMS)?
In this implementation, when we use non-maximum suppression NMS, it replaces all the values in the neighborhood with the maximum value in that region. It is helpful because it suppresses all the values around a corner, and prevents us from interpreting a single corner as multiple corners.
And one disadvantage could be it might be slow on large images and small filter size as it has it compare all the values in the range of filter dimensions over almost all the pixels.
]

Part 1: Harris corner detector

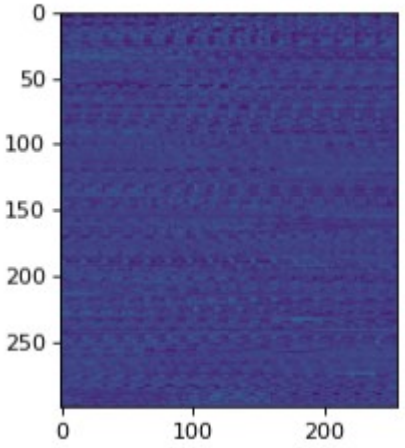
[What is your intuition behind what makes the Harris corner detector effective?

The Harris corner detector is effective as it detect points based on the intensity variation in a local neighborhood. A small window region around the feature should show a large intensity change when compared with windows shifted in any other direction to be considered as a corner point.

]

Part 2: Normalized patch feature descriptor

[insert visualization of normalized patch descriptor from pa2.ipynb here]

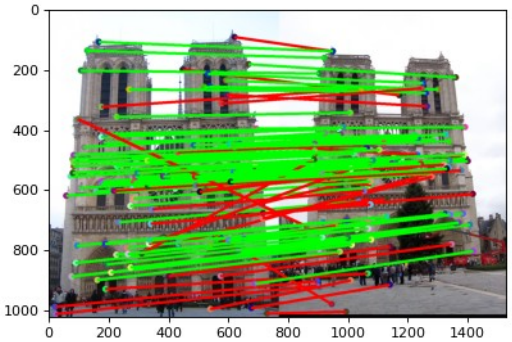


[Why aren't normalized patches a very good descriptor?

The normalized patches are only dependant on the pixel Intensity around the interest point. These are highly sensitive to illuminance, translation, rotation or scaling. Hence, they are not very good descriptors.
]

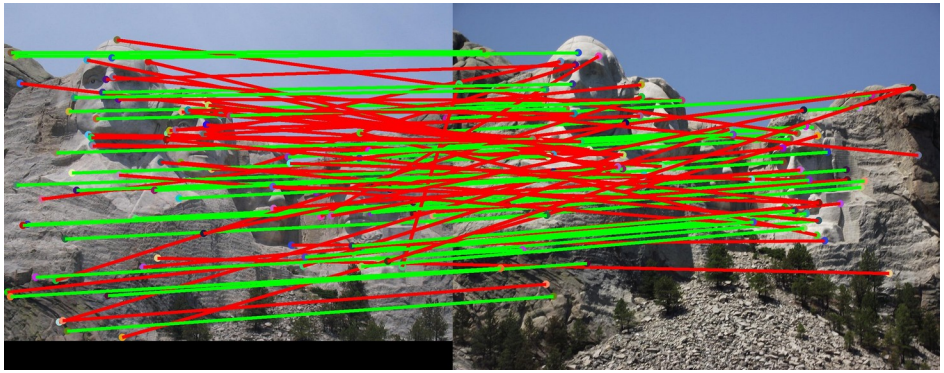
Part 3: Feature matching

[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from pa2.ipynb here]



matches (out of 100): [116]
Accuracy: [0.689655]

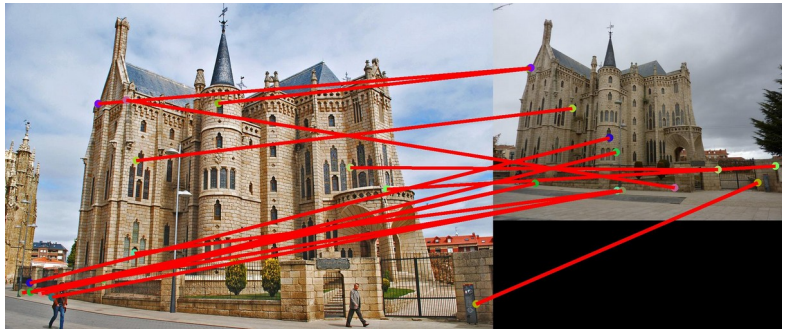
[insert visualization of matches for Mt. Rushmore image pair from pa2.ipynb here]



matches: [94]
Accuracy: [0.410000]

Part 3: Feature matching

[insert visualization of matches for Gaudi image pair from pa2.ipynb here]



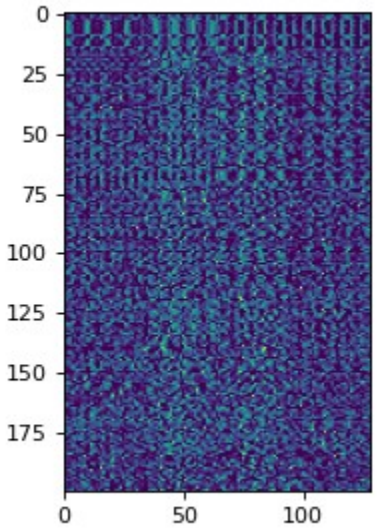
matches: [14/100]
Accuracy: [0]

[Describe your implementation of feature matching here
Initially, the distances between 2 features are computed.
Then, a ratio test is performed, which returns an array of matches(pair of features) and array of confidence of each match. As per the text book took the nndr threshold above 0.8 to consider the feature pair as a match.
This way feature matching is achieved.

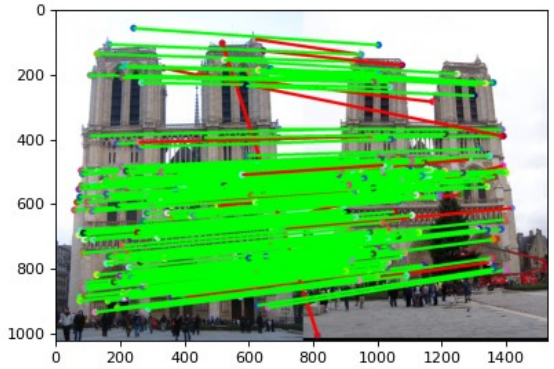
P.S. In the gaudi image, though I can see alot of matching features, all were marked red and accuracy was returned 0 by the Tests.
]

Part 4: SIFT feature descriptor

[insert visualization of SIFT feature descriptor from pa2.ipynb here]



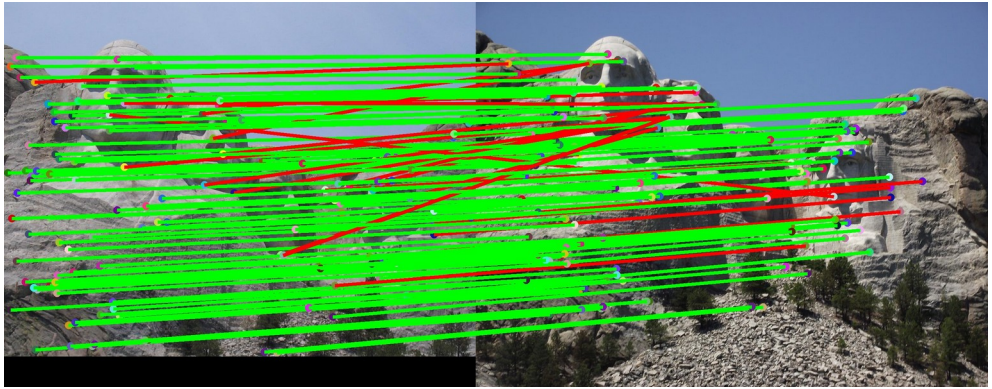
[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from pa2.ipynb here]



matches (out of 100): [180]
Accuracy: [0.905556]

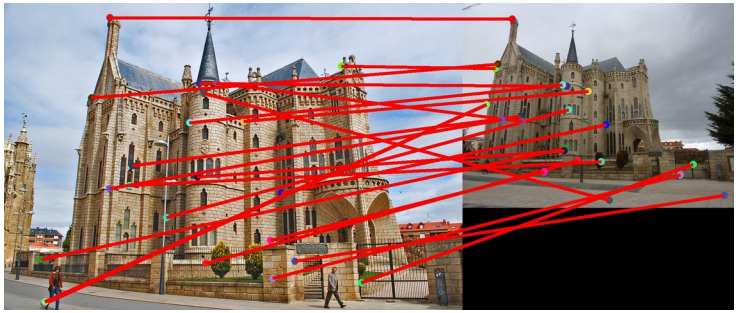
Part 4: SIFT feature descriptor

[insert visualization of matches for Mt. Rushmore image pair from pa2.ipynb here]



matches: [124]
Accuracy: [0.806452]

[insert visualization of matches for Gaudiimage pair from pa2.ipynb here]



matches: [22]
Accuracy: [0.0]

Part 4: SIFT feature descriptor

[Describe your implementation of SIFT feature descriptors here:
My Sift feature descriptor function takes in an image, and the X and Y coordinates of all the interest points, and returns an array of feature vectors. To get the feature vectors, I'm first computing the x and y gradients of the image, then calculating the magnitutes and orientations of the image. For every point in the interest point, I'm calculating a feature vector, by inserting a patch of size of feature-width over it and calculating the histogram. Each of this feature is then normalised,and raised to power of(1/2) and returned]

[Why are SIFT features better descriptors than the normalized patches?
Sift feature descriptors are histograms in a local patch around interest points. They capture the information around an interest point in a histogram. However a simple normalized patch is truly based only on Image pixel intensities around an interest point. It doesnt perform well even with a slight rotation or scaling of the image. Whereas, SIFT descriptors generally perform well even on different illumination, rotation,translation or scaling of the image.
]

Conclusion

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

Rotation invariance can be achieved by assigning a dominant orientation for each keypoint, and by doing all further calculations relative to this dominant orientation. This effectively cancels out the effect of rotation, making it rotation invariant.

Scale Invariance can be achieved by initially extracting features at a variety of scales, by performing the same operations at multiple resolutions in a pyramid of Image and then matching features at the same level.(works only for small scale changes). And a much more efficient way is to extract the features that are stable in both location and scale.This way scale invariance can also be achieved.]

Extra Credit

My default implementation of get_sift_features is a vectorized version. It runs in 3.52sec with an Accuracy of 90.55%

The feature matching uses numpy broadcasting technique to return the matches and the confidences efficiently.

```
Your vectorized SIFT implementation takes 3.52 seconds to run on Notre Dame
180 matches from 2462 corners
You found 180/100 required matches
Accuracy = 0.905556
Your vectorized feature matching pipeline achieved 90.56% accuracy to run on Notre Dame
"Correct"
```