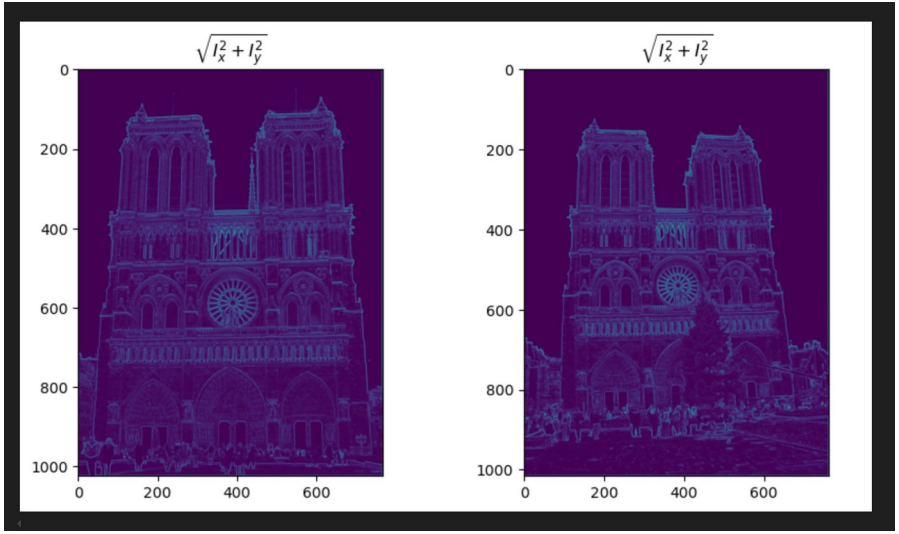
# CS 6476 Project 2

[Ruize Cao]
[rcao73@gatech.edu]
[rcao73]
[904012084]

# Part 1: Harris corner detector

[insert visualization of \sqrt($I_x^2 + I_y^2$) for Notre Dame image pair from proj2.ipynb here]

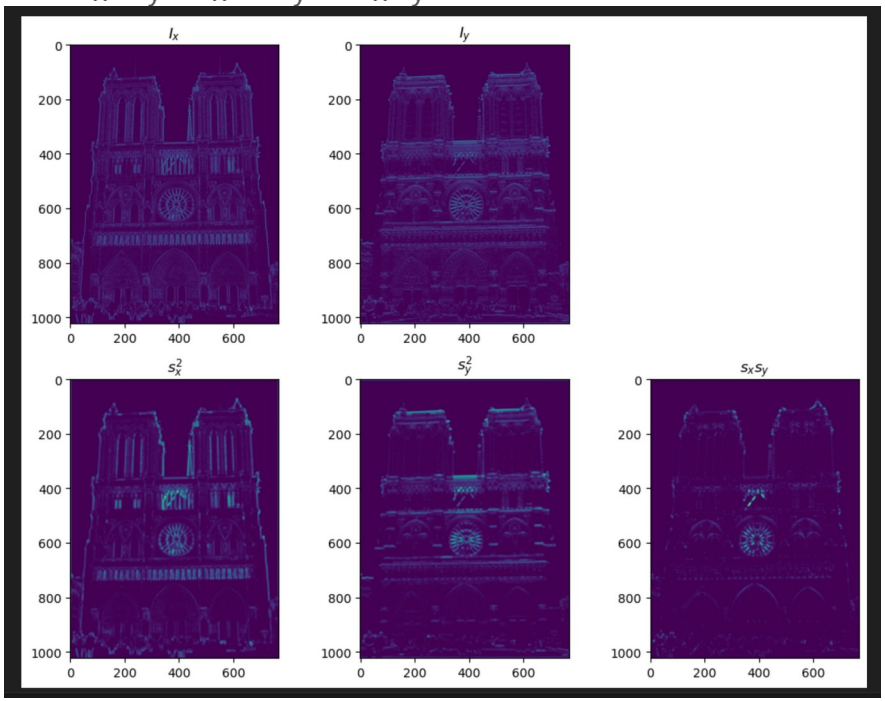



[Which areas have highest magnitude? Why?
At the corner of the object.
Here is a significant change in intensity in both the xx and yy directions. This means that both IxIx and IyIy will be large, resulting in a large value of Ix2+Iy2Ix2+Iy2

.

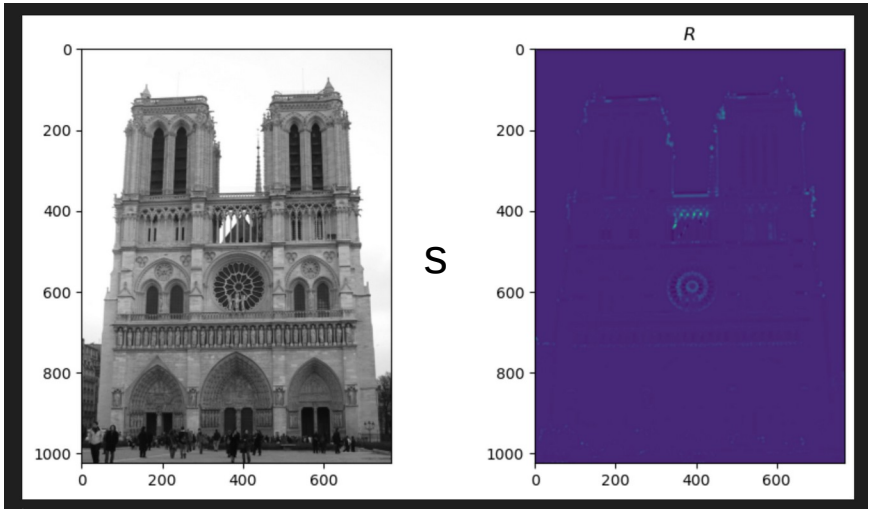
# Part 1: Harris corner detector

[insert visualization of $I_x$, $I_y$, $s_x^2$, $s_y^2$, $s_xs_y$ for Notre Dame image pair from proj2.ipynb here]



# Part 1: Harris corner detector

[insert visualization of corner response map of Notre Dame image from proj2.ipynb here]

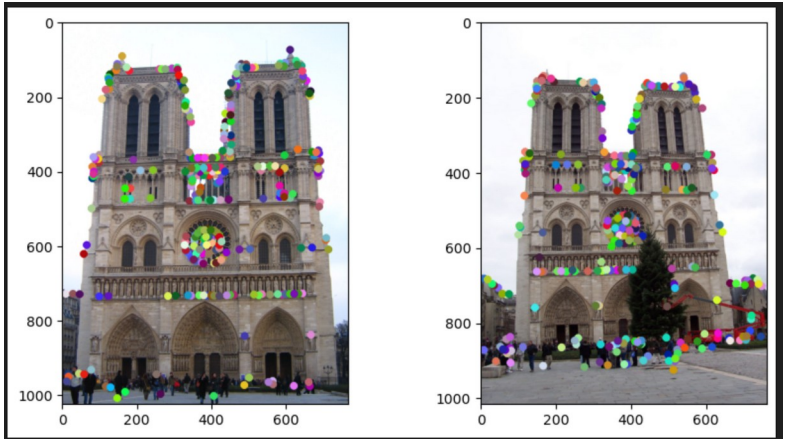



[Are gradient features invariant to both additive shifts (brightness) and multiplicative gain (contrast)? Why or why not? See Szeliski Figure 3.2]

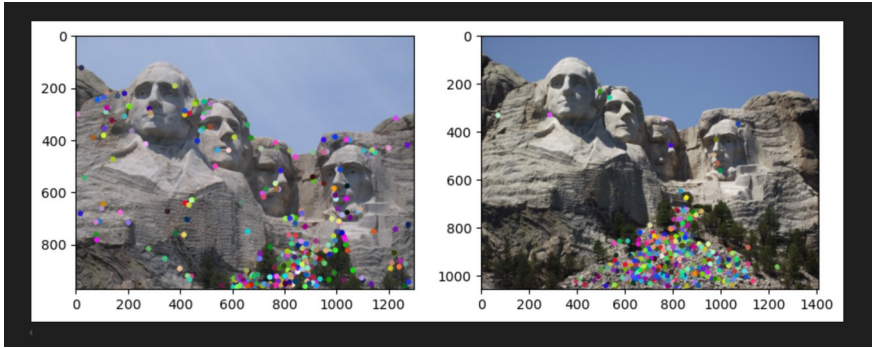
The gradient features are invariant to shifts, however, are variant to multiplication gain. Since gradient calculate the how much intensive changes along certain direction, additive shifts, will not alter the changes along a direction. However, multiplication gain will scale the changes in certain direction, hence, scale the gradients

# Part 1: Harris corner detector

[insert visualization of Notre Dame interest points from proj2.ipynb here]

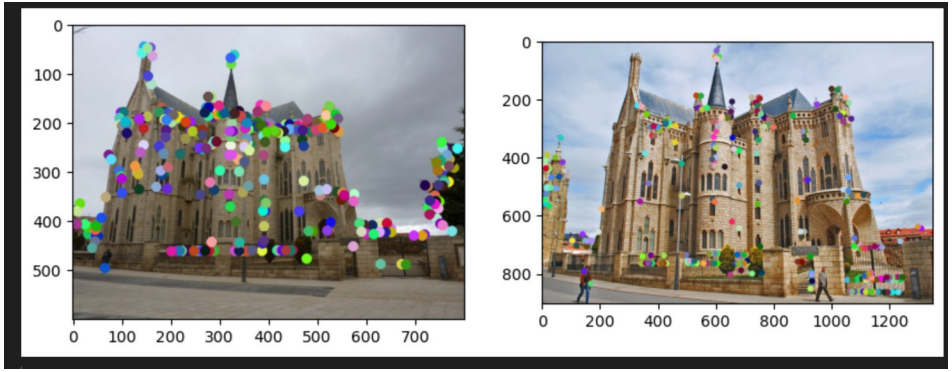



[insert visualization of Mt. Rushmore interest points from proj2.ipynb here]



# Part 1: Harris corner detector

[insert visualization of Gaudi interest points from proj2.ipynb here]




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

Advantages:

1. It can select the strongest response in each neighborhood, which is the essence of NMS. It suppresses all non-maximum values within the pooling window, leaving only the most prominent points.
2. Max pooling naturally selects the strongest response in each neighborhood, which is the essence of NMS. It suppresses all non-maximum values within the pooling window, leaving only the most prominent points.

Disadvantages:

1.Max pooling can sometimes be too coarse. It may discard valuable nearby maxima if they are close to each other but still represent distinct features. This can lead to missed interest points in areas where multiple important points exist close together (such as highly textured regions).

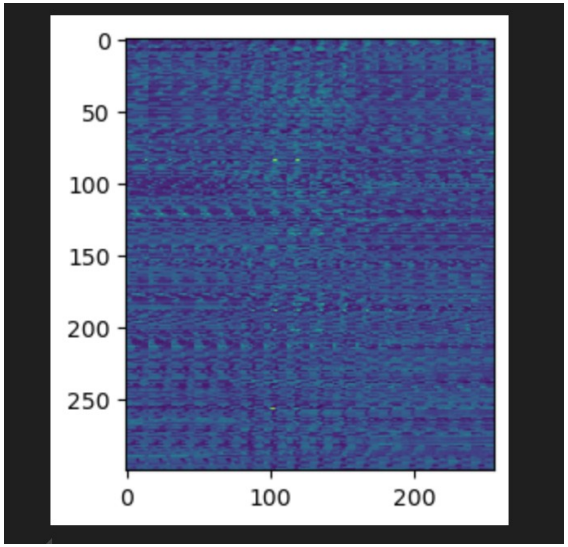
# Part 1: Harris corner detector

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

Since it computes the intensity changes in multiple directions, it can easily tell the corner points based on corner response function, where intensity changes most in multiple directions. What’s more, it is invariant to shifts, and uses non-maximum suppression to reduce noises

# Part 2: Normalized patch feature descriptor

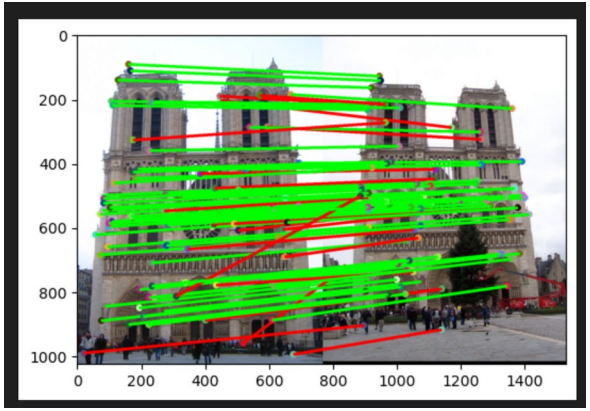
[insert visualization of normalized patch descriptor from proj2.ipynb here]




[Why aren't normalized patches a very good descriptor?]
It is sensitive to noise and minor changes like lighting variations, minor misalignment, since it only compare the intensity of values.

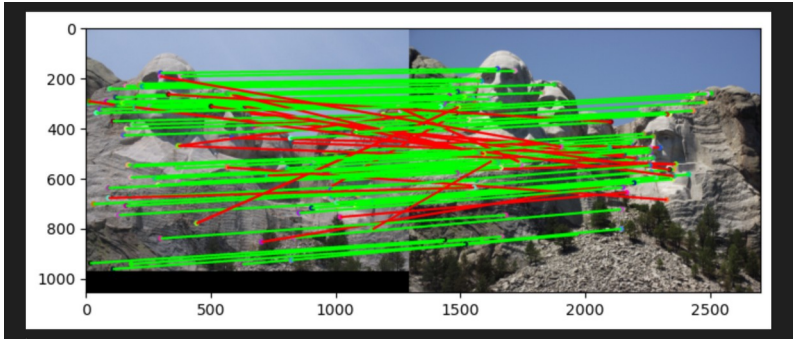
# Part 3: Feature matching

[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from proj2.ipynb here]




# matches (out of 100): [insert # 95]
Accuracy: [0.75]

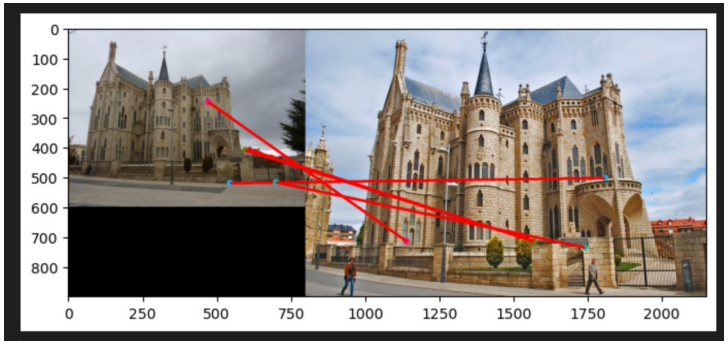
[insert visualization of matches for Mt. Rushmore image pair from proj2.ipynb here]




# matches: [insert # 96]
Accuracy: [insert 0.73]

# Part 3: Feature matching

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

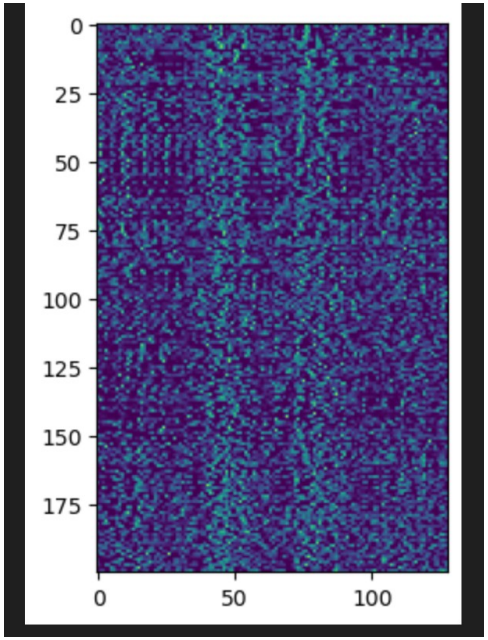



# matches: [insert # 5]
Accuracy: [0]

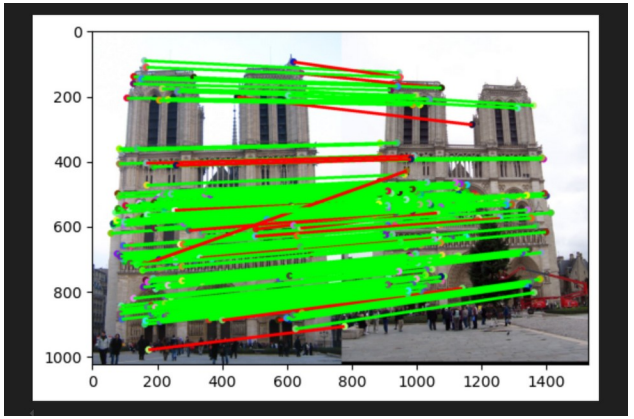
[Describe your implementation of feature matching here]
I firstly use compute_feature_distance(feature1,feature2) to calculate the distances from each point in feature1 to every point in feature2.
Then I use match_features_ratio_test(), to filter the the match. Basically, I sort the distances in descent order. Then I abstracted two matches with smallest values, from matches of each point in feature1 to every point in feature2. After that, I performed ratio test to discard invalid matches.

# Part 4: SIFT feature descriptor

[insert visualization of SIFT feature descriptor from proj2.ipynb here]



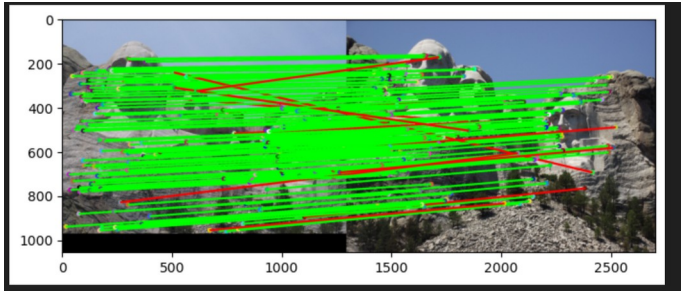

[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from proj2.ipynb here]




# matches (out of 100): [insert # 179]
Accuracy: [0.916201]

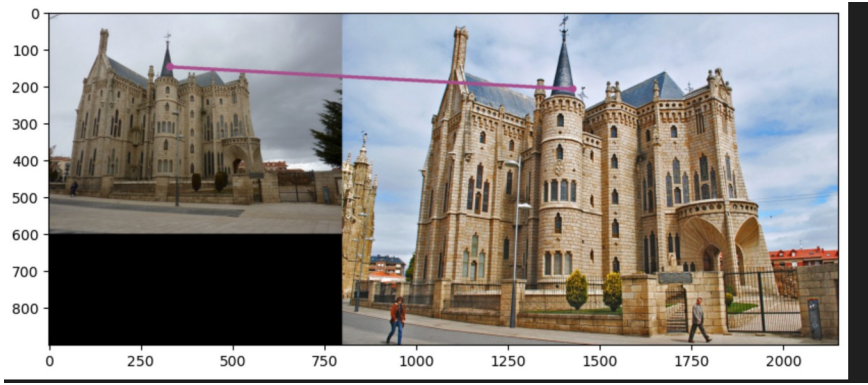
# Part 4: SIFT feature descriptor

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




# matches: [insert # 149]
Accuracy: [0.919463]

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




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

# Part 4: SIFT feature descriptor

[Describe your implementation of SIFT feature descriptors here]

1.I firstly used compute_image_gradients(image_bw) to gain the Ix and Iy at each pixel

2. Then I used get_orientations_and_magnitudes() to gain the orientation and magnitude of gradient at each pixel

3. Then I used get_feat_vec() to gain the feature vector of a patch which centers at each selected interest points

4. Based on the feature vector gained from step3, I performed match_features_ratio_test() to compute the distance between different matches and performed ratio test to discard invalid matches.

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

Since SIFT actually compute the gradient difference of potential match in different directions. Therefore, it is less sensitive to noise and small changes.

What's more, it is also invariant to brightness shifts

# 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?]
Since the geometric features in Guadi image pair consists of mostly straight lines, there are less corners than those in Notre Dame and Rushmore pairs. And SIFT is based on these corner points, thus SIFT perform worse.

# 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]?

I this the pipeline worked well, however, there are still some characteristics that make it difficult to match.

For example, if the some parts of buildings are blocked, or the two photos were taken in large perspective differences

# Conclusion

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

Since our SIFT only takes in one scale and does not assign a dominant orientation at each key points. To solve that, we can generate multiple blurred versions of the image at different scales for SIFT, so that our SIFT can be scale-invariant. For rotation, we can assign a dominant orientation at pixels and calculate the SIFT relative to that dominant orientation.