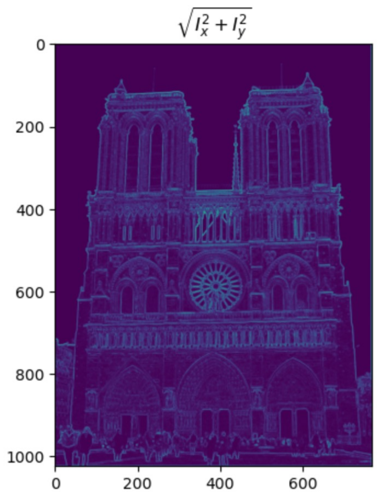
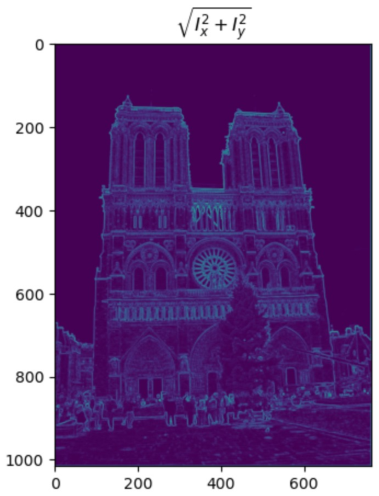
# CS 4476/6476 Project 2

Nathan Wang
nwang334@gatech.edu
nwang334
903633600

## Part 1: Harris corner detector

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



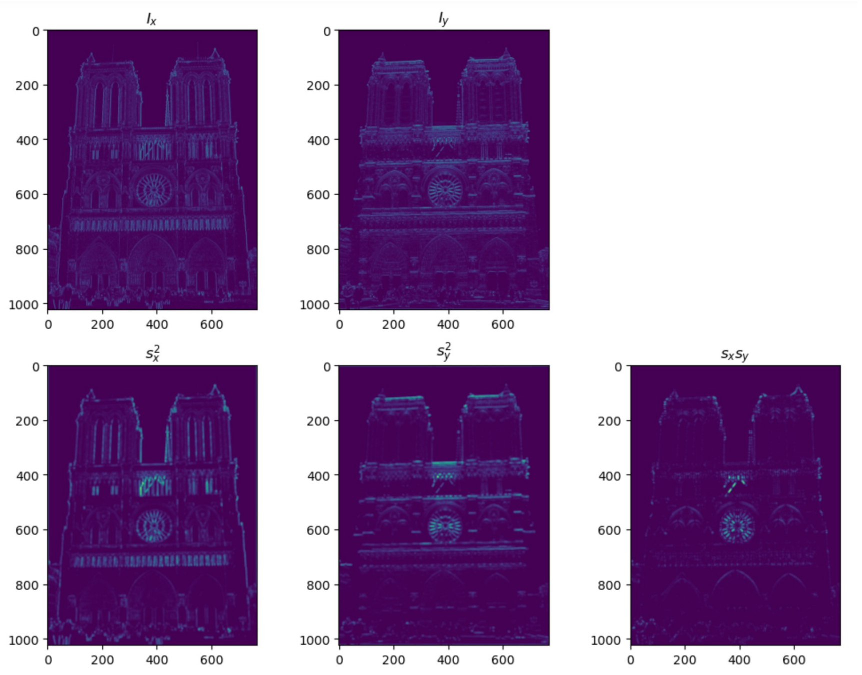


[Which areas have highest magnitude? Why?

The edges of each image have the highest magnitude because we run a Sobel filter (X and Y kernels) over each image. Sobel filters work for edge detection because it finds the gradient of each pixel from light to dark.

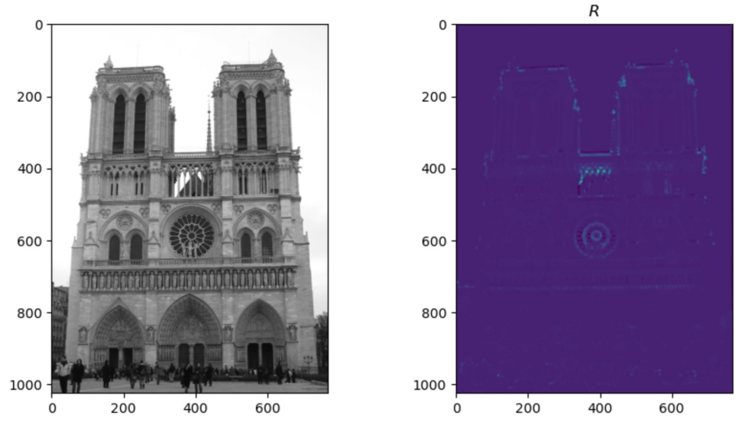
## Part 1: Harris corner detector

[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]

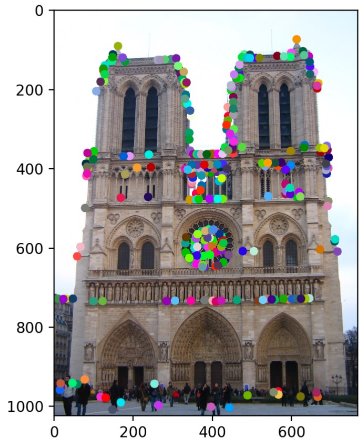


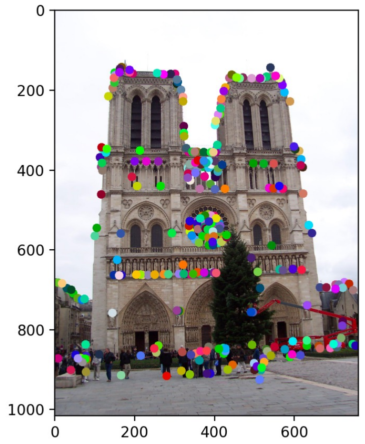
[Are gradient features invariant to both additive shifts (brightness) and multiplicative gain (contrast)? Why or why not? See Szeliski Figure 3.2]

Gradient features are invariant because gradient direction is not affected by multiplicative or additive shifts to each pixel. This is due to the fact that we use derivatives, so brightness and contrast shifts will not affect Harris detectors much (just threshold for multiplicative).

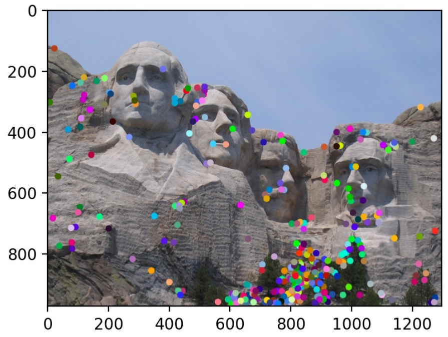
## Part 1: Harris corner detector

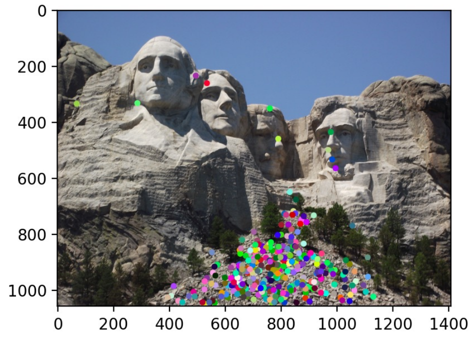
[insert visualization of Notre Dame interest points from proj2.ipynb here]





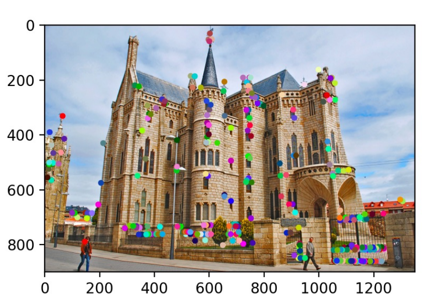
[insert visualization of Mt. Rushmore interest points from proj2.ipynb here]

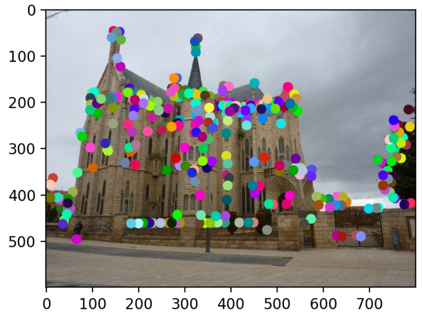




# Part 1: Harris corner detector

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





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

The advantages of maxpooling for NMS is its easy implementation, removes shift, rotational, and scale shifts, and reduces computational cost by taking only important features (the peaks).

The disadvantages of maxpooling for NMS is that we take the local maxima, so if features in the region all have high magnitudes in relation to other regions, only the highest point will be kept.

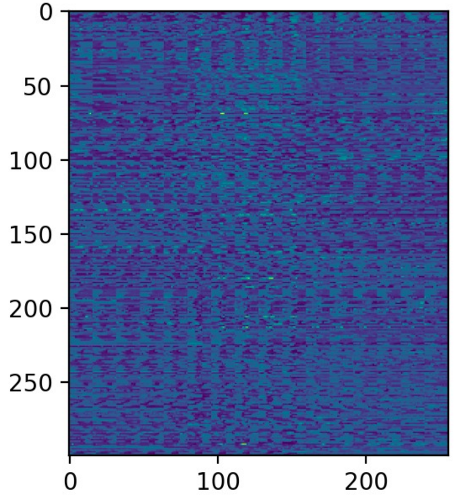
## Part 1: Harris corner detector

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

By sliding a window over the image and calculating gradients, we can find the corners by locating regions where the intensity changes most. Corner detection is good feature because there are changes in either direction and is thus more unique than edges. This makes corners translation and rotationally invariant.

## Part 2: Normalized patch feature descriptor

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

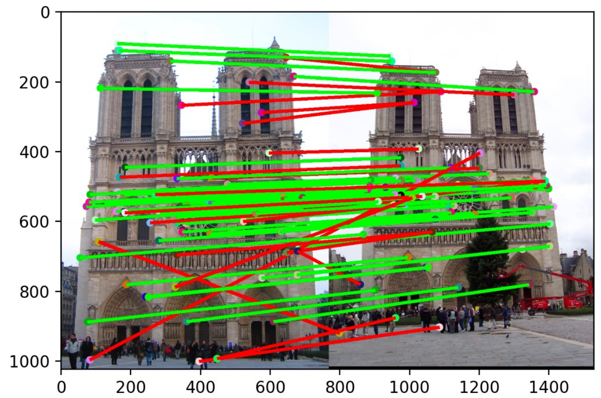


[Why aren't normalized patches a very good descriptor?]

Normalized patches aren’t a very good descriptor because they just take every point in the patch and normalizes the vector. This does not take into account any important features or gradients of the whole image.

## Part 3: Feature matching

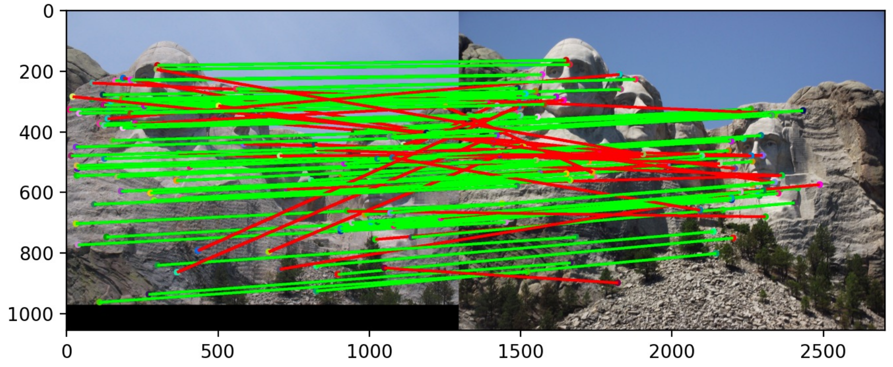
[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from proj2.ipynb here]



# matches (out of 100): 74

Accuracy: 0.52000

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

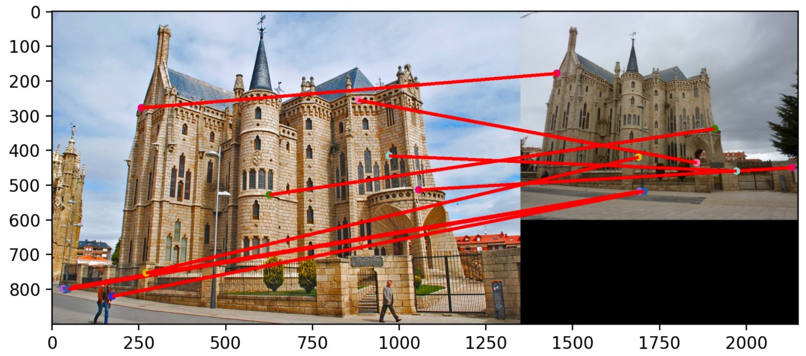


# matches: 101

Accuracy: 0.71000

## Part 3: Feature matching

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



# matches: 9

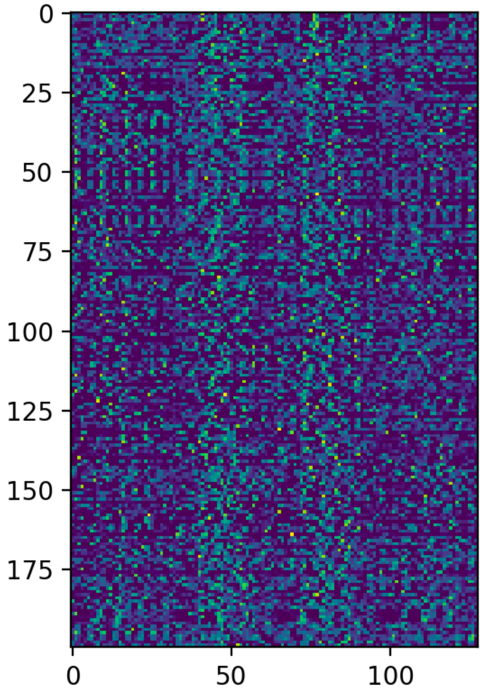
Accuracy: 0.0000

[Describe your implementation of feature matching here]

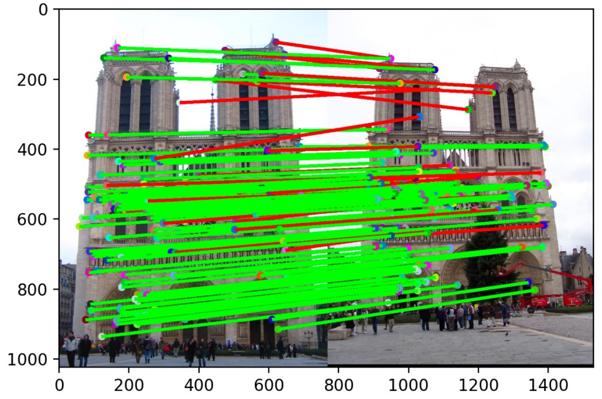
First, I calculate the distances between every features using compute_feature_distances(). Then, I loop over dists on the first point, argsort the list of distances to other features, and find the distances from the current point to the closest and second closest feature. Then I calculate NNDR by dividing the first closest distance by the second closest. If the NNDR is below the threshold I chose (0.8), I add the indices of each feature to the result and put in 1/NNDR as the confidence. I chose to do 1/NNDR since the confidence should increase as NNDR decreases.

# 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]

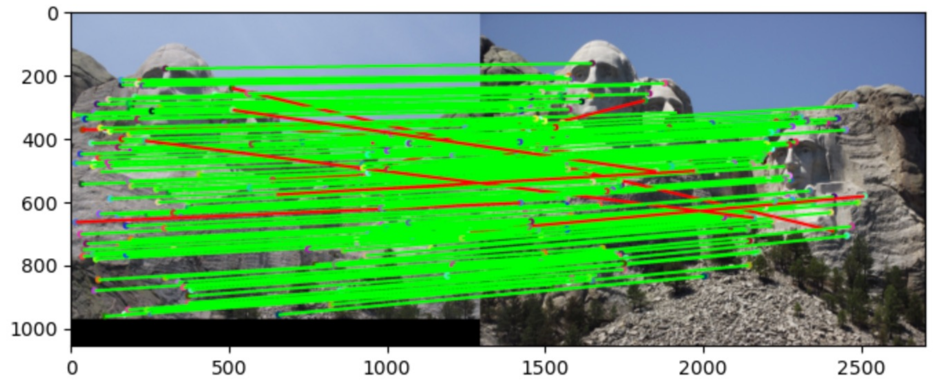


# matches (out of 100): 146

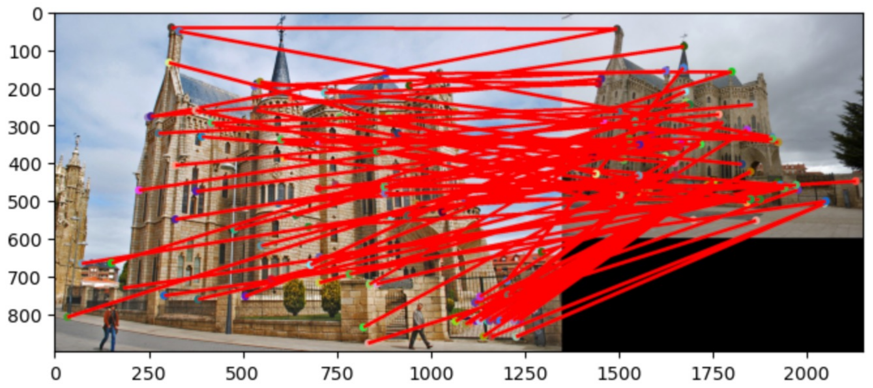
Accuracy: 0.835616

# Part 4: SIFT feature descriptor

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



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



# matches: 165
Accuracy: 0.933333

# matches: 91
Accuracy: 0.000000

## Part 4: SIFT feature descriptor

[Describe your implementation of SIFT feature descriptors here]

First, I compute the image gradients using the compute_image_gradients method from Part 1. Then, I calculate the magnitudes and orientations using Ix and Iy gradients. Finally, I loop over each feature point and calculate the SIFT feature vector. To do this, I get the 16x16 patch around the keypoint and then make histograms with centers -7pi/8, -5pi/8 … 7pi/8 on 4x4 cells. Based on the post on Piazza, I had to subtract 1e-6 to the bins.

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

SIFT features are better descriptors because it uses gradients at each interest point to determine feature descriptors. This way, we look at more unique points and base our features off of distances to the other corner points.

## 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?]

Our SIFT implementation performs worse on the Gaudi image pair than the other image pairs because of the shift in scale. Our version of SIFT does not use blurring or scale space images. In order to account for the change in image sizes, we would have to change the corresponding sizes of regions around each corner interest point.

## 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?]

Our SIFT does not account for different scales or rotation of objects in space. To make it so, we would have to create a scale space of the image (account for region sizes using Gaussians or Laplacian and scale) and subtract each keypoint’s rotation from the orientations. Since our SIFT features use gradient orientations, rotating the image without accounting for rotation would change the gradients.