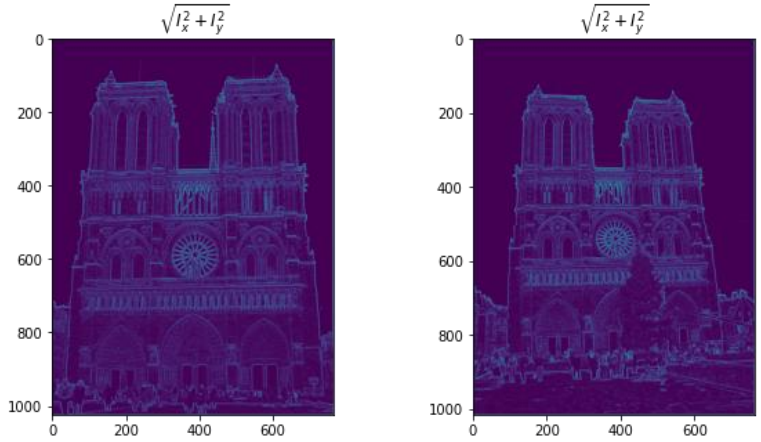
CS 5330 Programming Assignment 2

[Sasank Potluri]
[potluri.sas@northeastern.edu]
[potluri.sas]
[002925014]

Part 1: Harris corner detector

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

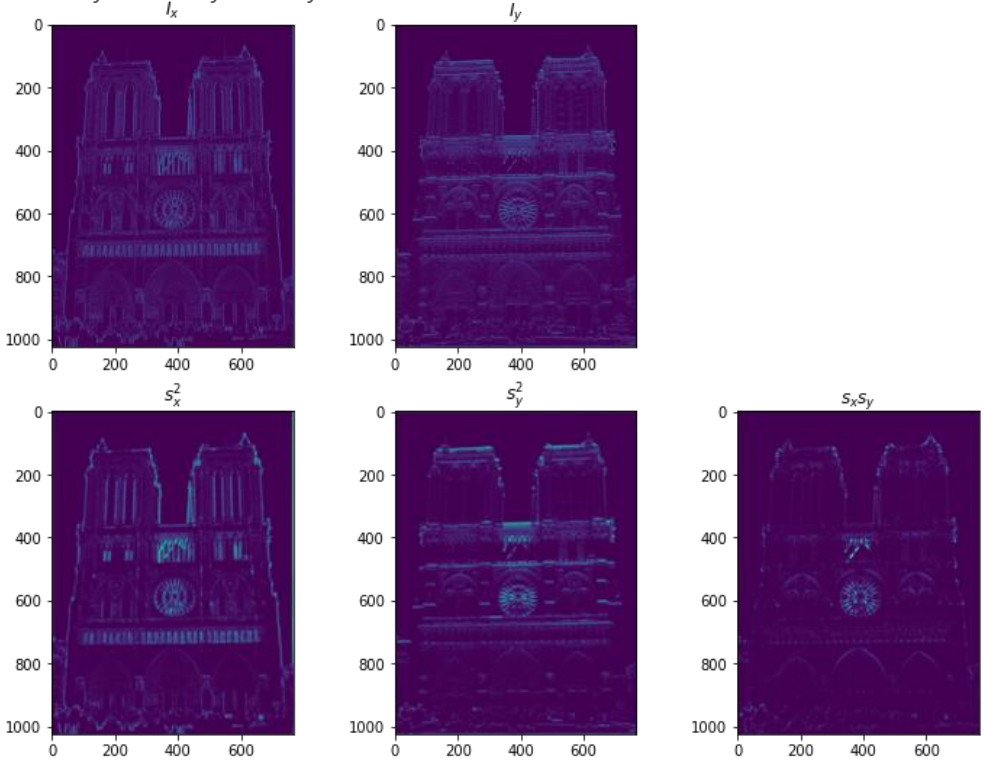


[Which areas have highest magnitude? Why?]

The places where there is a rapid change in the color shows the highest magnitude in the images. This is why you cannot see any bright region in the place where sky is but a lot of bright lines where people are. This is because we are trying to calculate the image gradients by multiplying the image with sobel filter which has a row/ column of low followed by zeros and followed by high values, this is why the image shows high brightness when there is a rapid change in color-depth

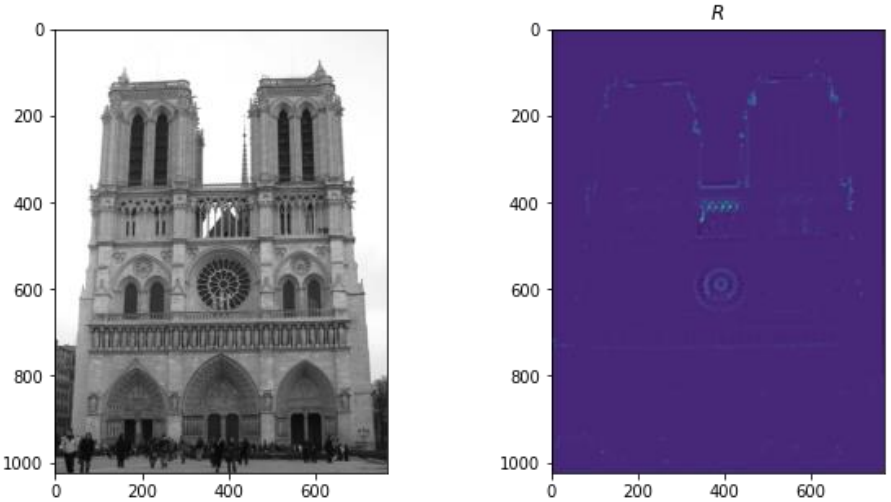
Part 1: Harris corner detector

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



Part 1: Harris corner detector

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



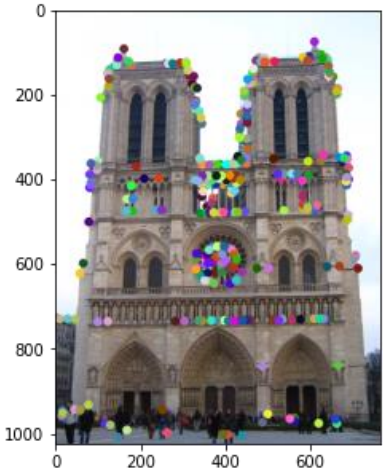
[Are gradient features invariant to both additive shifts (brightness) and multiplicative gain (contrast)? Why or why not? See Szeliski Figure 3.2]

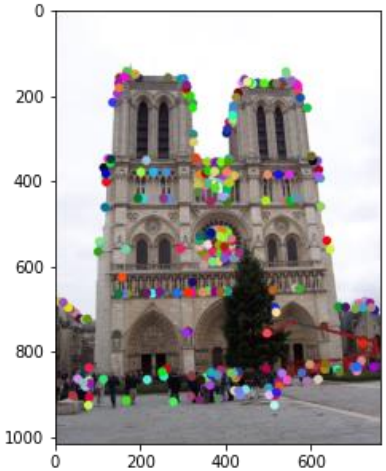
By adding constant brightness gradient features will not change as we are working with the differences between consequetive pixels. If there is a multiplicative gain then this would show difference, but to compensate that we normalize our output. In this case we are using eigen values, which is a measure of direction of data so in this particular the brightness and contrast wouldn't matter, only the direction matters. In the image we can clearly see that it doesn't find any corners at points that are mostly dark or light but find the points where the trasition is happening

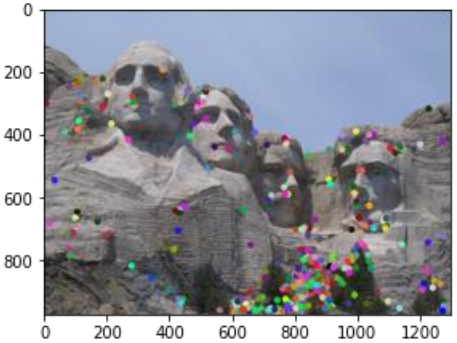
Part 1: Harris corner detector

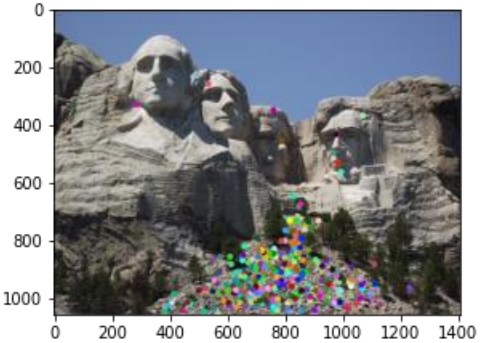
[insert visualization of Notre Dame interest points from pa2.ipynb here]

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



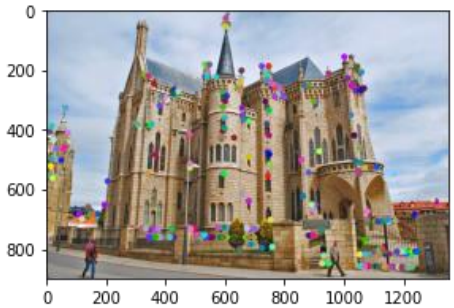


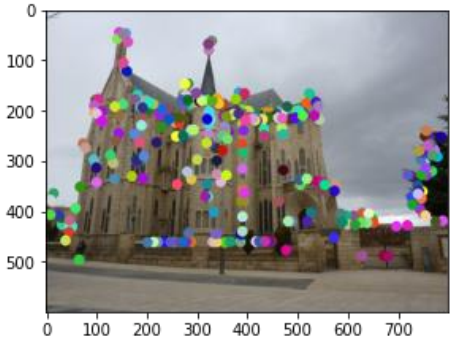




Part 1: Harris corner detector

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





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

NMS is mainly used to remove duplicates of features. It picks only the highest R value in a local window for a given kernel size, this is important because many times the same corner could be spread over multiple features or a features could be denseley packed over one particular spot. Inorder to prevent that we use maxpooling for NMS, causing our features to be more spread over the image. The disadvantages of this is that we loose the information around the highest values and also this adds another step to our pipeline causing our algorithm to take longer

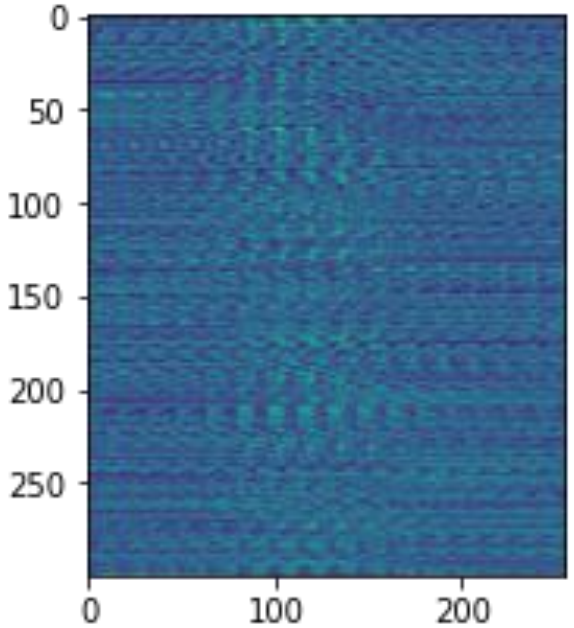
Part 1: Harris corner detector

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

Harris corner detector is based on the change in the intensities which is why it is invariant to intensities, and it finds the corners based on the direction of gradients by making use of eigen values. It doesn’t calculate the eigen values but makes use of its properties instead which makes it fast and accurate at the same time. Im the above images it can be clearly seen as it doesn’t pick features at sky or at the floor but it picks at designs on buildings and windows

Part 2: Normalized patch feature descriptor

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

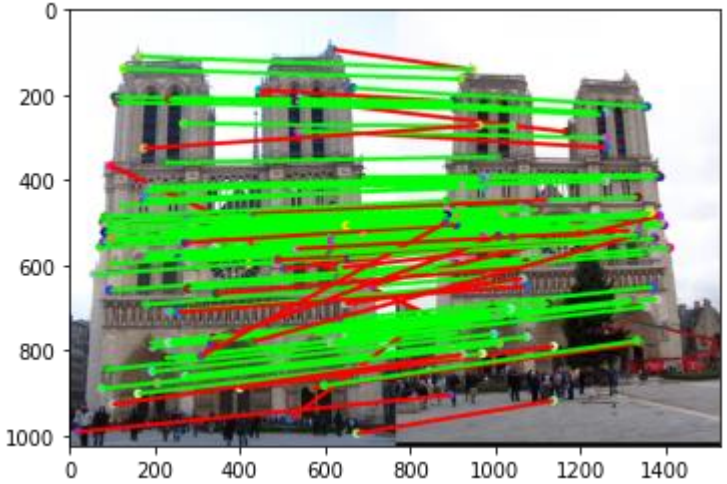


[Why aren't normalized patches a very good descriptor?]

When you normalize a descriptor any value that is eccentric to the normal distribution curve gets normalized and this is not good for a feature descriptor. Eccentricities in descriptor is caused because of the feature so a very good way to match these features is to see if both the points are having same kind of eccentric values or not. Also normalizing feature descriptor will cause you to have the seond nearest neighbor close to the first nearest neighbor which means it is hard for the algorithm to find a good feature

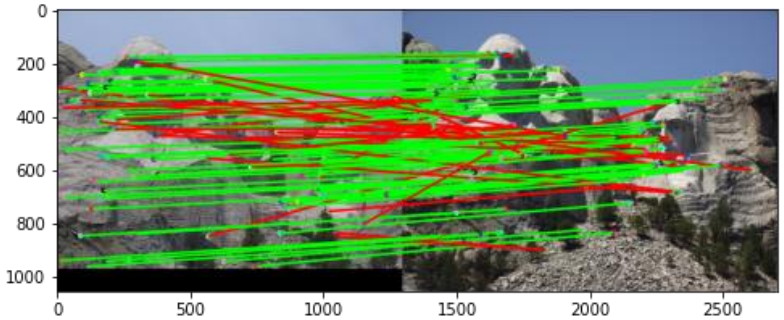
Part 3: Feature matching

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



matches (out of 100): [107 matches from 2464]
Accuracy: [0.766355]

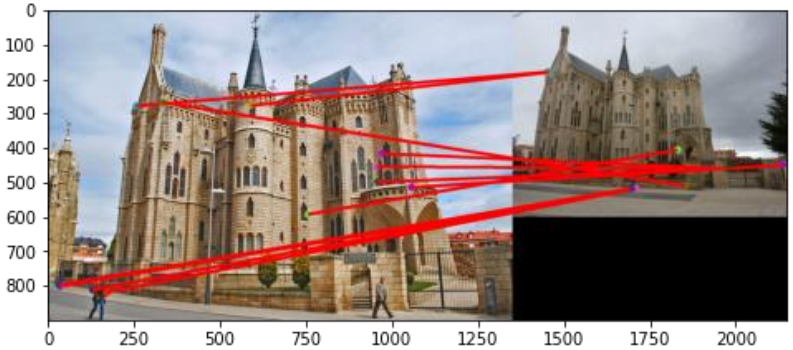
[insert visualization of matches for Mt. Rushmore image pair from pa2.ipynb here]



matches: [107 matches]
Accuracy: [0.728972]

Part 3: Feature matching

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



matches: [12 matches]
Accuracy: [0.000]

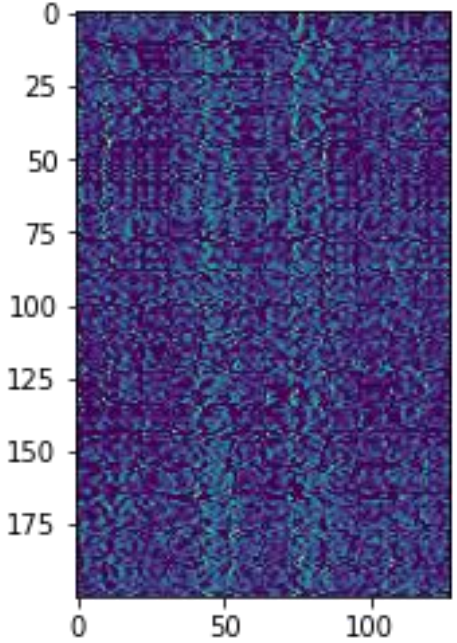
[Describe your implementation of feature matching here]

First I had to calculate the distances between the features, to do this I ran each feature in the image1 with all the features in the 2nd image to calculate the distances between them which are then appended to a list. Idid this for all features in 1st image using a for loop and calculated the MXN matrix.

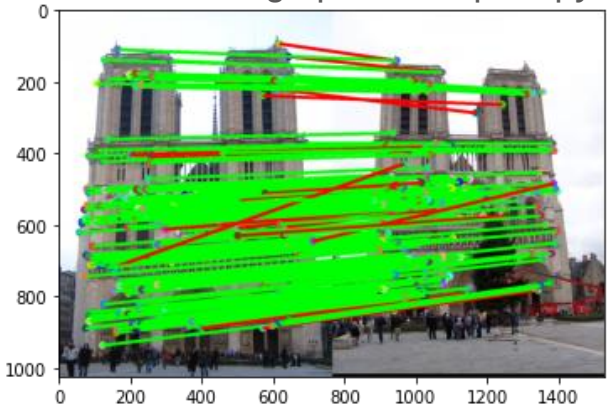
Second I had to calculate the ratio of distances for all the features in 1st image. I did this by running a for loop for all the features of 1st image and sorted them then took the top 2 values to find out their ratio. If their ratio is higher than the given threshold then I considered them to be the same feature and appended its index value.

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]

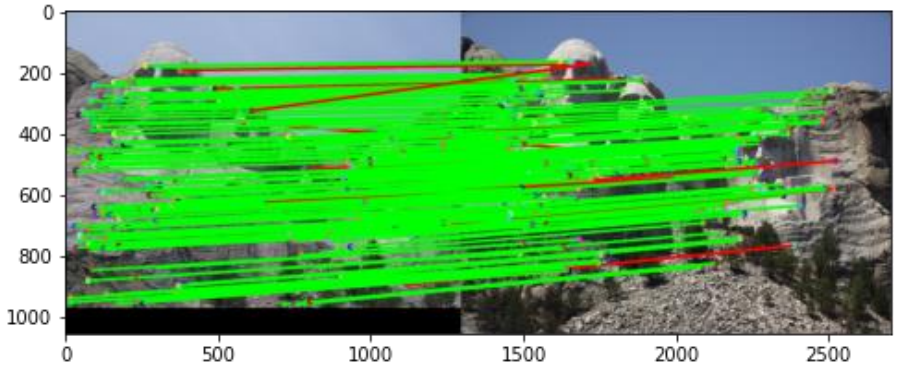


matches (out of 100): [198 matches]

Accuracy: [0.919192]

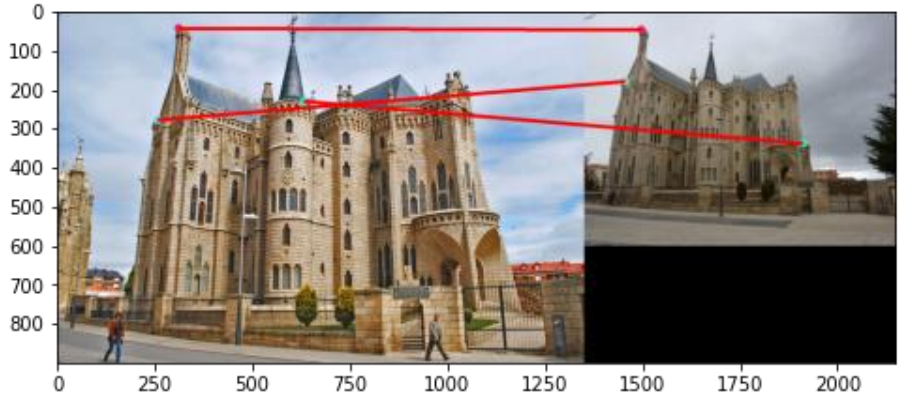
Part 4: SIFT feature descriptor

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



matches: [177 matches]
Accuracy: [.937853]

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



matches: [3 matches]
Accuracy: [0.000]

Part 4: SIFT feature descriptor

[Describe your implementation of SIFT feature descriptors here]

First, I need to calculate the magnitudes and orientation of the image, I did this by using the intensity values and converting them using sqrt and arctan function in numpy.
Next I ran the 16X16 patches around the feature points by running it in the form of 4X4 patches to calculate the histogram and appended the results
The histogram arrays were then smoothened by normalizing them this is done using np.norm and sqrt command.
The smoothened feature descriptors were then used for comparision between them by running the this algorithm on all keypoints for both the images

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

SIFT feature extractor doesn’t mainly focus on every aspect of the feature, it does this by collecting all the values and making a histogram for the local patch. This makes the descriptor to get the general structure rather than every detail which is prone to noise. This makes SIFT invariant to brightness, translation and out of plane rotation upto some extend.These characters can be clearly seen first image where the overall contrast is not the same but the descriptor works After generating the feature vector it is also normalized causing the descriptor to be invariant to contrast as well but SIFT is not scale invariant.
Normalized patches would be contrast invariant but it cannot function well when the feature selected is even a little to the left or right, Since SIFT takes all the values into consideration it would have little to no difference for such noises.

Conclusion

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

Our version of SIFT is not invariant to rotation because we do not give the orientation in which the feature is being described. To make them rotation invariant we could calculate the second descriptor at other possible angles and then match the descriptor, This make the process computationally very expencive

Our harris corner detector only detects whether a pixel in the image is a feature or not, it doesn’t find the right scale for the found out features. To find the right scale we can use the glob feature detector instead which records the image on a different scale dimension and also find the best fit scale for the particular feature