EE379K: Assignment #2

Due: Mar. 1 2024, 11:59pm TX

Introduction

The overall goal of this homework is to implement a local feature matching algorithm. You can discuss in small groups, but turn in individual solutions and indicate collaborators. Turn in assignments by Friday, Mar. 1. Submit to Canvas a .zip file containing (1) a questions.pdf with answers for Part 1 questions, (2) a writeup.pdf for Part 2 code; and (3) a `code/` directory containing your code without any large result files or subfolders. The pdf can be created using latex or converting a word document to pdf or any other method you prefer, as long as it is organized and easy to read. The submission template with latex and code for this homework can be found here: https://classroom.github.com/a/mCG_ijF3.

Info: `Szeliski` refers to the second edition of *Computer Vision: Algorithms and Applications*, which can be found here.

1 Part 1 Questions (30%)

Instructions

- Three graded questions.
- Write code where appropriate.
- Feel free to include images or equations.
- **We do NOT expect you to fill up each page with your answer.** Some answers will only be a few sentences long, and that is okay.

1.1 Q1

The Harris Corner Detector is commonly used in computer vision algorithms to find interest points from which to extract stable features for image matching.

How do the eigenvalues of the 'M' matrix change if we apply a low-pass filter to the image? What if we apply a high-pass filter? Describe qualitatively.

Answer: If we apply a low-pass filter to the image, the eigenvalues of the M matrix will **decrease**. This is because the low-pass filter will smooth the image, which will reduce the gradient magnitudes and thus the eigenvalues of the M matrix. The opposite is true for a high-pass filter, where the eigenvalues of the M matrix will **increase** because the high-pass filter will keep the high frequencies in the iamge, which will increase the gradient magnitudes and thus the eigenvalues of the M matrix.

1.2 Q2

Given an interest point location, the SIFT algorithm converts a 16×16 patch around the interest point into a 128×1 feature descriptor of the gradient magnitudes and orientations therein. Write pseudocode *with matrix/array indices* for these steps.

Notes Do this for just one interest point at one scale; ignore the overall interest point orientation; ignore the Gaussian weighting; ignore all normalization post-processing; ignore image boundaries; ignore sub-pixel interpolation and just pick an arbitrary center within the 16×16 for your feature descriptor. Please just explain in pseudocode how to go from the 16×16 patch to the 128×1 vector.

```
# You can assume access to the image, x and y gradients, and their magnitudes/
                                            orientations.
image = imread('example.jpg')
grad_x = filter(image, 'sobelX')
grad_y = filter(image, 'sobelY')
grad_mag = sqrt( grad_x .^2 + grad_y.^2 )
grad_ori = atan2( grad_y, grad_x )

# Takes in a interest point x,y location and returns a feature descriptor
def SIFTdescriptor(x, y)
    descriptor = zeros(128,1)

    # loop through a 4x4 grid of 4x4 patches
    for i in range(-2, 2):
        for j in range(-2, 2):
            # i,j represents the 4x4 patch index
            patch_descriptor = zeros(8,1)
            for m in range(4):
                for n in range(4):
                    # m,n represents the 4x4 pixel index
                    # get the gradient orientation and magnitude
                    grad_magnitude = grad_mag[x+i*4+m, y+j*4+n]
                    grad_orientation = grad_ori[x+i*4+m, y+j*4+n]
                    # convert the orientation to the 8-bin histogram index
                    bin_index = floor(grad_orientation // (pi/4))
                    # add the magnitude to the corresponding bin
                    patch_descriptor[bin_index] += grad_magnitude
            # add the 8-bin histogram to the descriptor
            descriptor.append(patch_descriptor)
    return descriptor
```

1.3 Q3

Distance metrics for feature matching.

(a) Explain the differences between the geometric interpretations of the Euclidean distance and the cosine similarity metrics. What does this tell us about when each should be used?

(b) Given a distance metric, what is a good method for feature descriptor matching and why?

Answer:

A: Difference between Euclidean distance and cosine similarity: The Euclidean distance measures the distance between two points in a space, while the cosine similarity measures the angle between two vectors. The Euclidean distance is sensitive to the magnitude of the vectors, while the cosine similarity is not. This means that the Euclidean distance will be large if the vectors are far apart in space and cosine similarity will be large is the direction of the vectors is vastly different. This tells us that the Euclidean distance should be used when the magnitude of the vectors is important, while the cosine similarity should be used when the direction of the vectors is important.

B: Good method for feature descriptor matching: Given a distance metric, a good method for feature descriptor matching is to use the nearest neighbor algorithm. This is because the nearest neighbor algorithm is simple and efficient, and it is also very effective for high-dimensional data. The nearest neighbor algorithm will find the closest neighbors (or feature descriptor) in terms of the specified distance metric. There is a chance that a multiple feature descriptors will be close to the target feature descriptor, so the first closest neighbor may not be the best match. This can be accounted for by dividing the distance of the first closest neighbor by the distance of the second closest neighbor. If this ratio is less than a certain threshold, then the first closest neighbor is a good match. This is called the ratio test, and it is a good method for keeping matches with a high confidence level.

2 Writeup

2.1 Notre Dame

- Image 1 - Number of interest points: 338
- Image 2 - Number of interest points: 291
- Matches: 156
- Accuracy on 50 most confident: 92%
- Accuracy on 100 most confident: 64%
- Accuracy on all matches: 50%

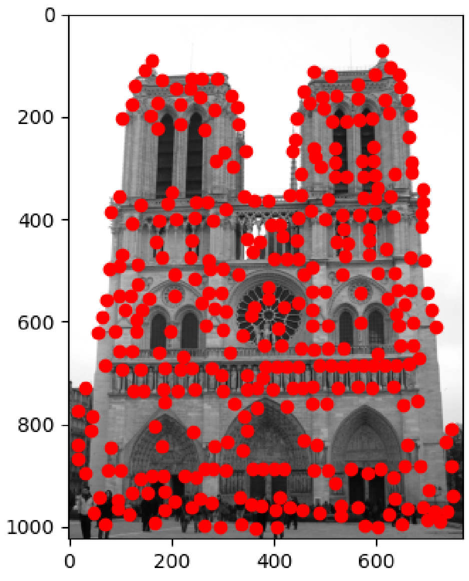

Figure 1: Notre Dame Image 1 Interest Points

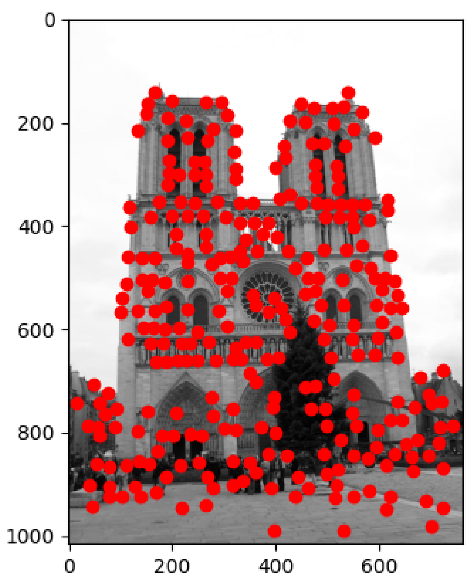

Figure 2: Notre Dame Image 2 Interest Points

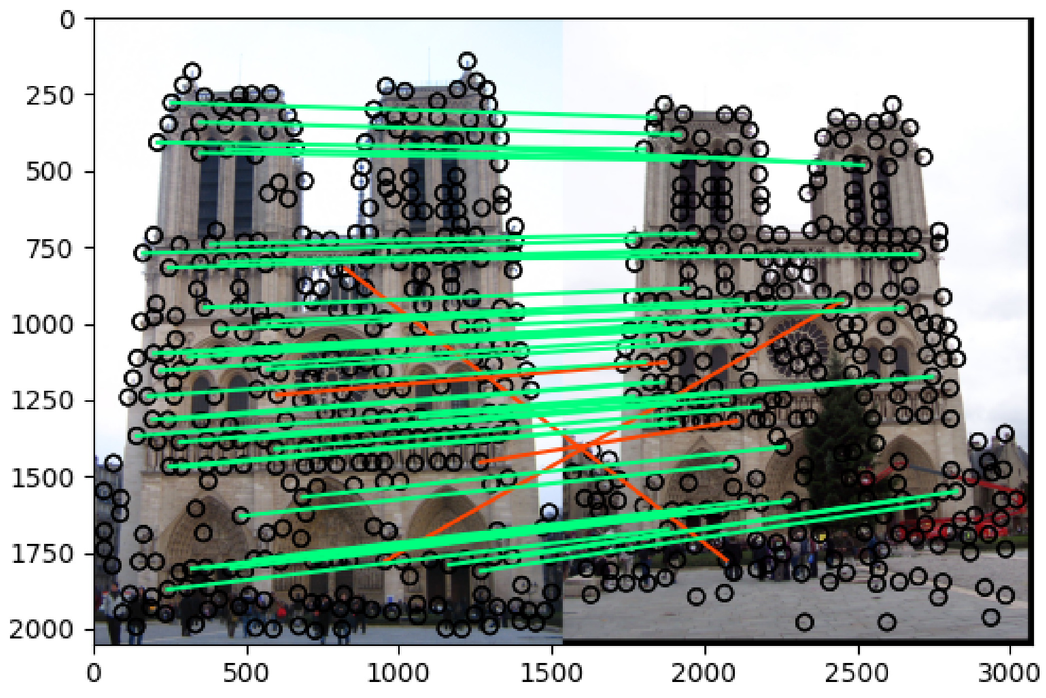

Figure 3: Notre Dame Matches

2.2 Mount Rushmore

- Image 1 - Number of interest points: 572
- Image 2 - Number of interest points: 700
- Matches: 249
- Accuracy on 50 most confident: 100%
- Accuracy on 100 most confident: 83%
- Accuracy on all matches: 40%

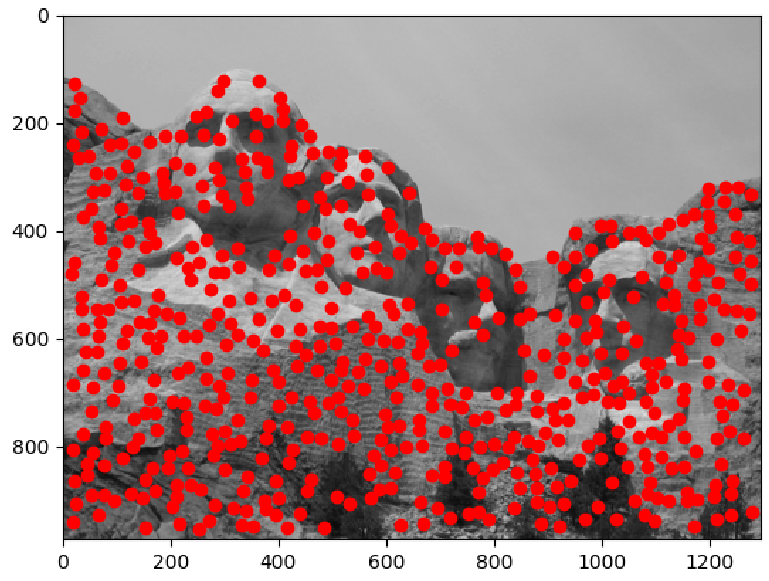

Figure 4: Mount Rushmore Image 1 Interest Points

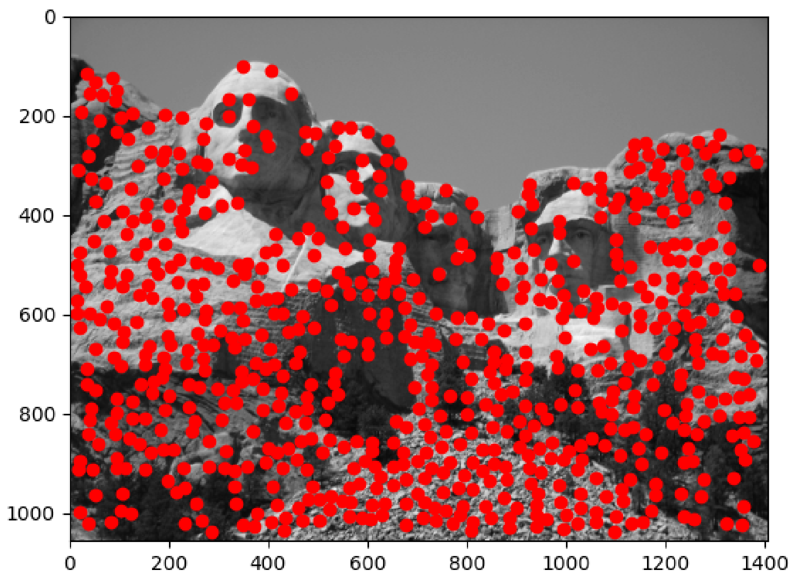

Figure 5: Mount Rushmore Image 2 Interest Points

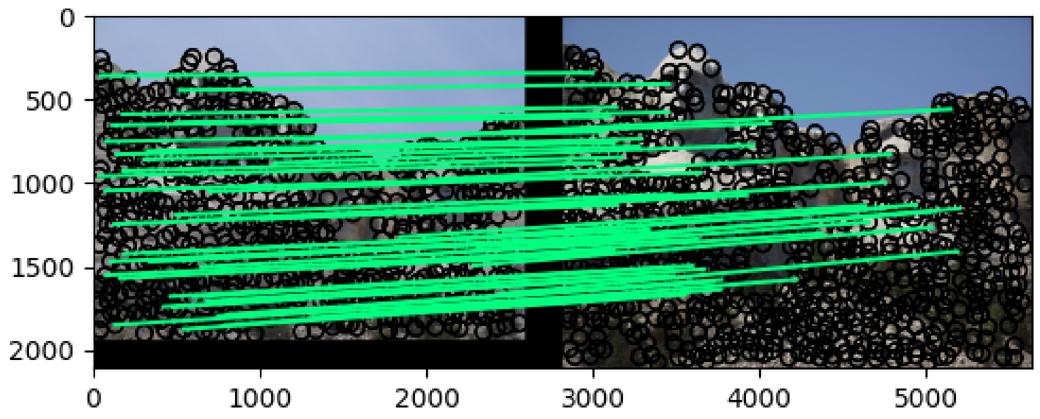

Figure 6: Mount Rushmore Matches

2.3 Episcopal Gaudi

- Image 1 - Number of interest points: 173
- Image 2 - Number of interest points: 543
- Matches: 62
- Accuracy on 50 most confident: 6%
- Accuracy on all matches: 9%

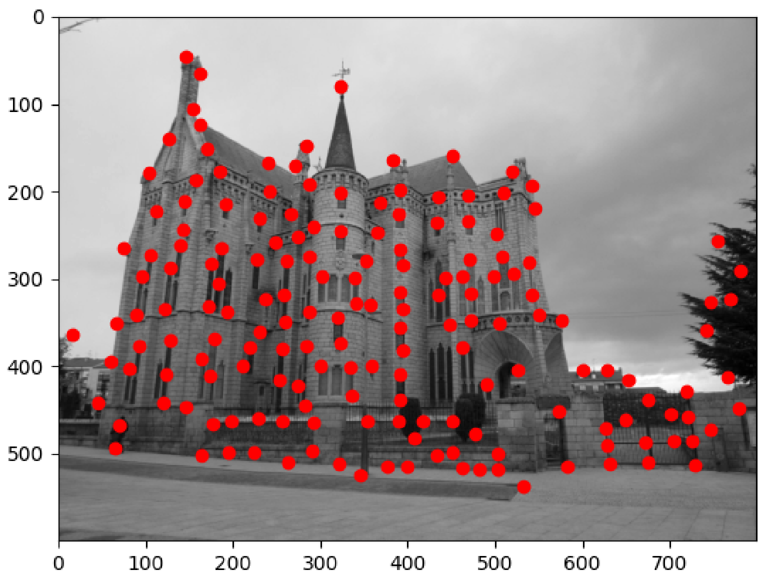

Figure 7: Episcopal Gaudi Image 1 Interest Points

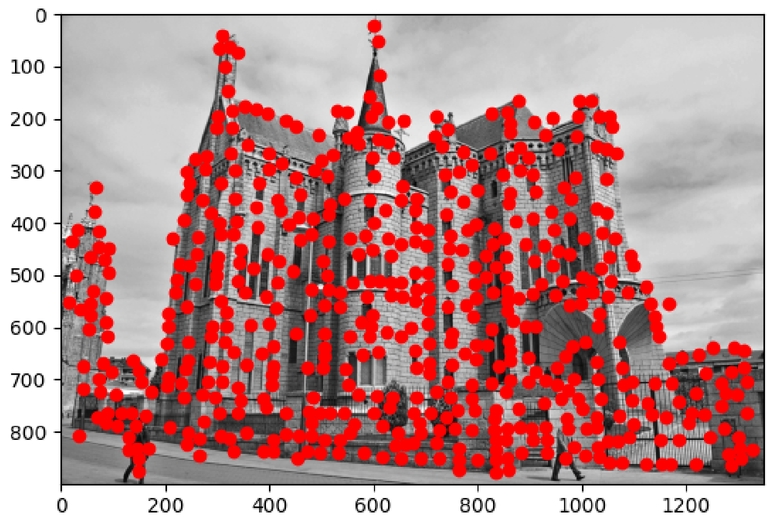

Figure 8: Episcopal Gaudi Image 2 Interest Points

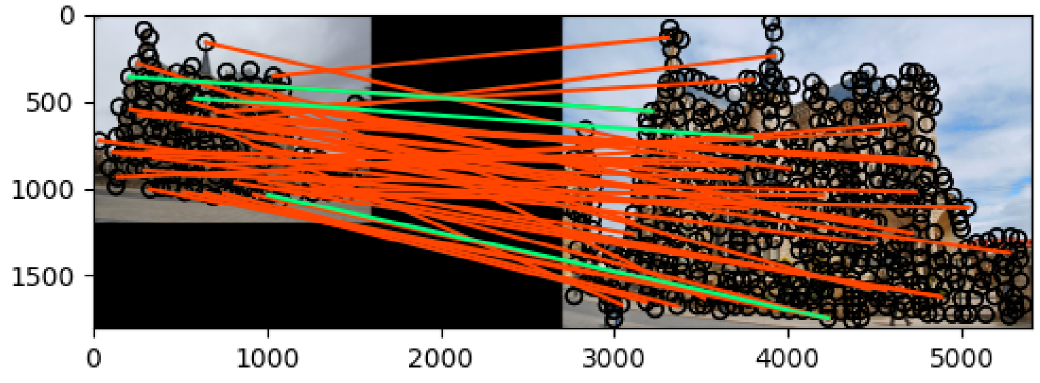

Figure 9: Episcopal Gaudi Matches

2.4 Analysis

The key point match accuracy performs incredibly well on Notre Dame and Mount Rushmore, but very poorly on Episcopal Gaudi.

This makes sense since Episcopal Gaudi's Figure 1 and Figure 2 differ significantly. Particularily, the Figure 2 is much more crisp and sharp. Thus the number of interest points is much higher in Figure 2 as the corners are easier to detect. Furthermore, the sizes of the images are also significantly different. Figure 1 is significantly smaller than figure 2 (compared to the relative sizes of figure 1 compared to figure 2 for Notre Dame and Mount Rushmore). I believe that multi-scale keypoint detection and multi-scale feature generation could help improve the accuracy of the matches for Episcopal Gaudi.

The interest points for Notre Dame and Mount Rushmore look correct. For the Notre Dame, the corners (such at the top most right and left corners) are captured well. For Mount Rushmore, the corners of the faces (such as the nose tisp) are captured well. The matches also look correct. Something interesting is that Mount Rushmore Figure 2 has many more points than Figure 1, due to the pile of rocks in the bottom. However, these points don't get matched, which demonstrates good performance of the feature detection and matching algorithms.

In class, we learned that for feature detection, we can either add a 1 to the bins, or the $magnitude$. I noticed that when adding magnitude, the performance actually became worse. I also tried scaling the magnitude before adding it to the bin. For the scaling factor, I used a 16x16 Gaussian kernel. As a result, the points closer to the center point would contribute a higher magnitude to their respective bins, and vice versa. This also resulted in worse performance. I believe that the reason for this is that the magnitude of the gradient is not as important as the direction of the gradient for feature detection.