COMP408 Assignment 1, Prepared by Arda Kırkağaç, 49799

The goal of this assignment was to design a scale-invariant feature detection and matching algorithm using techniques described in lecture slides and Szeliski Chapter 4.1. We implemented a well-known SIFT pipeline for this. The first function was to detect corners, and the others were simply extract scale invariant features and match them both visually and mathematically. Here are my ".m" files and their explanations:

1) detect_corners.m

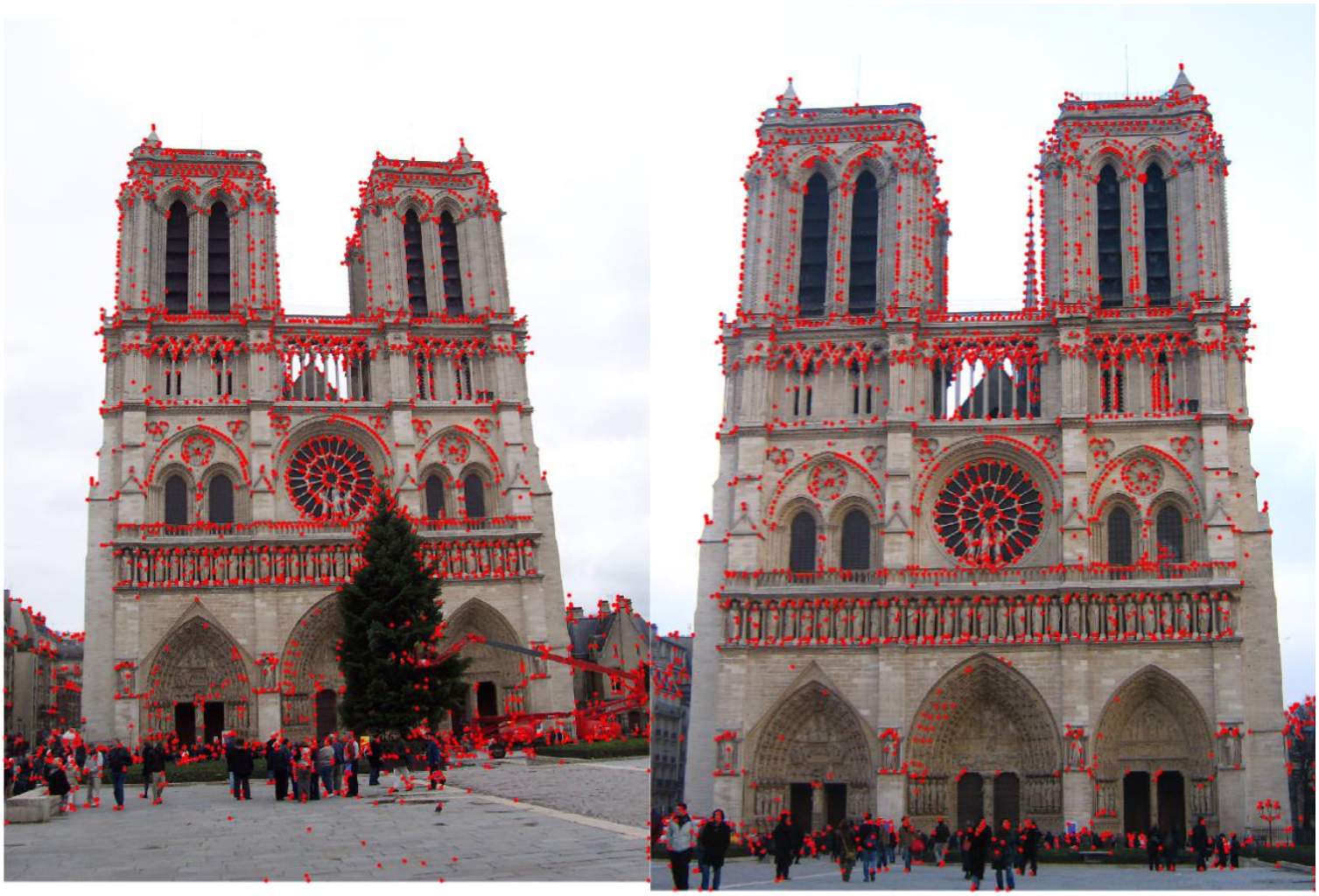
The file is in "KeypointDetect" folder. I used the Sobel filter to compute gradients in x and y direction, and then I further added Gaussian filtering for smoothing. While calculating the Harris value using difference matrices, 5x5 vicinities for each pixel was taken, reshaped as described in the slides, and the correlation matrix was computed. The code is shown below.

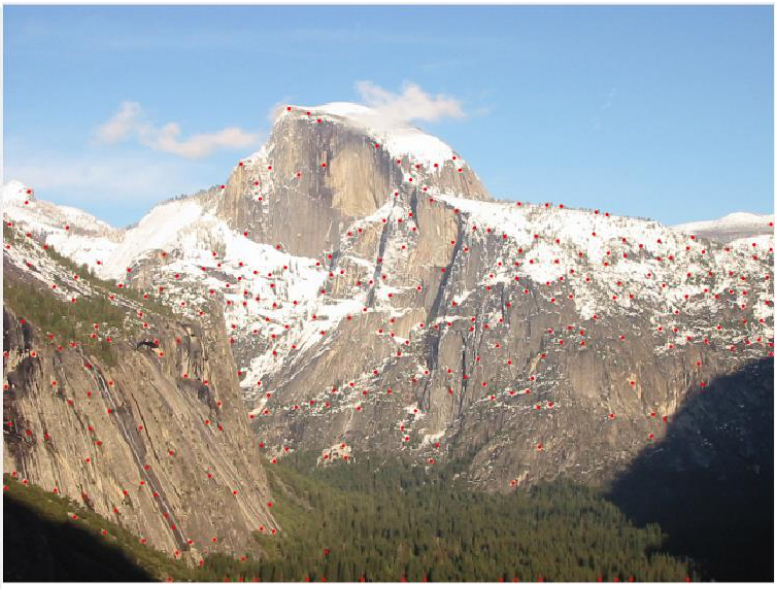
On the next pages, there are four pairs of images in total. You can see the dots pretty match each other in both images. My "detect_corners" function returns the coordinates of each keypoint, returns an image pyramid consisting of only one image with a fixed scale and a level of 1, making it compatible for further processing.

```
for i = 3:height-2
    for j = 3:width-2
        xVicinity = xGradient(i-2:i+2,j-2:j+2,:);
        xVicinity = reshape(xVicinity,[75,1]);
        yVicinity = yGradient(i-2:i+2,j-2:j+2,:);
        yVicinity = reshape(yVicinity,[75,1]);

        D = double([xVicinity,yVicinity]);
        C = (D')*D;
        Harris(i,j) = det(C)-0.04*(trace(C))^2;

    end
end
```

I applied a threshold on the matrix, hence values below a specific number were replaced with zeros. I used "colfilt" function to paint blocks with some maximum values, then take the difference matrix to obtain only that particular pixel with max Harris score value. Then I generated corresponding output for the pipeline. Harris threshold are subject to change for different purposes for images at different challenge levels. My outputs for the image pairs can be seen in the folder named "detect_corners output". Here are some examples below for four given image pairs (corners are represented with yellow and red dots):

```
%keep values above threshold
binaryHarris = Harris>2;
Harris = Harris.*binaryHarris;

%fill points with max values
maxHarris = colfilt(Harris, [15 15], 'sliding', @max);

%obtain logic difference so that we can locate max locations
a = (Harris == maxHarris & Harris > 0);
%if corner, this will find the coordinates
[posr, posc] = find(a > 0);
```





2) SIFTDescriptor

Here, we are calculating image gradients in both direction and for each point in image we compute two matrices, which are gradient magnitudes and gradient angles matrices. Those matrices will be used in calculating histograms and dominant directions just after.

We need to create 16x16 regions for each keypoint to generate SIFT features, which is already done by the prior code. Since `ComputeDominantDirection` function accepts input pixels as arrays rather than matrices, I needed to reshape our patches into row or column vectors before processing. This function will give us the dominant direction for the whole patch. This angle will be subtracted from the whole gradient angle matrix to obtain rotation invariance features. 36 is the pre-defined number of bins for this calculation.

For the next step, I divided the patch into 16 smaller patches, using exact values and expressions that are indicated in SIFT paper. For each small patch, gradient histograms were calculated. The output of the histogram function was concatenated with prior histogram values and with this procedure, our N*128 SIFT feature matrix was generated. The evaluation script indicates that it is properly working.

3) SIFTSimpleMatcher

This is a very straightforward code that implements the idea in paper. For each SIFT descriptor of the first image, all the descriptors of the other are processed to create a Euclidean distance matrix. If the smallest distance of two descriptors is smaller enough than the second-smallest distance, it is added as a match and relevant output is given in the visual matcher function. The evaluation script indicates that it is properly working.

4) MyTestPipeline

It is almost the same as "MatcherTester.m", only with slight differences in the feature vector (since ours has quite lower components). I used the SIFT pipeline that I had implemented for the other parts of the homework. My matching outputs are in a different folder named "myTestPipeline". Since my features are scale and rotation variant, in some images it will not yield any fruitful result as in the SIFT pipeline. However, for the image pairs in which only the viewing direction changes, I managed to obtain very similar and promising results. On the following page, some of my matcher outputs are shown. For a simpler visualization, threshold parameters were set to low values:

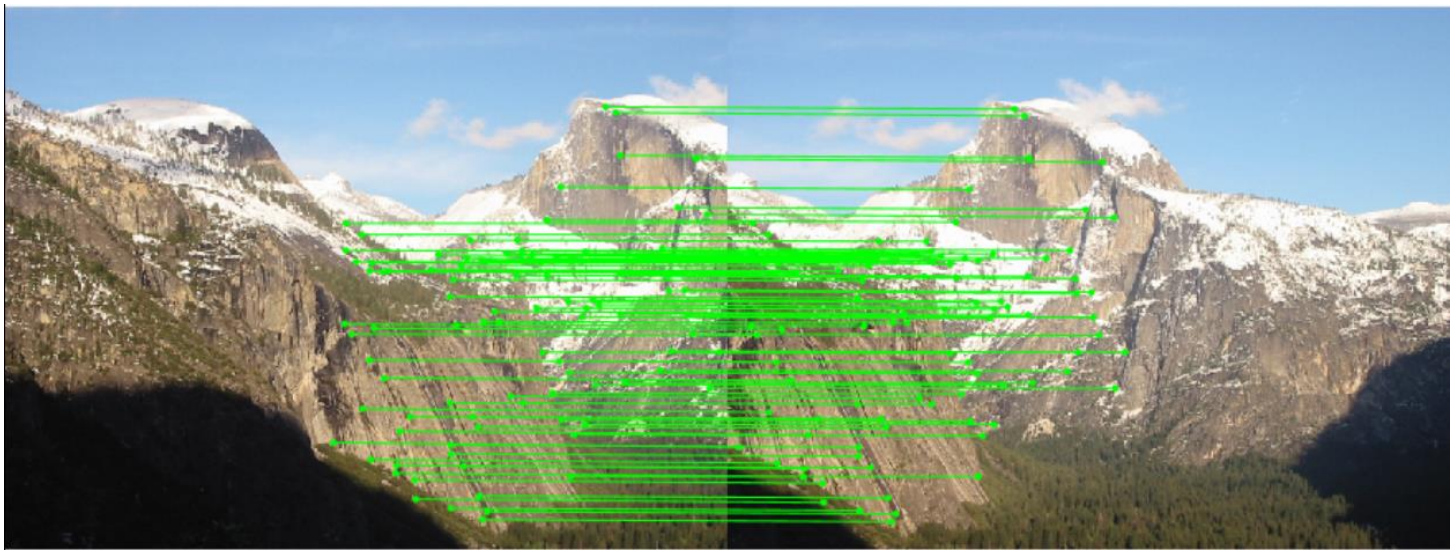
5) Results

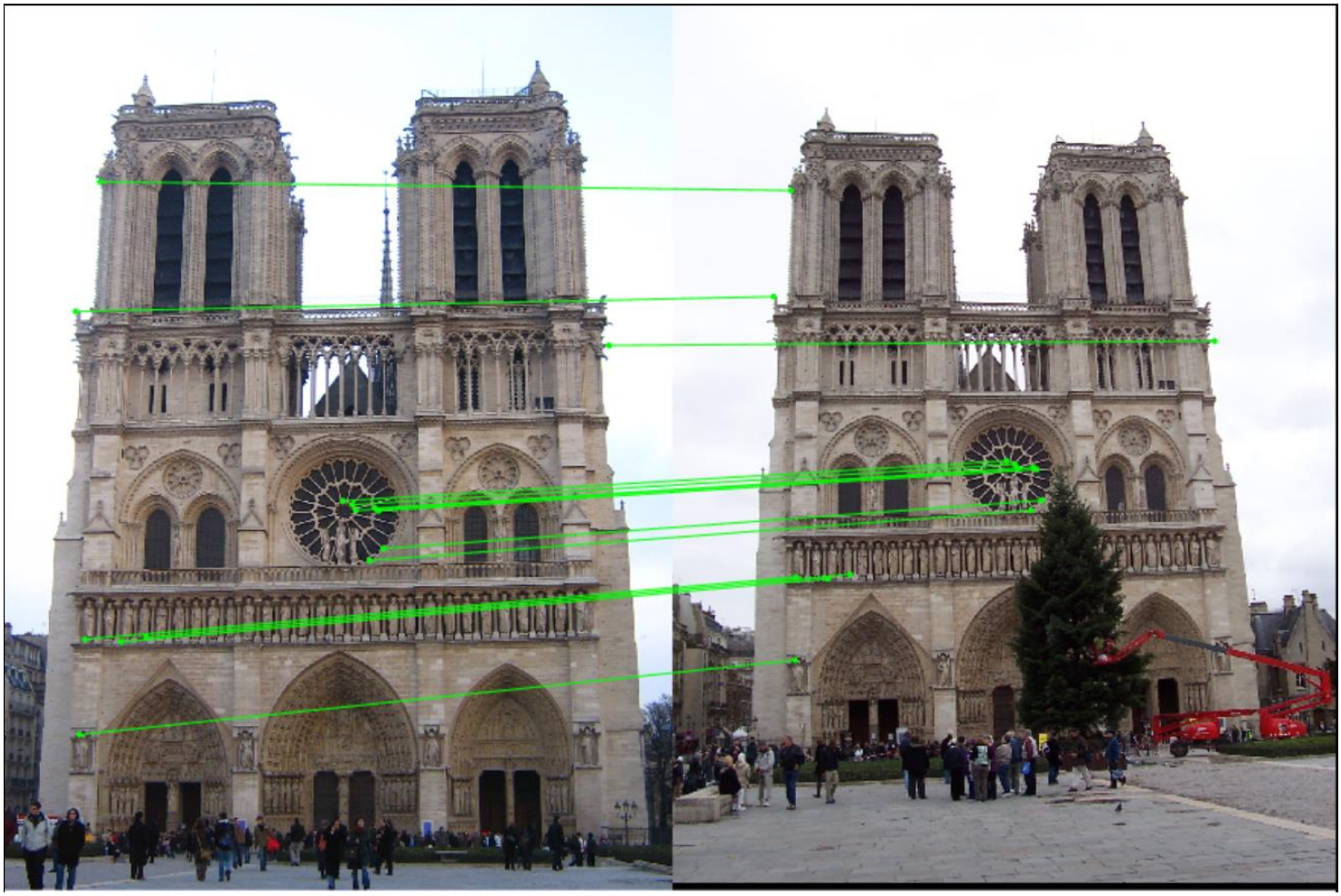
For a threshold of 0.7 in the tester, here is the table of how many matches I obtained with Harris corner detection algorithm and SIFT feature pipeline below (Almost all the matches are good matches as far as I can validate with my eyes):

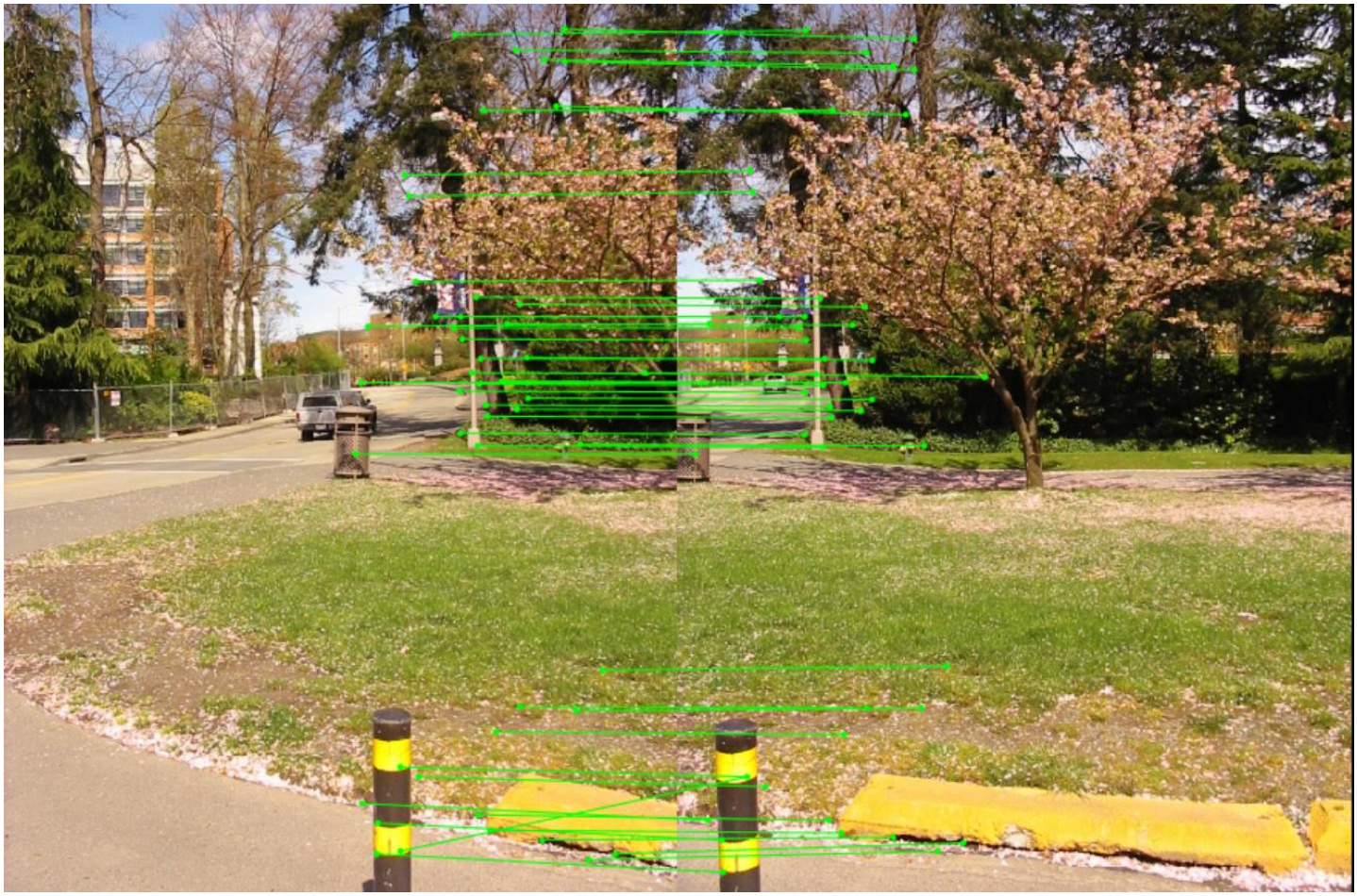
Image Set	# of actual Harris corners	# of matches with Harris corners	# of actual SIFT Descriptors	#of matches with SIFT descriptors
yosemite1	293	69	274	89
yosemite2	266		235	
yard1	138	34	168	37
yard2	91		115	
shrek_reference	224	0	228	5
shrek_test	97		437	
uttower1	174	22	334	60
uttower2	183		323	
trees_002	522	50	355	50
trees_003	655		367	
notredame1	1341	23	3505	145
notredame2	1448		2866	

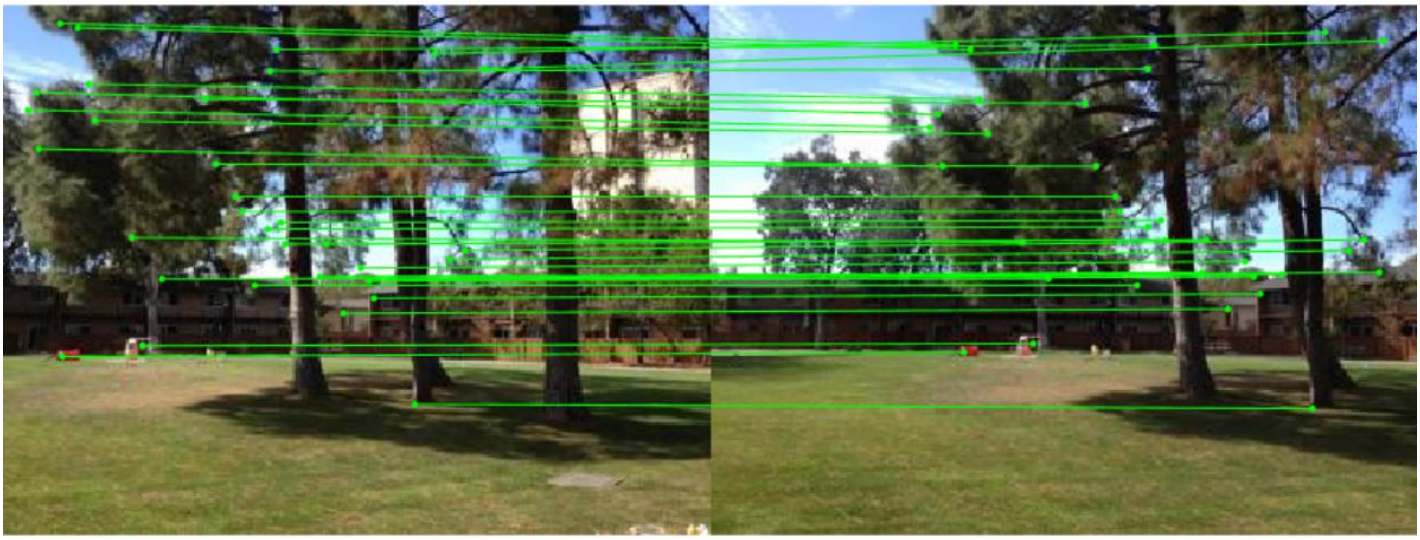
The values above are for a high value of threshold for implemented script. If I decrease the Harris threshold value, the number of good matches increases. In intact images such as "Yosemite" or "yard", it is really working as good as (better in some cases since I detect lots of corners in images) already given feature detection code. Note that the images of Notre Dame Church were adding a bit of rotation and scale changes. In fact, our Harris detector and the pipeline we use for it is not invariant to scale at all. However, the pipeline still works for some corners, possibly where those variances do not matter much. For this image, we can roughly say that there are three times more good matches existing in SIFT implementation. In all the other images, number of corners/descriptors and number of matches mainly change as I vary Harris corner algorithm threshold (I use a value of 5 in normal) and matcher threshold (set to 0.7 normally). As shown in the table below, the number of corners and descriptors increase with the increased image resolution. My code's performance drops to zero when it comes to a completely scaled and rotated image. Other than that, the ratio of matches and detections is similar.

As you can see in my corner detection outputs in the first part of this report, there are some unnecessary detections, while this is well balanced and implemented in SIFT feature algorithm. In two different folders of my submission, all detections and all matching visualizations are provided. Here are a few of my matching outputs using uniscale Harris corner based algorithm:

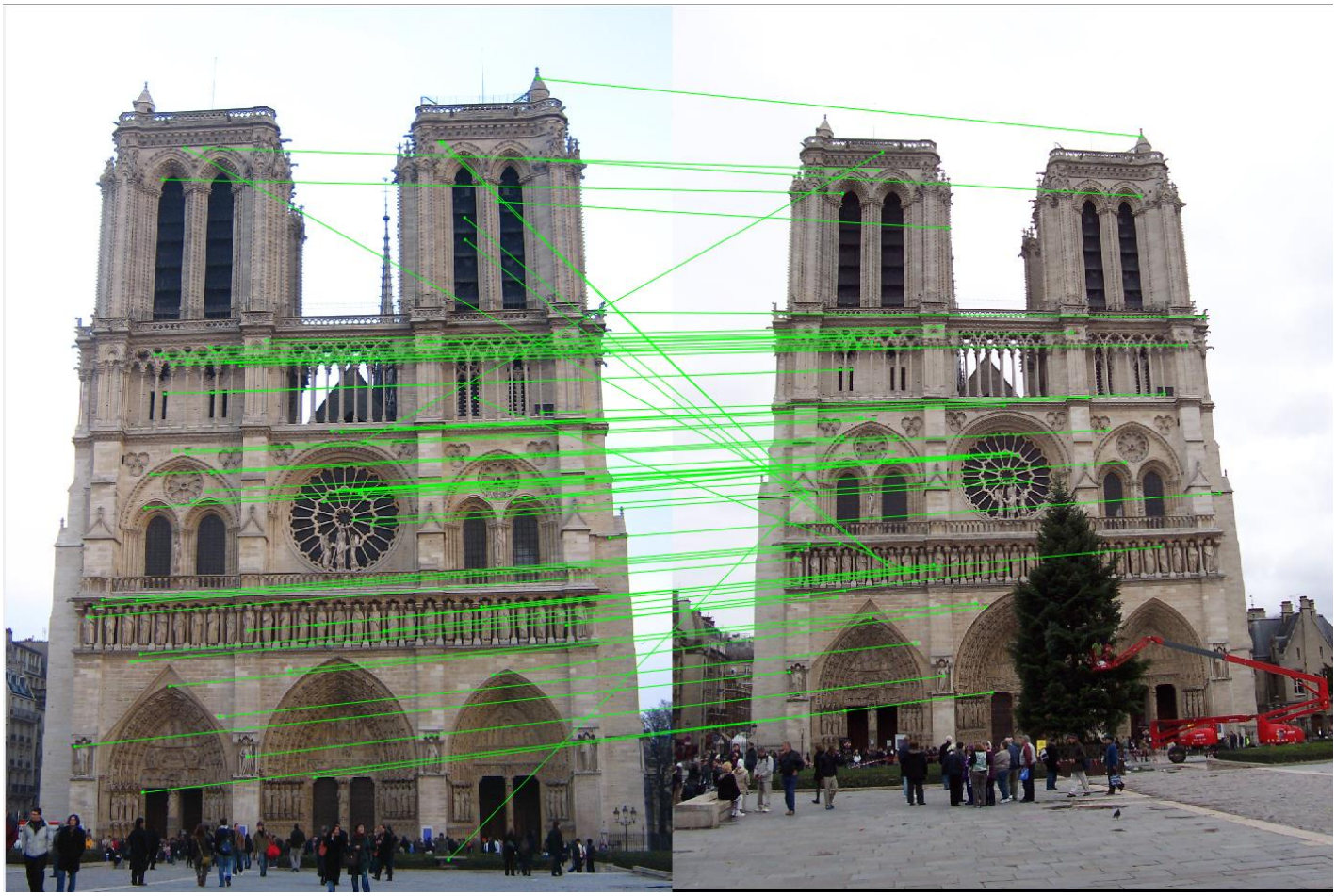


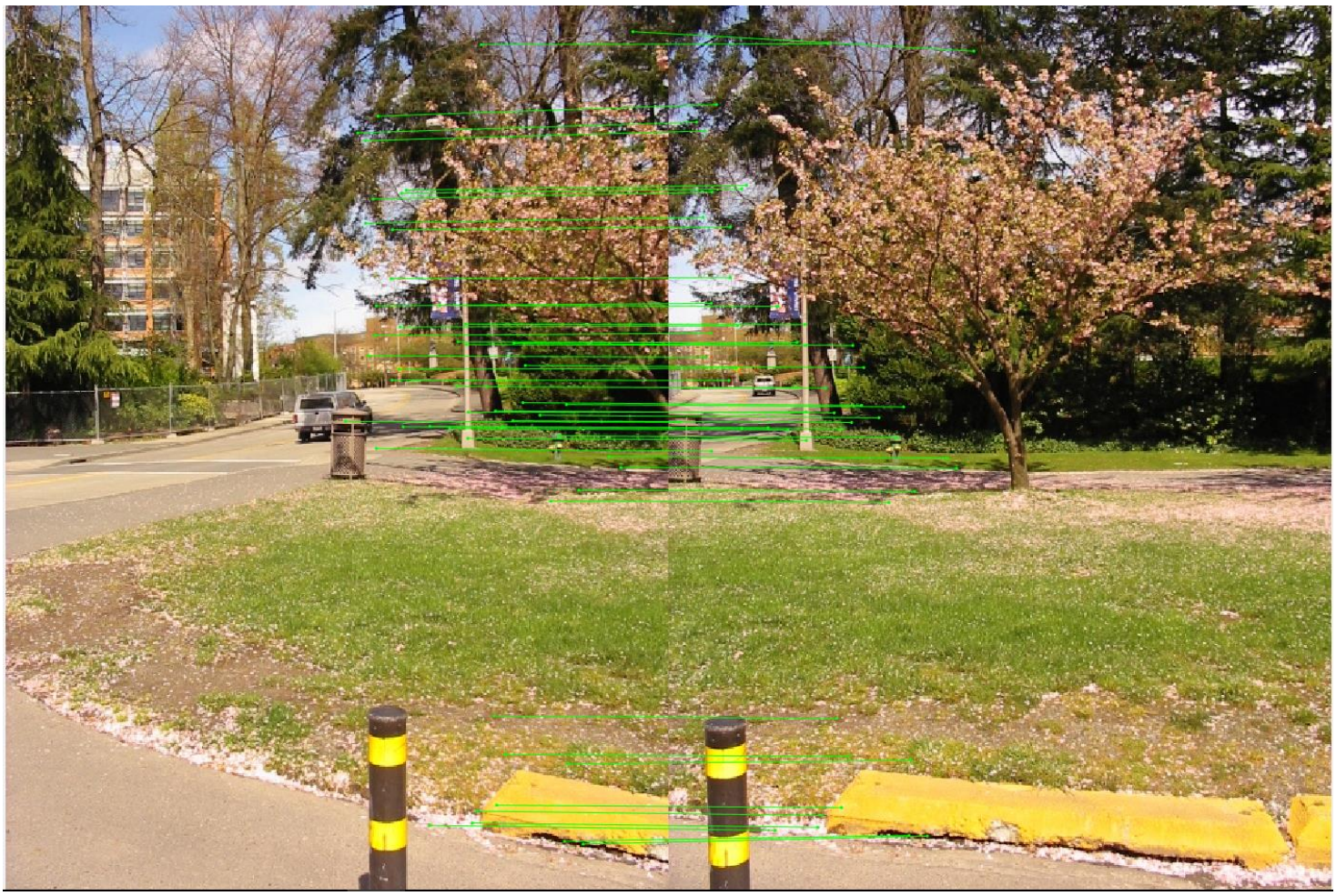


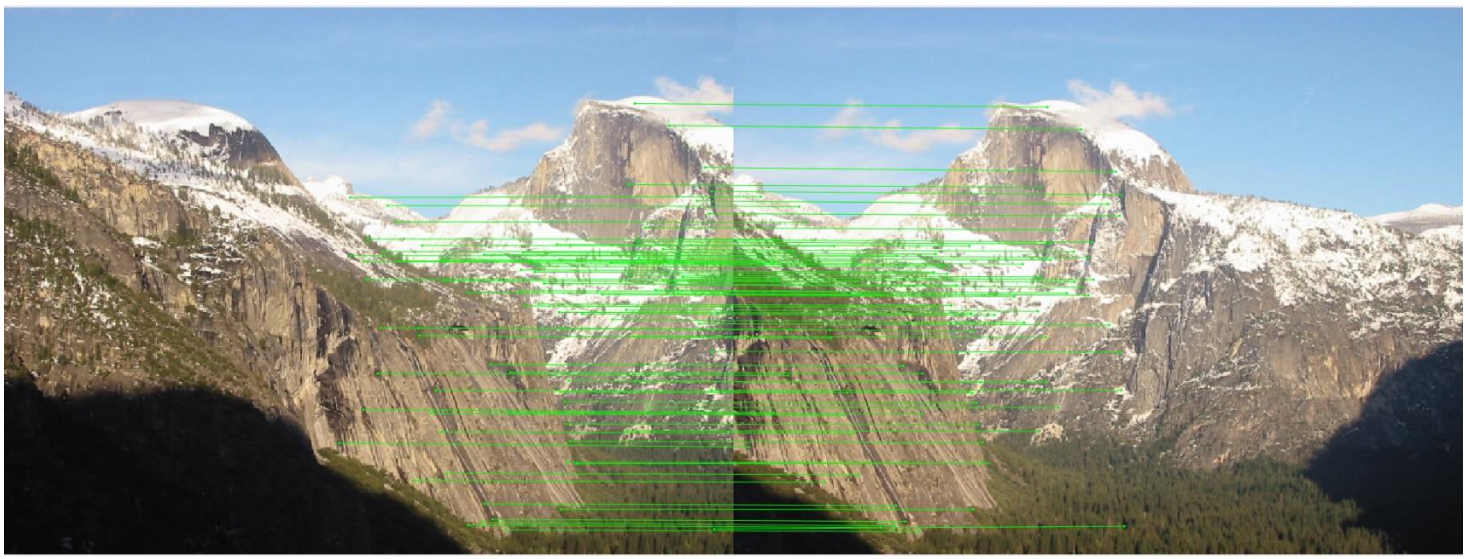




Here are the matching results obtained using "detect_features.m" and the remaining pipeline (Note that all of the results are put in the folder MatcherTester). This is important since descriptor extraction and the matching function was left to students to implement:







6) Comments and Evaluations

We see the biggest difference between the Notre Dame output of two feature detection algorithms, since the scale change is obvious between the two images being compared. Also note that scale invariant pipeline has some mismatches, probably due to the simplicity of the matching algorithm. We can easily see that scale invariant pipeline has more matches in almost each image, probably because of detected corners in various scales. However, with a threshold which lets an increased number of corners, my algorithm also returns as many matches as the scale invariant pipeline. Furthermore, since in all images the views are quite far, the corners will be intuitively small, and our uniscale detector favors this. If the objects were too close or too large, it would have difficulties in finding the "correct" corners.

The hardest matching for both pipelines was the "shrek" case, in which there are two completely different (highly rotated and highly scaled) images. Although it is impossible to extract matches for my pipeline (with 0.7 threshold value), scale invariant pipeline still managed to achieve a few matchings, provided in my submission. There are lots of mismatches with increasing values of threshold, so I decided to keep the threshold the same and not to include those results.

In conclusion, my algorithm's performance is around 60-70% of the provided detection algorithm, and scale invariancy mainly creates the difference. In order to capture corners better, the Gaussian smoothing at the beginning of my corner detection function could be reduced. In my submission, the final values are delivered so that they yield the most acceptable results.

You can find my written assignment below:

COMP408 HW#2 Written Assignment, Arda KIRKATAC 49799

We were given k observations of N-dimensional random vector f.
We want to show that the principal eigenvector of the correlation matrix of f is equal to the solution of the following optimization problem:

$$v_1 = \underset{\|v\|=1}{\operatorname{argmax}} \sum_{k=1}^{K} (f_k v^T)^2$$

$$E\left[\sum_{k=1}^{K} (f_k v^T)^2\right] = E\left[\sum_{k=1}^{K} v f_k^T f_k v^T\right] = K\, v\, E\left[\frac{\sum f_k^T f_k}{K}\right] v^T = K\, v \Sigma v^T$$

We need to maximize $v \Sigma v^T$ above, where Σ is a positive semi-definite matrix

$\underset{\|v\|=1}{\operatorname{argmax}} (v \Sigma v^T)$ corresponds to V_1 in principle components analysis.

Proof: The matrix Σ above can be written as $\Sigma = U^T D U$ for a unitary matrix U. Here, the matrix D is diagonal with entries ≥ 0. If we substitute $V = xU$, then $\|x\| = 1$ since U is also unitary.

$U^T D U \rightarrow x U U^T D U U^T x^T = x D x^T$ (I, I)

j^{th}: $\delta_j = [0 \ \dots \ 1 \ \dots \ 0]$

Lagrange multipliers ($\max \ v \Sigma v^T = x D x^T$ subject to $x x^T = 1$)

- $\mathcal{L}(x, \lambda) = x D x^T - \lambda (x x^T - 1)$ ↓ partial derivative wrt x.
- $\nabla_x \mathcal{L} = 2 D x^T - 2 \lambda x^T = 0$

$\rightarrow D x^T = \lambda x^T$ (eigenvalue, eigenvector) and this means $x = \delta_j$ for some j. (explanation below, D diagonal, $\|x\| = 1$)

- $v \Sigma v^T = \delta_j D \delta_j^T = D_{jj}$, then $D_{jj} = \max_p D_{pp}$, and this last expression shows that v will be the principal eigenvector of Σ.

$\|x\| = 1$

$\rightarrow D x^T = \lambda x^T \rightarrow x D x^T = \lambda \|x\|^2 = \lambda$. This shows that we need to maximize λ and we can pick x only as the vector corresponding to the highest eigenvalue of D (hence eigenvector of D)

Therefore $x = \delta_j$ for D_{jj} (highest eigenvalue of D)