Computer Vision

Homework Assignment 2 : Feature Extraction

Autumn 2018

Nicolas Marchal

Feature Extraction (Harris Corner)

• I used the Matlab function `gradient` to compute the gradient.

Note : I slightly blurred the image using the function `imgaussfilt` to get better results.

• To compute the Harris response, I first used a double `for` loop to find the matrix H for every pixel. I got the Harris response using $K = \frac{det(H)}{trace(H)}$ for each pixel. This implementation gave excellent results but the computation took around 90 seconds per image. To improve the performances, I used the function `movsum` and avoided going through any `for` loops (not even the determinant). This took down my computing time to less than a second per image (see table 1).

I then plot a histogram (see figure 1) to see where most values of K are located. I will only keep a few hundreds points with the highest value of K. Using the histogram, I set a threshold $\lambda = 0.004$ for potential corners.

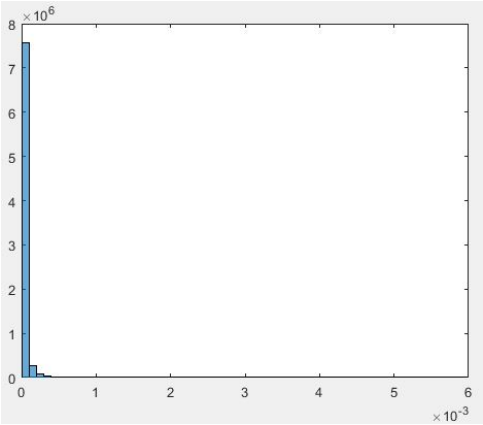
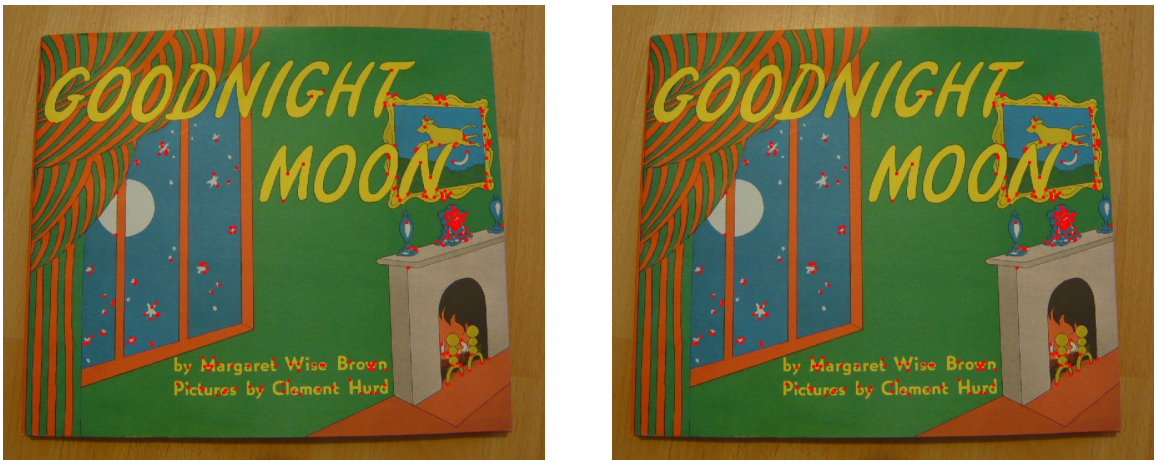


Figure 1: Histogram of values of K (truncated at 0.007)

• Finally I apply a Non-Maximum-Suppression in a 3x3 square area around potential corners to have my final corners. I also forbid any point closer than 4 pixels to be border to be a valid corner. This is done to simplify the work when we later chose a 9x9 descriptor (avoid the descriptor being out of the picture). The results are given below :

	Image 1	Image 2
Time to compute Harris Corners before optimization (seconds)	94.84	92.29
Time to compute Harris Corners after optimization (seconds)	0.6586	0.5697
Number of potential corners (before Non-Max-Supp)	825	2570
Number of corners (after Non-Max-Supp)	280	656

Table 1: extractHarrisCorner.m performance



(a) Image 1 (b) Image 2

Figure 2: Result from Harris Corner Detection

Feature Descriptor

We use a very simple descriptor : which is a 9x9 patch centered around each pixel where a corner was identified (in function `extractDescriptor`).

SSD Feature Matching

I used a simple approach to match the descriptors. For every corner in image 1, I look for the most similar corner in image 2. This is done by calculating the sum of squared difference (SSD) between a corner in image 1 and all the corners in image 2. The corner in picture 2 that gives the smallest SSD is a potential match with the one in the first image. I need to chose a threshold under which I agree that the corners are similar enough to be matched. To select this threshold, I plot the histogram of the values of the smallest SSD for all corners and I decided that 0.1 is a reasonable threshold (see figure 3).

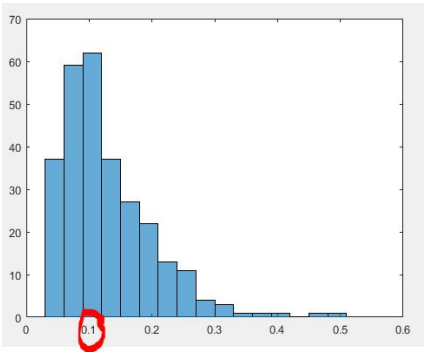


Figure 3: Histogram of SSD values

This worked well but some corners are very similar (see gigure 5) and the matching was thus not perfect. To get a perfect matching I introduced a criteria based on the ratio of SSD :

$$T = \frac{SSD(C_1, C_2)}{SSD(C_1, C_2')} < 0.4 \quad \Rightarrow \quad \begin{cases} C_2 \text{ is the best SSD match for corner } C_1 \\ C_2' \text{ is the second best SSD match for corner } C_1 \end{cases} \tag{1}$$

If there exist 2 similar corners, this ration will be close to 1. It is important to avoid matching ambigious corners, in order to have a perfect matching. This is especially crucial in this photo where corners of letters (see figure 5a and 5c) and stars (see figure 5b) are very similar.

When we choose T = 0.5, we get 52 matches, but there is 4 mistakes as seen in yellow on figure 4. Most mistakes come from the fact that there are still too many ambigious corners, especially with the text (see figure 5)

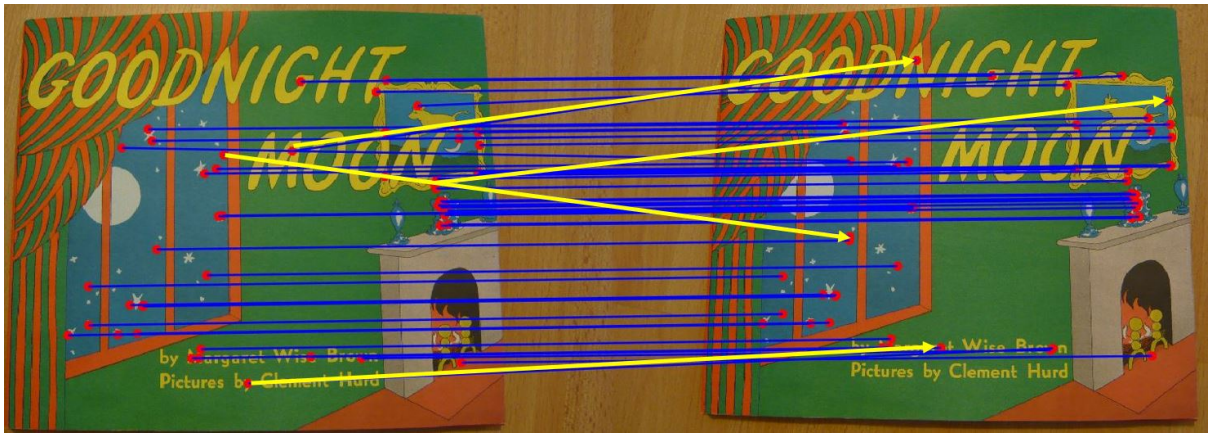
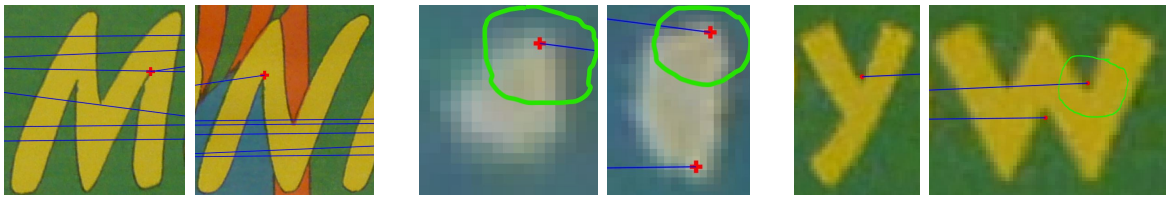


Figure 4: Almost Perfect Matching (T = 0.5)



(a) M and N (b) Small Star (c) y and w

Figure 5: Ambigious corners

The choice of T = 0.4 allowed me to have a perfect matching between the image corners as seen in figure 6. Figure 7 is a zoom on some elements on figure6. The precision of the mapping on the clock (figure 7a) is especially interesting.

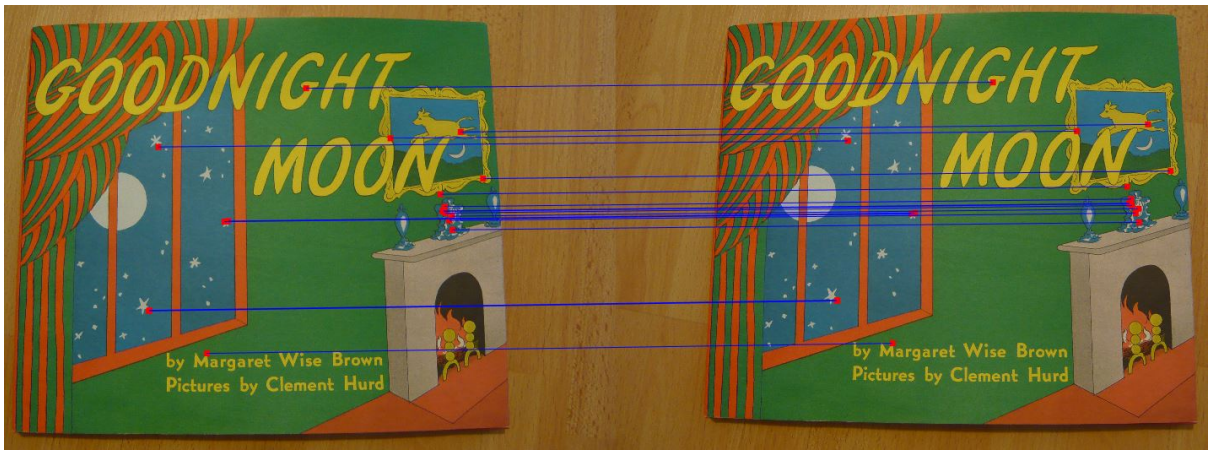
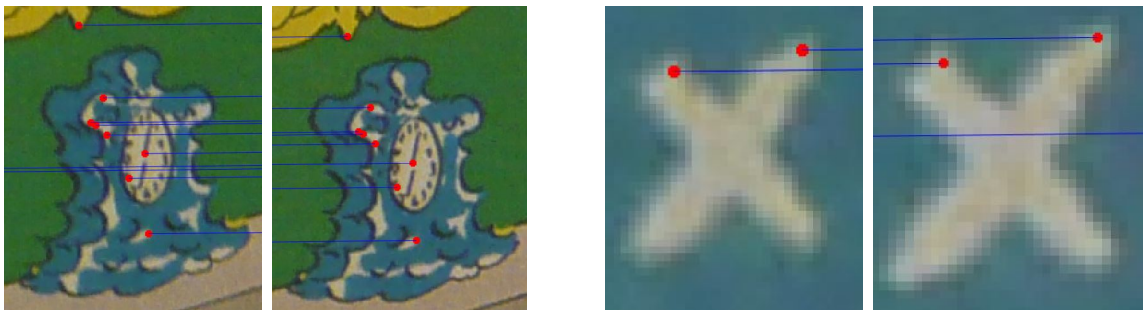


Figure 6: Perfect Matching (T=0.42)



(a) Clock Figure 7: Zoom (b) Star

0.1 SIFT features

Following the tutorial, we can get a very good matching using the SIFT detescriptor available at `http://www.vlfeat.org`.. The result is given below (figure8) :

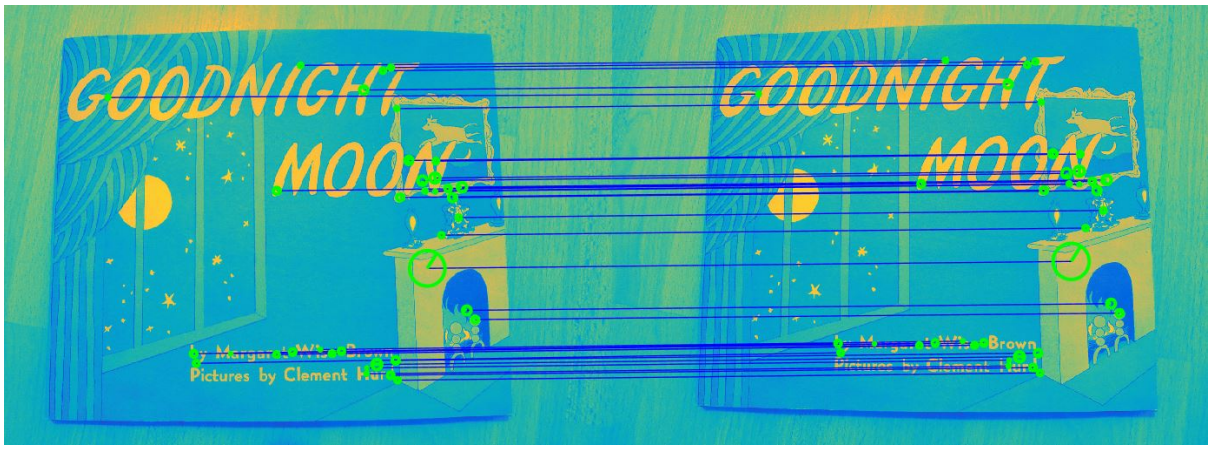


Figure 8: Sift Descriptor Matching 43 points (Threshold = 300)

We see that once more, all the points are on the drawing (none are from the background). This SIFT detector was able to match more points from the text and less from the clock. The biggest difference is that by adjusting the threshold in the function `vl_ubcmatch` we can obtain much more points having a 100% correct assignment. Figure 9 has so many matches (364) that we can not distinguish them all. However you can see that all the lines are parallel so there is 100% correct match.

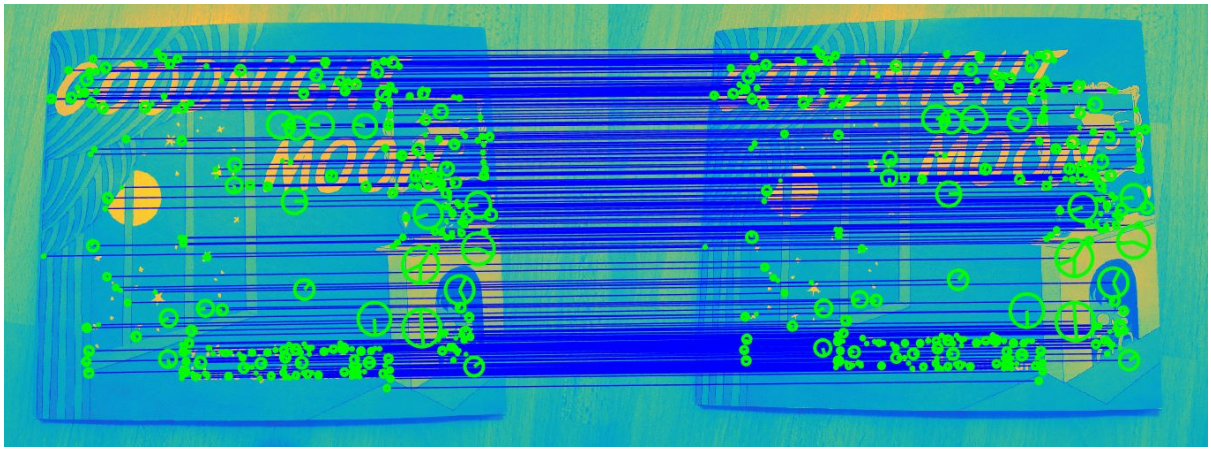


Figure 9: Sift Descriptor Matching 364 points (Threshold = 75)