Computer Vision
Exercise 1

Xingze Tian

October 5, 2017

1 Harris corner detection

I implemented the feature extractor in `extractHarrisCorner.m` with the following steps:

- **Smooth the image to reduce noise**
  I used matlab function `imgaussfilt(A,sigma)` to perform this step. This function *filters image A with a 2-D Gaussian smoothing kernel with standard deviation specified by sigma*[1]. Alternatively the function `fspecial('gaussian')` can be used which allows user-defined kernel size. For simplicity, the previous approach is taken.

(a) original (b) sigma = 0.5 (c) sigma = 2 (c) sigma = 5

Figure 1: Effects of blurring with variations in sigma

  With the variations in image size, different sigmas should be chosen. The bigger the sigma value, the more blurred the image would be.

- **Compute gradients in x and y directions**
  The matlab function `imgradientxy(I)` is used, which *returns the directional gradients,* I_x *and* I_y*, the same size as the input image* I[2].

- **Compute Harris response at each pixel**

  1. **Fill points outside the boundaries as 0**: to avoid index-out-of-matrix problem, I preset the values outside the boundaries of I_x and I_y as 0. This is done through inserting the original matrix into an all-zero matrix with an addition column or row on each boundary.

  2. **Compute Harris matrix**: for each pixel, I retrieved the gradients I_x and I_y of every pixel in its 3x3 neighbourhood. Then the Harris matrix is calculated as:

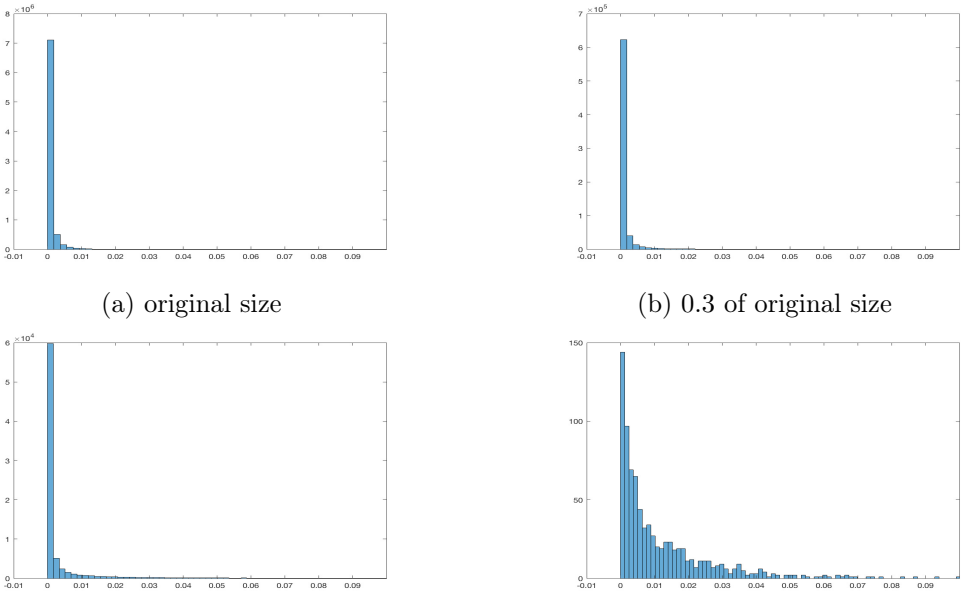
$$H = \sum \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_x}^2 \end{bmatrix}$$

  3. **Compute Harris response**: for each Harris matrix, the Harris response is thus calculated using matlab functions `det()` and `trace()`:

$$K = \frac{det(H)}{trace(H)}$$

[1] https://ch.mathworks.com/help/images/ref/imgaussfilt.html
[2] https://ch.mathworks.com/help/images/ref/imgradientxy.html

- **Threshold the response image**
  The input image can be resized to speed up the computation process. To determine the threshold value, I plotted the Harris response histograms based on the input image sizes:



(a) original size

(b) 0.3 of original size

(c) 0.1 of original size

(c) 0.01 of original size

Figure 2: Harris response histogram of different image size

It is clear from the histograms that the more compressed the image is (fewer pixels after resizing), the more difficult it is for the Harris response to converge and thus harder to decide the value of the threshold.

Based on the histogram, I set the threshold to be 0.05. The images after threshold are shown in Figure 3:

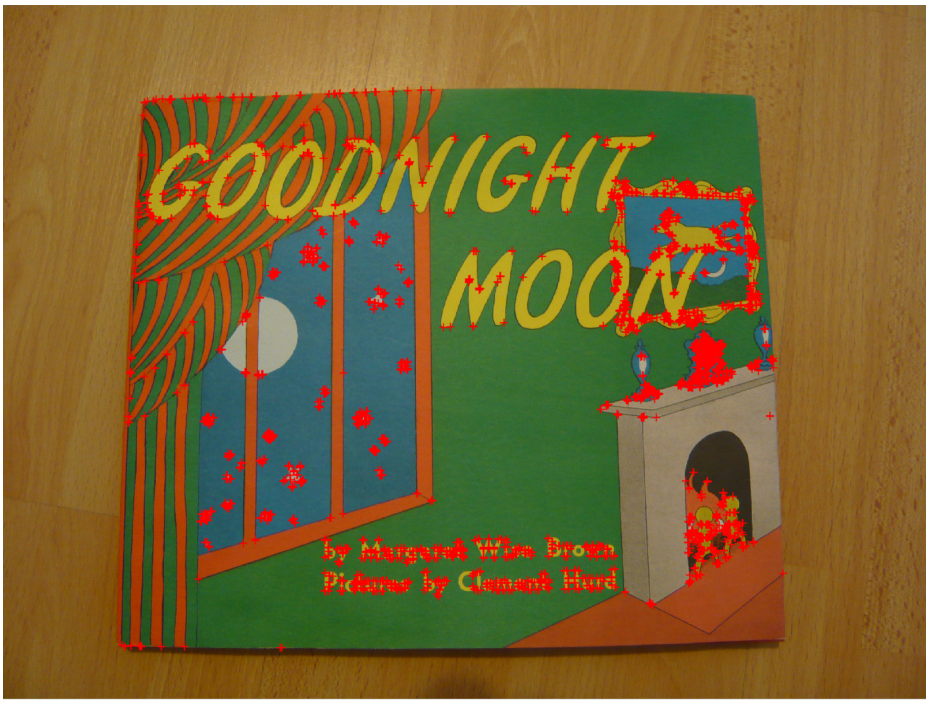


Figure 3: Before non maximum suppression: image size = 1, threshold = 0.05

- **Apply non-maximum suppression**
  For each pixel bigger than the threshold, I compared its value within its 3x3 neighbourhood. To avoid out-of-index problems, I filled the points outside the boundaries as 0 using the same technique specified in step 1. For every pixel with the largest value in its neighbourhood, I extracted it as the key point and added its coordinates to the corners matrix.

  The resulting image is as shown in Figure 4:

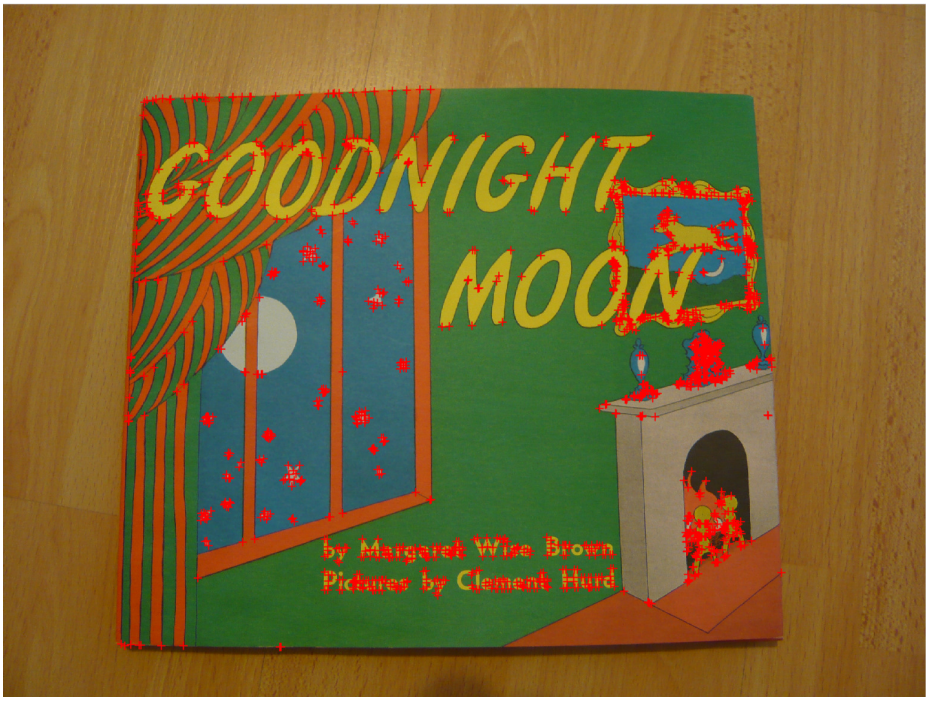
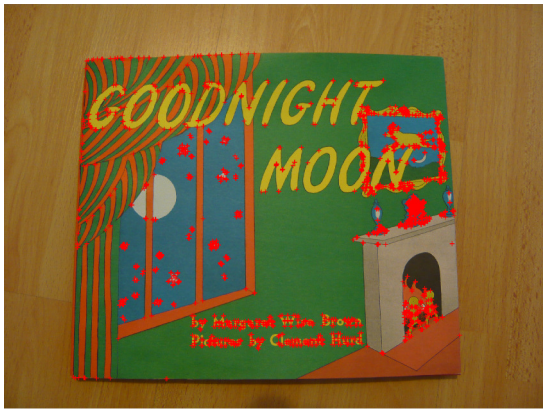
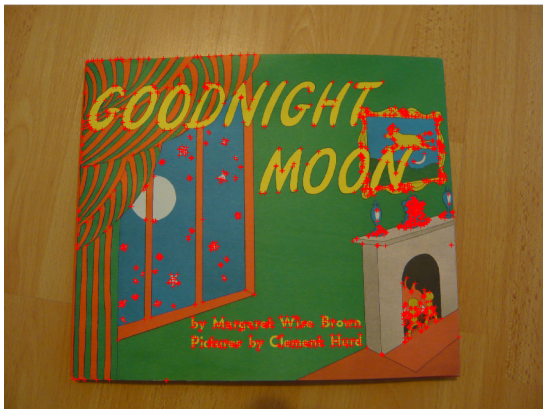


Figure 4: After non-maximum suppression:image size = 1, threshold = 0.05

The comparison of features extracted with and without non-maximum suppression is shown in Figure 5:



(a) before non-maximum suppression



(b) after non-maximum suppression

Figure 5: Image size = 1, threshold = 0.05

It can be seen in Figure 5, Figure 6 and Figure 7 that the image before applying non-maximum suppression is "redder", i.e. includes more feature points than the image after non-maximum suppression. In particular, for segments with more corners (for example the characters at the lower part and the fireplace), the non-maximum suppressed image contains fewer repeating or overlapping points. This is because for segments with more corners, the pixels around the

(a) before non-maximum suppression (b) after non-maximum suppression

Figure 6: Image size = 0.5, threshold = 0.05

(a) before non-maximum suppression (b) after non-maximum suppression

Figure 7: Image size = 0.1, threshold = 0.05

real corner generally have higher Harris response. In this case, non-maximum suppression helps to distinguish between the key points just around the corners and the real corners, and thus gives more accurate results.

2 Image patch extraction

The patch descriptor is implemented in `extractDescriptor.m`. To avoid index-out-of-matrix problem, similar approach is used as before by filling the points outside the boundaries as 0 (in this case, it is needed to add 4 extra columns/rows on each side as we want to extract a 9x9 patch). For each key points detected in part 1, I extracted its 9x9 neighbourhood to a descriptor vector and appended the vector to the returning matrix.

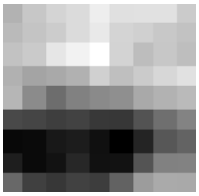


Figure 8: An example patch extracted from Figure 3

3 Feature matching

The descriptors are matched in `matchDescriptors.m`. For each key point in image1 I computed its sum of squared differences (SSD) with every key point in image2 using the matlab expression `sum(difference(:).^2)`, where difference denotes the difference between each entry of the two descriptors.

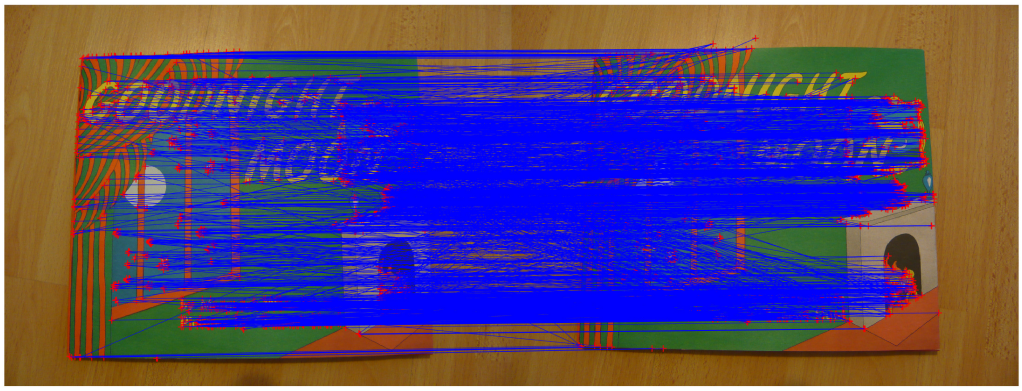
For each SSD value, I first compared it with the threshold. If the SSD is smaller than the threshold, I then searched the point which gives the smallest SSD. I used two variables `minimum` and `match_candidate` to keep track of the comparison. For each key point in image1, the key point in image2 that gives the smallest SSD below the threshold would be selected, and the index pair would be added to the resulting matrix.

Ideally, for two key points to match, the SSD should be 0, and the closer the threshold to 0, the more accurate the matching would be. To determine the best value of the threshold, I experimented on several different settings:

(b) Threshold = 0.05

(c) Threshold = 0.1

It is clearly shown from the graphs, when threshold is 0.05, some matches are found but quite a few features are missing; when the threshold is 1, too many features are matched and some of them are quite inaccurate. In this case, I set my threshold to be 0.1 which gives sufficient good matching features.



(c) Threshold = 1

Figure 9: Feature matching results using different thresholds

4 Comparison with SIFT

Based on the tutorial of VLfeat, I first turned the image to the required format using `single()`, then applied `vl_sift()`.

4.1 Detector Comparison

1. **Number of Features**
   SIFT detector detects similar amount of points when the threshold of the Harris detector is set to be 0.005, and thus gives more detected features (my threshold is 0.05).

(a) My Harris Detector (b) SIFT Detector

Figure 10: Image size = 1

2. **Feature Point Locations**
   Both of the detectors can detect most of the corners in the image. However, as presented in the histograms in part 1, at threshold 0.005 a large amount of points would be detected, but also less accurate. Although the SIFT detector missed fewer features, it contains more noise than my Harris detector and detected more non-corner points as feature points. In a filtered version (see Figure 12 b) the accuracy has improved, and the SIFT detector gives quite good results.

3. **Scale Invariant**
   As shown in Figure 5, 6 and 7, with the change in the size of the image, the number of detected points changes a lot accordingly. SIFT presents more tolerance with the scale change: With

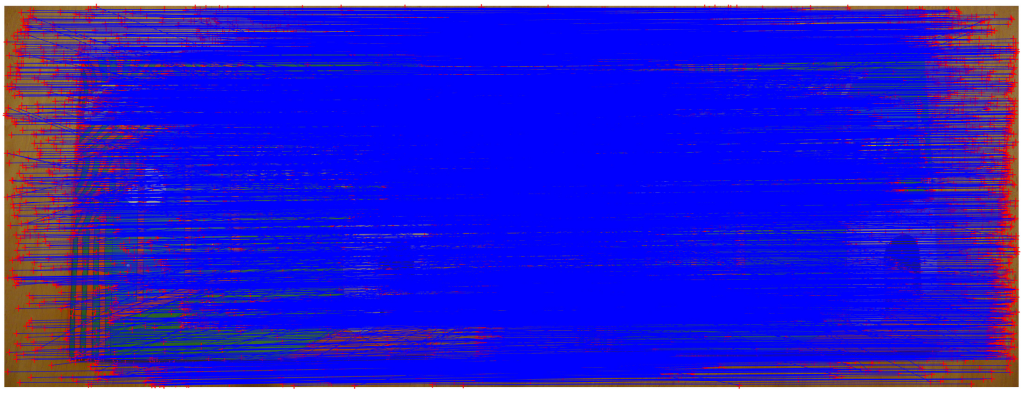
(a) Image size = 1 (b) Image size = 0.5 (c) Image size = 0.3

Figure 11: Image size = 1

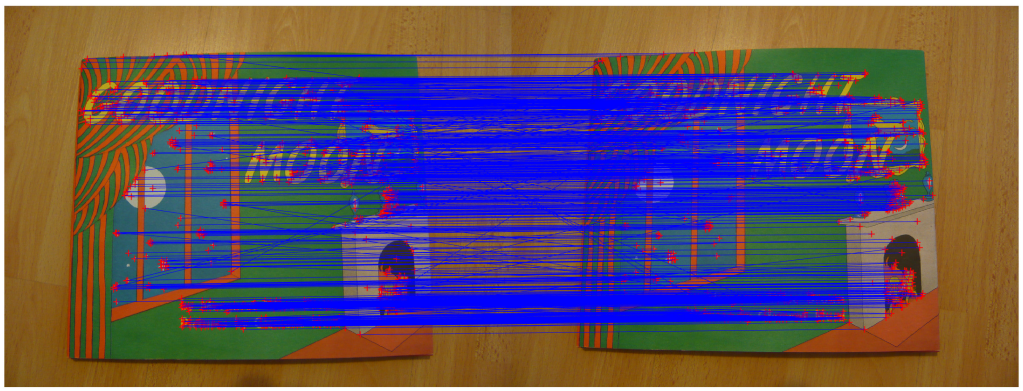
the change in image size, the number of features detected doesn't change much. Thus SIFT is more scale invariant then the Harris detector.

4.2 Matching comparison

SIFT gives better matching than the Harris detector in both quantity and quality. As can be seen in Figure 9 and Figure 12, SIFT finds more matchings and the matching lines are more parallel (which indicates more accurate matchings).



(a) No peak threshold



(b) peak threshold = 0.02

Figure 12: SIFT results with variations in peak threshold

The peak threshold *filters peaks of the DoG scale space that are too small (in absolute value)*[3].

[3]http://www.vlfeat.org/overview/sift.html

5 My SIFT Descriptor

My SIFT descriptor is implemented in `extractSIFTDescriptor.m`, though due to time limit, a small part is unfinished. This function takes in the extracted corners using the Harris detector and returns the descriptors for each key point in a matrix, and the descriptors are matched using the same technique as in part 3.

For every pixel around the 16x16 neighbourhood of each key point, I calculated the gradient orientations (`atan(Iy/Ix)`) and magnitudes (sqrt(Ix^2 + Iy^2)). Then I divided the 16x16 matrix into 4x4 blocks, where each block is a 4x4 matrix. For each block, I constructed a weighted histogram of the orientations using the magnitudes as weights in 8 bins, applied Gaussian filter to it and appended the weighted histogram to my descriptor patch.

And finally the 1x128 patch is added to the returning matrix. The only part I didn't have time to implement is to organize the orientations into the blocks which can be easily added in future time. The resulting matching in theory should be better than the Harris descriptor, as first it contains more dimensions and thus a richer definition of the neighbourhood, and second as the descriptor analyses the neighbourhood more carefully, a more accurate description should be extracted.