Computer Vision Lab Assignment 1 Report

lijiaj@student.ethz.ch

October 2023

The whole assignment consists of two tasks: (1) Harris Corner Detection and (2) Feature Description & Matching, where the second task depends on the results of the first task. Therefore, I will organize the report following the same structure.

1 Harris Corner Detection

Harris corner detection is an algorithm for detecting corners in an image and can be divided into four main steps as suggested in the given lab slides.

The first step is to compute the image gradients that represent the intensity changes both in horizontal and vertical directions. The specific formulas for calculating the image gradients are shown in eq. (1) and eq. (2). When implementing the computation of image gradients, I employed the convolution operation with two 3x3 convolution kernels in eq. (3) and eq. (4).

$$I_x = \frac{I(x+1, y) - I(x-1, y)}{2} \tag{1}$$

$$I_y = \frac{I(x, y+1) - I(x, y-1)}{2} \tag{2}$$

$$\text{kernel}_x = \begin{bmatrix} 0 & 0 & 0 \\ -0.5 & 0 & 0.5 \\ 0 & 0 & 0 \end{bmatrix} \tag{3}$$

$$\text{kernel}_y = \begin{bmatrix} 0 & -0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix} \tag{4}$$

Next, we need to blur the image gradients and then construct the auto-correlation matrix M. The main reason behind the image gradients blurring is image gradients are very sensitive to noise, which may result in false corner detection later in the algorithm. The auto-correlation matrix M contains the information about changes in intensity values along all directions from I_x and I_y. The auto-correlation matrix M is defined in eq. (5), where $w(x, y)$ is the Gaussian window function for blurring. The Gaussian window function has a hyperparameter σ, namely the standard deviation. In the code, I first computed the I_x^2, I_y^2, and $I_x I_y$, then applied the Gaussian window function to each of them, and finally constructed the auto-correlation matrix M.

$$M = \sum_{(x,y)} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{5}$$

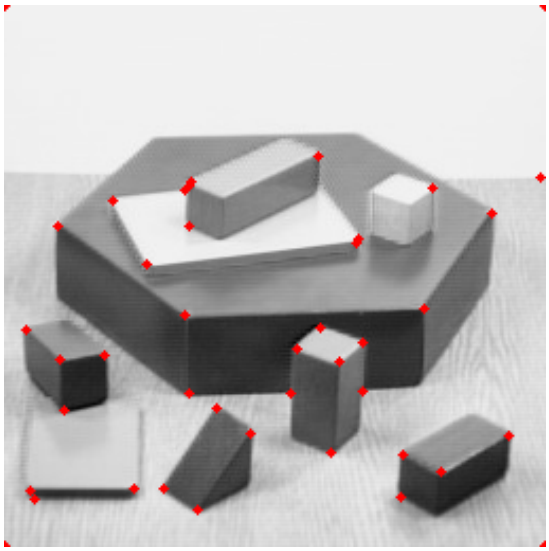
With the auto-correlation matrix M, the Harris response C, a matrix with elements indicating "cornerness" or corner strength of every pixel, can be easily computed. The Harris response C can be characterized by the determinant and trace of the auto-correlation matrix M as in eq. (6), which makes its computation much more efficient. The k in the formula is another hyperparameter that we need to choose reasonably.

$$C = \det(M) - k * \text{trace}(M)^2 \tag{6}$$

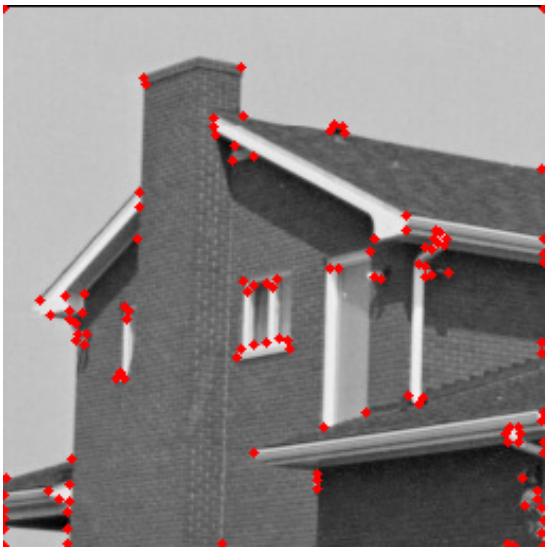
Finally, the non-maximum suppression (NMS) technique needs to be applied on the Harris response C obtained from the previous step to achieve a more refined corner detection result, because not only the real corners but also the pixels around corners can get a high Harris response value. The NMS technique uses a sliding window to find the pixels that have the highest Harris response value in their neighbourhood and consider them as corner candidates. Furthermore, if the Harris response values of these corner candidates are larger than a predefined threshold (another hyperparameter), they are detected as corners.

In general, the Harris corner detection algorithm contains three hyperparameters: the σ value of the Gaussian blur function, k in the Harris response formula, and the threshold t in non-maximum suppression (NMS). The recommended value ranges are σ: $[0.5, 1, 2]$, k: 0.04 to 0.06, and t: 10^{-6} to 10^{-4}. In the given code, the default values are σ: 1, k: 0.05 and t: 10^{-5}.

Although the Harris corner detection algorithm has shown considerable results under the default hyperparameter settings, I observed that some corner pixels were ignored in the "blocks" image and some non-corner pixels were mistakenly marked as corners in the "house" image (in Figure 1). Based on this, I made further attempts to adjust the hyperparameters. I found that when σ is set to 3, k is set to 0.05, and t is set to 10^{-6}, the detection result is significantly improved, as shown in Figure 2.

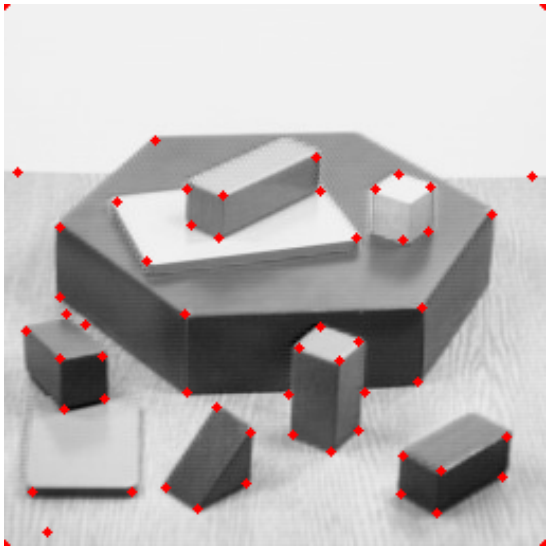


(a) "blocks" image

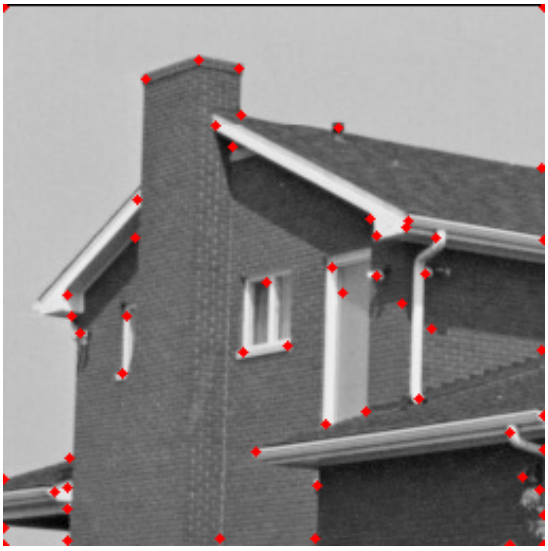


(b) "house" image

Figure 1: Harris detection results under default hyperparameters



(a) "blocks" image



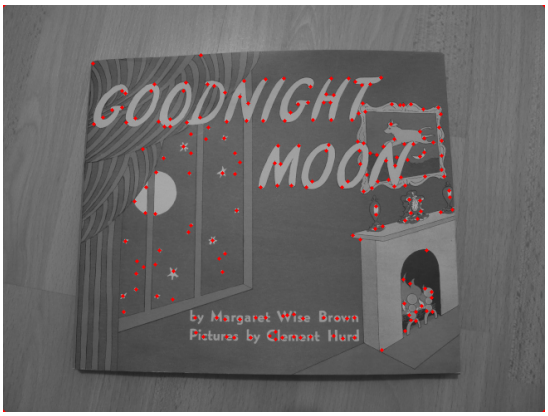
(b) "house" image

Figure 2: Harris detection results under specially-chosen hyperparameters ($\sigma = 3$, $k = 0.05$, $t = 10^{-6}$)

2 Feature Description & Matching

Due to the invariance to transformations such as rotation and scaling, the real corners in the image are considered as good features that can describe the local unique patterns of the image. However, the corner pixels themselves found only by Harris corner detection is not informative enough. In order to better describe these key points, we need to utilize the pixel information in their neighborhoods to construct a more descriptive feature. With these features, we might be able to find close features in different images and match them together, which provides the basis for tasks such as image stitching and object recognition. The Harris corner detection results of the pair of images used for feature matching are below in Figure 3 and we can see most key points are detected reasonably in both images, which makes the feature matching possible.

(a) The first image



(b) The second image

Figure 3: Harris detection results of two images used for feature matching

2.1 Feature Description

Specifically, in the `extract_descriptors.py` file, the `extract_patches` function is designed to extract for each key point its surrounding 9x9 pixel block as features. However, due to the existence of boundaries of the image, there is no complete 9x9 pixel neighbourhood for some key points on the edge. Thus, I implemented the `filter_keypoints` function, which selects those key points that can obtain a complete 9x9 pixel patch by checking if their distances to the image boundaries are at least 4 pixels.

2.2 Feature Matching

After successfully extracting image features, we can try to find similar patterns between them by matching the features in different images. In the `match_descriptors` function of the `match_descriptors.py` file, I implemented three matching strategies, which all employ the Sum of Squared Differences (SSD) defined in eq. (7) as the distance measure. For this reason, I have specially implemented an `ssd` function in `match_descriptors.py` file to compute SSD.

$$\mathrm{SSD}(p, q) = \sum_i (p_i - q_i)^2 \tag{7}$$

First, there is "one-way nearest neighbors matching", which simply matches each feature in image 1 to the nearest feature in image 2. This algorithm was easily implemented by the `argmin` function with parameter `axis = 1` from the `numpy` library. The one-way matching result is in Figure 4.

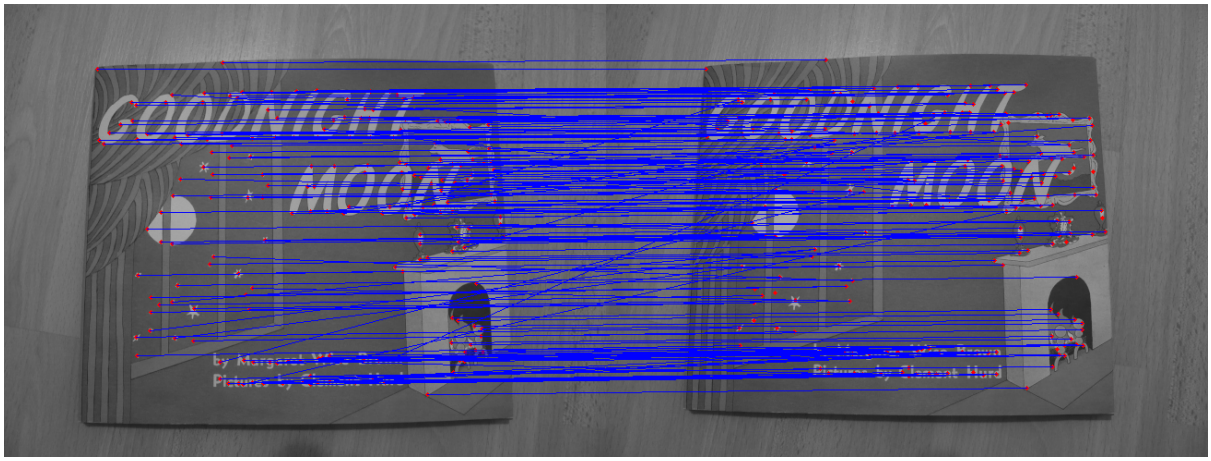


Figure 4: One-way nearest neighbours matching result

Next is "mutual nearest neighbors matching". This strategy performs a "one-way" match from both directions of image 1 and image 2, and only keeps matching pairs with consistent results twice. The code of this strategy was completed by first calling the `argmin` function from the `numpy` library twice with different `axis` values and looping over two returned list to find common matching pairs as the valid ones. Figure 5 is the mutual matching result.

Finally, there is "ratio testing matching", which performs "one-way" matching and further calculates the ratio of the distance of each feature to its nearest neighbor and second-nearest neighbor, retaining only those matching pairs whose ratios are lower than

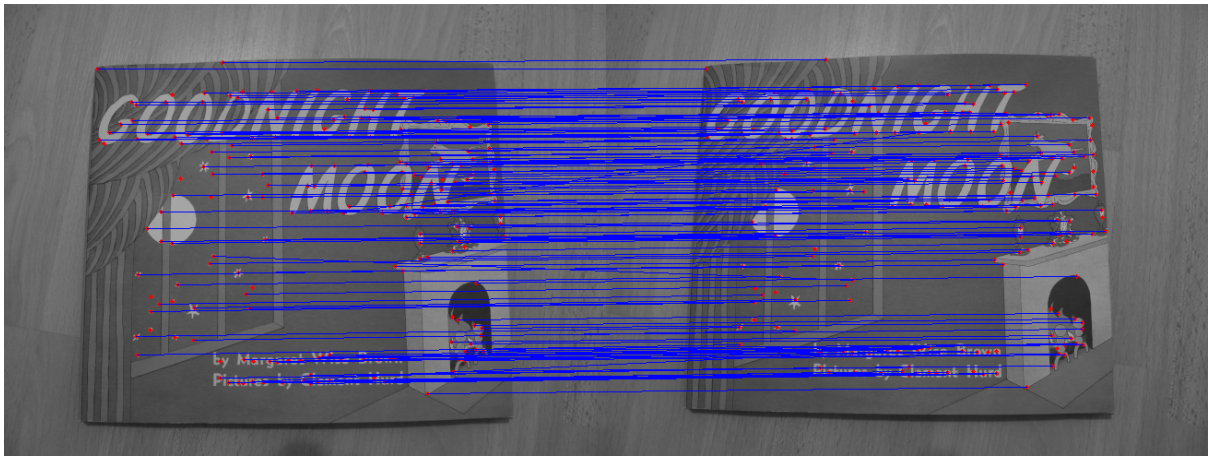


Figure 5: Mutual nearest neighbours matching result

a given threshold. In the code, the most difficult task of identifying the nearest and second-nearest neighbours was tackled by calling the `partition` function from the `numpy` library. The ratio testing matching result is shown in Figure 6 with the hyperparameter threshold set to the default value 0.5.

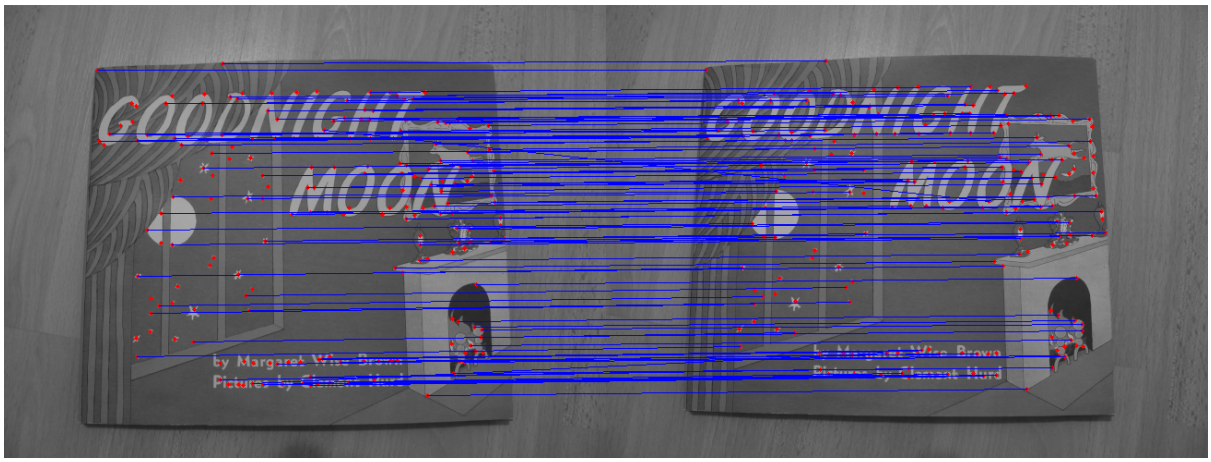


Figure 6: Ratio testing matching result with default threshold 0.5

Comparing the images obtained using three different matching strategies, we can clearly see that there are many intersecting line segments in the "one-way" matching results, which indicates a large number of incorrect matches. Although "mutual" matching reduces some false matches, "ratio testing" matching provides superior results. Moreover, considering that the "ratio testing" matching at the default thresholds already performed quite well, I decided not to tune it further.