Computer Vision Exercise 2: Local Features

Soomin Lee (leesoo@student.ethz.ch)

October 9, 2020

1. Detection

Image gradients Image gradients can be obtained by convolving the image with the Sobel filter:

$$I_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * I, \tag{1}$$

$$I_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * I, \tag{2}$$

which is implemented using `conv2`.

```
sobel_1 = [-1 0 1];
sobel_2 = [1 2 1];

Ix = conv2(sobel_2', sobel_1, img, 'valid');
Iy = conv2(sobel_1', sobel_2, img, 'valid');
```

Local auto-correlation matrix The local auto-correlation matrix can be computed by conducting element-wise multiplications and then filtering it with the Gaussian kernel (standard deviation of σ), which is implemented using `imgaussfilt`.

```
Ixx = double(Ix .^ 2);
Iyy = double(Iy .^ 2);
Ixy = double(Ix .* Iy);

wIxx = imgaussfilt(Ixx, sigma);
wIyy = imgaussfilt(Iyy, sigma);
wIxy = imgaussfilt(Ixy, sigma);
```

Harris response function The Harris response function $C(i,j) = det(M_{i,j}) - kTr^2(M_{i,j})$ is calculated for all pixels by following code:

```
C = (wIxx.*wIyy - wIxy.^2) - k*(wIxx + wIyy).^2;
```

Detection criteria The keypoints are obtained by filtering the pixels with the predefined threshold T, and then choosing local maximum points.

```
score = C;
score(score<thresh) = 0;

BW = imregionalmax(score);
[i, j] = find(BW);
corners = [i, j]';
```

In the above steps, there are 3 parameters that can be tuned, namely σ of the Gaussian kernel, k from the Harris response function, and the threshold T for detecting keypoints. After some manual tuning, following combination of the parameters turned out to be appropriate for the tasks:

$$\sigma = 2, \quad k = 0.06, \quad T = 0.001. \tag{3}$$

This particular setting is used for Figure 1 - 4 and throughout the rest of the tasks. The algorithm is able to detect the corners in general as intended, however, one can notice that there might be some ambiguous matches between them.

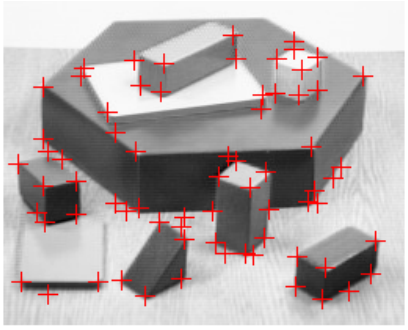

Figure 1: Detected keypoints for Image 1

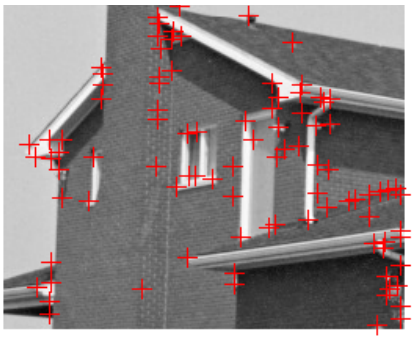

Figure 2: Detected keypoints for Image 2

2. Description & Matching

Local descriptors If a keypoint is too close to an edge, a 9×9 patch around the keypoint cannot be extracted. Therefore, only the keypoints that have at least 4 pixels before the edges are used.

```
r = 4;
[width, height] = size(img);
mask1 = (keypoints(1,:)>=1+r) & (keypoints(1,:)<=width-r);
mask2 = (keypoints(2,:)>=1+r) & (keypoints(2,:)<=height-r);
mask = mask1 & mask2;
keypoints = keypoints(:, mask);
descriptors = extractPatches(img, keypoints, r*2+1);
```

SSD one-way nearest neighbors matching `pdist2` function is used to calculate the SSD between the descriptors:

```
distances = pdist2(double(descr1)', double(descr2)', 'squaredeuclidean');
```

Figure 3: Detected keypoints for Image 3

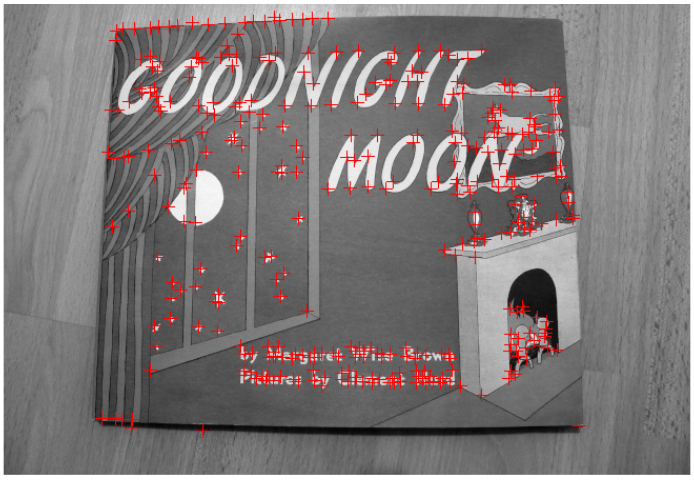

Figure 4: Detected keypoints for Image 4

Then for each feature in the first image, the closest feature from the second image was matched to it.

Mutual nearest neighbors The one-way nearest neighbor matching method is applied to both the first image and the second image, and each match was only included when it is valid in both cases. One can confirm that the number of valid matches decreases (in this particular example, from 343 to 274) compared to those when using the one-way nearest neighbor matching method, and also that some of the inaccurate matches are removed as show in Figure 5 and Figure 6.

```
[~, match1] = min(distances,[],2);
[~, match2] = min(distances,[],1);
matches = [];
for idx=1:length(match1)
    if match2(match1(idx)) == idx
        matches = [matches; idx match1(idx)];
    end
end
matches = matches';
```

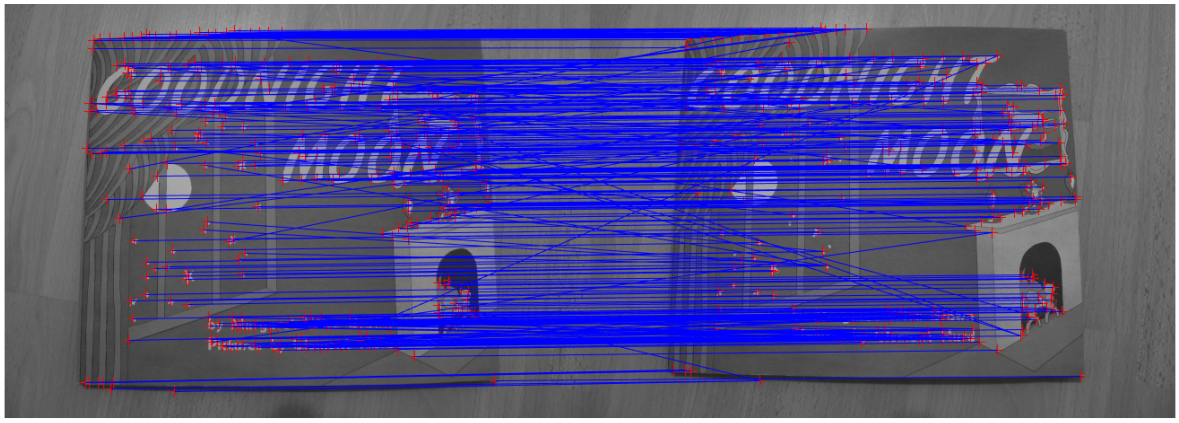

Figure 5: Matching of keypoints when using the one-way nearest neighbor method.

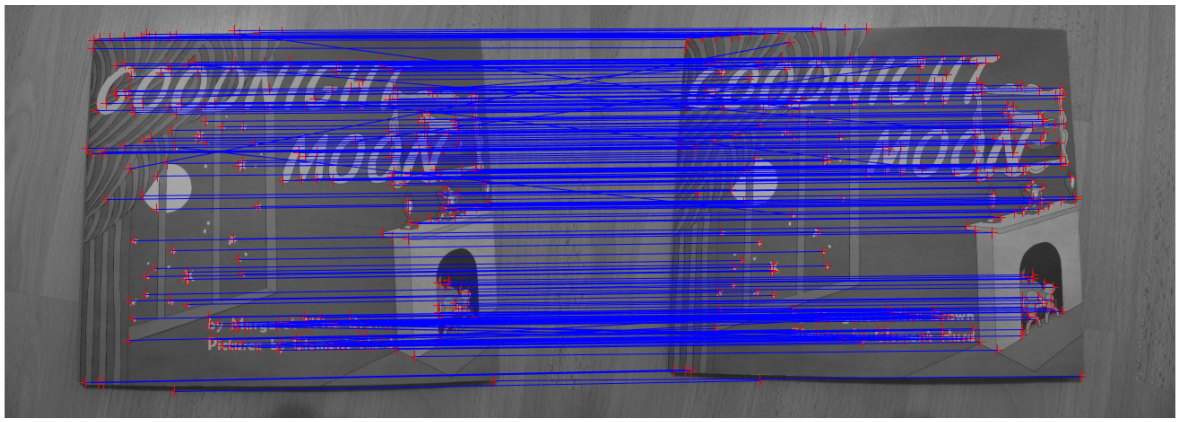

Figure 6: Matching of keypoints when using the mutual nearest neighbor method.

Ratio test As an alternative method to get rid of ambiguous matches, each match is considered valid only when the ratio between the first and the second nearest neighbor is lower than a given threshold (in the code, the threshold is set as 0.5). One can observe that the number of valid matches decreases even further in this case compared to the other two (to 216), and the change is also noticeable as in Figure 7.

```
matches = [];
[~, I] = mink(distances,2,2);
for idx=1:length(descr1)
    if distances(idx, I(idx,1)) < distances(idx, I(idx,2))*0.5
        matches = [matches; idx I(idx,1)];
    end
end
matches = matches';
```

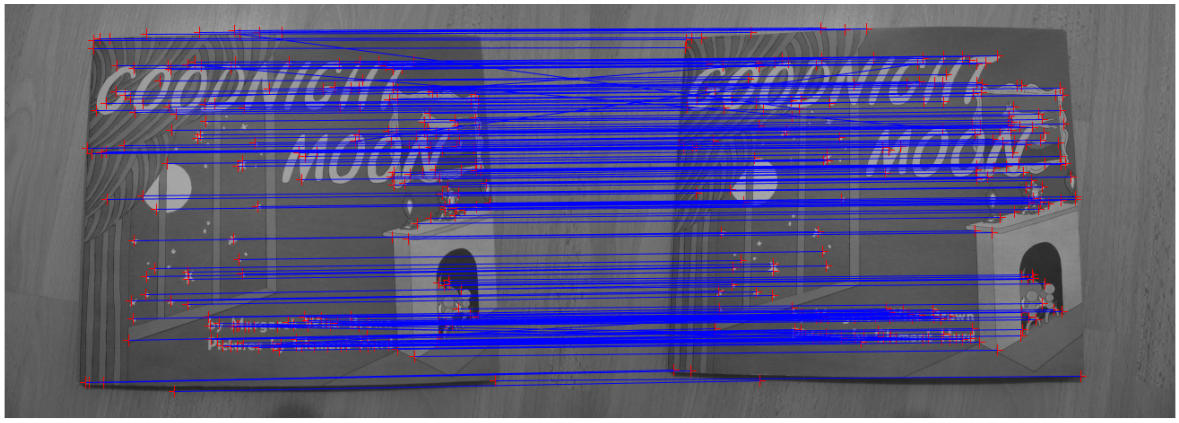

Figure 7: Matching of keypoints when using the ratio method.