Computer Vision Assignment 01

Luca Ebner (ebnerl@student.ethz.ch)

October 7, 2020

1. Detection

For the detection of any kind of features, the first thing to do is to extract so-called *keypoints*. To obtain the keypoints of an image, the following steps have to be computed:

1. Image gradients
2. Local auto-correlation matrix
3. Harris response function
4. Detection criteria

1.1 Image gradients

Presume we have a greyscale image. In order to get an idea of how pixels are changing, for example at a corner of an object inside the image, the gradients of the image are computed. The underlying formula here is a midpoint-approach.

$$I_x(i,j) = \frac{I(i,j+1) - I(i,j-1)}{2}, I_y(i,j) = \frac{I(i+1,j) - I(i-1,j)}{2} \tag{1}$$

To compute these expressions from a given image, we can simply use the `conv2` function provided by Matlab.

```
gradx_filter = [0.5, 0, -0.5];
grady_filter = [0.5; 0; -0.5];
I_x = conv2(img, gradx_filter, 'same');
I_y = conv2(img, grady_filter, 'same');
```

1.2 Local auto-correlation matrix

After computing the image gradient matrices $\mathbf{I_x}, \mathbf{I_y}$, we compute the auto-correlation matrix $\mathbf{M_p}$ for each pixel in the image. The auto-correlation matrix $\mathbf{M_p}$ of a pixel p is given by

$$M_p = \sum_{p'} w_{p'-p} \begin{bmatrix} I_x(p')^2 & I_x(p')I_y(p') \\ I_y(p')I_x(p') & I_y(p')^2 \end{bmatrix}, \tag{2}$$

where $w_{p'-p}$ is a weighting factor. In this exercise, we choose the weighting to be Gaussian with standard deviation σ. Also for this computation, Matlab provides a nice function called `imgaussfilt`, which computes the entries of each $\mathbf{M_p}$ matrix out of the Gaussian weighted sum over all pixels in the image.

```
M_p_11 = imgaussfilt(I_x.^2, sigma);
M_p_12 = imgaussfilt(I_y.^2, sigma);
M_p_22 = imgaussfilt(I_x.*I_y, sigma);
```

1.3 Harris response function

The Harris response function is defined as

$$C(i,j) = \det(M_{i,j}) - k\,\mathrm{Tr}^2(M_{i,j}), \tag{3}$$

where $k \in [0.04, 0.06]$ is an empirically determined constant. Since we've already computed $M_{i,j} = M_p$ as stated above, the computation of $\mathbf{C}$ is straightforward.

1.4 Detection Criteria

At this point we have computed the Harris response function of each pixel in the image. We now want to extract the pixels for which the following conditions hold:

1. The Harris response function is above a certain threshold
2. The Harris response function is at a local maximum (Matlab: `imregionalmax`)

If these two conditions are true, then the found pixel is a keypoint. Note that these two conditions do not exclude the image border corners from becoming keypoints! This has to be taken into account when extracting the keypoint patches in section 2.1.

Varying the threshold T, the constant k and the standard deviation σ.[1]

To fine-tune the detection algorithm, we can vary T, k and σ.

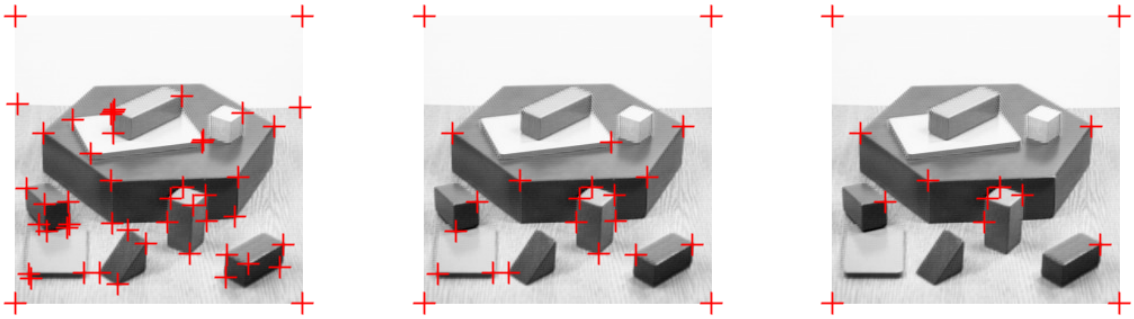

Figure 1: Varying T: If we lower the threshold T, then we also allow "weaker" corners to be considered as keypoints. From left to right: $T = 0.00001, 0.00005, 0.0001$. $(k = 0.04, \sigma = 1)$

[1] For completeness, the same tests with variations of T, k and σ on the other image of this exercise can be found in the Appendix at the end of this document.

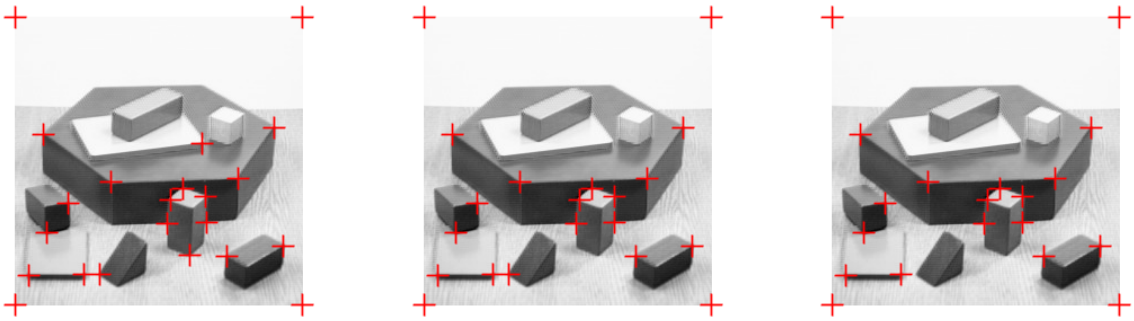

Figure 2: Varying k: $k \in [0.04, 0.06]$ is the so-called sensitivity factor, an empirically determined constant. From left to right: $k = 0.04, 0.05, 0.06$. In this case, $k = 0.04$ led to the best results. ($T = 0.00005, \sigma = 1$)

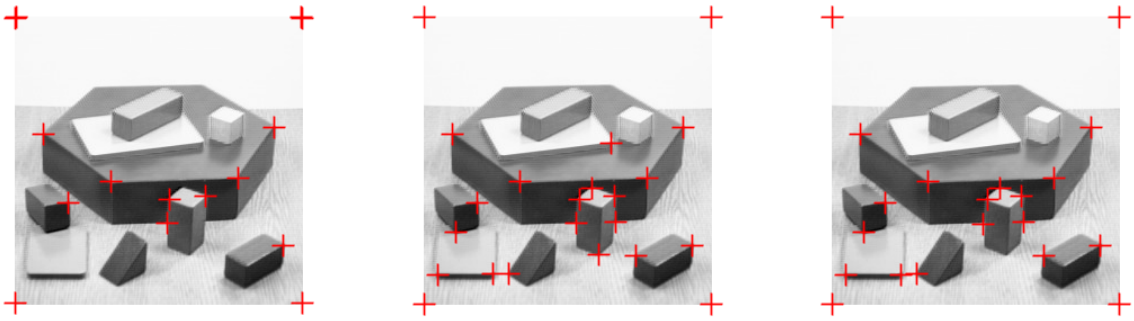

Figure 3: Varying σ: σ is the standard deviation used in the Gaussian weighting for the $\mathbf{M_p}$ matrices. The higher σ is, the more weight is given to the surrounding pixels. If σ is too small, the entries of $\mathbf{M_p}$ are more subjected to noise which stems from the original image. If σ is too big, the difference between the Harris response functions of closeby pixels starts to "fade" out. (Imagine trying to find keypoints from a very blurry image.) From left to right: $\sigma = 0.5, 1, 1.5$. In this case, $\sigma = 1$ led to the best results. ($T = 0.00005, k = 0.04$)

2. Description and Matching

From here on we use the detection parameters $T = 0.00002, k = 0.04$, and $\sigma = 1.5$.

2.1 Local descriptors

Imagine you have taken two pictures of the same object and now you want to compare them by matching their keypoints. In order to match keypoints, we need something to describe them, a so-called descriptor. To achieve this, we first consider a 9×9 pixel patch around each keypoint.
To get the descriptors from these patches, the function `extractPatches` was given in the exercise code framework. The output of this function is a $81 \times q$ matrix for q columns of extracted patch descriptors, where each descriptor consists of $9 \times 9 = 81$ entries.

```
1 descriptors = extractPatches(img, keypoints, patchSize);
```

2.1 SSD one-way nearest neighbors matching

If we have a descriptor vector for a keypoint in image 1, we can compare it to all keypoint descriptors in image 2. The two keypoints that match best are the ones that minimize the sum-of-squared-distances

$$SSD(p, q) = \sum_i (p_i - q_i)^2, \tag{4}$$

where p, q are the two descriptor vectors of the keypoints and i denotes the vector indices. This equation can easily be computed with the Matlab function `pdist`:

```
distance = sum( pdist2(descr1',descr2','squaredeuclidean') );
```

The result of such a matching can be seen in Figure 4.

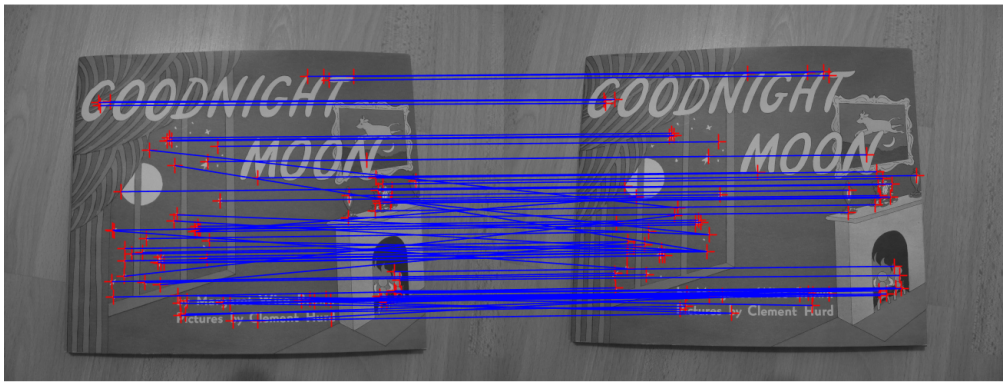

Figure 4: One-way keypoint matching between two images. Note that, since the two images have almost the same object position and rotation, the lines that connect the two keypoints of a match should be close to horizontal. Lines that are not horizontal correspond to false keypoint matching.

2.3.1 Mutual nearest neighbors

In order to ensure better matching, we can not only check if the descriptor of the second keypoint is the best fit for the descriptor of the first keypoint, but also the other way around.

In other words, we first do a one-way matching as before. Afterwards we check if also for the second keypoint it is still true that the descriptor of the first keypoint is the best matching. We can do this by again minimizing Equation 4. If the pair resulting from the one-way matching is identical to the reverse matching, we then consider this a mutual match. The result of this matching can be seen in Figure 5.

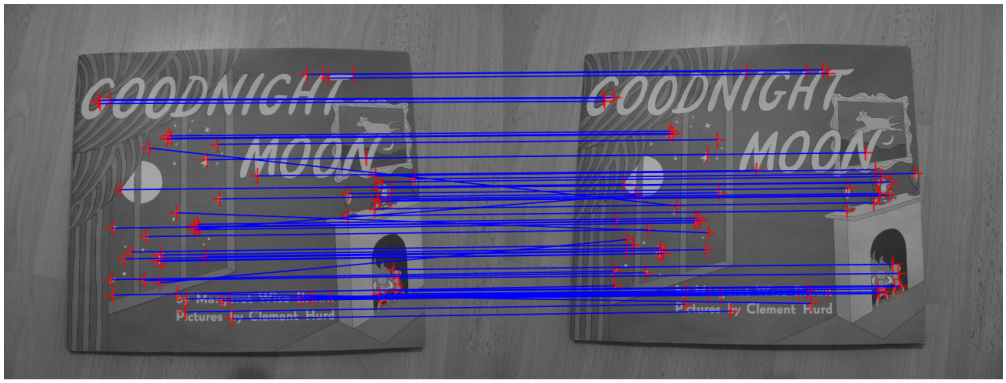

Figure 5: Mutual keypoint matching between two images. Note that compared to Figure 4 we see much less lines that are not horizontal, meaning we have less faulty matches.

2.3.1 Ratio test

Another way of improving keypoint matching is the Ratio Test. When searching the best descriptor fit between a given keypoint from image 1 and the keypoints from image 2, we only consider the match valid if the ratio between the best and second best matches is within a certain ratio.
For this, we compute the SSD values of the best and second best match. If we name SSD_1 the smallest (best fit) value and SSD_2 the second smallest (second best fit) value, then we only consider the one-way match valid if $\frac{SSD_1}{SSD_2} < t$, where t is the threshold. The result of this matching can be seen in Figure 6.

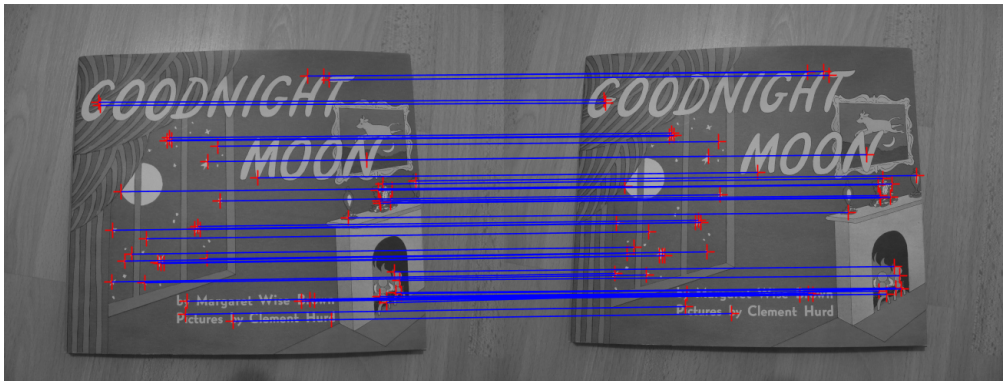

Figure 6: Ratio test keypoint matching between two images. Note that compared to Figure 5 we can see even less faulty matches. ($t = 0.5$)

3. Appendix

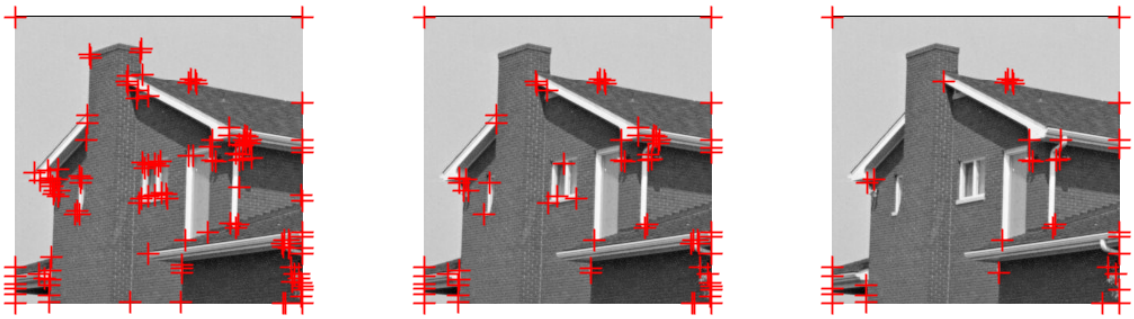

Figure 7: Varying T: From left to right: $T = 0.00001, 0.00005, 0.0001$. $(k = 0.04, \sigma = 1)$

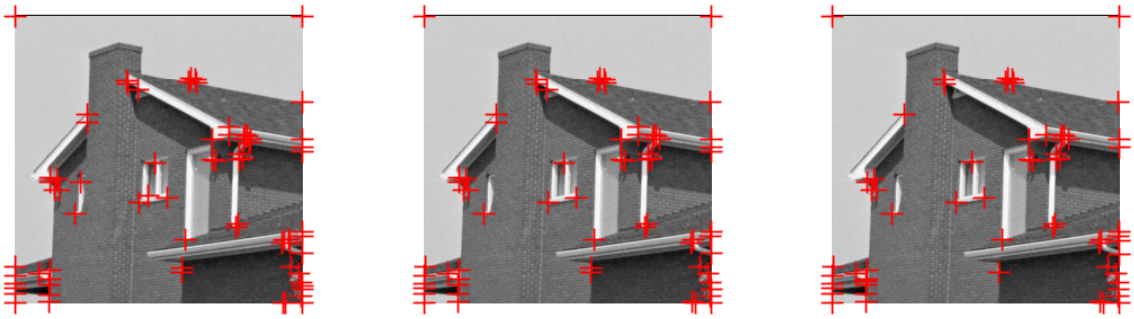

Figure 8: Varying k: From left to right: $k = 0.04, 0.05, 0.06$. $(T = 0.00005, \sigma = 1)$

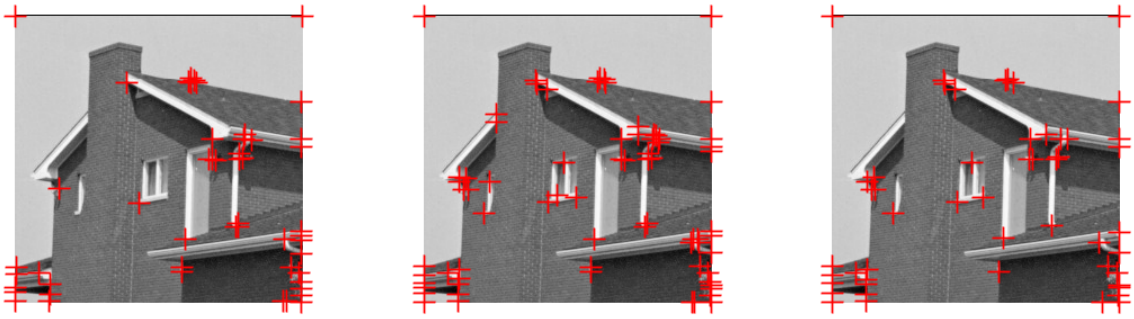

Figure 9: Varying σ: From left to right: $\sigma = 0.5, 1, 1.5$. $(T = 0.00005, k = 0.04)$