Computer Vision
ETH Zürich
Fall 2020

Lab Assignment 2 - Local Features

Mokhles Bouzaien

In this lab, we will implement feature detectors such as corners detection as a first step and then pixel-wise matching using different methods i.e., one-one, mutual and ratio neighbors matching.

1 Detection

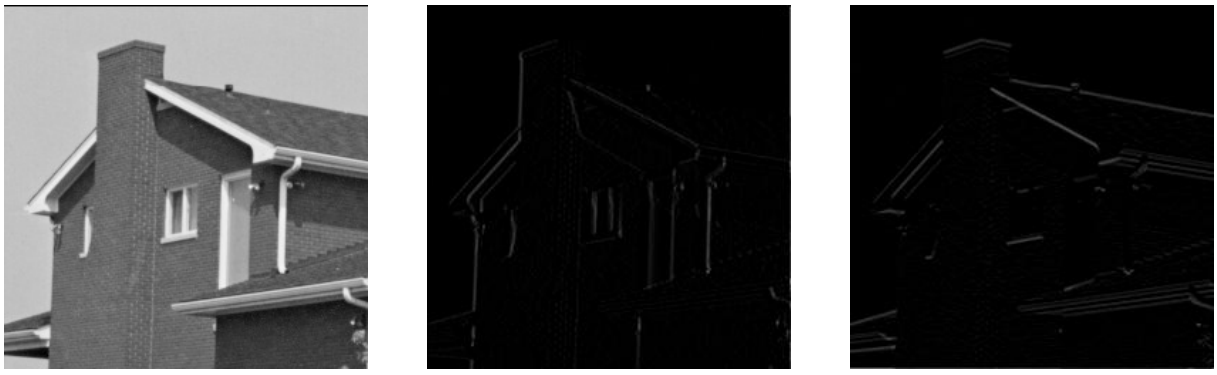
1.1 Image gradients

The first step is to compute intensity gradients in x and y directions. Given an input image I, this is done by applying 2D convolutions with specific filters such that

$$I_x = I * \frac{1}{2}\begin{pmatrix}-1 & 0 & 1\end{pmatrix} \Longrightarrow I_x(i,j) = \frac{1}{2}\left(I(i,j+1) - I(i,j-1)\right)$$

$$I_y = I * \frac{1}{2}\begin{pmatrix}-1 & 0 & 1\end{pmatrix}^T \Longrightarrow I_y(i,j) = \frac{1}{2}\left(I(i+1,j) - I(i-1,j)\right)$$

In MATLAB, this is simply done using the `conv2` function with `img` and `filter` as arguments. A third argument `'same'` is added to keep the same image size after applying the 2D convolution, i.e., 256×256 pixels. The output of this step can be observed in figure 1.



(a) Original image (b) Gradient in x direction (c) Gradient in y direction

Figure 1: Original image and intensity gradients

As we can see, the gradient is used to extract horizontal and vertical lines depending on its direction.

1.2 Local auto-correlation matrix

Now, we will create the local auto-correlation matrix defined for each pixel (i, j) by

$$M(i,j) = \sum_{p'} \omega_{p'} \begin{pmatrix} I_x(p')^2 & I_x(p')I_y(p') \\ I_x(p')I_y(p') & I_y(p')^2 \end{pmatrix}$$

where p' are the neighbor points of (i, j). So we introduce $\texttt{n\_mask} = \left(\begin{smallmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{smallmatrix}\right)$ that allows to sum all the neighbor pixel intensities using a 2D convolution.

1.3 Harris response function

The Harris response function is defined for each pixel (i, j) by

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

where k is an empirically determined constant.

1.4 Detection criteria

Given the Harris response function, the final step for corner detection is to threshold on the values of C and only select local maxima in 3×3 neighborhood.

Choosing the appropriate parameter values. Using a high value of σ i.e., $\sigma = 2$ resulted in an extremely blurry image which in inconvenient for corner detection because high frequency contents (edges and corners) are no more visible. So $\sigma = 1$ is the most appropriate value to remove noise and keep important features. After trying different configurations, $k = 0.056$ and $T = 1.5 \times 10^{-6}$ seem to give acceptable results, as shown in figure 2.

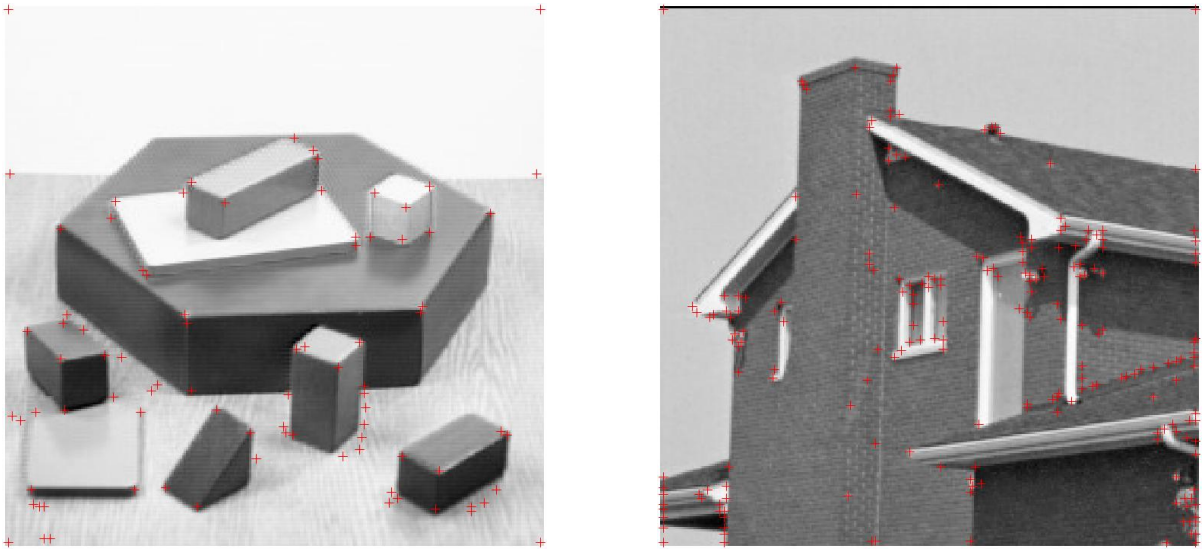


Figure 2: Detected key points for both images

The main issue with the detected key points is the misdetection of some edge points that do not represent corners especially in the second image. This is mainly due to the low resolution of the image and the non-uniform texture of the house.

2 Description & Matching

2.1 Local descriptors

The first step is to filter out the key points that are too close to the edges, i.e. keep only points $M_i(x_i, y_i)$ such that $(x_i, y_i) \in [50, 600] \times [50, 470]$. As we can see in figure 3, no key points near the edges were selected. Then, a 9×9 patch around each descriptor was extracted. The patches will be compared across the 2 images in order to do the matching.

```
Ky = keypoints(1,:) > 50 & keypoints(1,:) < 470;
Kx = keypoints(2,:) > 50 & keypoints(2,:) < 600;
K = [Ky;Kx];
```

2.2 SSD one-way nearest neighbors matching

In order to match corresponding descriptors, the first step was to use the `ssd` function to calculate euclidian distances between all the descriptors (each descriptor is a 81-dimension vector). For the one-way nearest neighbors matching, each feature of the first images is matched with the nearest one (considering the calculated distances) of the second images as shown in figure 3.

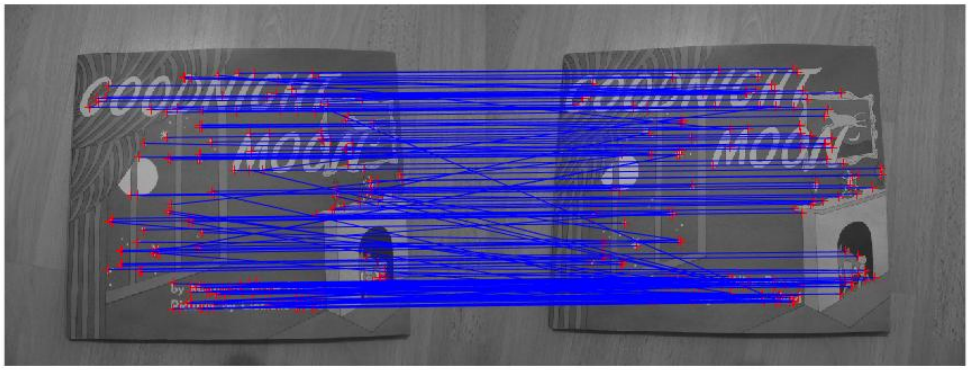


Figure 3: One-way nearest neighbors matching

The code used for this part is

```
distances = ssd(descr1, descr2);
[s1,s2] = size(distances);
[~,b] = min(distances,[],2);
matches = [[1:s1]; b'];
```

2.3 Mutual nearest neighbors / Ratio test

Mutual nearest neighbors is similar to the one-way matching but the verification is done also when we swap the images. The transpose matrix of the distances is used for this. As we can see in figure 4, nearly only parallel lines are considered, which means points are matched only if they are mutually the nearest neighbors.

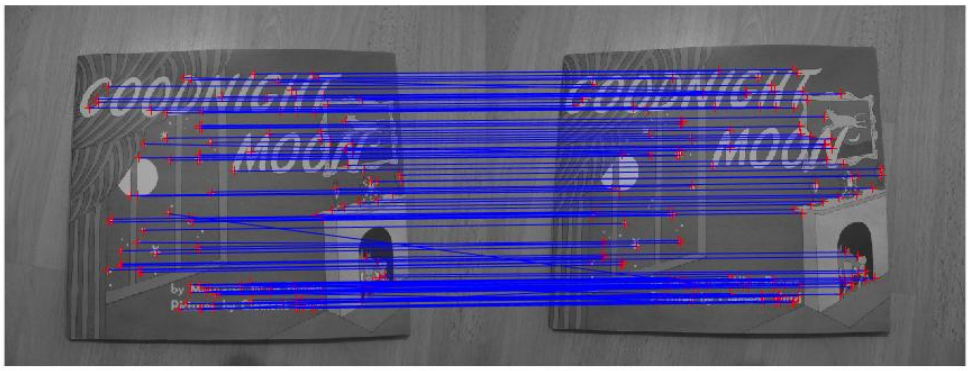


Figure 4: Mutual nearest neighbors matching

Finally, the ratio method is considered. It consists of using the one-way matching and then only keeping matchings whose distances are below a threshold ($t = 0.5$).

```
matches = matches(:,a<=t);
```

In that way, only matchings with high similarities are considered. Results are shown in figure 5.

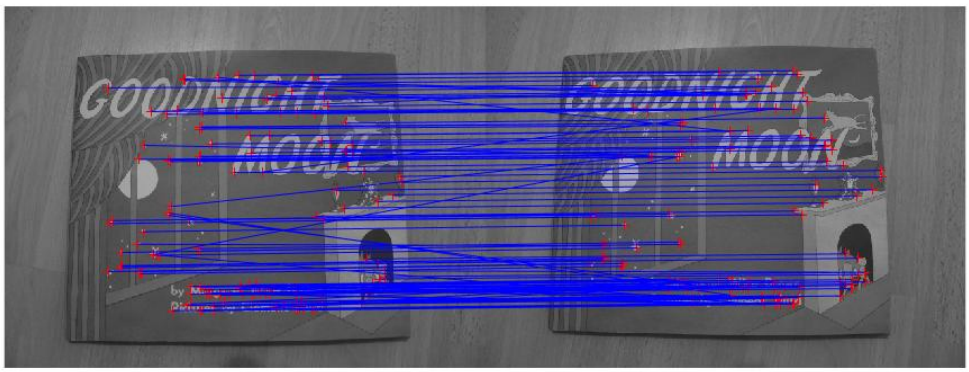


Figure 5: Mutual nearest neighbors matching