Prof. M. Pollefeyes

Filippo Ficarra: Lab 2 - Feature extraction and Optical flow

fficarra@student.ethz.ch, 22–938–062.

08/10/2023 - 23:04h

1 Introduction

In this lab we aim to extract features from images, such as corners, using Harris detection and match descriptors between two different photos of the same thing.

2 Harris corner detection

The idea of Harris corner detection is to analyze the change of intensity of pixels in a window. Using this idea we can basically identify three different regions:

- flat: there is no change of intensity in all the directions
- edge: there is no change of intensity in the direction of the edge
- corner: large change of intensity in all the directions

To perform this analysis we define a window $\boldsymbol{W}$ and we define the SSD error as:

$$E(u,v) = \sum_{(x,y)\in\boldsymbol{W}} \left[I(x+u,y+v) - I(x,y)\right]^2$$

using Taylor approximation we can approximate the error with:

$$E(\Delta x, \Delta v) \approx [\Delta x, \Delta y]\, M \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix},\ M = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Note that I_y and I_y are respectively the partial derivative of the image with respect to x and y.

2.1 Image derivatives

Since images are discrete we need to compute some approximation of the derivative. The approximation used in this case is the following:

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

We can exploit the convolution operations to compute these derivatives, using the following filters:

$$\mathrm{filter_x} = \begin{bmatrix}\begin{bmatrix}-0.5 & 0 & 0.5\end{bmatrix}\end{bmatrix},\ \mathrm{filter_y} = \begin{bmatrix}\begin{bmatrix}-0.5\end{bmatrix}, \begin{bmatrix}0\end{bmatrix}, \begin{bmatrix}0.5\end{bmatrix}\end{bmatrix}$$

This convolutions have been done in python in the following way:

Listing 1: Image gradients

```
filter_x = np.array([[1/2, 0, -1/2]])
filter_y = np.array([[1/2],[0],[-1/2]])

I_x = signal.convolve2d(img, filter_x, mode='same')
I_y = signal.convolve2d(img, filter_y, mode='same')
```

2.2 Gaussian blur and local-auto correlation matrix

The next step is to introduce blur in the image to make the detection more robust. Applying a Gaussian Blur filter prior to performing corner detection serves the purpose of decreasing image noise, thereby enhancing the outcome of corner-detection.

To do so we use a gaussian kernel g, thus the matrix M is the following:

$$M = g * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Furthermore to compute the response function C we will need just this 3 elements:

Listing 2: Local auto-correlation matrix elements

```
I_xx_blur = cv2.GaussianBlur(I_x**2,(5,5),sigma,borderType=cv2.BORDER_REPLICATE)

I_yy_blur = cv2.GaussianBlur(I_y**2,(5,5),sigma,borderType=cv2.BORDER_REPLICATE)

I_xy_blur = cv2.GaussianBlur(I_x*I_y,(5,5),sigma,borderType=cv2.BORDER_REPLICATE)

# 3. Compute elements of the local auto-correlation matrix "M"

g_xx = I_xx_blur
g_yy = I_yy_blur
g_xy = I_xy_blur
```

2.3 Harris response and corner detection

The Harris detector uses the following function to score the presence of corners:

$$C = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 = \det(M) - k(trace(M))^2$$

therefore if

- $\lambda_1 \sim 0$ and $\lambda_2 \sim 0 \implies C \ll 0$ and the region is flat, since the intensity of the pixels doesn't really change in that region
- $\lambda_2 \gg \lambda_1 \implies C < 0$ and we detect an horizontal edge
- $\lambda_1 \gg \lambda_2 \implies C < 0$ and we detect a vertica edge
- $\lambda_1 \sim \lambda_2$ and both large, $\implies C > 0$ and we detect a corner

Alternatively to use $\det(M)$ and $trace(M)$, we can rewrite the response function like this:

$$C = g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - k[g(I_x^2) + g(I_y^2)]^2$$

and then we can reuse the components derived above:

Listing 3: Harris response function

```
C = (g_xx * g_yy - (g_xy**2) - k * (g_xx + g_yy)**2)
```

In order to detect corners we need just to compare every entry of the response with a threshold and perform non-maximum suppression. A non-maximum suppression process allows selecting a unique feature in each neighborhood.

This has been performed in python like this:

Listing 4: Local auto-correlation matrix elements

```
corners = np.argwhere((C > thresh) & (C == ndimage.maximum_filter(C, (3,3))))
corners = corners[:, [1, 0]] # convention row = y, column = x
```

2.4 Results

In this section I'll show the result of the detection for the different parameters Sigma (Gaussian Blur), k (Response function) and the threshold.

As expected, when one decreases the threshold, the number of corners detected is greater but not so accurate.

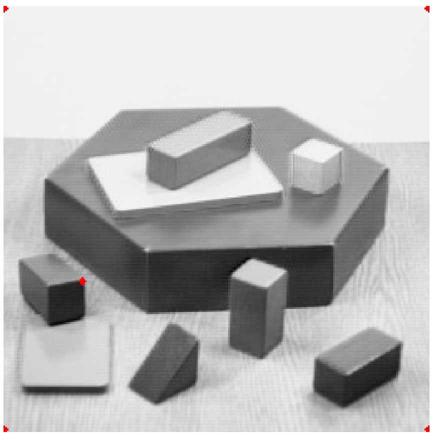

Figure 1: k = 0.04, sigma = 0.5, t = 1e-4 and keypoints = 5

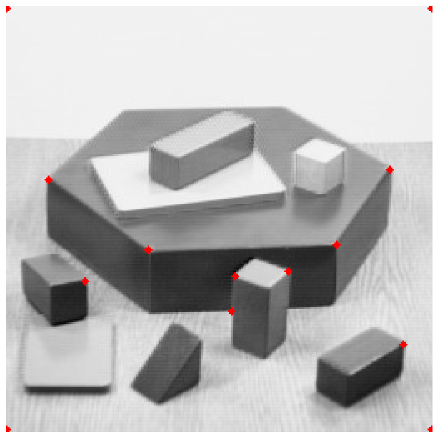

Figure 2: k = 0.04, sigma = 0.5, t = 5e-5 and keypoints = 13

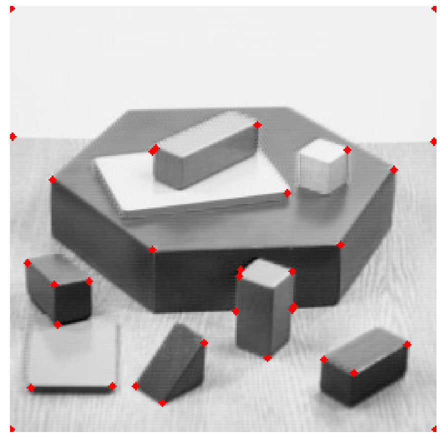

Figure 3: k = 0.04, sigma = 0.5, t = 1e-5 and keypoints = 34

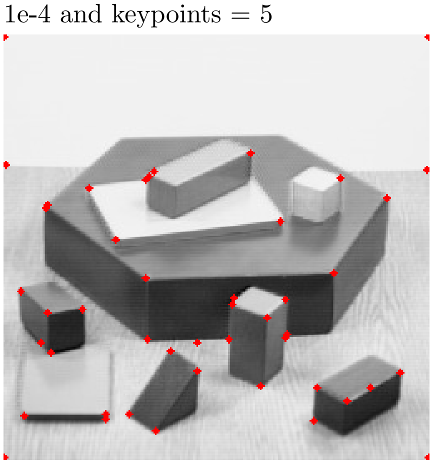

Figure 4: k = 0.04, sigma = 0.5, t = 5e-6 and keypoints = 46

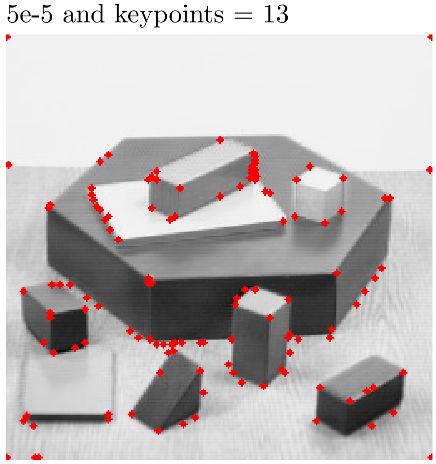

Figure 5: k = 0.04, sigma = 0.5, t = 1e-6 and keypoints = 127

A greater effect was given increasing k to 0.06 with sigma 1.0 or 2.0. Increasing k reduces the response function so the number of points should be lower, while increasing the value of sigma

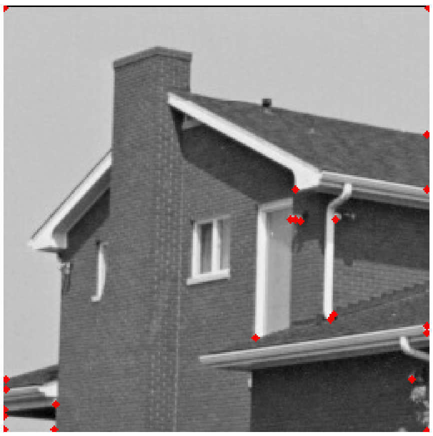

Figure 6: k = 0.04, sigma = 0.5, t = 1e-4 and keypoints = 23

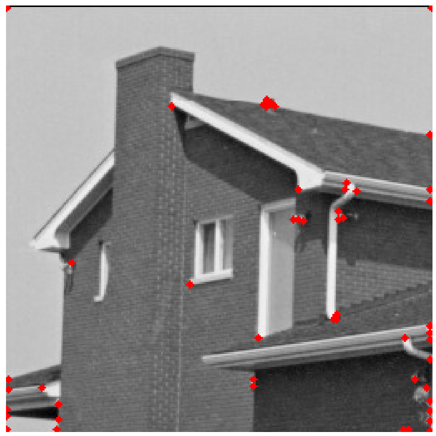

Figure 7: k = 0.04, sigma = 0.5, t = 5e-5 and keypoints = 51

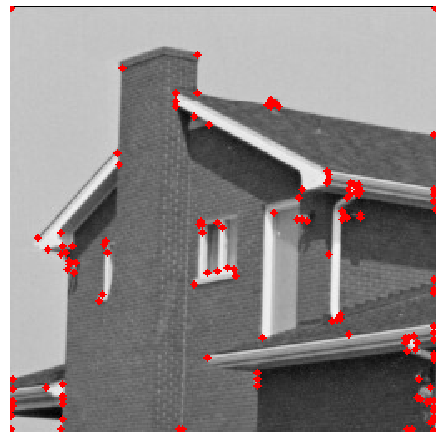

Figure 8: k = 0.04, sigma = 0.5, t = 1e-5 and keypoints = 134

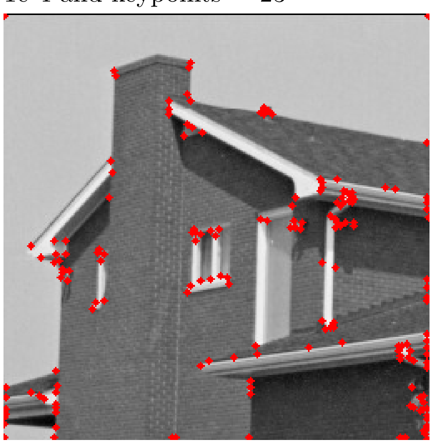

Figure 9: k = 0.04, sigma = 0.5, t = 5e-6 and keypoints = 173

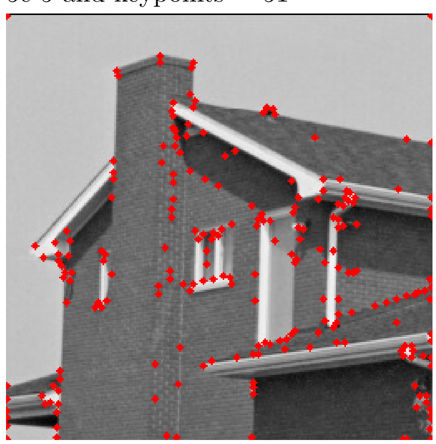

Figure 10: k = 0.04, sigma = 0.5, t = 1e-6 and keypoints = 222

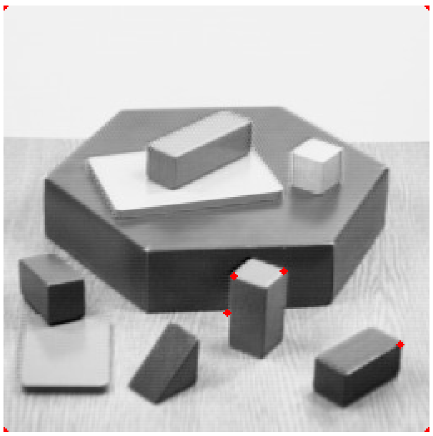

Figure 11: k = 0.06, sigma = 2.0, t = 1e-4 and keypoints = 8

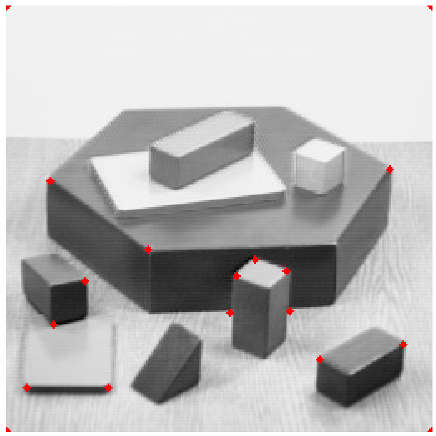

Figure 12: k = 0.06, sigma = 2.0, t = 5e-5 and keypoints = 18

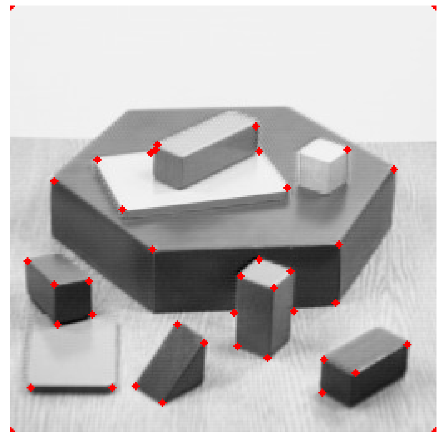

Figure 13: k = 0.06, sigma = 2.0, t = 1e-5 and keypoints = 41

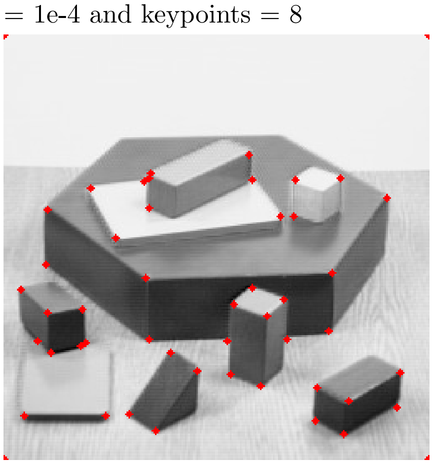

Figure 14: k = 0.06, sigma = 2.0, t = 5e-6 and keypoints = 50

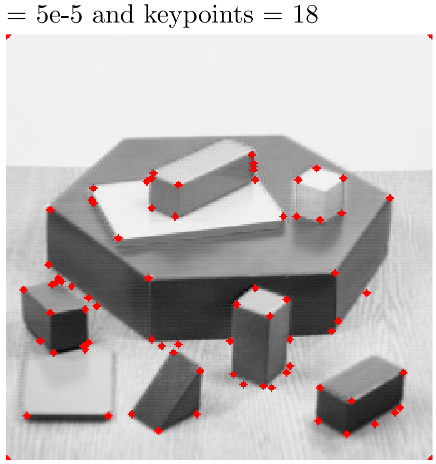

Figure 15: k = 0.06, sigma = 2.0, t = 1e-6 and keypoints = 79

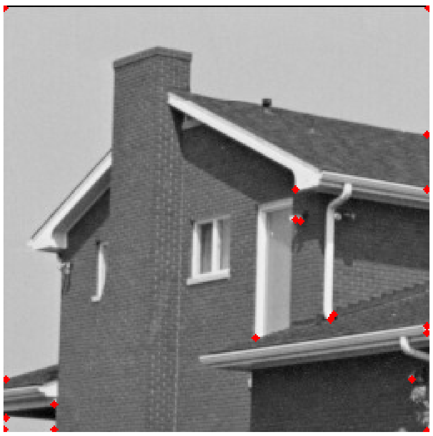

Figure 16: k = 0.06, sigma = 2.0, t = 1e-4 and keypoints = 19

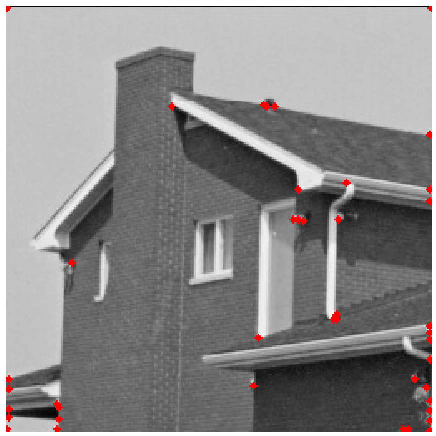

Figure 17: k = 0.06, sigma = 2.0, t = 5e-5 and keypoints = 42

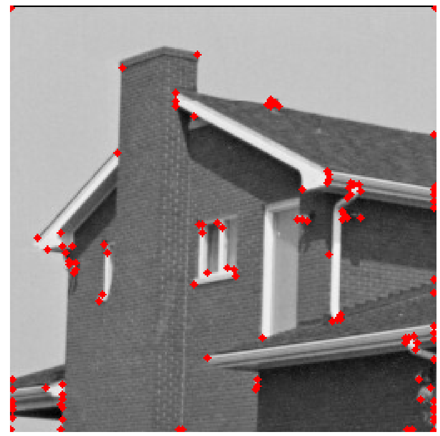

Figure 18: k = 0.06, sigma = 2.0, t = 1e-5 and keypoints = 115

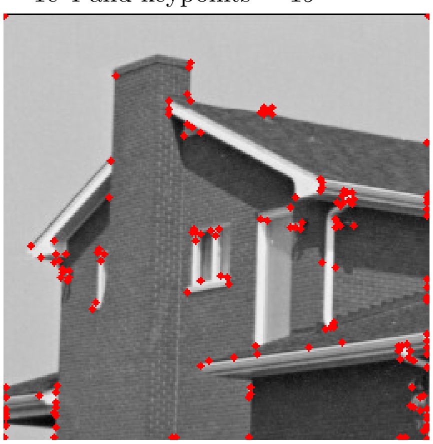

Figure 19: k = 0.06, sigma = 2.0, t = 5e-6 and keypoints = 154

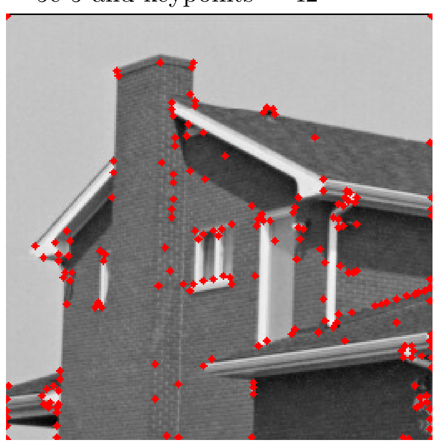

Figure 20: k = 0.06, sigma = 2.0, t = 1e-6 and keypoints = 191

3 Matching

The evaluation was done with these parameters $HARRIS_SIGMA = 1.0, HARRIS_K = 0.05 and HARRIS_THRESH = 1e-5$