Computer Vision: Lab Session n.3
Edge detection

Francesca Canale 4113133
Filippo Gandolfi 4112879
Marco Giordano 4034043

09/04/2019

1 Introduction

The aim of this third lab is to create a Laplacian of Gaussian Operator used to evidence the edges of the given images and to implement a function to actually identify the edges.

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision because semantic information are encoded in edges (e.g. to recognize objects and to recover geometry and viewpoint).
We have been given three different images to work with:

Figure 1: Original image

Figure 2: Original image

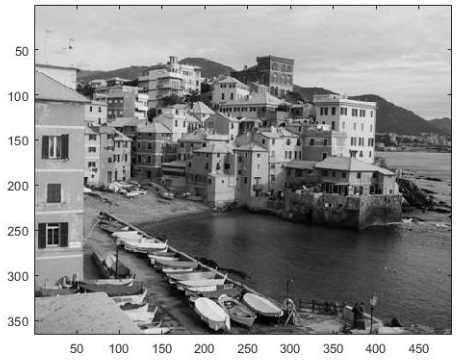

Figure 3: Original image

2 Laplacian of Gaussian Operator

Laplacian operator is the most commonly used second derivative-based edge operator. The principle of edge detection based on double derivative is to detect as edge only those points which possess local maximum in the gradient values. To reduce the noise, Laplacian of Gaussian (LOG) operator can be used. First, it performs the Gaussian smoothing, which is followed by the Laplacian operation.

$$\Delta^2 G_\sigma = \frac{1}{2\pi\sigma^2}\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right)e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

To implement the Laplacian of Gaussian operator we wrote a function *laplacian_operator()* that takes as input the standard deviation desired. Using MATLAB function *meshgrid()* it creates a kernel with a spacial support equal to six times the standard deviation and it computes the operator implementing the *formula (1)*.
Below we report the image and the surface of the operators obtained for different values of the standard deviation σ.

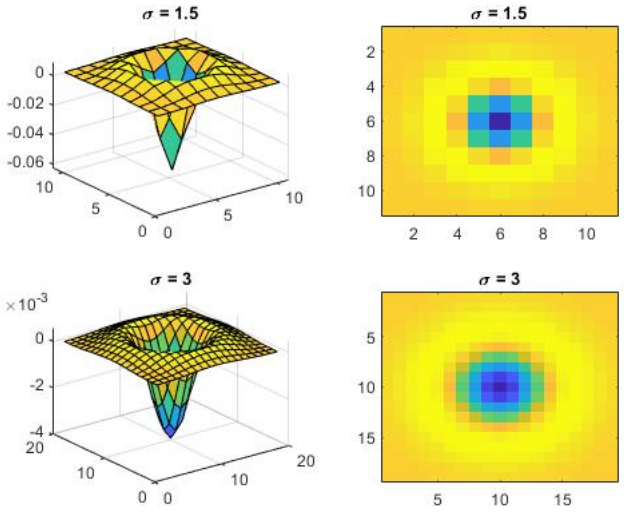

Figure 4: Laplacian of Gaussian operator

3 Convolution with Laplacian of Gaussian Operator

To convolute the original images with the Laplacian of Gaussian operator previously implemented, we used function *conv2()* that performs a two-dimensional convolution. This is done to make the edges of the images highlighted, emphasizing the changes of intensities between near pixels.

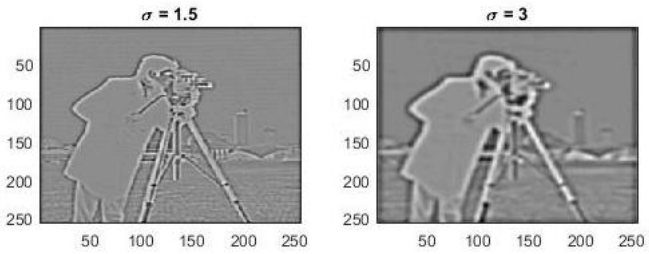

Figure 5: Convolution result

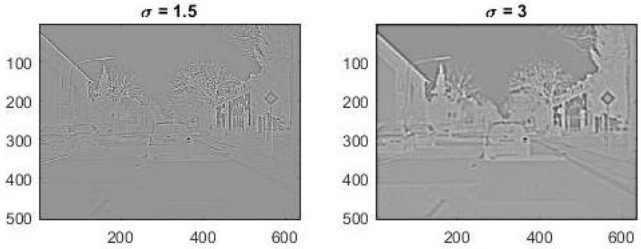

Figure 6: Convolution result

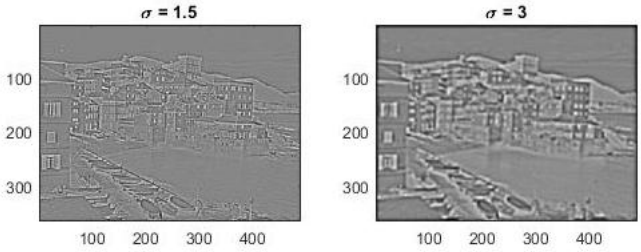

Figure 7: Convolution result

4 Detection of zero-crossings and application

In this section we proposes a zero-crossing detector that attempts to provide a more precise localization of edges than the one obtained with the LOG. To do this, we find the zero-crossing of the image obtained at the previous step; this because a zero-crossing in the second derivative means a change of intensity in the real image. We need a threshold to say if a zero-crossing is taken into account for the edge detection. There are four types of zero-crossing:

- Transition + - : when the curve is positive before the zero value and negative after the zero value

- Transition - + : when the curve is negative before the zero value and positive after the zero value
- Transition + 0 - : as 1, but passing through real 0 value
- Transition – 0 + : as 2, but passing through real 0 value

In this lab we don't consider the 3 and 4 transition because matrix will practically never have exactly zero value.

4.1 Algorithm

To detect zero-crossings we implemented a function *edge_detection* that takes as inputs the desired threshold and the convoluted image. It initializes with all zeros the matrix *edge_matrix* that will contain the edges information. First, the pixels of the convoluted image are scanned along all the rows and then along all the columns. The algorithm implemented is the following: check if the intensity of the actual pixel is greater than the threshold and the intensity of the next one is less than -threshold (transition + -) or the opposite (for the transition - +) and if this is true we set to 1 the value of the corresponding pixel in *edge_matrix*. The result is a matrix that correspond to a binary black and white image containing ones where the algorithm finds edges in the input image and zeros elsewhere. Below we report the images obtained for different values of the threshold: greater the threshold, less the number of details of the image.

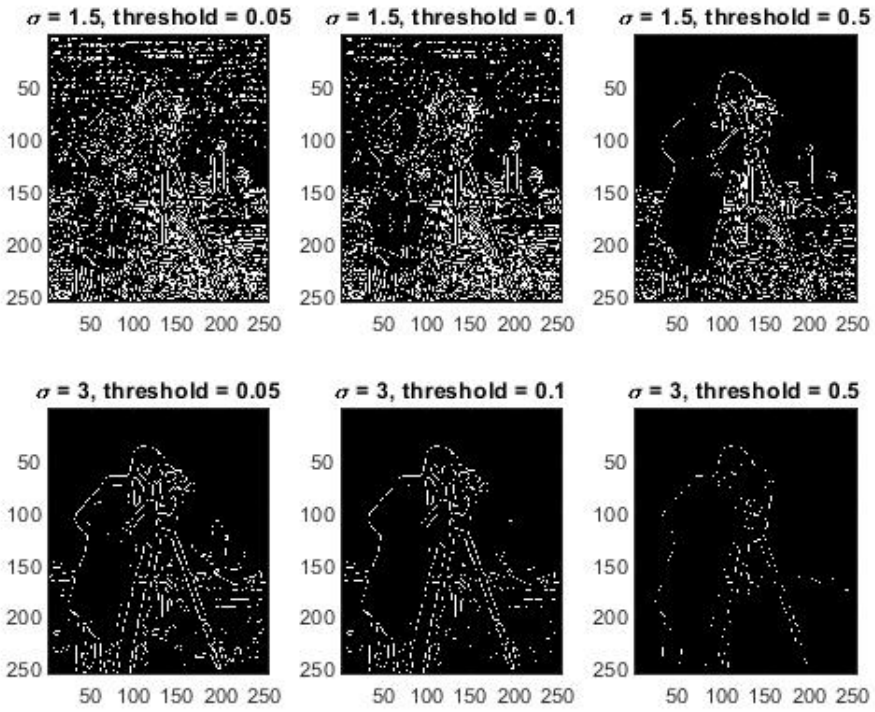

Figure 8: Edge detection

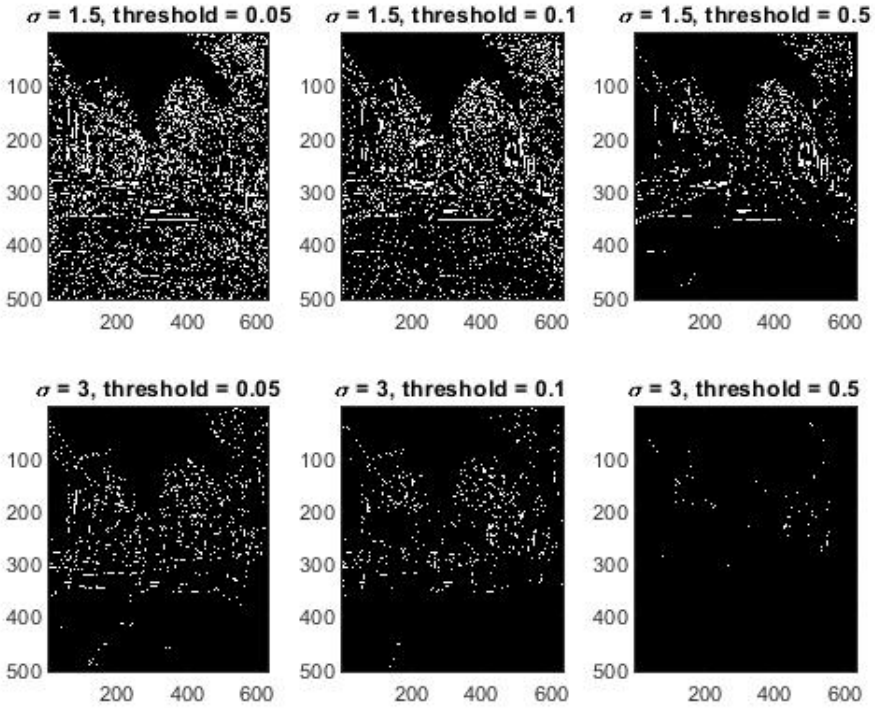

Figure 9: Edge detection

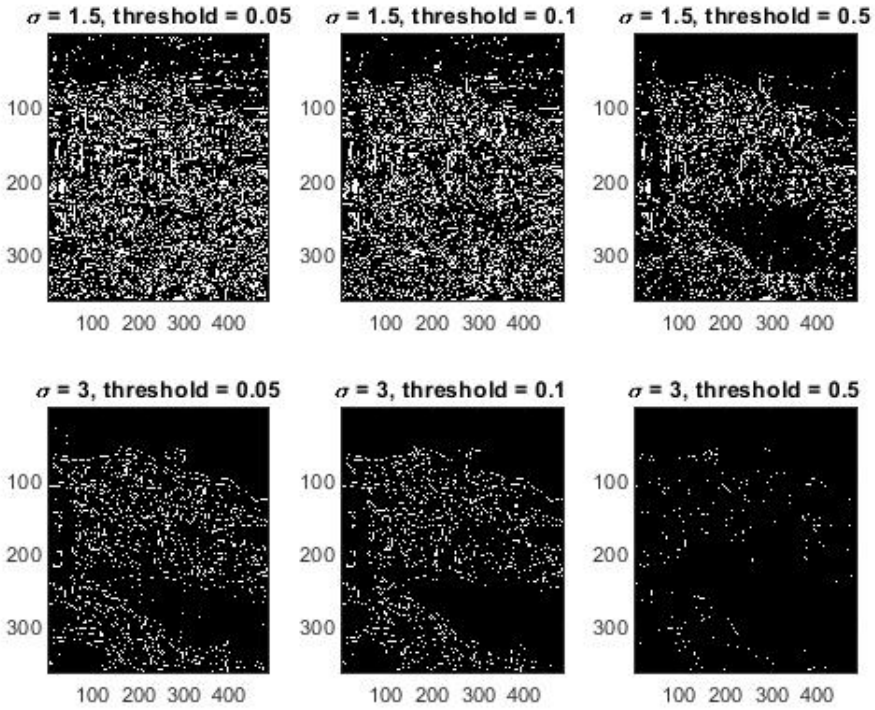

Figure 10: Edge detection

5 Comparison

At the end, we compare our images with the ones obtained with the default MATLAB function *edge(I, method)*.
This function takes as input an image I and an edge detection method and returns the same type of image of the algorithm previously described.
Since the function uses the most suitable standard deviation and threshold values for the image taken as input, to make a sensible comparison we tried to replicate these values in our algorithm. The default standard deviation value is 2, while the threshold depends on the image.

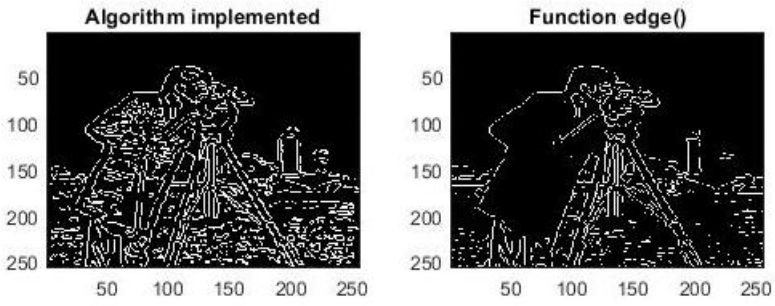

Figure 11: Comparison result with threshold = 0.0051

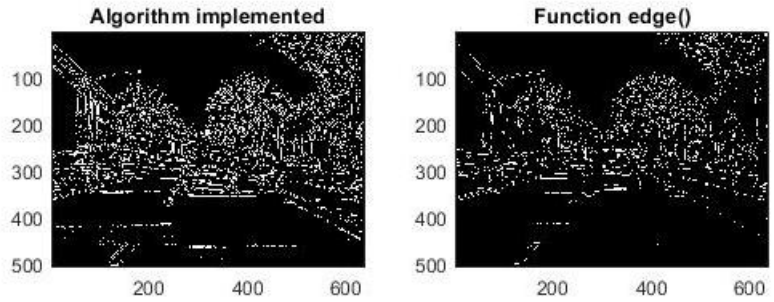

Figure 12: Comparison result with threshold = 0.0029

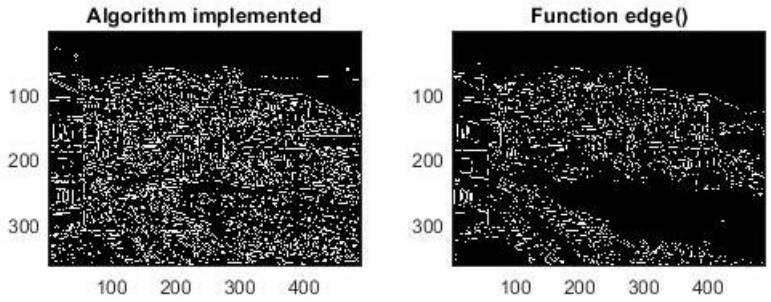

Figure 13: Comparison result with threshold = 0.0063