COMPUTER VISION LABORATORY

SESSION 3: Edge detection

June 17, 2020

Arlotta Andrea 4089306
Cantoni Francesca 4698289

Contents

Chapter 1

Abstract

This report aims to present different types of simple edge detection, which use gradient of the image, and to show in particular Marr and Hildreth method.

The code takes as input one image, which will be transformed in a gray-scale if necessary, and returns a subplot as shown below:

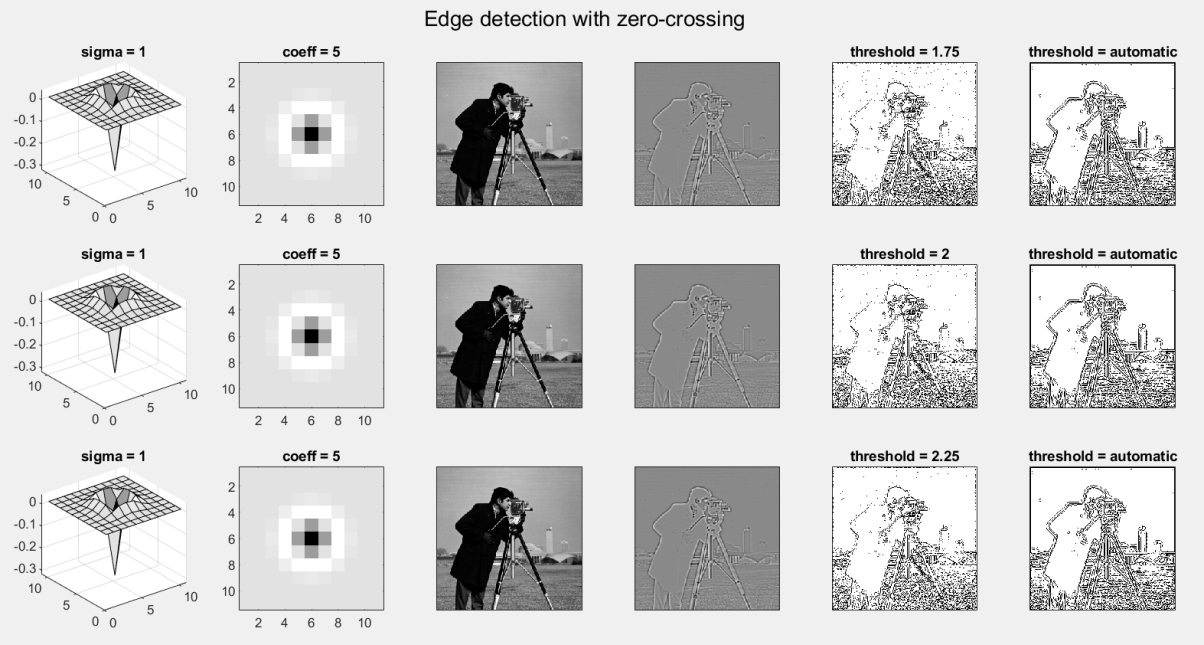

Figure 1.1: Example of final subplot

Analyzing the subplot column by column it is possible to find:

1. **3D Laplacian of Gaussian operator:** this operator can be implemented with different values of σ and spatial domain
2. **1D representation of LoG:** previous surface seen from the above
3. **input image:** chosen input image in grayscale levels
4. **convolved image:** contains information about the gradient of the original image
5. **our edge map:** edges found using our code
6. **Matlab function edge map:** edges found implementing a specific Matlab function that use the LoG method

Description of the paragraphs:

Firstly the section **'Introduction'** gives to the reader a brief explanation about what edge detection is and why it is useful in image processing. After that there is a short presentation of the main methods for detecting edges and boundaries.

The second section **'Theoretical information'** goes deeper on gradient edge detection method and in particular on the one that uses the Laplacian of Gaussian operator (implemented in our Matlab script).

Chapter 4, called **'Matlab'**, can be considered as a guide for the Matlab code that we implemented. During this section it is explained how to use it and how to set all the variables of the *cycles()* function.

In the last chapter, named **'Conclusion'**, are presented all the results, the screen-shots and the comments on the work done.

Chapter 2

Introduction

2.1 2D gradient

Edge detection is a way to extract image features that contain important semantic associations. These information allow to recover the geometry of the objects, through the boundaries detected in the image, and then to recognize them.

Figure 2.1: Example of objects recognition

To perform this image processing is necessary to follow these steps:

1. **Filtering:** the image is passed through a filter in order to remove the noise [1]
2. **Differentiation:** highlights the locations in the image where the intensity changes are significant
3. **Detection:** localizes the points detected in the previous stage

[1] Noise can be due to the undesirable effects introduced during sampling or quantization steps, or due to blurring produced during the shot.

2.1.1 Types of edge

Qualitatively an edge is a rapid variation of intensity that occur between two regions with a high difference of pixel values and detect it is the main point of this objects recognition. Edges can be modeled according to their intensity profiles, as shown in the following figure.

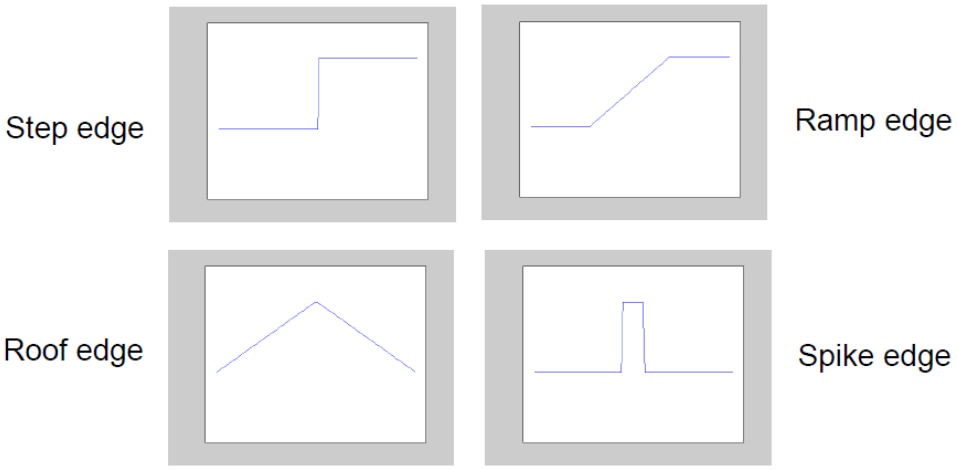

Figure 2.2: Different types of edge

Each different edge model represents a particular behaviour, as follow:

- **step:** model of the ideal edge in which the gray level changes abruptly
- **ramp:** model of the real edge in which the gray level changes gradually
- **roof:** function that describes two ramps close together
- **spike:** function that describes two steps close together

The main strategies to perform edge detection are:

- **Objects' boundary:**[2] this strategy provides isolated edges using image gradient. The considered types are:

[2]For a more in-depth analysis see section **3.2 Simple edge detection using gradient magnitude**.

  - **Marr and Hildreth:** simplest way to perform edge detection but doesn't keep into account the direction of the gradient, so each pixel is individually evaluated without correlation with the surrounding ones
  - **Canny:** the magnitude of the gradient of each pixel is compared with the one of the neighbors along a specific direction in order to determine if the considered pixel is or not an edge

- **Abstraction of boundaries:** this strategy allows to abstract the edges into a symbolic form. The simplest type is:
  - **Hough transform:** use the voting technique[3] to provide the line fitting between edges. Using this method increase the overall complexity but allows to obtain better results

[3]This is a technique that allow to decrease computational complexity through the reduction of possible combinations of features that need to be check in order to perform the line fitting.

Chapter 3

Theoretical information

During this chapter we will discuss in detail edge detection performed with gradients and in particular the Marr and Hildreth method which uses the Laplacian of Gaussian operator.

3.1 Gradient

Detecting discontinuities is the main point of edges detection and it can be found computing a 2D first derivative of the image.

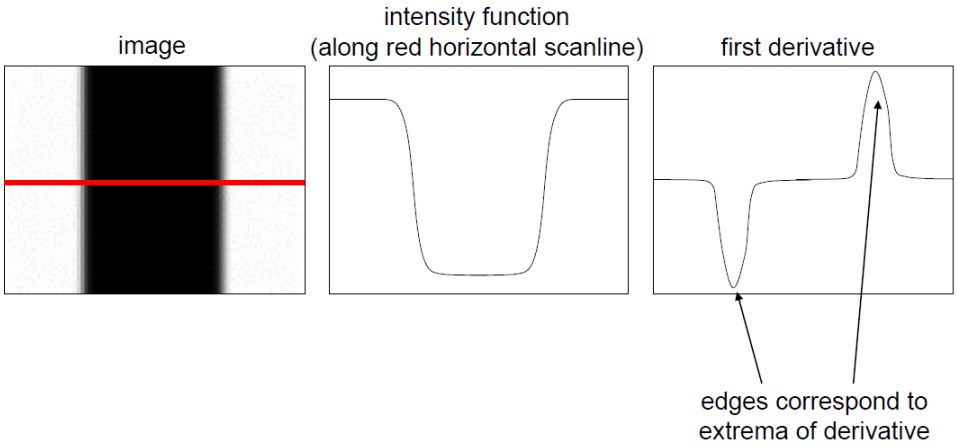

Figure 3.1: Example of edge detection through a first derivative of 1D signal

Perform this operation is equal to execute the **image gradient** so, calling the input image f, can be describe in that way:

$$\nabla f(x,y) = [\ \frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}\ ]^T \ = \ [I_x, I_y]^T \tag{3.1}$$

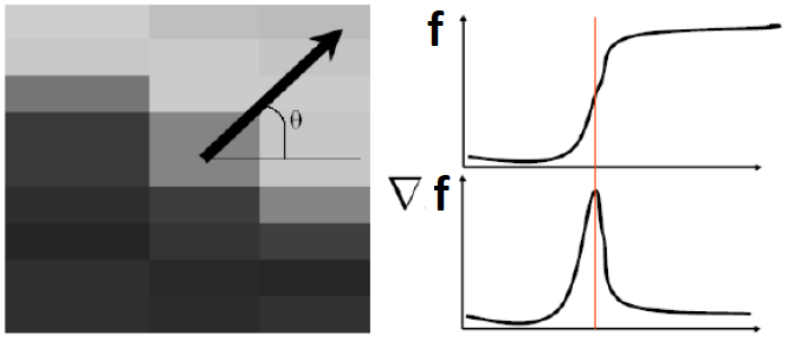

Figure 3.2: Example of 2D gradient on a generic image f

Considering the previous relation it is possible to define:

- **magnitude:** provides information about the edge strength

$$||\nabla f|| = \sqrt{(\frac{\delta f}{\delta x})^2 + (\frac{\delta f}{\delta y})^2} \tag{3.2}$$

- **direction:** points into the direction perpendicular to the edge direction

$$\theta = tan^{-1} \ (\frac{\delta f}{\delta x} / \frac{\delta f}{\delta y}) \tag{3.3}$$

The three main cases of 2D gradient are shown in the following figure:

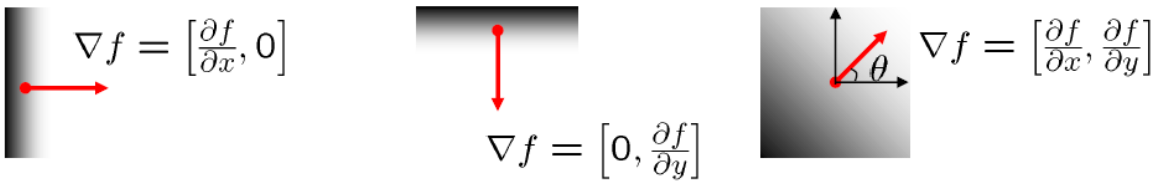

Figure 3.3: Three main cases of 2D gradient

Since images are represented by a discrete set of pixels, it is necessary to resort to an approximation[1] for estimate the derivative of the 2D input signal.

[1]The approximation is due to the fact that images are discrete so δ can't tent to zero but the minimum value it can reach is $\delta x = 1$.

This relation is provided by the *discrete approximation of the derivative*:

$$\text{partial derivative w.r.t. the rows} \longrightarrow \frac{\delta f(x,y)}{\delta x} \approx \frac{f(x+1,y) - f(x,y)}{1} \tag{3.4}$$

$$\text{partial derivative w.r.t. the columns} \longrightarrow \frac{\delta f(x,y)}{\delta y} \approx \frac{f(x,y+1) - f(x,y)}{1} \tag{3.5}$$

Equations **3.4** and **3.5** can be considered as a convolution between the image **I(x,y)** and a kernel $[1,-1]$ and $[1,-1]$ as follow:

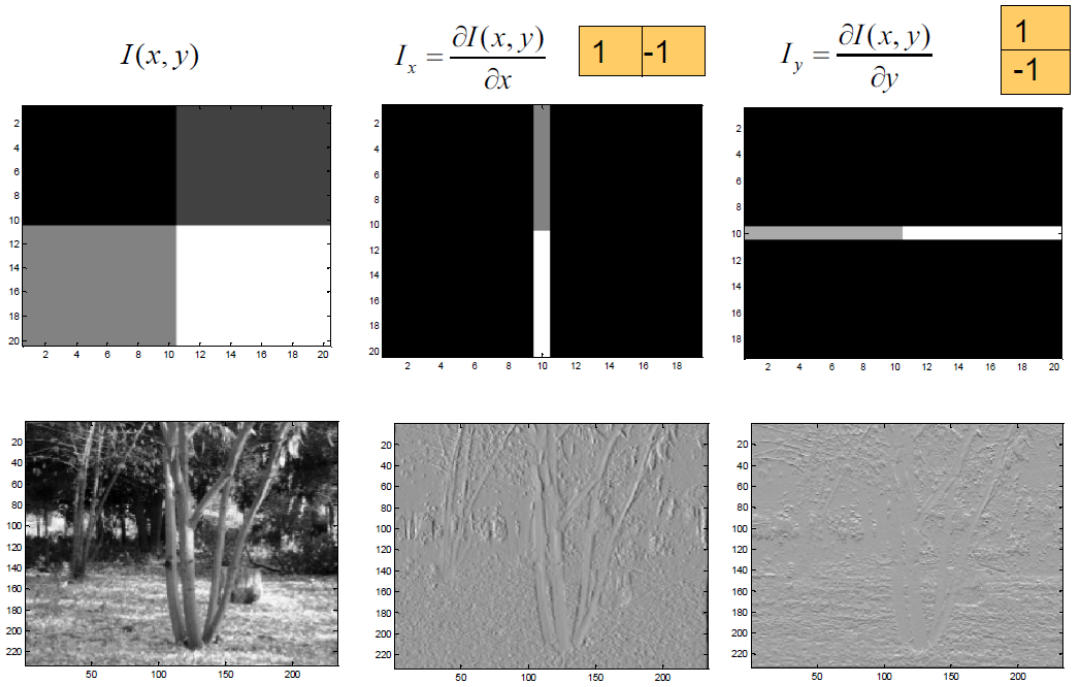

Figure 3.4: Examples of derivative computed applying a mask to I

From the above images it is possible to see that this operation gives a large positive response to an image configuration that is positive on one side and negative on the other side. On the other hand it returns a large negative response to the mirrored image.

3.2 Simple edge detection using gradient magn.

The main steps performed for the edge detection, through gradient magnitude, are the following:

1. computes gradient vector at each pixel by convolving image with the horizontal and vertical derivative filters
2. computes the gradient magnitude at each pixel
3. if the value obtained by the previous step exceeds the chosen threshold, reports the associated pixel as an edge point

It's easy to understand that, when these steps are performed on an image where noise is present, they produce an undesirable result because during differentiation operation high frequencies signals are amplified.

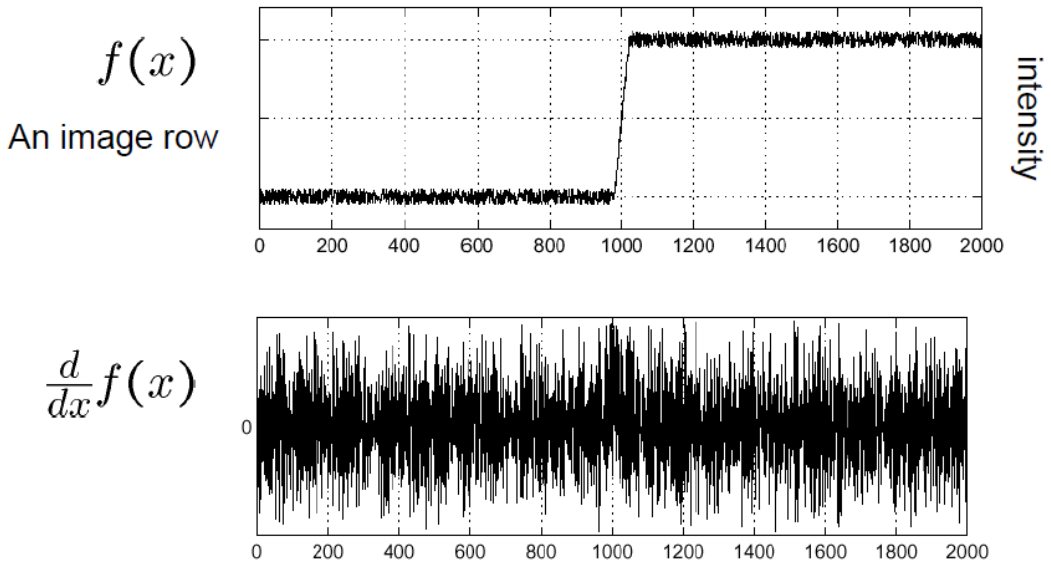

Figure 3.5: Representation of image noise issue

From **Figure 3.5** it is evident the necessity of apply a filter, that reduces the effect produced by the noise, before proceeding to search for the edges.

The most common filter used for this purpose is the Gaussian filter that is a smoothing low-pass one.

Taking into account the previous considerations, edge detection can be performed by:

1. filtering the image with the Low-pass Gaussian filter using a convolution
2. deriving the smoothed image in order to highlights the significant intensity transitions
3. computing the gradient magnitude at each pixel
4. localizing the locations in which the magnitude of the gradient are greater than the chosen threshold

For sake of clarity it is shown an example that displays the first and second step, in which f is a 1D function and h is the 1D mask of the Gaussian filter:

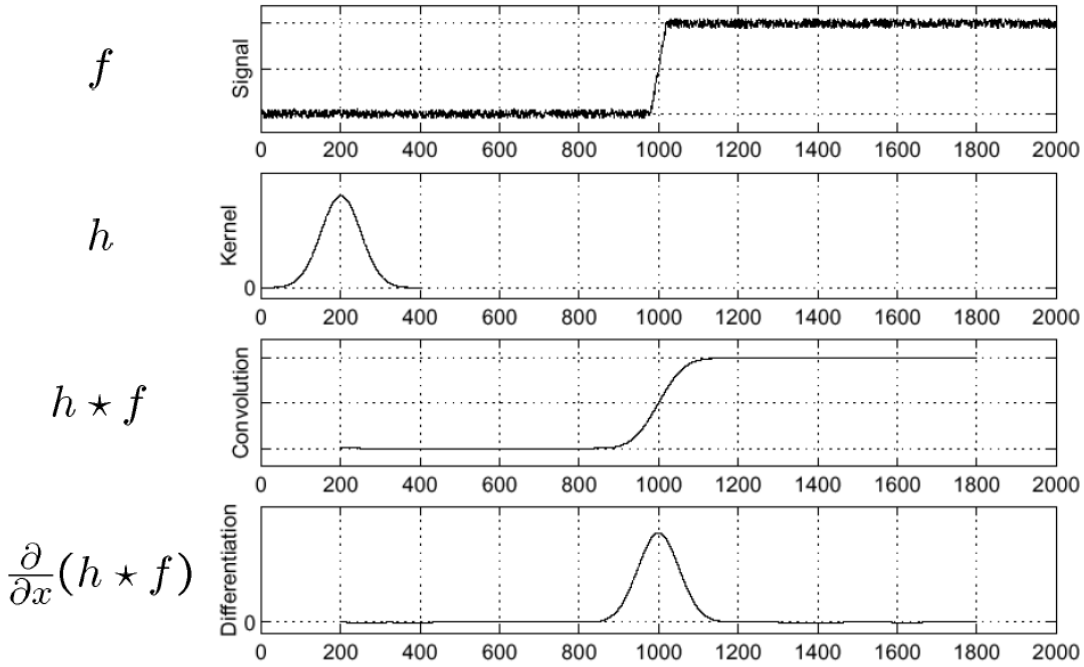

Figure 3.6: Smoothing the image before differentiating it

Since we are applying two consecutive linear operations we can switch them by order in the following way:

$$\frac{\delta}{\delta x}(h \otimes f) \quad = \quad (\frac{\delta}{\delta x} \; h) \otimes f \tag{3.6}$$

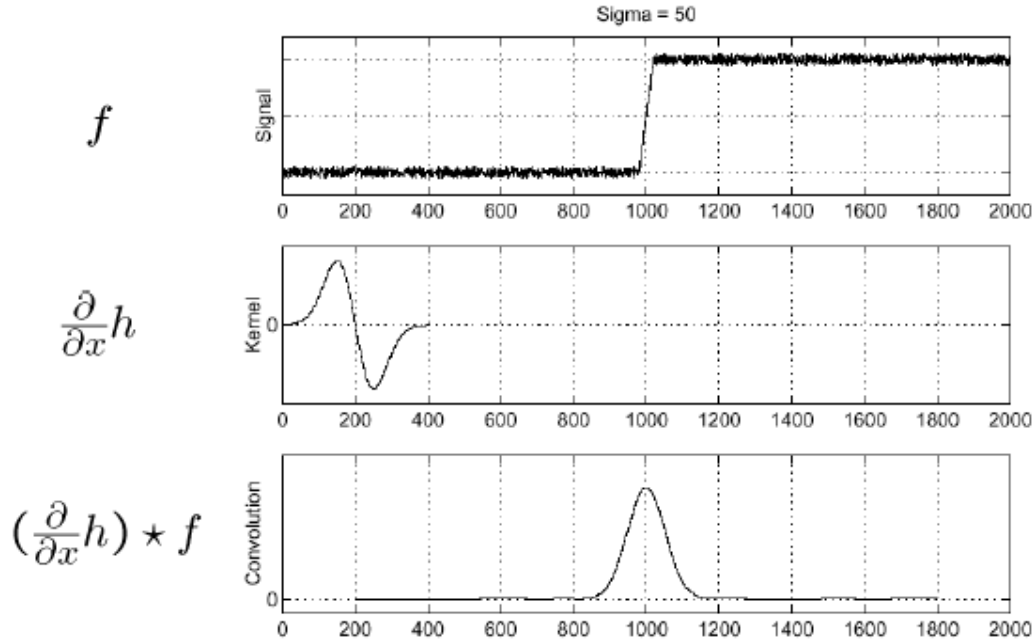

Figure 3.7: Differentiate the Gaussian filter and then convolve it with the image

It's possible to check, comparing **Figure 3.6** and **Figure 3.7**, that the two sides of the equation provide the same result.

3.2.1 Marr and Hildreth method

The Marr and Hildreth is the gradient based edge detector implemented for this laboratory session and it can be represented as follow:

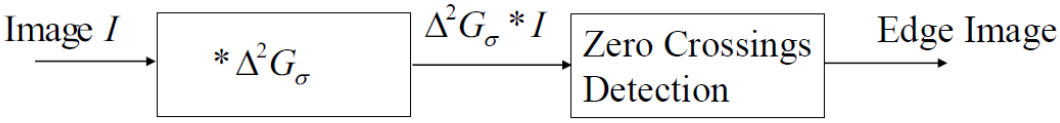

Figure 3.8: Main steps of Marr and Hildreth method

This technique is composed by two main steps:

1. **Smoothing the image:** allows to reduce high frequencies signal and therefore also the noise. This operation is performed through the convolution between the image and the Laplacian of Gaussian operator[2] (LoG)

2. **Zero crossing detection:** allows to detect abrupt changes of the 2D signal, obtain by the previous step, and to store the location in which the peak of this second derivative function is higher than a given threshold

We proceed now to go deeper to the up mentioned steps.

3.2.1.1 Smoothing the image

As already stated in the previous section, smoothing the image, described by the 2D function f, is a fundamental step for obtain a result as good as possible. The best way to do it is by a Gaussian filter[3] as follow:

$$S = G_\delta \otimes f \tag{3.7}$$

in which the Gaussian filter is described with this relation $G(x,y)_\sigma = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}}$

[2]This operator will be soon described.

[3]This is the optimal smoothing filter because places strong weight on center pixels and less weight on distant pixels, like a Gaussian function, as wanted.

REMARK: when the smoothing operation is performed, pixels have a greater propensity to look like their neighbouring pixels so the resultant derivative will have a smaller magnitude. This effect is due to the fact that the derivative operation measure the tendency of pixels to look different from the surrounding ones.

Since we are working with linear operators, is it possible to swap them as follows:

$$\nabla^2 S = \nabla^2 (G(x,y)_\sigma \otimes f) = \nabla^2 G(x,y)_\sigma \otimes f \tag{3.8}$$

in which the $\nabla^2 (G(x,y)_\sigma$ is described by this equation:

$$\textbf{Laplacian of Gaussian operator} \longrightarrow \nabla^2 G_\sigma = \frac{1}{2\pi\sigma^2} \left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right) e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.9}$$

Considering a 1D function, the operations performed up to now can be displayed as follow:

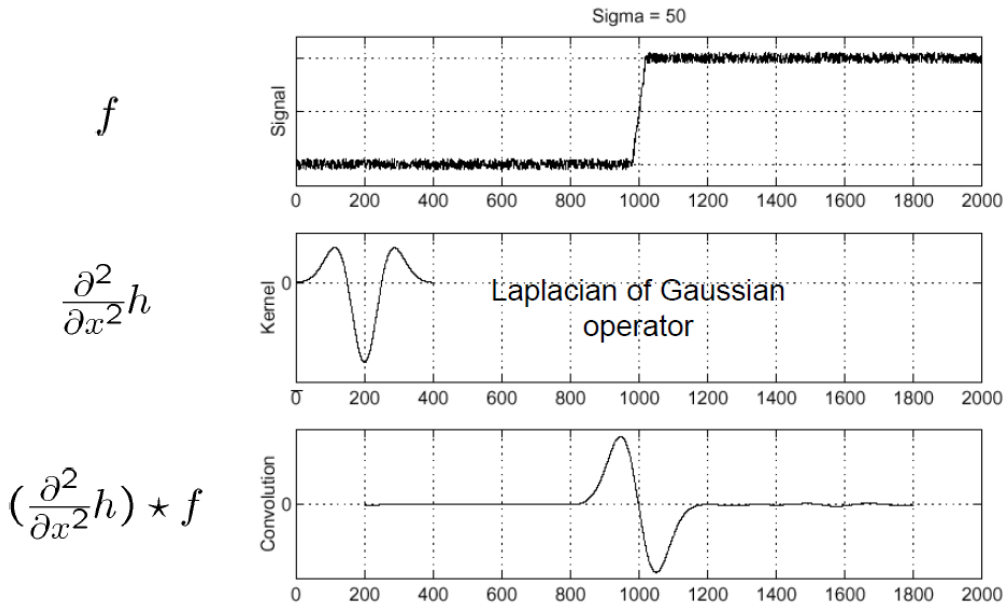

Figure 3.9: Effect of the LoG on a 1D function

It is important to remind the reader that σ value and mask dimension have a big impact on the resulting function, therefore also on the final edge map, in particular small kernels perform noise reduction operation instead big kernels provide an estimation of intensity over a large region[4].

3.2.1.2 Zero crossing detection

The second derivative of a function maps variations between two opposite peaks. Let consider a 1D function:

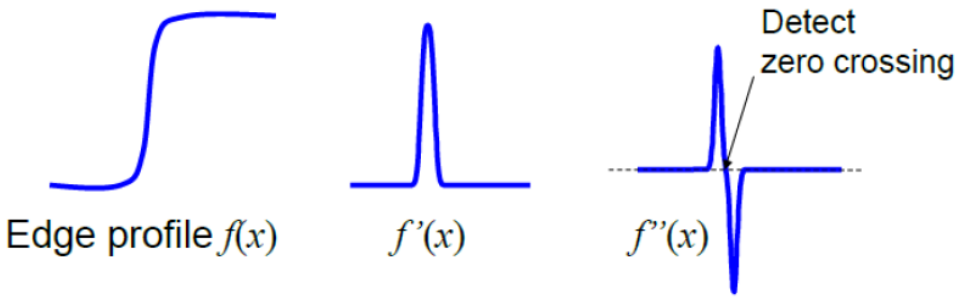

Figure 3.10: Second derivative of 1D edge profile

From the figure above it is possible to understand that, in order to search edges, is necessary to execute a zero crossing detection.

Zero crossing detection looks for adjacent pixel locations where the sign changes values. The four possible cases are:

- [+ , -]
- [- , +]
- [+ , 0 , -]
- [- , 0 , +]

In correspondence of this event the value of the edge map is set to the maximum intensity:

[4]These effects can be seen on images displayed in section **5.3 Results**.

$$Edge(x,y) = \begin{cases} 1 & \text{if N(x,y)} > \text{threshold} \\ 0 & otherwise \end{cases} \tag{3.10}$$

in which N(x,y) represents the gradient magnitude in (x,y) location.

Chapter 4

Matlab

4.1 Function tree

The code consists of a *main.m* file that allows the user to choose the desired input image[1] and the value of each parameter[2]:

1. *cycles.m*
   calls *edgedetection()* function[3] and performs changes in parameter values based on user's parameters set

   1.1. *edgedetection.m*
   transforms input image into monochrome one, if necessary, and displays a subplot that compares the results obtained using our *LoG.m* and *zeroCrossing.m* functions with the Matlab one, called *edge()*[4]. The comparison is made performing the operations on the same input image

   i. *LoG.m*
   provides the Laplacian of Gaussian (LoG) operator value

   ii. *zeroCrossing.m*
   performs the zero crossing detection on the filtered image and it returns the edge map

[1] It's possible to process one or more images at the same time, each one with different set of parameters.
[2] For more information see next subsection.
[3] The number of calls of this function depends on the value set by the user.
[4] For the edge() function we decided to use the automatic value for the threshold parameter.

4.2 Parameters

In this section are described all the variables that can be set through the user interface **cycles(INimage,mainLoop,sig,sigstep,coeff,th,thstep,row)**

This function calls the following variables:

- **INimage:** input image [5]
- **mainLoop:** number of subplot displayed maintaining the values of sigma and spatial dimension unchanged, and varying just the threshold value at each iteration
- **row:** number of row inside each subplot
- **sig:** value of sigma related to the LoG operator
- **sigstep:** increment value of sigma added to the previous value at each *mainloop* iteration
- **coeff:** value that influences the dimension of the spatial support [6]
- **th:** value of the edge detection threshold
- **thstep:** increment value of the threshold added to the previous value. After $\#mainLoop*row$ edge detection, sigma increases by *sigstep* and the threshold is reset to *th*. At this point the iteration starts again

[5] This image can be RGB or gray scale.

[6] The relation between sigma and the dimension of the spatial support is $spat\_supp = ceil(sigma * coeff)$.

Chapter 5

Conclusion

5.1 Execution time

In this section it is shown the performance time[1] of the overall program and of the main functions[2]. The table below shows that *zeroCrossing.m* has the largest Self Time between all the functions; beside this function and the *edgedetection.m* one, it is possible to observe that most of the execution time is spent by functions involved in plotting and formatting graphs. The *LoG* function is not present in the ranking due to its short execution time: 0.004 s (Total Time), 0.002 s (Self Time).

Profile Summary

Generated 07-Apr-2019 21:52:53 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
main	1	3.384 s	0.001 s	
cycles	1	2.817 s	0.004 s	
edgedetection	3	2.813 s	0.543 s	
close	1	0.566 s	0.005 s	
subplot	63	0.552 s	0.374 s	
zeroCrossing	9	0.547 s	0.547 s	
title	45	0.535 s	0.468 s	
close>request_close	1	0.512 s	0.014 s	
closereq	3	0.483 s	0.398 s	

Figure 5.1: most time-expensive functions of the code

[1]All these values have been evaluated using the Matlab tool *performance time*.

[2]The execution time has been estimated processing three images, specifically the *'cameraman.tif'*, *'boccadasse.jpg'*, *'car.bmp'*, each one processed three times in order to have a better understanding of the effects caused by different parameter values.

5.2 Test

In order to test the code we upload the RGB photo *camerman.tif* [3], through a relative path, with these characteristics: width and height of 256 pixels and dimension of 65,1 kB . However it is possible to test the script with other two images by changing the first input of the function *cycles('INimage#', ...)* in the *main.m*.

The possible input images are shown below:

Figure 5.2: input image on which we will refer in the next subsection

Figure 5.3: other 2 possible input images

[3]We choose this image as test because in our opinion it displays the most representative results of the performed image processing.

5.3 Results

We now present and comment the outcomes of our test related to the change of one or more parameter values of the main function **edgedetect(INimage,sig,sigstep,coeff,th,loop)**:

- **sig:** this value is equal to σ and performs some changes inside *LoG(sigma,coeff)* function. The figures **5.4** and **5.5** compare 8 different algorithm outputs that differ each other only for the implemented threshold value.

  The values chosen for the first figure are around zero, while the second figure presents a series of high values.

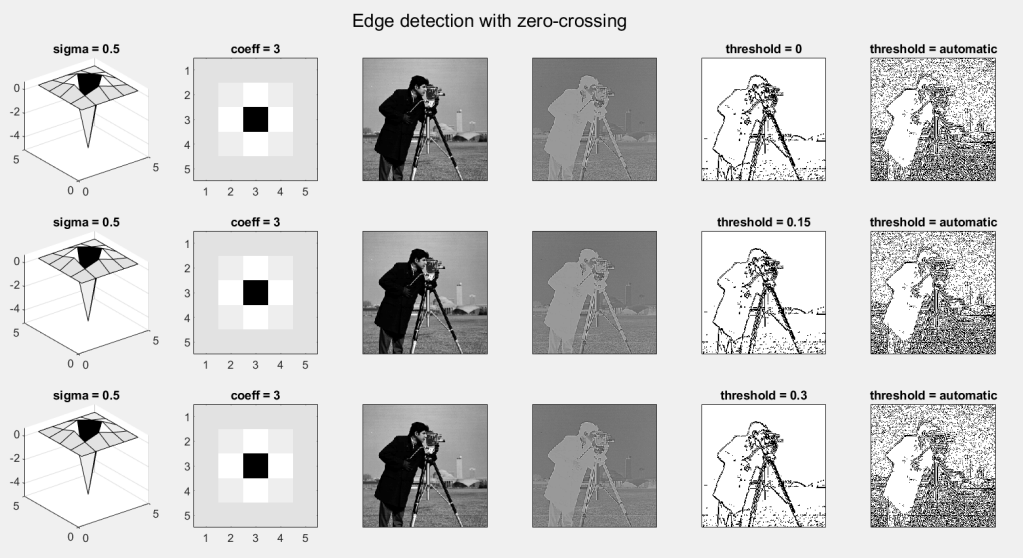

Figure 5.4: Output holding sigma at a value of 0.5 and letting vary the threshold

Small values of sigma, in correspondence of a small kernel dimension, generates sharp filters that cut off the majority of noise, as well as details,and this enabling the detection of the more relevant intensity variations.

Comparing **Figure 5.4** and **Figure 5.5** it is possible to notice that, despite the threshold variation, there's no much edge map variation due to the fact that σ has a small value.

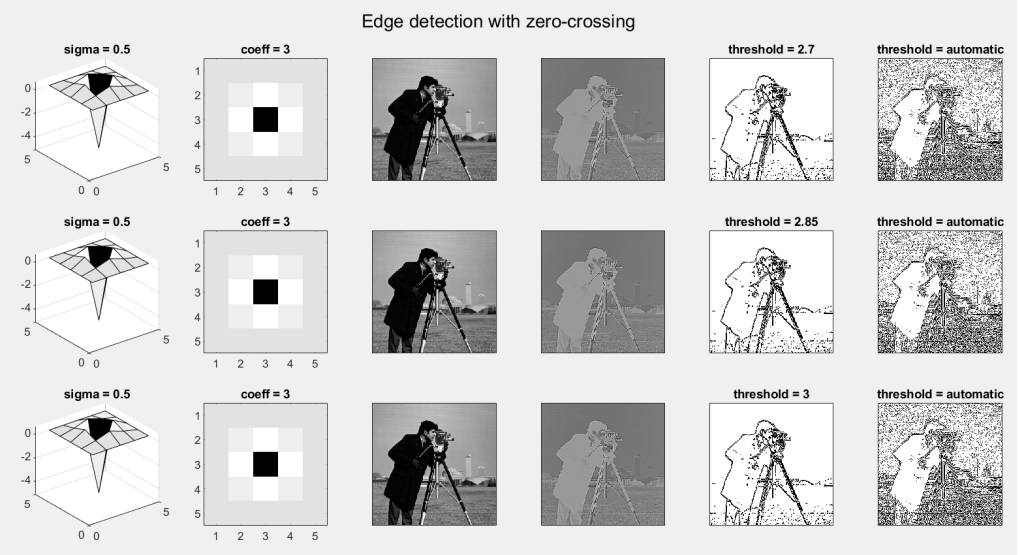

Figure 5.5: Output holding sigma at a value of 0.5 and letting vary the threshold

- **coeff:** this parameter modify the kernel dimension as shown in the following equation:

$$spat\_supp = ceil\ (sigma * coeff) \tag{5.1}$$

The spatial support has a weight on the final result in fact it defines the number of pixels took in account in each step of the convolution.

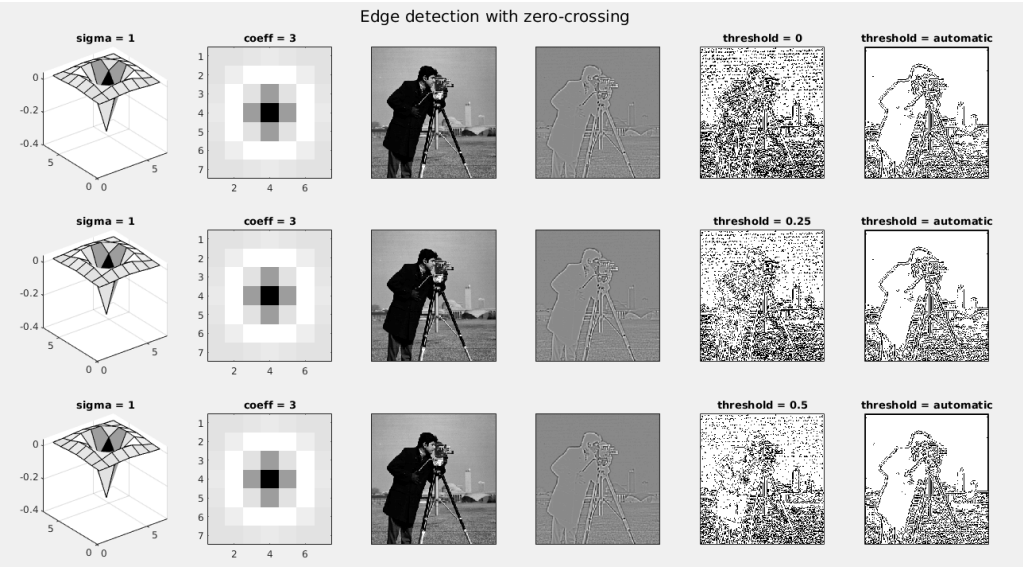

Figure 5.6: Edge detection with a 7x7 LoG filter

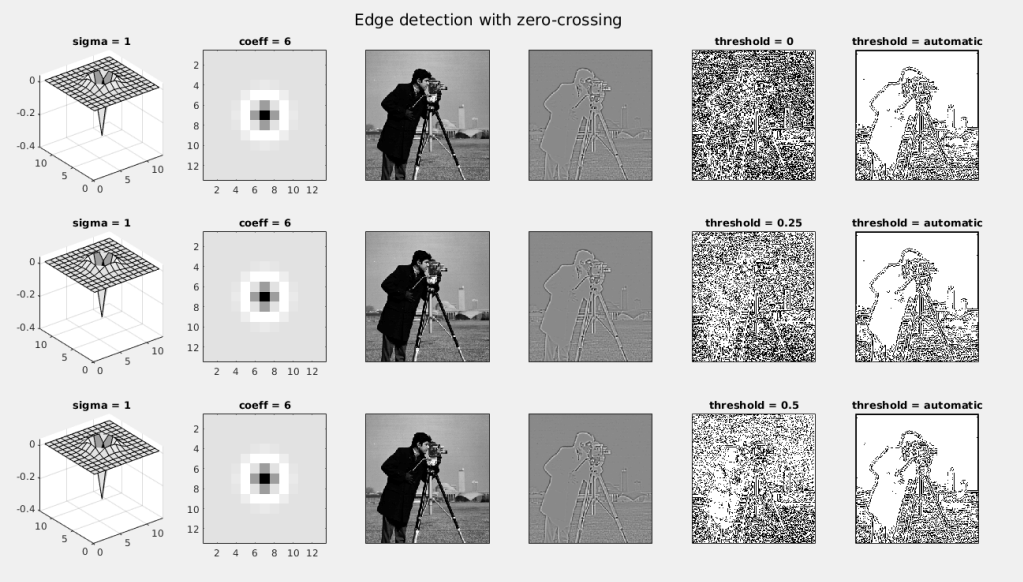

Figure 5.7: Edge detection with a 13x13 LoG filter

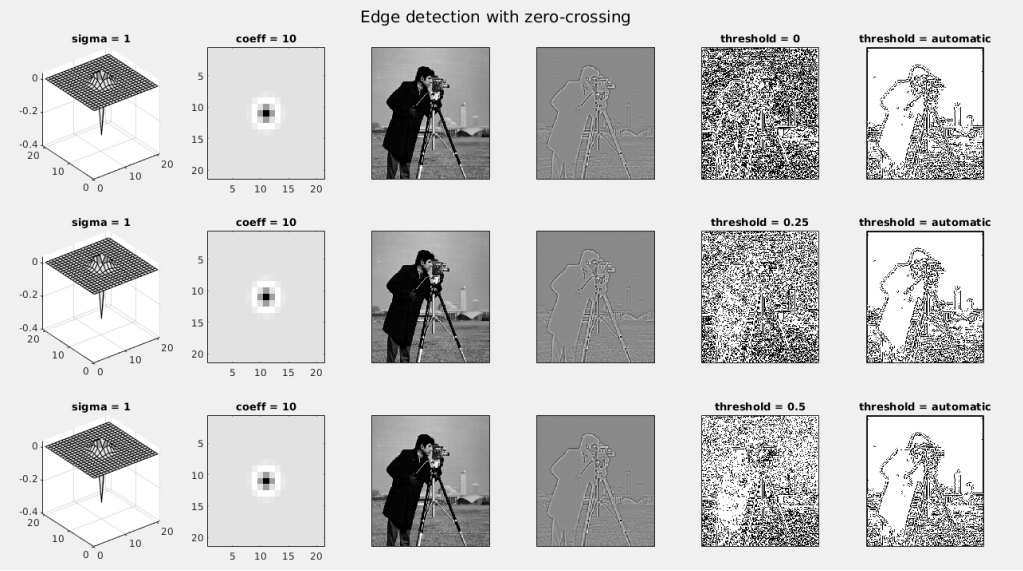

Figure 5.8: Edge detection with a 21x21 LoG filter

Figures **5.6**, **5.7**, **5.8** are brought as an example of the aforementioned properties of the kernel. By increasing *coeff* we can clearly observe that the filter let pass more information, and consequently the process considers as edges also little variations.

- **th:** this parameter indicates the minimum level of intensity variation that must be marked as edge.

  To support what up mentioned we show 9 significant subplots each one with a different threshold values.

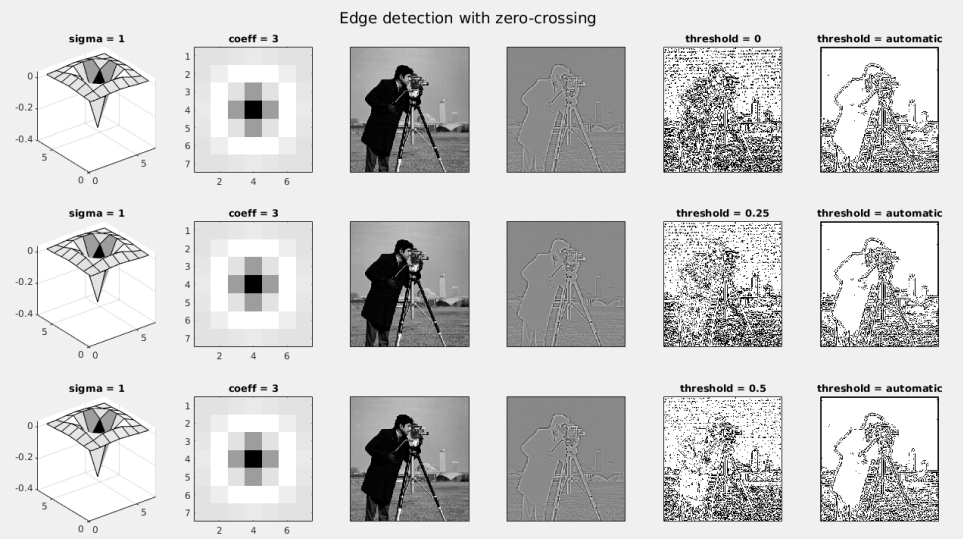

Figure 5.9: Resulting subplot by varying the threshold with a constant step

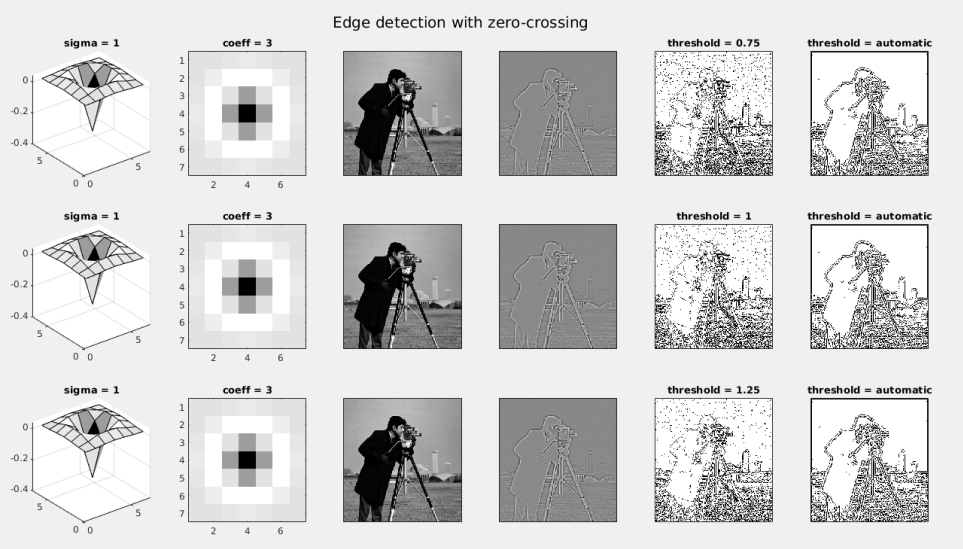

Figure 5.10: Resulting subplot by varying the threshold with a constant step