Computer Vision Laboratory Report n.3

EDGE DETECTION

April 7, 2019

Claudio Curti ID: 4216203
Nicola De Carli ID: 4198668
Alberto Ghiotto ID: 4225586
University of Genoa

Contents

Chapter 1

Introduction

1.1 Defining the objective

This project aims to get a clear and in-depth analysis of a very important topic of computer vision: objects' boundary detection. A clear understanding of this key concept is fundamental to the implementation of a working visual perception system.
In order to perform edge detection this experiment will focus on finding the zero crossing of the original image convoluted with the Laplacian of Gaussian.
With the purpose of highlighting all the different effects of all the existing parameters under study a set of distinct simulations with varying parameters was carried out.

Chapter 2

Theoretical Background

Since the noise values at each pixel are typically uncorrelated they would result as edges when applying edge detecting algorithms. Therefore, to achieve the edge detection goal, at first the image is filtered to remove noise using, in this case, a Gaussian filter. This algorithm works by considering the second derivative of the filtered image and then looking for the zero-crossing in order to find the local maxima, which corresponds to the edges. Since the differentiation is linear and shift invariant and convolution is associative, it is possible to compute the second derivative of the Gaussian filter, thus obtaining the Laplacian of a Gaussian (LoG), apply the convolution with the original image, and eventually check for the zero-crossing. In this way only one convolution needs to be performed at run-time on the image. In order to avoid considering too many small details in the resulting binary image, the resulting zero-crossing taken in consideration are only those with a slope greater than a certain fixed threshold. Therefore, there are two main parameters which determine the final outcome: the threshold and the standard deviation of the Gaussian filter. In the following section different combination of these parameters will be compared and their impact on the algorithm will be discussed.

Chapter 3

Implementation and Results

The code is organized using a script, which loads the grayscale image in the workspace and then calls all the developed functions. In order to carry out all the required manipulation the following functions were developed:

- **doLog**: which implements the Laplacian of Gaussian operator with the parameters indicated when calling the function and displays the results.
- **filterLoG**: which filters the loaded image by convolving it with the LoG and displays the results.
- **edgeDetection**: which creates a binary image with the edges of the provided image by detecting the zero crossing in the filtered image.
- **compareAlg**: which compares the result obtained by the "edgeDetection" function developed in this experiment with the one obtained by using the MATLAB function "edge".

3.1 Laplacian of Gaussian operator

In order to implement the analysis outlined in the introductory chapter the first operation to do was the implementation of the Laplacian of Gaussian Operator. As expressly requested by the text, the implemented function takes as input both the standard deviation and the support of the gaussian function, although the support must be an odd number at least six times larger than the support. To avoid possible incorrect input values, a control is applied at the beginning of the function in such a way that if an invalid support is passed to the function, the minimum feasible support is computed and used to estimate the Laplacian of Gaussian operator. The latter is computed according to the following formula, which correspond to the second derivative of a Gaussian function:

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

The LoG is eventually displayed both as 2D image and as a surface. In the following images it is possible to see different shapes of this operator varying the standard deviation and the support. From these images it is possible to see that the Laplacian of the Gaussian operator has negligible values over $3 * \sigma$ from the center, as inherited from the standard Gaussian function.

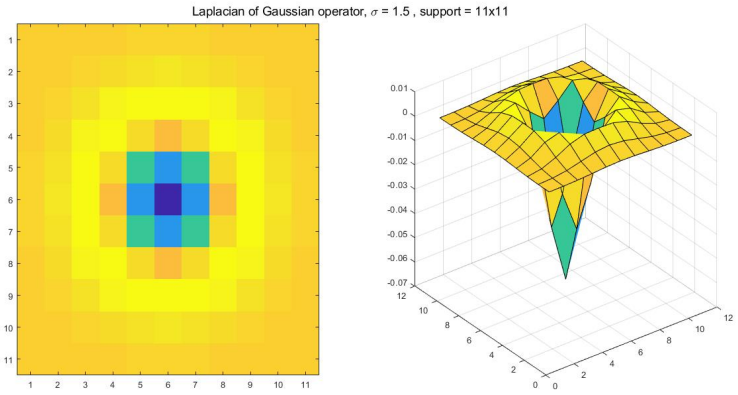

Figure 3.1: LoG operator with $\sigma = 1.5$, support = 11x11.

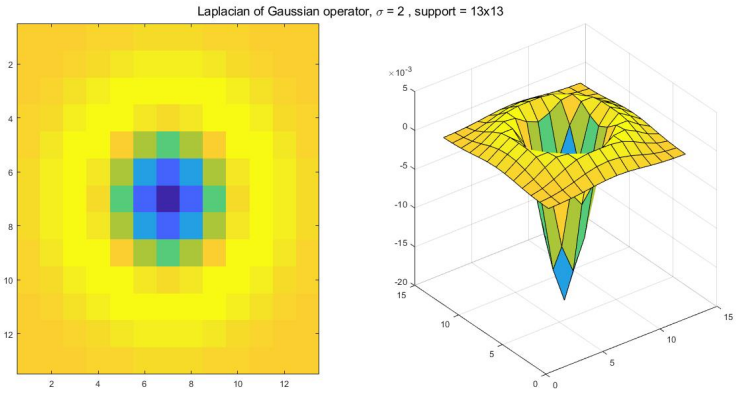

Figure 3.2: LoG operator with $\sigma = 2$, support = 13x13.

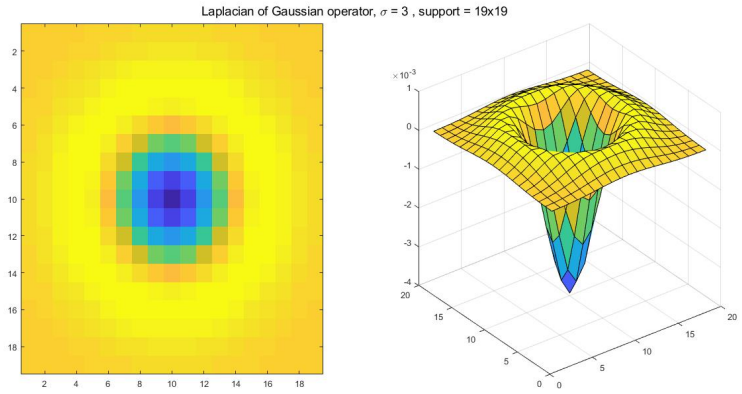

Figure 3.3: LoG operator with $\sigma = 3$, support = 19x19.

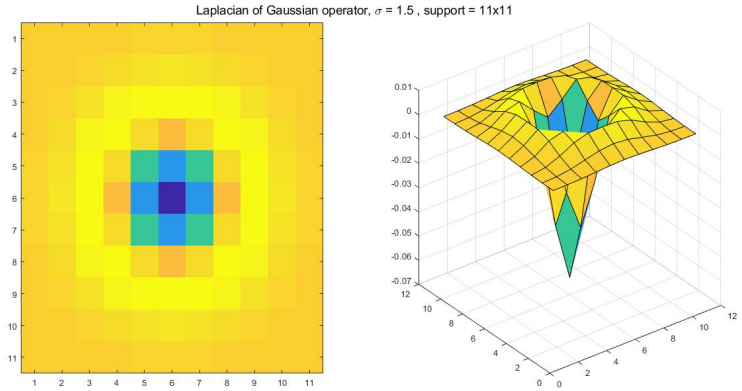

Figure 3.4: LoG operator with $\sigma = 1.5$, support = 11x11.

3.2 Filtering with LoG operator

The Laplacian of Gaussian filter is simply convoluted with the original image to produce an image which contains the structural properties needed to find the edges. Using the same values of the previous part (standard deviation and support), it is possible to see the different results of the convolution.

Figure 3.5: Filtered Image with $\sigma = 1.5$, support = 11x11.

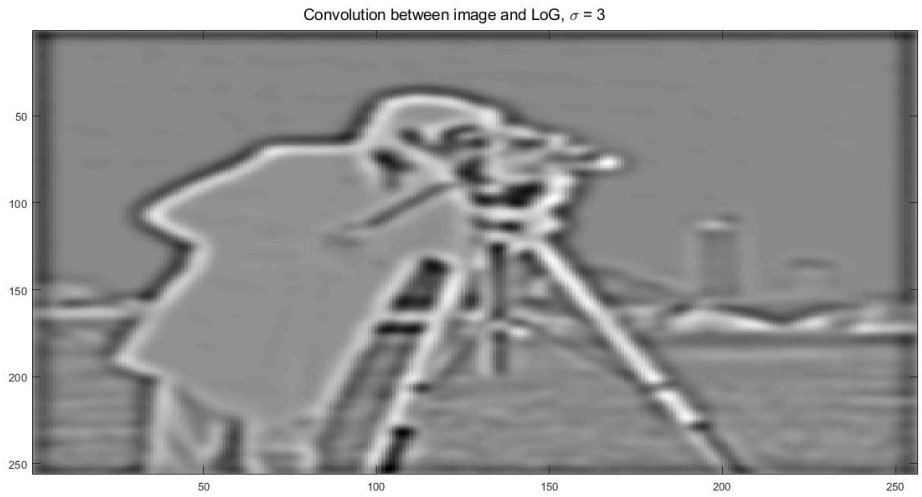

Figure 3.6: Filtered Image with $\sigma = 3$, support = 19x19.

Figure 3.7: Filtered Image with $\sigma = 1.5$, support = 19x19.

As it can be seen from Figure 3.7 above, incrementing the size of the support without changing the standard deviation doesn't change the resulting image in any way. This is the result of the truncation after about three standard deviation from the center due to its negligible effects.

Figure 3.8: Filtered Image with $\sigma = 2$, support = 13x13.

3.3 Edge detection

Once the convolution is computed, it is possible to proceed with the edge detection searching for the zero-crossing. This procedure is divided in two phases. In the first part of the algorithm, all the columns of the image are scanned and the edge is detected only if two adjacent pixels (horizontals) show a change of sign +- or -+ and the module of their difference (the slope) is greater than a threshold. The case in which the zero crossing pass for a pixel whose value is actually zero +0- and -0+ is also possible, although it is really unlikely, thus this possibility is also checked. Each time an edge is detected, a corresponding pixel arbitrarily chosen (in this case the left pixel) is saved to 1 (white) in a binary image with the same size of the original one. The second part is analogous to the first one, in this case the rows are scanned so the correspondence to look for is between adjacent vertical pixels. At the end of the whole process an image containing the edges corresponding to a certain threshold is produced. In the following images some examples of results with various sigma and thresholds are shown.

In the Figure 3.9 ($\sigma = 2$ and $th = 0.6$) there is an example of balanced edge detection.

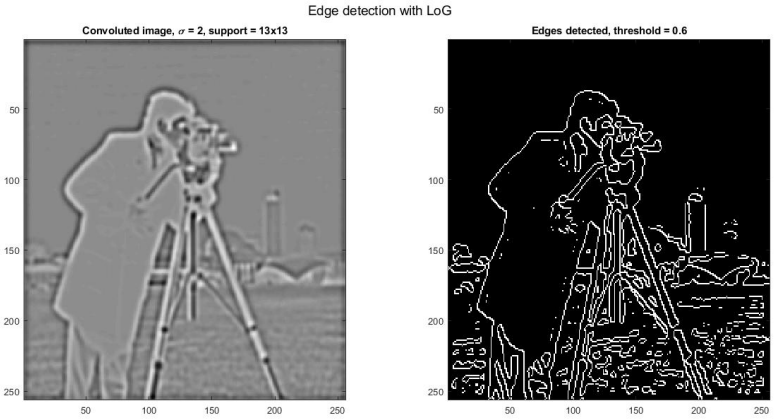

Figure 3.9: Edge detection with $\sigma = 2$, support = 13x13, threshold = 0.6.

In Figure 3.10 σ has been increased to 3, highlighting the fact that smoothing on a scale smaller than the width of the building on the background allows the algorithm to detect its edges, viceversa with greater sigma values (as in this case) the building is not visible anymore.

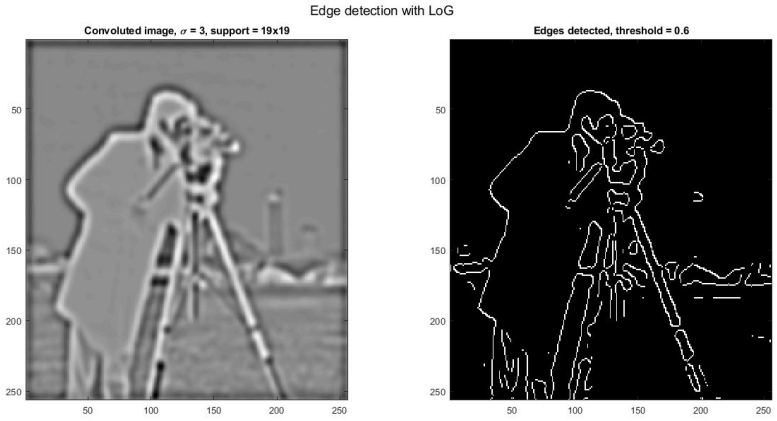

Figure 3.10: Edge detection with $\sigma = 3$, support = 19x19, threshold = 0.6.

In Figure 3.11, σ has been decreased to 1.5 and it is possible to notice the presence of much more noise.

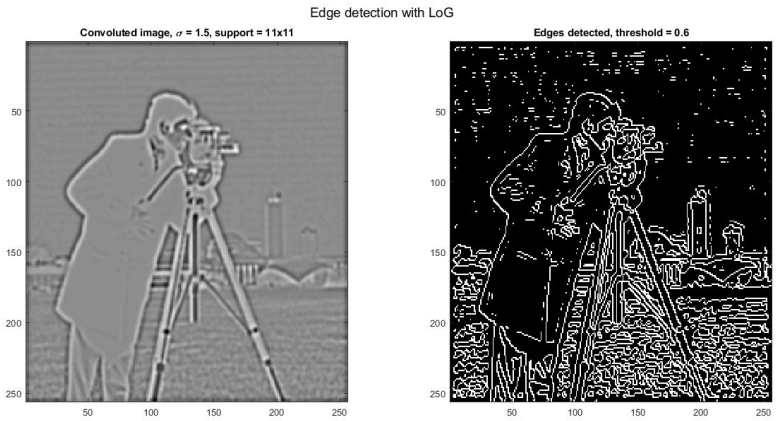

Figure 3.11: Edge detection with $\sigma = 1.5$, support = 11x11, threshold = 0.6.

Finally, in Figure 3.12, σ is again equal to 2, instead the threshold has been decreased to 0.2 and therefore a lot more details are visible.

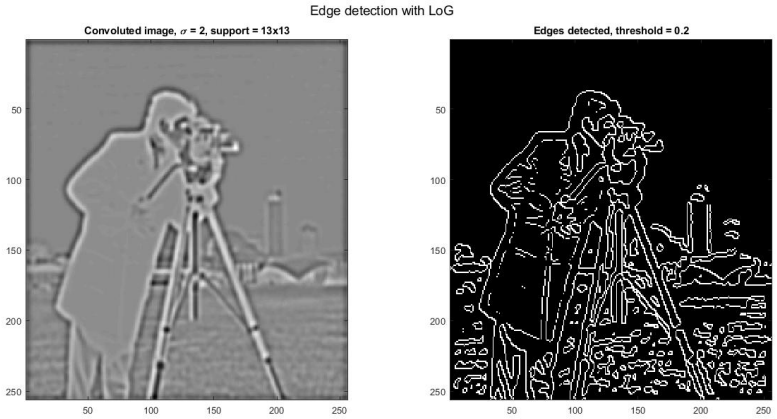

Figure 3.12: Edge detection with $\sigma = 2$, support = 13x13, threshold = 0.2.

3.4 Results comparison

A final comparison between the MATLAB predefined function "edge" and the user-defined function "edgeDetection" is shown in the following image, in which the goal has been to find the best combination of standard deviation and threshold to get the two output images as similar as possible.

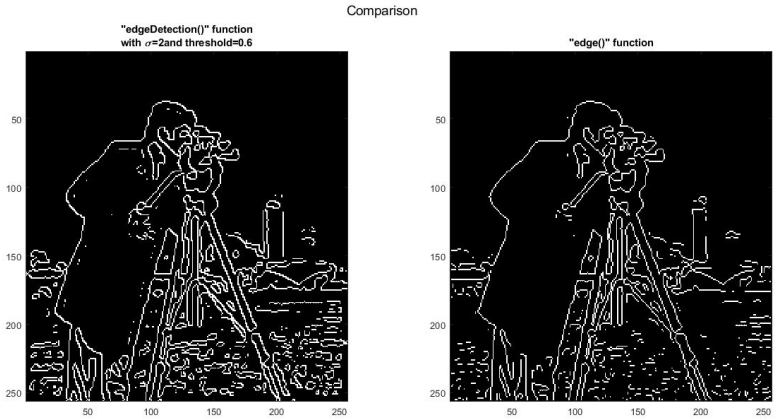

Figure 3.13: Comparison between MATLAB "edge" function and the one developed in this experiment.