A Review of Classic Edge Detectors

Haldo Spontón, Juan Cardelino

Instituto de Ingeniería Eléctrica, Universidad de la República, Uruguay ({haldos, juanc}@fing.edu.uy)

Abstract

In this paper some of the classic alternatives for edge detection in digital images are studied. The main idea of edge detection algorithms is to find where abrupt changes in the intensity of an image have occurred. The first family of algorithms reviewed in this work uses the first derivative to find the changes of intensity, such as Sobel, Prewitt and Roberts. In the second reviewed family, second derivatives are used, for example in algorithms like Marr-Hildreth and Haralick. The obtained results are analyzed from a qualitative point of view (perceptual) and from a quantitative point of view (number of operations, execution time), considering different ways to convolve an image with a kernel (step required in some of the algorithms).

Source Code

For all the reviewed algorithms, an open source C implementation is provided which can be downloaded from the IPOL web page of this article[1]. An online demonstration is also available, where the user can test and reproduce our results.

Keywords: edge detection; Sobel; Prewitt; Roberts; Haralick; Marr-Hildreth; derivatives

1 Introduction

Edge detection is one of the oldest and most basic operations in image processing, which is often used as a basic building block for more complex algorithms. The basic idea of edge detection is to detect abrupt changes in image intensity. Detecting those changes can be accomplished using first or second order derivatives.

In the 70's, edge detection methods were implemented using small operators (such as Sobel masks), attempting to compute an approximation of the first derivative of the image [16]. Section 2 describes such algorithms and serves as an introduction to a more sophisticated analysis of the edge detection process.

In 1980 Marr and Hildreth [9] argued that intensity changes are not independent of image scale, so edge detection requires the use of different size operators. They also argued that a sudden intensity change will be seen as a peak (or valley) in the first derivative or, equivalently, as a zero crossing in the second derivative. This algorithm is presented in Section 3. Haralick's algorithm [8] is an alternative

---

[1]http://dx.doi.org/10.5201/ipol.2015.35

approach based on the second derivative which is also reviewed in Section 3. This algorithm has the particularity of proposing a model to approximate the image locally around a point; then, using this model, an approximation to the second derivative of the image can be calculated analytically and finding edges is achieved by imposing a condition over the model parameters.

The objective of this work is to provide reproducible implementations of the abovementioned algorithms, along with a comprehensive quantitative evaluation of the behavior of the algorithms with respect to the parameters. Quantitative benchmarks are outside the scope of this work and will be addressed in the future. In addition, we will focus on low-level edge detection algorithms, which means that they will output a disconnected and unordered set of edgels (individual pixel facets where boundaries occur). Algorithms that perform edge integration into objects boundaries are of higher level of abstraction and not considered in this paper.

Along with this paper, a detailed and well commented source code is presented, which implements the described algorithms. In Section 4 some common mathematical developments are presented. Results of the implemented algorithms are presented in Section 5, along with examples to compare their performance. Conclusions are detailed in Section 6.

2 Algorithms Based on the First Derivative

The algorithms based on first derivatives studied in this paper share a common structure, and only differ in the type of filtering used to compute those derivatives. Figure 1 shows a block diagram of that common structure.
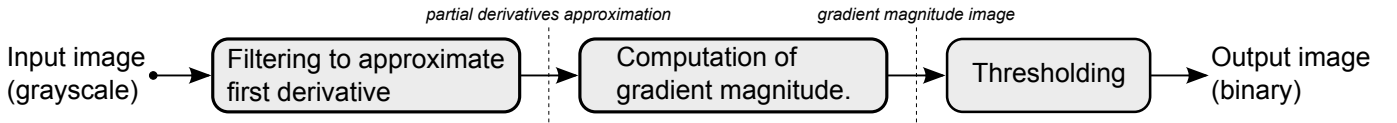

Figure 1: Block diagram of first derivative edge detection algorithms.

The usual tool to find the magnitude and direction of the intensity changes of an image f is the gradient operator (denoted as ∇), defined as the vector

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}. \tag{2.1}$$

The magnitude (M) and direction (α) of the gradient vector ∇f at location (x, y) are calculated as

$$\begin{cases} M(x, y) = \sqrt{f_x^2 + f_y^2} \\ \alpha(x, y) = \tan^{-1}\left(\dfrac{f_y}{f_x}\right). \end{cases} \tag{2.2}$$

The direction of an edge at an arbitrary location (x, y) of the image is orthogonal to the direction $\alpha(x, y)$ of the gradient vector.

To obtain the gradient, the partial derivatives $\partial f / \partial x$ and $\partial f / \partial y$ need to be computed at every pixel of the image. When dealing with digital images, numerical approximations of these derivatives are computed in a neighborhood of each point. In the following, the methods of Roberts, Prewitt and Sobel will be studied, whose main difference is how they perform this calculation.

2.1   The *Roberts* Operators

The Roberts edge detector was introduced in 1963 by L.G. Roberts in the context of 3D reconstruction [15], and it was based on the computation of the first derivative of the image. The most usual approach to approximate the first derivative is to use the first order Taylor expansion with a small value of h. Thus $f'(x)$ is computed as

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h}. \tag{2.3}$$

Digital images are commonly seen as a discretization of a continuous image, where a pixel is evenly sampled on a grid. If we denote the grid sizes by Δx and Δy, the continuous coordinates x and y can be written as $x = i\Delta x$ and $y = i\Delta y$. When no information about the spatial sampling is available, the size of the grid it is commonly assumed to be $\Delta x = \Delta y = 1$. Thus, in the discrete space the first derivative of the image is computed as:

$$f_x = \frac{\partial f(x,y)}{\partial x} \cong f(i+1, j) - f(i, j) \tag{2.4}$$

and

$$f_y = \frac{\partial f(x,y)}{\partial y} \cong f(i, j+1) - f(i, j). \tag{2.5}$$

Equations (2.4) and (2.5) can be implemented for all values of x and y by filtering the image $f(x, y)$ with the 1-D masks shown in Figure 2. However, in order to be able to compute diagonal edges, 2-D masks are needed. The *Roberts operators* are one of the earliest attempts to use 2-D masks for this purpose. These operators are based on computing the diagonal differences implemented by filtering an image with the masks in Figure 3. By convention in these figures, x-coordinates grow from left to right and y-coordinates grow from top to bottom. It is also usual to scale the mask values, in order to have gain equal to 1.

-1
1

-1	1

Figure 2: One-dimensional masks used to implement equations (2.4) and (2.5).

-1	0
0	1

0	-1
1	0

Figure 3: *Roberts cross-gradient* 2-D masks.

In order to use the same convolution code for all algorithms, our implementation uses 3×3 matrices, adding a row and a column of zeros to the matrices in Figure 3, which produces the same result.

2.2   The *Prewitt* Operators

Masks of size 2×2, although conceptually simple, are not symmetrical with respect to the central points. Having symmetrical edges is a desirable property and can only be achieved with oddly sized

masks, the smallest of them being the 3×3. These masks provide more information to find the direction of the edges, because they take into account information on opposite sides of the central point.

The simplest digital approximation of the partial derivatives using masks of size 3×3 is obtained by taking the difference between the third and first rows (or columns) of the 3×3 region. The difference between the third and first rows approximates the derivative in the x-direction, and the difference between the third and first columns approximates the derivative in the y-direction. These approximations can be implemented by filtering the image with the two masks shown in Figure 4. These masks are called *Prewitt operators* [14].

$-\frac{1}{3}$	$-\frac{1}{3}$	$-\frac{1}{3}$
0	0	0
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$

$-\frac{1}{3}$	0	$\frac{1}{3}$
$-\frac{1}{3}$	0	$\frac{1}{3}$
$-\frac{1}{3}$	0	$\frac{1}{3}$

Figure 4: Normalized *Prewitt* 2-D masks of size 3×3.

The left hand mask can be constructed (up to a scale factor) as the convolution of two filters: a vertical derivative $[-1\ 0\ 1]^T$ and an horizontal moving average $[1\ 1\ 1]$. The same reasoning could be applied to the right hand mask. Hence, these operators can be interpreted as the consecutive application of an horizontal low pass filter with a vertical derivative filter, and vice versa. Thus, they show smoothing properties in the direction orthogonal to the gradient.

2.3 The *Sobel* Operators

Sobel operators are a slight variation of the Prewitt operators which use more weight on the central coefficients of the masks. It can be shown that using more weight in the center location provides better image smoothing. This variation is implemented using masks shown in Figure 5.

$-\frac{1}{4}$	$-\frac{1}{2}$	$-\frac{1}{4}$
0	0	0
$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$

$-\frac{1}{4}$	0	$\frac{1}{4}$
$-\frac{1}{2}$	0	$\frac{1}{2}$
$-\frac{1}{4}$	0	$\frac{1}{4}$

Figure 5: *Sobel* 2-D masks of size 3×3.

Sobel masks can be seen (up to a scale factor) as the convolution of a vertical derivation mask $[-1\ 0\ 1]^T$ with an horizontal smoothing filter $[1\ 2\ 1]$ (closer to a Gaussian response than Prewitt), and vice versa. Hence, these operators have better smoothing properties, as mentioned in the preceding paragraph.

The computational cost of applying Prewitt and Sobel masks is exactly the same. Thus, it is preferable to use Sobel operators, because edges are better localized, and its filters are also less aliased, because of the weighted shape of the kernel.
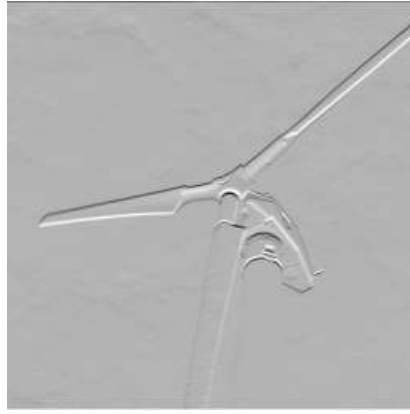
2.4 Computation of the Edges

As mentioned before, edges can be seen as abrupt changes in intensity, which correspond to high values of the gradient. An example of this behavior is shown in Figure 6, where gradient was computed using Sobel operators.

(a) Grayscale image.
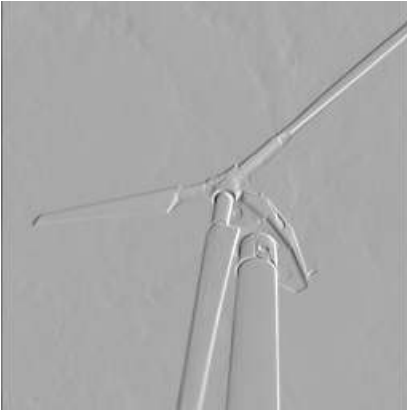
(b) Horizontal derivative f_x.

(c) Vertical derivative f_y.

(d) Gradient module image $M = \sqrt{f_x^2 + f_y^2}$.

Figure 6: Example computation of the gradient. Original image and gradient image computed using Sobel operators (operators in Figure 5).

Then, once the respective operators are applied, an approximation of the gradient (stored in two matrices) is obtained, containing the approximation of the partial derivatives f_x and f_y. The gradient magnitude image M is calculated using Equation (2.2) (Figure 6(d)).

Finally, the edges image is obtained by thresholding the gradient magnitude image, i.e. for a pixel at position i, j in the image the algorithm computes

$$\text{edges}[i, j] = \begin{cases} 1, & \text{if } M[i, j] \geq \text{th} \\ 0, & \text{if } M[i, j] < \text{th} \end{cases}$$

where th is the threshold. A black and white image is obtained, in which edge points are indicated in white (see Figure 7). It is noted that different threshold values lead to different results: thicker or more edges for smaller thresholds, fewer and sparser edges for larger thresholds. This threshold, and the others mentioned below, are defined as a percentage of the maximum value of the gradient image. The advantage of this approach is to adapt the threshold to the dynamic range of the image. Therefore, the parameter $threshold$ in the algorithms takes values between 0 and 1.

2.5   Pseudo-code

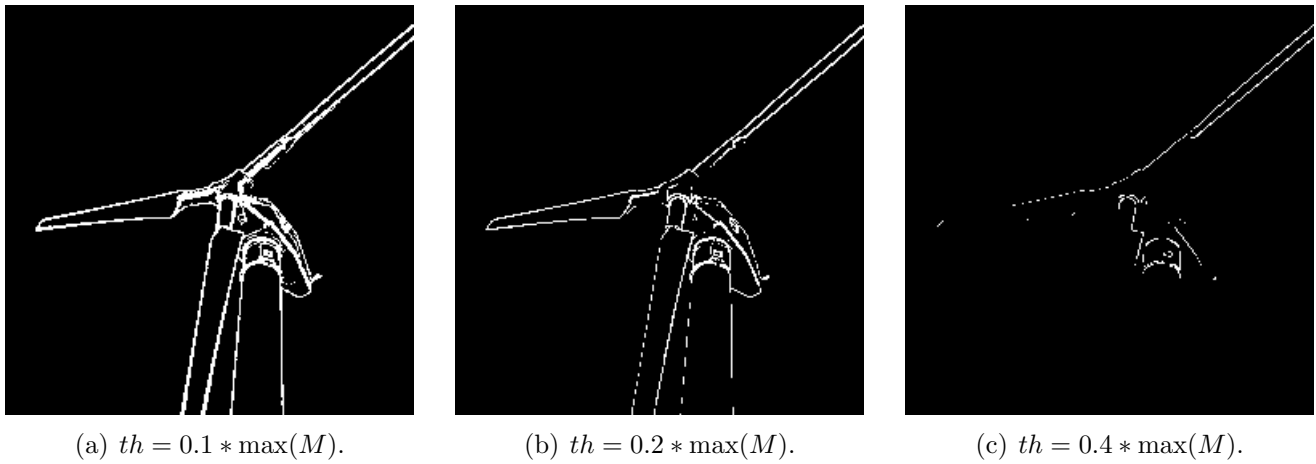The pseudo-code of the implemented algorithms is shown in Algorithm 1.

(a) $th = 0.1 * \max(M)$.   (b) $th = 0.2 * \max(M)$.   (c) $th = 0.4 * \max(M)$.

Figure 7: Thresholded images. Different values of the threshold applied to the magnitude of the gradient.

Algorithm 1 First derivative edge detection algorithms.

Require: im input image, threshold th.

1: Define $operator_x$ and $operator_y$ ▷ Roberts, Prewitt or Sobel.
2: $g_x \leftarrow$ convolution(im,$operator_x$)
3: $g_y \leftarrow$ convolution(im,$operator_y$)
4: $max_M \leftarrow 0$
5: **for all** pixel i in image **do**
6:      $M[i] \leftarrow \sqrt{g_x^2 + g_y^2}$ ▷ Gradient magnitude.
7:      **if** $M[i] > max_M$ **then**
8:          $max_M \leftarrow M[i]$
9:      **end if**
10: **end for**
11: **for all** pixel i in image **do**
12:      **if** $M[i] \geq th \times max_M$ **then**
13:          $edges[i] \leftarrow 1$
14:      **else**
15:          $edges[i] \leftarrow 0$
16:      **end if**
17: **end for**
18: **return** $edges$

---

3 Algorithms Based on the Second Derivative

The edge detection methods discussed in the previous section are simply based on filtering the image with different masks, without taking into account the characteristics of the edges or the noise in the image. The algorithms that will be presented in this section (Marr-Hildreth [9] and Haralick [8]) are based on the second derivative of the image, and take steps to reduce noise before detecting edges in the image.

3.1 The *Marr* and *Hildreth* Algorithm

The Marr-Hildreth algorithm is a method of detecting edges in digital images. It is based on finding zero crossing points of the second derivative of the image. This computation can be done in several ways, but in this work we implement two of these ways (see block diagram in Figure 8): convolving the image with a Gaussian kernel and then approximating the second derivative (Laplacian) with a 3×3 kernel, or convolving the image with a kernel calculated as the Laplacian of a Gaussian function. There are more ways to perform the same tasks, for example, using recursive Gaussian filters [6].

The algorithm is divided in two steps, which will be described later:

1. Convolution of the image with:

   - a Laplacian of Gaussian (LoG) kernel, (or)
   - a Gaussian kernel and then a Laplacian operator.

2. Search of zero crossing points in the filtered image.

In order to implement this algorithm some auxiliary functions are needed, such as Gaussian kernel and Laplacian of a Gaussian kernel generation, and 2-D convolution of an image with a given kernel, using different boundary conditions. These operations will be discussed in detail in Section 4.

3.1.1 Gaussian and LoG Kernels

The main idea of Marr-Hildreth's algorithm is to convolve the input image $f(x,y)$ with a LoG kernel;

$$g(x,y) = [\nabla^2 G(x,y)] \star f(x,y), \tag{3.1}$$

and then finding the zero crossings of $g(x,y)$ to determine the location of edges in $f(x,y)$. Because these are linear processes, Equation (3.1) can be written also as

$$g(x,y) = \nabla^2 [G(x,y) \star f(x,y)], \tag{3.2}$$

indicating that the image can be smoothed with a Gaussian filter first, and then compute the Laplacian of the result[2]. Then, the algorithm requires the generation of both Gaussian and LoG kernels (see Section 4).

The Marr-Hildreth edge-detection algorithm may be summarized as follows:

1. Filter the input image with a $n \times n$ Gaussian lowpass filter obtained by sampling the Gaussian kernel (see Equation (4.1)).

---

[2]The difference between these approaches lies in the compromise between the accuracy in the calculation and the computational cost (see Section 4.1.3).
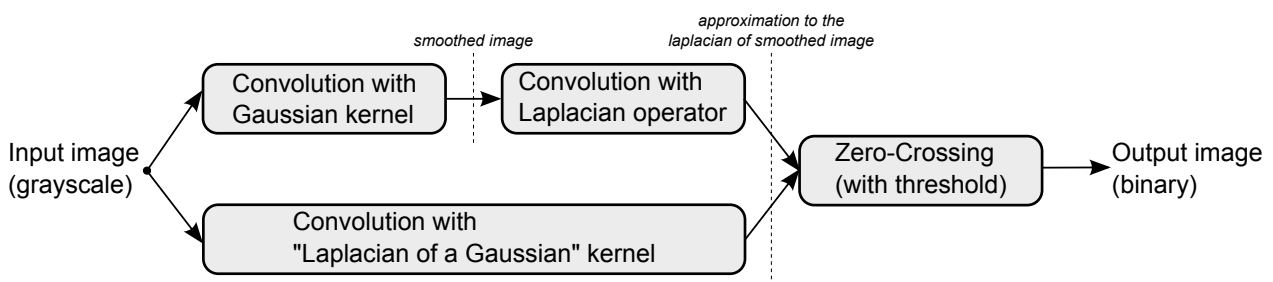

Figure 8: Block diagram of the Marr-Hildreth algorithm.

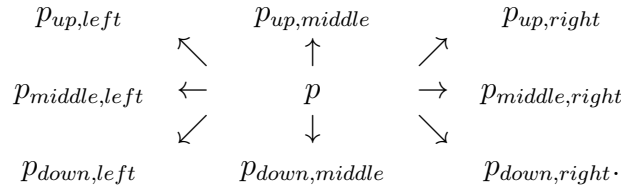2. Compute the Laplacian of the image resulting from step 1, using, for example, the 3×3 mask[3]:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.3}$$

3. Find the zero crossings of the image from step 2.

3.1.2 Zero Crossing

A zero crossing at pixel p implies that the signs of at least two opposite neighboring pixels are different. There are four cases to test: left/right, up/down, and the two diagonals. In this case two conditions must hold: first, the signs of the opposite pixels must differ, and second, their difference in absolute value must be greater than a certain threshold.

The zero-crossing threshold (th_{ZC}) is given as a percentage of the maximum value max_L of the Laplacian image for both Gaussian and LoG kernels. Each pixel p has eight neighbors, named according to their position as follows:

$$
\begin{array}{ccc}
p_{up,left} & p_{up,middle} & p_{up,right} \\
\nwarrow & \uparrow & \nearrow \\
p_{middle,left} \leftarrow & p \rightarrow & p_{middle,right} \\
\swarrow & \downarrow & \searrow \\
p_{down,left} & p_{down,middle} & p_{down,right}.
\end{array}
$$

Then a pixel p is considered an edge pixel if any of the following conditions is true (for simplicity the Laplacian image is denoted as $\mathcal{L}$):

- $(\text{sign}(\mathcal{L}[p_{up,left}]) \neq \text{sign}(\mathcal{L}[p_{down,right}]) \ \& \ |\mathcal{L}[p_{up,left}] - \mathcal{L}[p_{down,right}]| > th_{ZC} * max_L$

- $(\text{sign}(\mathcal{L}[p_{up,middle}]) \neq \text{sign}(\mathcal{L}[p_{down,middle}]) \ \& \ |\mathcal{L}[p_{up,middle}] - \mathcal{L}[p_{down,middle}]| > th_{ZC} * max_L$

- $(\text{sign}(\mathcal{L}[p_{down,left}]) \neq \text{sign}(\mathcal{L}[p_{up,right}]) \ \& \ |\mathcal{L}[p_{down,left}] - \mathcal{L}[p_{up,right}]| > th_{ZC} * max_L$

- $(\text{sign}(\mathcal{L}[p_{middle,left}]) \neq \text{sign}(\mathcal{L}[p_{middle,right}]) \ \& \ |\mathcal{L}[p_{middle,left}] - \mathcal{L}[p_{middle,right}]| > th_{ZC} * max_L$.

Zero crossing detection is the key feature of the Marr-Hildreth edge detection method. The technique presented in the previous paragraph is attractive for its simplicity of implementation and its low computational cost. In general it yields good results, but for applications that require accurate edge localization, more advanced methods for finding zero crossings with subpixel accuracy could be used (e.g. marching squares [10]).

3.1.3 Pseudo-code

The pseudo-code of Marr-Hildreth's algorithm is shown in Algorithm 2.

3.2 The *Haralick* Algorithm

In this section we present a detailed description of Haralick's original work [8]. The main idea of this algorithm is identical to that of the previous method: find zeros in the second derivative of the image. In this method, however, the input image is smoothly approximated through local bi-cubic polynomial fitting. Then, when calculating the second derivative analytically, it is possible to find an equivalent expression to find the zeros of the second derivative of the polynomial as a function of its parameters.

---

[3]Steps 1 and 2 can be merged into one, using a $n \times n$ LoG lowpass filter obtained by sampling Equation (4.7).

Algorithm 2 Marr-Hildreth edge detection algorithm.
___
Require: im input image, standard deviation σ, kernel size n and zero-crossing threshold t_{ZC}.
1: **if** Gaussian kernel **then**
2:    $kernel \leftarrow$ generate_gaussian_kernel(n,σ)
3:    $im_{smooth} \leftarrow$ convolution($im,kernel$)
4:    Define Laplacian operator $laplacian$ ▷ See mask definition in Equation (3.3)
5:    $im_{lap} \leftarrow$ convolution($im_{smooth},laplacian$)
6: **else**
7:    $log\_kernel \leftarrow$ generate_log_kernel(n,σ)
8:    $im_{lap} \leftarrow$ convolution(im,log_kernel)
9: **end if**
10: $max_L \leftarrow 0$
11: **for all** pixel i in image im_{lap} **do**
12:    **if** $im_{lap}[i] > max_L$ **then**
13:        $max_L \leftarrow im_{lap}[i]$
14:    **end if**
15: **end for**
16: **for all** pixel i in image im_{lap}, except borders **do**
17:    **for all** pair (p_1,p_2) of opposite neighbors of p in im_{lap} **do**
18:        **if** $(\text{sign}(im_{lap}[p_1]) \neq \text{sign}(im_{lap}[p_2]))$ **and** $(|im_{lap}[p_1] - im_{lap}[p_2]| > th_{ZC})$ **then**
19:            $edges[i] \leftarrow 1$
20:        **else**
21:            $edges[i] \leftarrow 0$
22:        **end if**
23:    **end for**
24: **end for**
25: **return** $edges$
___

3.2.1 Bi-cubic Polynomial Fitting

Here, we present the interpolation method used in the original work of Haralick, although there are other options to do this [7].

The surrounding neighborhood of a point (x_0,y_0) in the image I is approximated using the following bi-cubic polynomial f as

$$f(x,y) = k_1 + k_2(x-x_0) + k_3(y-y_0) + k_4(x-x_0)^2 + k_5(x-x_0)(y-y_0)+$$
$$k_6(y-y_0)^2 + k_7(x-x_0)^3 + k_8(x-x_0)^2(y-y_0) + k_9(x-x_0)(y-y_0)^2 + k_{10}(y-y_0)^3. \quad (3.4)$$

To solve this fitting problem, it is necessary to take more neighbors than coefficients to be adjusted. As there are 10 coefficients to compute, the smallest neighborhood of odd size that accomplishes this has size 5×5. Having 10 coefficients and 25 data points leads to an overdetermined system, which can be solved using least squares or any other suitable fitting technique.

In Haralick's work, this polynomial approximation is expressed in a local coordinate system centered on each pixel. In this way, the center of the neighborhood (x_0,y_0) is always $(0,0)$, and the coordinates (x,y) are offsets from this center point (e.g. in a neighborhood of size 5×5, x and y take values between -2 and +2). Thus, the expression of the polynomial is

$$f(x,y) = k_1 + k_2 x + k_3 y + k_4 x^2 + k_5 xy + k_6 y^2 + k_7 x^3 + k_8 x^2 y + k_9 xy^2 + k_{10} y^3. \quad (3.5)$$

Then the solution is computed using least squares, and the local expression of the polynomial ensures that the values of the k_i coefficients are computed in the same way for every pixel in the image.

Thus, they can be computed by convolving the image with a set of fixed and precomputed masks, which results in a great speed-up of the algorithm. In the following we will state the approximation problem and show how to derive those masks from its solution.

Consider 25 points in a small neighborhood of point $(0,0)$. This gives us an equal number of equations to find 10 coefficients. As mentioned before, this leads to an overdetermined system which can be solved by least squares. By substituting the 25 data points into the polynomial Equation (3.5), the following system of equations is obtained,

$$f_1 = f(x_1, y_1) = k_1 + k_2 x_1 + k_3 y_1 + k_4 x_1^2 + k_5 x_1 y_1 + k_6 y_1^2 + k_7 x_1^3 + k_8 x_1^2 y_1 + k_9 x_1 y_1^2 + k_{10} y_1^3$$
$$f_2 = f(x_2, y_2) = k_1 + k_2 x_2 + k_3 y_2 + k_4 x_2^2 + k_5 x_2 y_2 + k_6 y_2^2 + k_7 x_2^3 + k_8 x_2^2 y_2 + k_9 x_2 y_2^2 + k_{10} y_2^3$$
$$\vdots$$
$$f_{25} = f(x_{25}, y_{25}) = k_1 + k_2 x_{25} + k_3 y_{25} + k_4 x_{25}^2 + k_5 x_{25} y_{25} + k_6 y_{25}^2 + k_7 x_{25}^3 + k_8 x_{25}^2 y_{25} + k_9 x_{25} y_{25}^2 + k_{10} y_{25}^3.$$

Using matrix notation, the system can be rewritten as

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{25} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & \cdots & y_1^3 \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 & \cdots & y_2^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{25} & y_{25} & x_{25}^2 & x_{25} y_{25} & y_{25}^2 & \cdots & y_{25}^3 \end{bmatrix} \times \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_{10} \end{bmatrix} \Rightarrow \mathbf{f} = \mathbf{A}\mathbf{k}.$$

Then, the least squares problem can be solved by using the normal equations,

$$(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{f} = \mathbf{k}.$$

Let us define a matrix $\mathbf{B} \in R^{10 \times 25}$ such that $\mathbf{B} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$, thus

$$\mathbf{k} = \mathbf{B}\mathbf{f}.$$

$$\begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,25} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,25} \\ \vdots & \vdots & \ddots & \vdots \\ b_{10,1} & b_{10,2} & \cdots & b_{10,25} \end{bmatrix} \times \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{25} \end{bmatrix} = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_{10} \end{bmatrix}.$$

For each coefficient (i from 1 to 10),

$$k_i = b_{i,1} f_1 + b_{i,2} f_2 + b_{i,3} f_3 + \cdots + b_{i,25} f_{25} \quad \Rightarrow \quad k_i = \mathbf{b_i} \mathbf{f}, \tag{3.6}$$

where $\mathbf{b_i}$ is the vector containing the $i - th$ row of B, i.e.

$$\mathbf{b_i} = \begin{bmatrix} b_{i,1} & b_{i,2} & \cdots & b_{i,25} \end{bmatrix}. \tag{3.7}$$

Let us rearrange $\mathbf{b_i}$ as a 5×5 matrix and denote it by $\mathbf{mask_i}$ as follows

$$\mathbf{mask_i} = \begin{bmatrix} b_{i,1} & b_{i,2} & \cdots & b_{i,5} \\ b_{i,6} & b_{i,7} & \cdots & b_{i,10} \\ \vdots & \vdots & \ddots & \vdots \\ b_{i,21} & b_{i,22} & \cdots & b_{i,25} \end{bmatrix}. \tag{3.8}$$

We will compute one value of k_i for each pixel on the image. Thus we can store these coefficients in a matrix $\mathbf{K_i}$ of the same size as the original image. Then, using the mask shown in Equation (3.8) we can obtain $\mathbf{K_i}$ as the convolution of the original image with $\mathbf{mask_i}$.

$$\mathbf{K_i} = f \star \mathbf{mask_i}. \tag{3.9}$$

As we expressed the approximation in a local coordinate system, the coefficients $b_{i,j}$ will be the same for every pixel, thus, this convolution is performed against a fixed and precomputed mask $\mathbf{mask_i}$. The numeric values of these 5×5 masks used in our experiments are shown in Table A.1.

3.2.2 Analytical Calculation of the Second Derivative

The main idea of Haralick's algorithm, as stated in the original work, is to find "negative sloped zero crossings of the second derivative". This means that it is actually looking for ascending step edges. According to Haralick's definition, a pixel will lie on such an edge if the following three conditions hold[4]:

1. The first derivative is non zero.

2. The second derivative is zero.

3. The third derivative is negative.

   After the neighborhood of each point of the image is approximated using the bi-cubic polynomial expression in Equation (3.5), the idea is to compute the derivatives of the image in the direction of the gradient using that approximation.

   Let θ be the gradient angle, defined clockwise with respect to the y-axis[5]. We approximate it using the first order terms of the polynomial (3.5) obtaining the following expression

$$\sin(\theta) = -\frac{k_2}{\sqrt{k_2^2 + k_3^2}}$$
$$\cos(\theta) = -\frac{k_3}{\sqrt{k_2^2 + k_3^2}}. \tag{3.10}$$

   Let us consider the line r passing by $(0,0)$ with angle θ. The coordinates of each point (x, y) on this line can be described in terms of the angle θ and the distance ρ to $(0,0)$ as follows

$$x = \rho \sin \theta$$
$$y = \rho \cos \theta.$$

If we substitute the above expressions in the bi-cubic polynomial of Equation (3.5), we obtain

$$f_\theta(\rho) = C_0 + C_1\rho + C_2\rho^2 + C_3\rho^3, \tag{3.11}$$

where

$$\begin{aligned}
C_0 &= k_1 \\
C_1 &= k_2 \sin(\theta) + k_3 \cos(\theta) \\
C_2 &= k_4 \sin^2(\theta) + k_5 \sin(\theta)\cos(\theta) + k_6 \cos^2(\theta) \\
C_3 &= k_7 \sin^3(\theta) + k_8 \sin^2(\theta)\cos(\theta) + k_9 \sin(\theta)\cos^2(\theta) + k_{10}\cos^3(\theta).
\end{aligned} \tag{3.12}$$

   The derivatives are obtained as follows

$$\begin{aligned}
f_\theta'(\rho) &= C_1 + 2C_2\rho + 3C_3\rho^2 \\
f_\theta''(\rho) &= 2C_2 + 6C_3\rho \\
f_\theta'''(\rho) &= 6C_3.
\end{aligned}$$

---

[4]Please note that there is a typo in Haralick's original paper. The statement of the idea is presented at the beginning of Section III, in the first column of page 42. The three conditions are presented at the end of the same section, in the second column. There it reads $f_\alpha''(\rho) < 0$, $f_\alpha''(\rho) = 0$ and $f_\alpha'(\rho) \neq 0$. Clearly, the first and second conditions can't be true at the same time, thus the first one should be $f_\alpha'''(\rho) < 0$.

[5]Although it seems a non standard choice, we followed exactly the original work.

The simplest way to impose the above-mentioned conditions is starting with the third one, i.e. that the third derivative has to be negative. Thus,

$$f_\theta'''(\rho) = 6C_3 < 0 \quad \Rightarrow \quad C_3 < 0. \tag{3.13}$$

The second condition that the second derivative is equal to zero becomes

$$f_\theta''(\rho) = 2C_2 + 6C_3\rho = 0. \tag{3.14}$$

Ideally, if the edge point falls exactly on a grid location, the polynomial function should be zero at $\rho = 0$. But due to the finite nature of the image grid, the edge could be located at non integer locations. Thus, the condition is relaxed asking that the polynomial becomes zero in a neighborhood of radius $\rho_0 > 0$ centered at $\rho = 0$. With this relaxation, condition (3.14) becomes

$$\exists \rho \in (-\rho_0, \rho_0) \text{ such that } 2C_2 + 6C_3\rho = 0. \tag{3.15}$$

Thus, the solution is $\rho_* = -\frac{C_2}{3C_3}$ and imposing $|\rho_*| < \rho_0$ we obtain the second condition

$$\left| \frac{C_2}{3C_3} \right| < \rho_0. \tag{3.16}$$

Finally, the first condition is that the first derivative is non zero,

$$f_\theta'(\rho) = C_1 + 2C_2\rho + 3C_3\rho^2 \neq 0. \tag{3.17}$$

If we evaluate f_θ' on the solution ρ_* using the above formula, we obtain

$$f_\theta'(\rho_*) = C_1 - \frac{C_2^2}{3C_3}. \tag{3.18}$$

By construction, θ is the direction of the gradient, which means that the edge will always be an ascending step. Thus by definition the function $f_\theta(\rho)$ is non-decreasing in a neighborhood of $\rho = 0$, which means that $f_\theta'(0) \geq 0$. This in turn implies that $C_1 \geq 0$. As shown before, imposing the first condition implies that $C_3 < 0$, and we shown that $C_1 \geq 0$, thus Equation (3.18) yields that $f_\theta'(\rho_*)$ will always be positive. This means that the first condition always holds by construction and does not need to be verified.

In Haralick's work ρ_0 is a fixed threshold which acts as a parameter of the algorithm. Its value must be greater than zero and less than the size of the grid (i.e. $\rho_0 < 1$). In this work we obtained the best results with values between 0.4 and 0.6.

3.2.3 Algorithm

The Haralick edge detection algorithm is summarized in the following 4 steps:

1. For each pixel in the image, find the coefficients $k_1 \ldots k_{10}$, as shown in Section 3.2.1.

2. Compute $\sin(\theta)$ and $\cos(\theta)$ (equations (3.10)).

3. Compute C_2 and C_3 (equations (3.12)).

4. If $\left| \frac{C_2}{3C_3} \right| < \rho_0$ and $C_3 < 0$, then the candidate is an edge point.

Algorithm 3 Haralick edge detection algorithm.

Require: im input image (width w, height h), edge point condition threshold ρ_0.

1: Define the ten 5×5 masks $(mask_1, \ldots, mask_{10})$ used to determine coefficients k_1 to k_{10} ▷ See Table A.1.

2: **for** $j = 1$ **to** 10 **do**

3:     $K_j \leftarrow$ **convolution**$(im, mask_j)$ ▷ K_j **is the convolution image of Equation** (3.9).

4: **end for**

5: **for all pixel** i **in image** im **do**

6:     **for** $j = 1$ **to** 10 **do**

7:         $k_j \leftarrow K_j(i)$ ▷ $K_j(i)$ **is the value of image** K_j **at location** i.

8:     **end for**

9:     $C_2 \leftarrow \frac{k_2^2 k_4 + k_2 k_3 k_5 + k_3^2 k_6}{k_2^2 + k_3^2}$

10:    $C_3 \leftarrow \frac{k_2^3 k_7 + k_2^2 k_3 k_8 + k_2 k_3^2 k_9 + k_3^3 k_{10}}{(\sqrt{k_2^2 + k_3^2})^3}$

11:    **if** $|C_2/3C_3| < \rho_0$ **and** $C_3 < 0$ **then**

12:        $edges[i] \leftarrow 1$

13:    **else**

14:        $edges[i] \leftarrow 0$

15:    **end if**

16: **end for**

17: **return** $edges$

3.2.4   Pseudo-code

The pseudo-code of Haralick's algorithm is shown in Algorithm 3.

4   Common Mathematical Operations

All implemented algorithms use some common mathematical operations that are independent of the algorithms themselves. These operations, although basic, have a great impact on the algorithm outcome, and thus they need to be implemented with care. In this section those operations are reviewed.

4.1   Kernel Generation

Some of the algorithms presented above require the use of a Gaussian kernel or a LoG kernel. These kernels are generated by sampling the corresponding analytical function, which in each case depends on the standard deviation σ of the Gaussian function. The result is an array of size n by n.

4.1.1   Gaussian Kernel

The Gaussian convolution mask is generated by sampling the 2-D Gaussian function (centered at $(0,0)$)

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2 + y^2}{2\sigma^2}} \tag{4.1}$$

where σ is the standard deviation (sometimes σ is called the *scale space constant*).

    The size of the kernel n and the standard deviation of the exponential function σ are both input parameters, but these are not strictly independent of each other. The value of n needs to be large

enough to ensure that no information is lost when creating the kernel G. To ensure this, n is taken equal to the first odd integer greater than 6σ. Larger values of n do not add more significant samples of G, and increase the number of operations in the convolution.

Figure 9 shows a Gaussian kernel, generated with $\sigma = 4$ and $n = 25$ (first odd integer greater than $6\sigma = 24$).
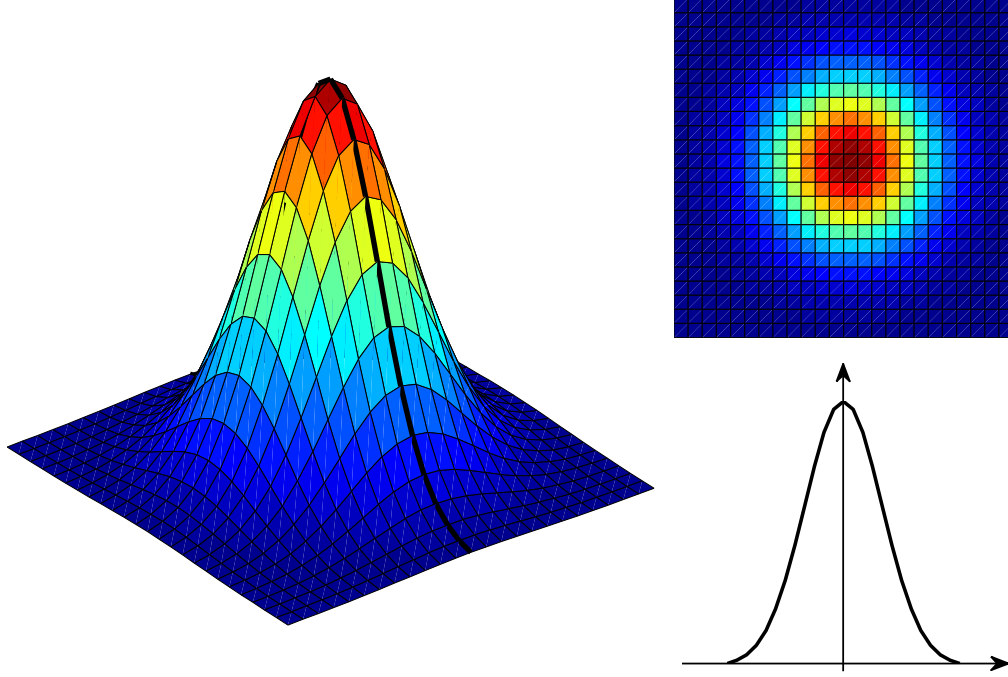

Figure 9: Gaussian kernel, $\sigma = 4$, $n = 25$. It is easy to see that the selected value of n is large enough to have a good approximation of the Gaussian function in the kernel.

4.1.2 LoG Kernel

The Laplacian of Gaussian $\nabla^2 G(x, y)$ can be obtained analytically first and then a discrete mask can be computed by sampling the analytical kernel. Using this kernel for edge detection involves only one convolution with the input image (unlike the Gaussian kernel case, in which two convolutions have to be performed, one with the kernel and another with the Laplacian operator), but the kernel support must be greater in order to obtain the same precision.

$$LoG \triangleq \nabla^2 G(x, y) = \frac{\partial^2}{\partial^2 x} G(x, y) + \frac{\partial^2}{\partial^2 y} G(x, y). \tag{4.2}$$

We first compute

$$\frac{\partial}{\partial x} G(x, y) = -\frac{1}{\sqrt{2\pi\sigma^2}} \frac{x}{\sigma^2} e^{-(x^2+y^2)/2\sigma^2}, \tag{4.3}$$

$$\frac{\partial}{\partial y} G(x, y) = -\frac{1}{\sqrt{2\pi\sigma^2}} \frac{y}{\sigma^2} e^{-(x^2+y^2)/2\sigma^2}, \tag{4.4}$$

$$\frac{\partial^2}{\partial^2 x} G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{x^2 - \sigma^2}{\sigma^4} e^{-(x^2+y^2)/2\sigma^2}, \tag{4.5}$$

$$\frac{\partial^2}{\partial^2 y}G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}}\frac{y^2 - \sigma^2}{\sigma^4}e^{-(x^2+y^2)/2\sigma^2}. \tag{4.6}$$

Therefore we obtain

$$LoG(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}}\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}e^{-(x^2+y^2)/2\sigma^2}. \tag{4.7}$$

Now the LoG kernel is generated by sampling the function defined in Equation (4.7). Figure 10 shows a LoG kernel, generated using the values $\sigma = 4$ and $n = 31$.
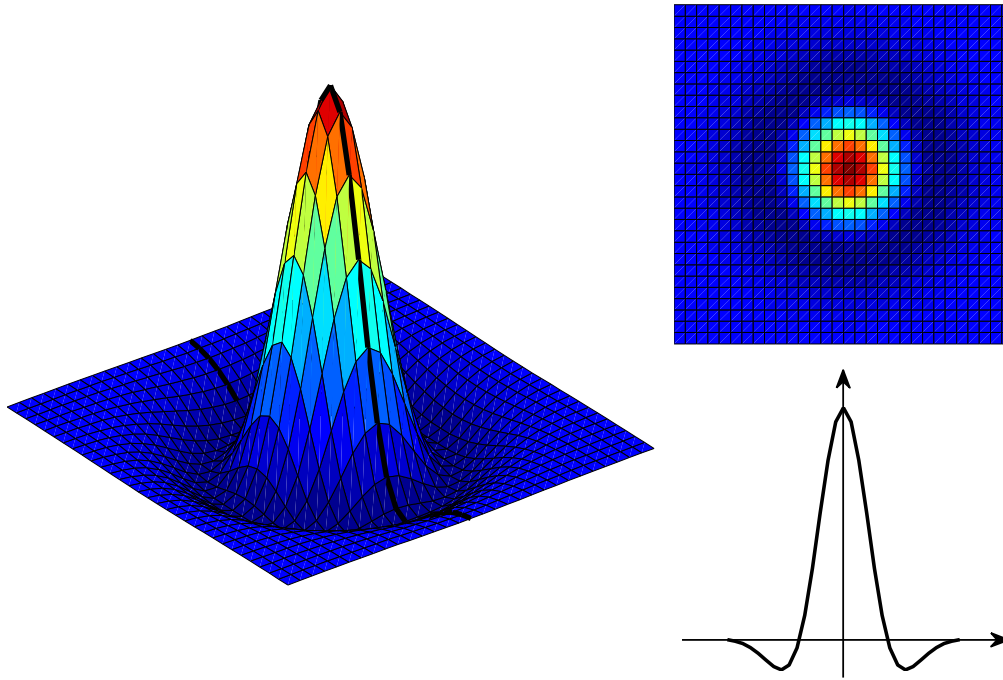
Figure 10: Laplacian of a Gaussian kernel, $\sigma = 4$, $n = 31$. The selected value of n is sufficient to have a good approximation of the LoG function in the kernel, but it is greater than in the case of the Gaussian kernel.

4.1.3 Gaussian and LoG Functions Comparison

As mentioned before, the size n of the kernel and the standard deviation σ of the exponential function are not independent. For the same σ value, the function LoG has wider support than the Gaussian function (i.e. the LoG function has a slower decay), so a greater value of n is needed to correctly sample the LoG kernel.

Figure 11 shows both functions generated with the same value of σ. It is clear that we must use a larger kernel size in the LoG function case (approximately 18% larger). For example, using $\sigma = 4$, the optimum value for n in the case of the Gaussian function is the first odd integer greater than 6σ, which is 25, and for the case of LoG function, would be 29.

4.2 Convolution

When performing a convolution, it is necessary to define the boundary conditions used to compute it near the edges of the image, and to get a valid output of the same size as the input image. In
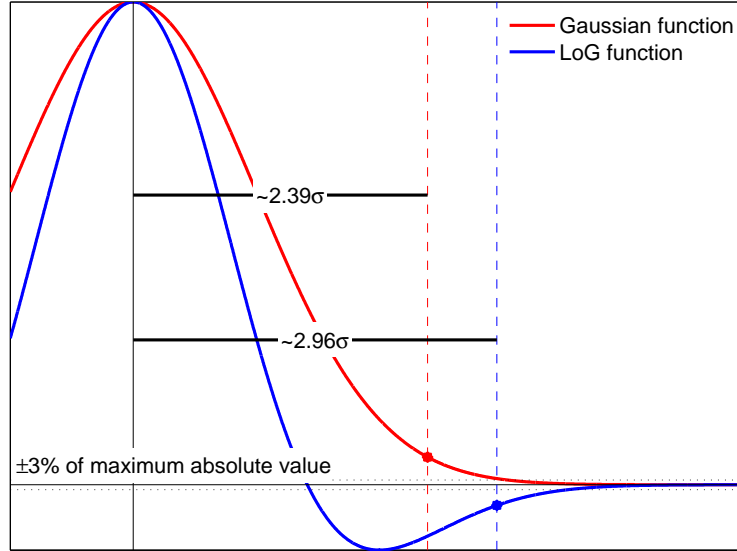
Figure 11: Comparison of the Gaussian and LoG functions.

this paper, two methods were implemented: zero-padding and boundary reflection. Zero-padding completes the borders of the image with zero valued pixels. Reflection completes those pixels with the corresponding symmetric pixel value relative to the edge of the image. Direct convolution is not the only way of filtering an image with a kernel, there are other methods such as FFT convolution or recursive filtering.

5 Results

The results of each algorithm are first shown for a simple image composed of a white square on a black background, see Figure 12.
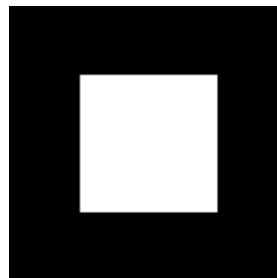
Figure 12: Test image 1: white square on black background (127×127 pixels).

Figure 13 shows the output of each algorithm including execution times[6]. In this simple case, the first derivative edge detection algorithms run better and faster. All algorithms show consistent results. Thicker edges appear in the results of Haralick, due to smooth image model that is imposed in this algorithm.

---

[6]Running on Intel Core i3 CPU (2.53GHz), 3 Gb RAM, standard laptop. Note that the execution time of the algorithms Roberts, Prewitt and Sobel are the same, because they run simultaneously.
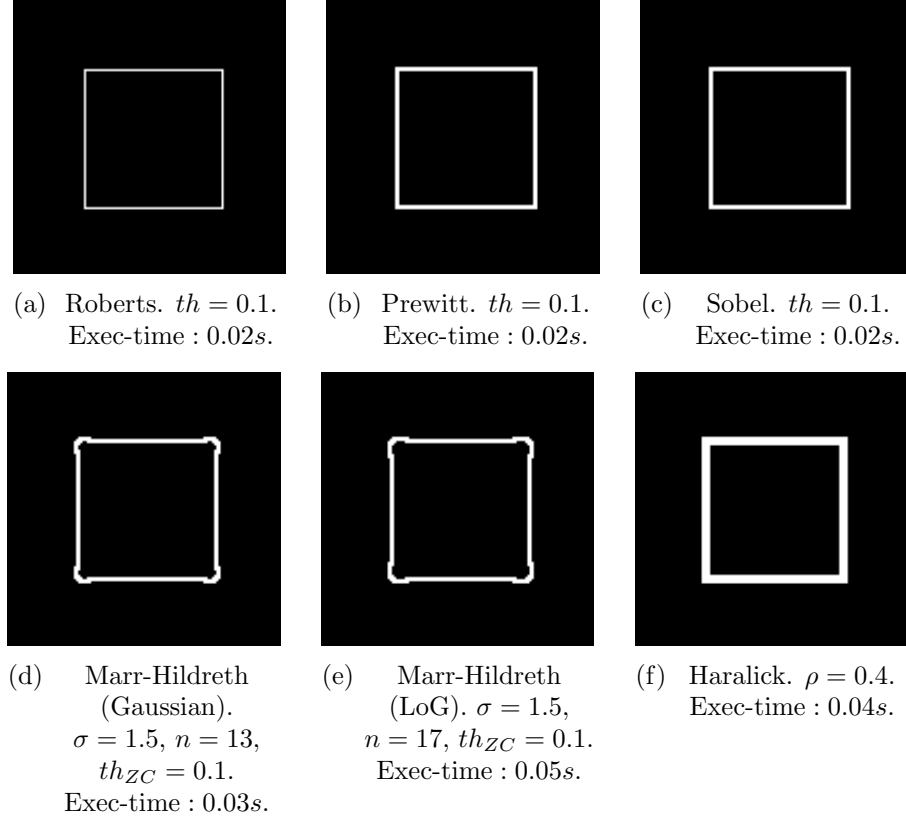
(a)   Roberts. $th = 0.1$.
Exec-time : $0.02s$.

(b)   Prewitt. $th = 0.1$.
Exec-time : $0.02s$.

(c)   Sobel. $th = 0.1$.
Exec-time : $0.02s$.

(d)   Marr-Hildreth
(Gaussian).
$\sigma = 1.5$, $n = 13$,
$th_{ZC} = 0.1$.
Exec-time : $0.03s$.

(e)   Marr-Hildreth
(LoG). $\sigma = 1.5$,
$n = 17$, $th_{ZC} = 0.1$.
Exec-time : $0.05s$.

(f)   Haralick. $\rho = 0.4$.
Exec-time : $0.04s$.

Figure 13: Results of the algorithms using Figure 12 as input image. The first three ((a), (b) and (c)) correspond to the first derivative methods (Roberts, Prewitt and Sobel), then the following two ((d) and (e)) are from the Marr-Hildreth algorithm using Gaussian and LoG kernels, and the last one ((f)) corresponds to the Haralick algorithm. Note the rounded corners on the Marr-Hildreth results, due to the Gaussian blur.

Figure 14: Test image 2: Windmill (1000×563 pixels).

Now the performance of each algorithm is analyzed using a natural image (*Windmill*) shown in Figure 14. The results of the first derivative edge detection algorithms are shown in figures 15(b) to 15(d). Figures 15(e) and 15(f) show the results obtained using the Marr-Hildreth algorithm (both Gaussian and LoG kernels), and Figure 15(g) shows the results obtained using the Haralick algorithm.
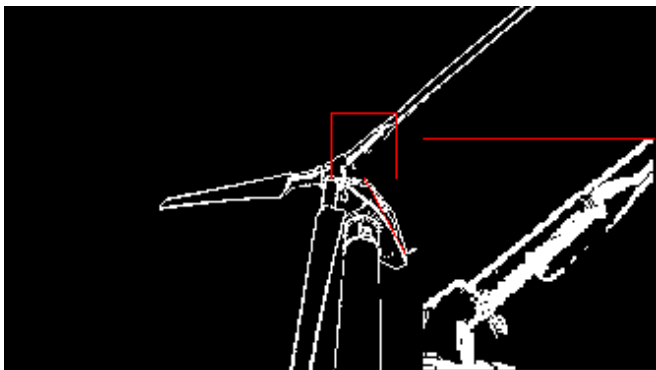
(a) Original.
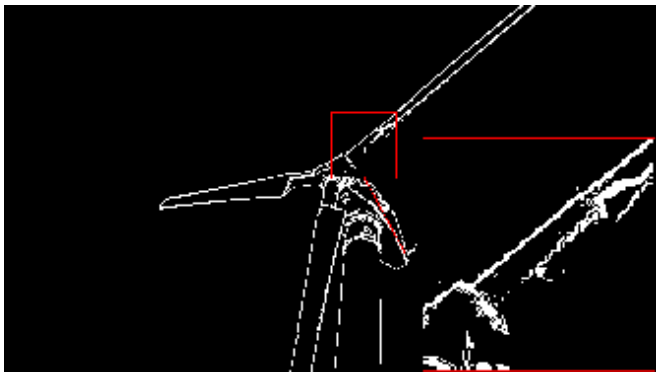
(b) Roberts. $th = 0.1$. Exec-time : $0.55s$.
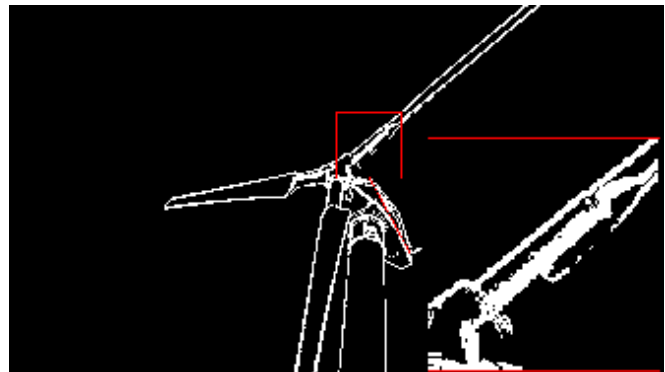
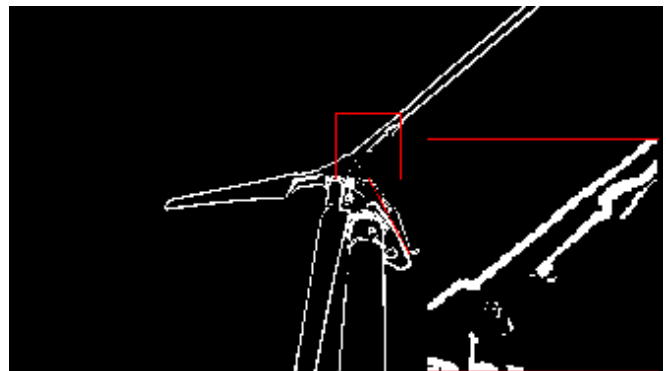(c) Prewitt. $th = 0.1$. Exec-time : $0.55s$.

(d) Sobel. $th = 0.1$. Exec-time : $0.55s$.

(e) Marr-Hildreth (Gaussian). $\sigma = 3$, $n = 25$, $th_{ZC} = 0.09$. Exec-time : $1.05s$.

(f) Marr-Hildreth (LoG). $\sigma = 3$, $n = 29$, $th_{ZC} = 0.13$. Exec-time : $1.44s$.

(g) Haralick. $\rho = 0.4$. Exec-time : $0.93s$.

Figure 15: Results of the first derivative, Marr-Hildreth and Haralick's algorithms on the *Windmill* image.

Table 5.1 summarizes the execution times for each of the algorithms. In this example the difference in performance between the first derivative and the second derivative algorithms is more meaningful. For each image in Figure 15, we show an area of interest, enlarging it to have a better view of the details. Note that none of the first derivative methods (even with a loose threshold) detect the lower edge of the blade (which is shaded). Neither the Haralick algorithm is able to detect it. However, the Marr-Hildreth algorithm detects it, using either a Gaussian or a LoG kernel. Another observation is that edges detected by Haralick's algorithm appear to be thicker than those detected with other ones. This may be due the regularity that Haralick assumes.

Algorithm	Execution time (s)
Roberts, Prewitt and Sobel	0.550 s
Marr-Hildreth (Gaussian)	1.050 s
Marr-Hildreth (LoG)	1.440 s
Haralick	0.930 s

Table 5.1: Execution time of the algorithms (including I/O) using the *Windmill* image from Figure 14 as input image (1000×563 pixels, 3 channels).

5.1 Further Examples

Further examples obtained with the implemented algorithms are shown in figures 16 and 17. More examples can be found in the online demo[7].

Some observations:

- The first derivative algorithms, although they work properly and run fast, have some problems like discontinuity of the edges. They are also very sensitive to noise, which make them detect spurious edges (More sophisticated methods help to improve this, e.g. Canny edge detector [2]).

- The Marr-Hildreth algorithm (in its two versions) achieved interesting results regarding edge details. This algorithm provides a first attempt to obtain regular edges, because of the filtering with a Gaussian kernel. This behavior can be seen in Lena's hair (Figure 16), or inside the oranges (Figure 17).

- As mentioned earlier, Haralick's algorithm is the first to assume greater regularity of the image in the neighborhood of a pixel (third order). This causes thicker edges, as shown in both examples. But, note that some edges are only detected with this algorithm, e.g. the lower edges of the oranges, which are in the shadow, in Figure 17. These edges are quite smooth, so that the Haralick model is well suited to them, and they do not represent an abrupt change in the intensity to be detected by other methods.

---

[7]http://dx.doi.org/10.5201/ipol.2015.35

(a) Original.

(b) Roberts. $th = 0.1$.

(c) Prewitt. $th = 0.1$.

(d) Sobel. $th = 0.1$.

(e) Marr-Hildreth (Gaussian). $\sigma = 2$, $n = 21$, $th_{ZC} = 0.15$.

(f) Marr-Hildreth (LoG). $\sigma = 2$, $n = 25$, $th_{ZC} = 0.15$.
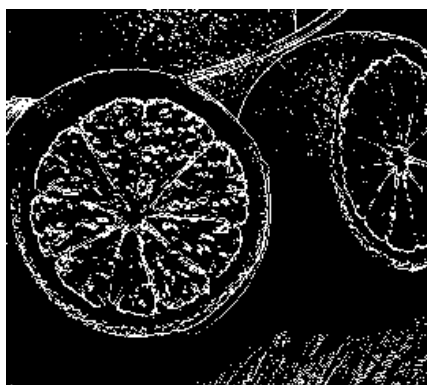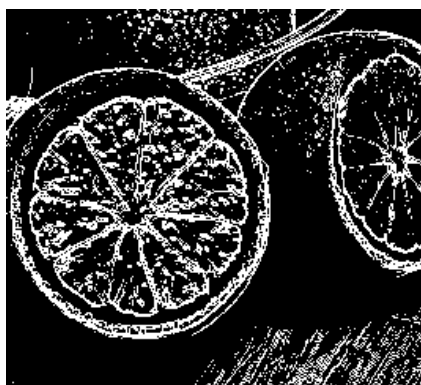

(g) Haralick. $\rho = 0.35$.

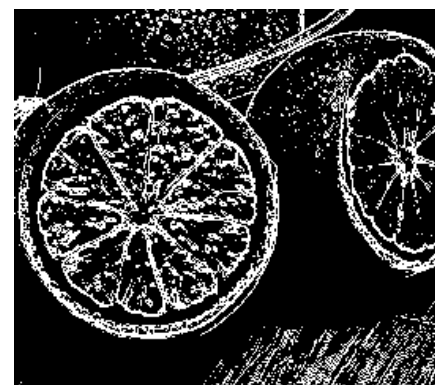Figure 16: Example: Lena (512×512 pixels).
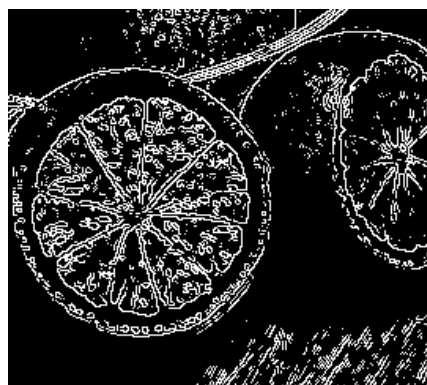
(a) Original.

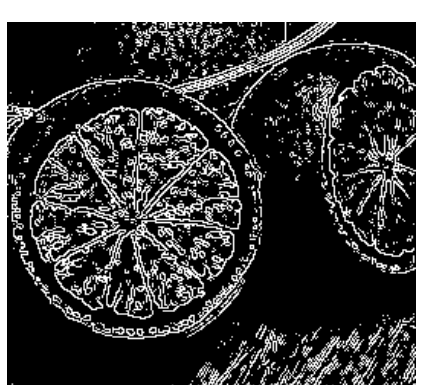(b) Roberts. $th = 0.1$

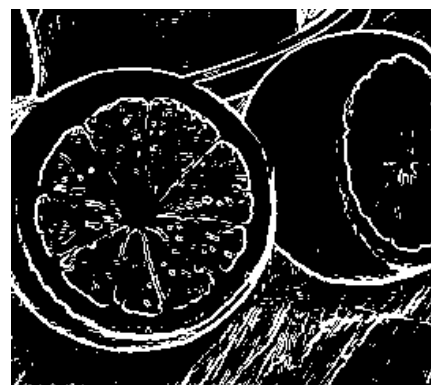

(c) Prewitt. $th = 0.1$

(d) Sobel. $th = 0.1$

(e)   Marr-Hildreth (Gaussian). $\sigma = 2$, $n = 21$, $th_{ZC} = 0.15$.

(f)   Marr-Hildreth (LoG). $\sigma = 2$, $n = 25$, $th_{ZC} = 0.15$.

(g) Haralick. $\rho = 0.4$.

Figure 17: Example: Oranges (536×480 pixels).

5.2   Effect of the Scale Parameter

In this section we will discuss the effect of the parameters for each of the reviewed algorithms. As a general rule, they all take one threshold parameter which acts as a scale parameter. As Figure 18 shows for the *Roberts* algorithm, the bigger that threshold is, the coarser is the edge map obtained.
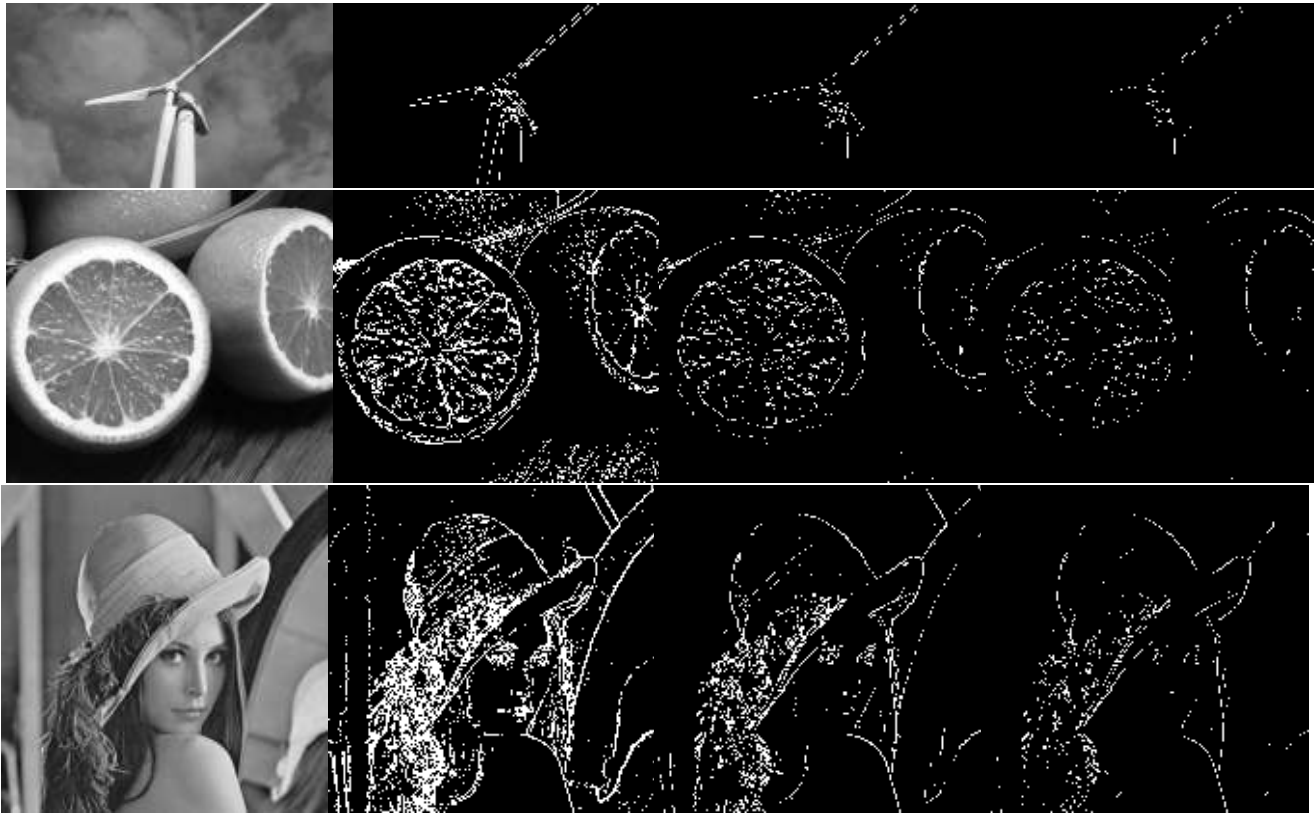

Figure 18: Results of the *Roberts* algorithm on our sample images. For each example we show the original image on the left and the results corresponding to parameter values of 0.1, 0.2 and 0.3 respectively.

In Figure 19 we show a summary of the threshold's behavior for this algorithm, in the whole range of possible values $[0, 1]$. For each image, this information is summarized in a saliency map which serves to assess the quality of the obtained hierarchical set of edges. The saliency map is constructed by sampling the scale range (using steps of 0.05) and computing the edge map corresponding for each scale. Then each edge is labeled with its scale of disappearance α^-, which is the maximum scale at which an edge is present in the output edge map. This scale is drawn with a *hot* colormap (black, red, yellow, white). Intuitively, this means that "hotter" edges last longer across scales and thus are more meaningful. These results show two important properties. First, the obtained stack of edges are causal, i.e. as the scale parameter increases the edge map gets coarser. This can be verified by observing the values of α^- assigned to each contour. If a contour has a high α^- value it means that it will survive for a long time in the detection process. On the other hand, small values mean that it will be removed at early stages. As the figure shows, the highest values of α^- are assigned to the most coarser segmentation. And second, the stack of edges is geometrically accurate, i.e. the borders present in each scale are a strict subset of the previous (finer) scale. If they were not included they will be seen as new contours in the saliency map. These results show empirical evidence that the obtained hierarchy holds the strong causality property introduced by Morel and Solimini [12].

In figures 20 and 21 we compare the three first derivative algorithms by means of their saliency maps. As the results show, they all present a similar behavior, however there are small differences in

Figure 19: Results of the *Roberts* algorithm on our sample images. For each example we show the original image on the left and a saliency map on the right. The value of each contour corresponds to the scale of disappearance α^- of each region.

the results of Roberts versus the other two algorithms. For example in the latter, edges surrounding the segments of the orange are better defined, because they have more relative weight than other spurious borders. This is explained by the smoothing introduced by Prewitt and Sobel masks, as commented in sections 2.3 and 2.3, which removes these spurious edges and thus improves the meaningfulness of the stable ones.

In figures 22 and 23 we compare the second derivative algorithms by means of their saliency maps. In the first place, we obtain very similar hierarchies for both Marr-Hildreth implementations,

Figure 20: Results of the first derivative algorithms on our sample images. For each example we show the original image on the left and the saliency map of each algorithm (Roberts, Sobel, Prewitt) respectively. The value of each contour corresponds to the scale of disappearance α^- of each region.

which validates the affirmation that the difference is merely about computational effort if both are correctly implemented. However, the LoG implementation seems to give slightly better results, this is probably because it provides a more accurate discretization of the differential operators than the Gaussian approach. See for instance the clothing sticks in Figure 23, where the boundaries in the LoG case seem better defined.

Figure 21: Results of the first derivative algorithms on our sample images. For each example we show the original image on the left and a saliency map of each algorithm (Roberts, Sobel, Prewitt) respectively. The value of each contour corresponds to the scale of disappearance α^- of each region.
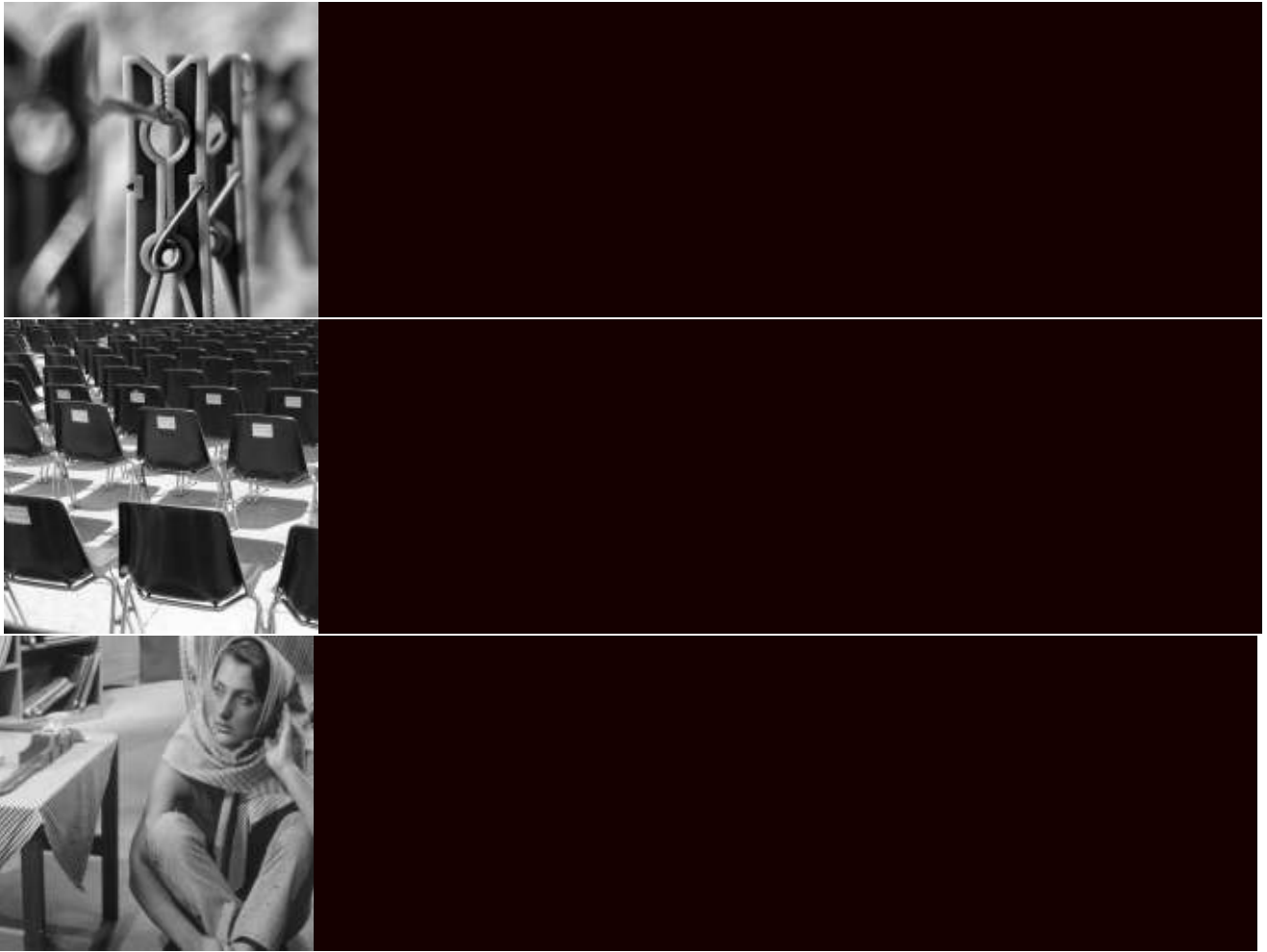
Figure 22: Results of the second derivative algorithms on our sample images. For each example we show the original image on the left and a saliency map of each algorithm (Marr-Hildreth-Gaussian, Marr-Hildreth-Log, Haralick) respectively. The value of each contour corresponds to the scale of disappearance α^- of each region.
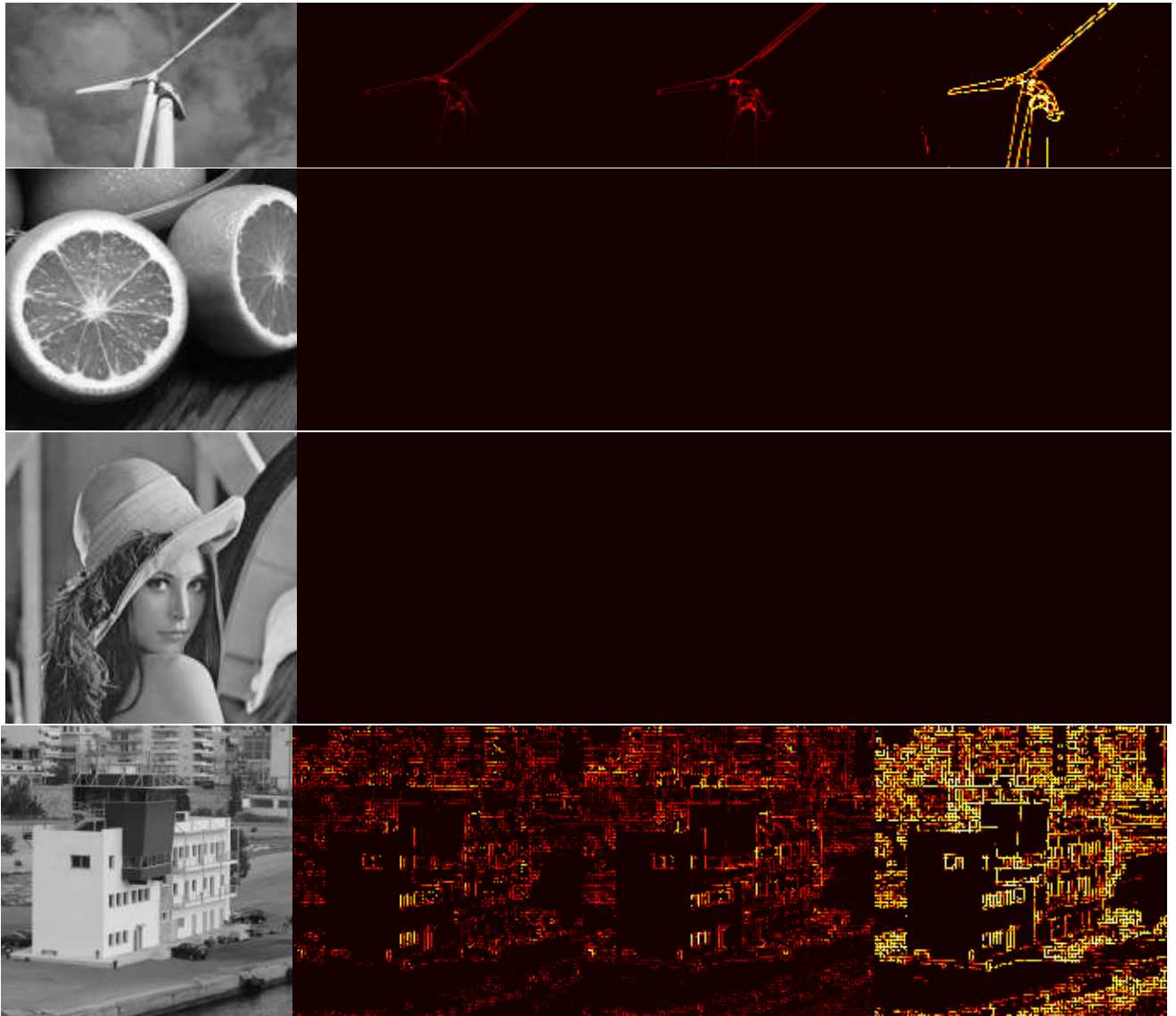
Figure 23: Results of the second derivative algorithms on our sample images. For each example we show the original image on the left and the saliency map of each algorithm (Marr-Hildreth-Gaussian, Marr-Hildreth-Log, Haralick) respectively. The value of each contour corresponds to the scale of disappearance α^- of each region.

5.3 Effect of the Smoothing Parameter

As we stated in the previous section, all the reviewed algorithms share a scale parameter (threshold), however, the Marr-Hildreth Algorithm also has a smoothing step which adds an additional parameter σ, which controls the amount of regularization. In Figure 24 we show the results using different σ values. As sigma increases more spurious edges are removed and the boundaries of the objects look smoother. However, if we keep increasing it we start removing meaningful edges (see the cloth band on the hat of Lena).
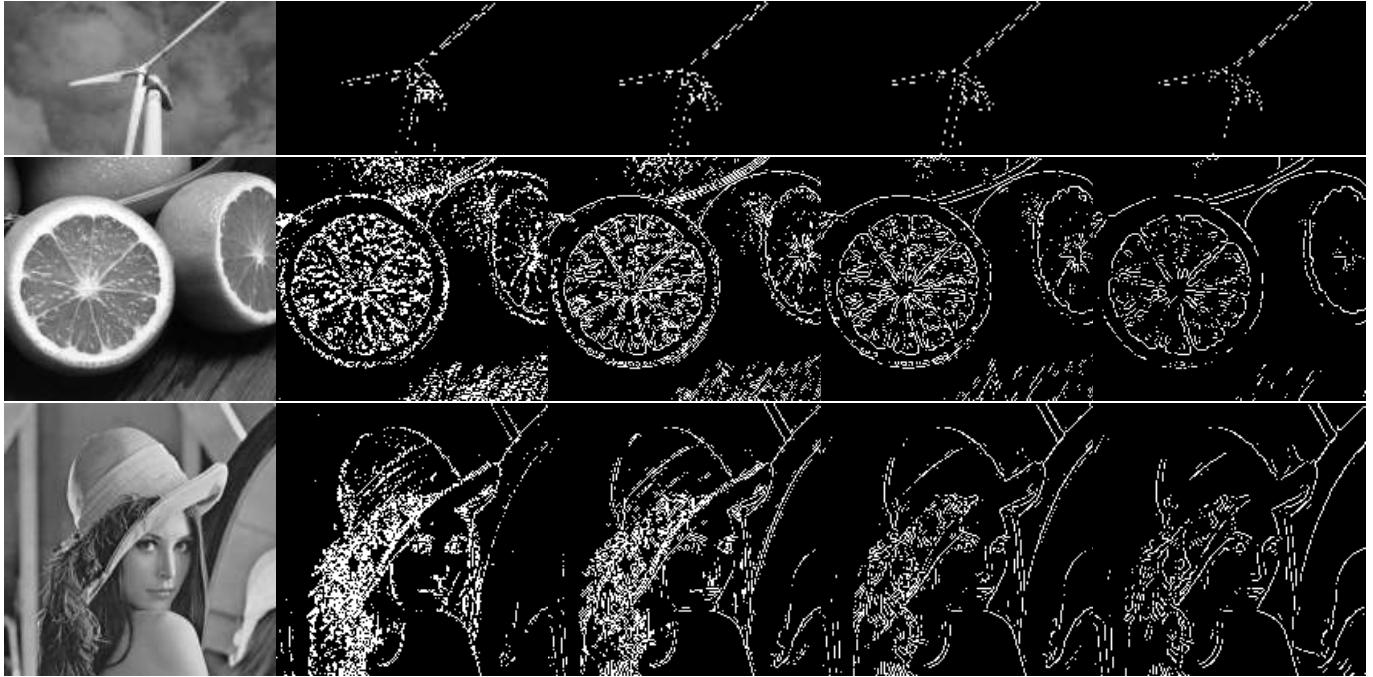

Figure 24: Effect of the σ parameter for the *Marr-Hildreth* (Gaussian) algorithm on our sample images. For each example we show the original image on the left and various consecutive results with different smoothing values ($\sigma = 1, 2, 3, 4$) on the right.

5.4 Discussion

In this section we will discuss the obtained results and contrast them with the claims made by the original authors. In the case of the first derivative algorithms, they are very simple and the claims made by the original authors are straightforward to verify. The Roberts algorithm was presented in his PhD Thesis ([15, Chapter IV]) as a step of a more complex 3D reconstruction system. For this reason, only qualitative results are shown on few images. The effects of the parameter or the discretization are not validated. The only observation made in that work is the presence of spurious edges due to noise (pg. 31), which are removed by a line fitting postprocessing. With respect to *Prewitt* and *Sobel* operators, they arise from an in-depth study of different ways to discretize the gradient. Prewitt [14] discussed and compared these discretizations qualitatively. In this work we tested only two of the possible discretizations, and we obtained very similar results using both operators. On the other hand, they show better performance than the *Roberts* mask.

In Haralick's work, there is an interesting quantitative validation over a synthetic grid. On that evaluation, he shows results of noise sensitivity as well as comparative performance with Prewitt

operators. As in this work we didn't focus on quantitative benchmarks, we didn't reproduce those results.

Regarding the Marr-Hildreth algorithm, the obtained results validate the original claims [9]:

- "A major difficulty with natural images is that changes can and do occur over a wide range of scales (Marr 1976a, b). No single filter can be optimal simultaneously at all scales, so it follows that one should seek a way of dealing separately with the changes occurring at different scales."

- "The reason is that strict bandlimiting gives rise to sidelobes in the spatial filter, and the consequence of these is that, in the zero-crossing image, strong intensity changes give rise to echoes as well as to the directly corresponding zero-crossings[...]"

- "Physical edges will produce roughly coincident zero-crossings in channels of nearby sizes. The spatial coincidence assumption asserts that the converse of this is true, that is the coincidence of zero-crossings is sufficient evidence for the existence of a real physical edge. If the zero-crossings in one channel are not consistent with those in the others, they are probably caused by different physical phenomena, so descriptions need to be formed from both sources and kept somewhat separate."

The first claim is validated in our experiments by varying the σ parameter, where we observed that no single value will allow us to obtain all the meaningful edges. The same reasoning applies to the zero crossing threshold t_{ZC}. The second observation is evidenced by the halos seen in the blades of the windmill, see inside left blade and left of the top blade in Figure 15(e).[8] The third claim is evidenced by the structure of the saliency maps of the threshold parameter (figures 22 and 23) and the smoothing parameter (Figure 25). In both maps we observe that meaningful edges are coincident across scales and the spurious ones are not.

---

[8]Please look at these issues in the electronic version or the online demo, as they are not noticeable in the printed version.

Figure 25: Effect of the σ parameter for the *Marr-Hildreth* (Gaussian) algorithm on our sample images. For each example we show the original image on the left and the generated saliency map on the right. σ is sampled regularly in the range $[0.5, 5]$ using a step of 0.5.

6   Conclusions

Some of the most traditional methods of edge detection in digital images were discussed and carefully implemented in this work. The implemented algorithms were tested with synthetic and real images, obtaining generally the expected results, taking into account the limitations of these methods.

The first derivative algorithms (Roberts, Prewitt and Sobel) have the advantage of having a very simple implementation. They also run extremely fast, because their main operation is a convolution with a very small kernel (2×2 or 3×3 pixels). The results obtained with these methods are quite good, considering their simplicity, but they have problems such as noise and discontinuity of the edges. Gaussian filtering would reduce noise, alleviate (but not solve) the discontinuity of the edges, and would make the comparison fairer with the second derivative methods.

The second derivative algorithms (Marr-Hildreth & Haralick) involve several more operations, since convolutions with larger kernels are performed. In real images, they have better behavior than first derivative algorithms.

Comparing the two versions of the Marr-Hildreth algorithm, the version with Gaussian kernel runs significantly faster that the LoG one. The latter, while slower (since it needs a larger kernel to achieve similar results), is more accurate because it makes no approximation to calculate the Laplacian (it is calculated analytically before creating the kernel). It is also possible to manage the size of the kernel, which represents a scale parameter of the algorithm, being able to obtain a highly detailed edge image using small kernels, or just more noticeable edges using larger kernels.

Haralick's algorithm, although it shares the Marr-Hildreth idea of finding zero crossings of the second derivative, has some quite different ideas. It is the only one of these algorithms which works with an approximation of the intensity of the image in the neighborhood of a pixel, using a bicubic polynomial function. This supposes a certain regularity in the image, which is not always true, and sometimes leads to detect edges where none of them exist due to ringing effects, and get some thicker edges too.

All algorithms have their advantages and disadvantages. Choosing one or the other may depend on requirements of the application. Furthermore, we must remark that we focused on low level edge detectors, and thus none of the implemented methods ensure edge connectivity. These algorithms only serve as the input for further edge integration steps. There are many proposals in the literature aiming to integrate edgels into meaningful object boundaries. Examples of these approaches are the Canny edge detector [2] or the meaningful level lines algorithm [3].

With respect to the parameters, these methods require the manual determination of the parameters, and it is very hard to find a single threshold value suitable for a broad category of examples. In this sense, *a contrario* methods [3] provide a better approach for setting parameters. For this reason, most modern approaches use a multiscale approach, were edges are detected at many different scales and the optimum scale is chosen a posteriori [13, 1]. This idea can be even extended to pick edges at different scales depending on the region of the image; this idea has been applied to regions [4].

6.1   Future Work

The goal of this work was to provide reference implementations for some of the most classical edge detectors. Some of them have been published for many years and are quite simple, however, it is surprisingly hard to find trustable implementations. We focused on low-level edge detection, which means that we do not consider post processing steps to integrate the detected edgels into meaningful lines. Extensive qualitative validation has been performed, showing the effects of the parameters and the scale-space properties of the algorithms in order to faithfully compare them. We consider this work as the first step towards a complete benchmark of edge detectors, which has to be completed with a systemic quantitative analysis. Suitable frameworks for this kind of analysis are

proposed in the literature [4, 11, 5], but this is clearly outside the scope of this work. In particular, quantitatively challenging the claims of the reviewed works will be a great contribution to the area, which is very weak with respect to experimental validation. From the original works reviewed, only Haralick presents a quantitative analysis but only over one synthetic image. Even at the moment of writing (2014), there are only a dozen edge detection and segmentation works presenting a serious quantitative analysis over a standard database of images.

A   Haralick Convolution Masks

(a) **mask$_1$**.

425	275	225	275	425
275	125	75	125	275
225	75	25	75	225
275	125	75	125	275
425	275	225	275	425

(b) **mask$_2$**.

-2260	-620	0	620	2260
-1660	-320	0	320	1660
-1460	-220	0	220	1460
-1660	-320	0	320	1660
-2260	-620	0	620	2260

(c) **mask$_3$**.

2260	1660	1460	1660	2260
620	320	220	320	620
0	0	0	0	0
-620	-320	-220	-320	-620
-2260	-1660	-1460	-1660	-2260

(d) **mask$_4$**.

1130	620	450	620	1130
830	320	150	320	830
730	220	50	220	730
830	320	150	320	830
1130	620	450	620	1130

(e) **mask$_5$**.

-400	-200	0	200	400
-200	-100	0	100	200
0	0	0	0	0
200	100	0	-100	-200
400	200	0	-200	-400

(f) **mask$_6$**.

1130	830	730	830	1130
620	320	220	320	620
450	150	50	150	450
620	320	220	320	620
1130	830	730	830	1130

(g) **mask$_7$**.

-8260	-2180	0	2180	8260
-6220	-1160	0	1160	6220
-5540	-820	0	820	5540
-6220	-1160	0	1160	6220
-8260	-2180	0	2180	8260

(h) **mask$_8$**.

5640	3600	2920	3600	5640
1800	780	440	780	1800
0	0	0	0	0
-1800	-780	-440	-780	-1800
-5640	-3600	-2920	-3600	-5640

(i) **mask$_9$**.

-5640	-1800	0	1800	5640
-3600	-780	0	780	3600
-2920	-440	0	440	2920
-3600	-780	0	780	3600
-5640	-1800	0	1800	5640

(j) **mask$_{10}$**.

8260	6220	5540	6220	8260
2180	1160	820	1160	2180
0	0	0	0	0
-2180	-1160	-820	-1160	-2180
-8260	-6220	-5540	-6220	-8260

Table A.1: 5×5 masks used to compute the coefficients of the bicubic fit.
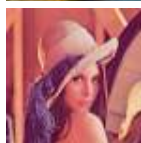
Image Credits
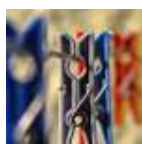
Cloth Sticks and *Windmill*, H. Spontón, 2012.

Chairs and *Piraeus*, R. Grompone, J. Jakubowicz, J.-M. Morel, G. Randall, 2010.

Barbara and *Lena*, Standard test images.

Oranges, Unknown source, freely available on the internet.

Acknowledgements

References

[1] P. Arbelaez, *Boundary extraction in natural images using ultrametric contour maps*, in Conference on Computer Vision and Pattern Recognition Workshop, June 2006, pp. 182–182. http://dx.doi.org/10.1109/CVPRW.2006.48.

[2] J. F. Canny, *A computational approach to edge detection.*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8 (1986), pp. 679–698.

[3] F. Cao, P. Musé, and F. Sur, *Extracting meaningful curves from images*, Journal of Mathematical Imaging and Vision, 22 (2005), pp. 159–181. http://dx.doi.org/10.1007/s10851-005-4888-0.

[4] J. Cardelino, V. Caselles, M. Bertalmío, and G. Randall, *A contrario selection of optimal partitions for image segmentation*, SIAM Journal on Imaging Sciences, 6 (2013), pp. 1274–1317. http://dx.doi.org/10.1137/11086029X.

[5] J. S. Cardoso and L. Corte-Real, *Toward a generic evaluation of image segmentation*, IEEE Transactions on Image Processing, 14 (2005), pp. 1773–1782. http://dx.doi.org/10.1109/TIP.2005.854491.

[6] R. Deriche, *Recursively implementing the Gaussian and its derivatives*, Tech. Report 1893, INRIA, May 1993. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.5904.

[7] P. Getreuer, *Linear Methods for Image Interpolation*, Image Processing On Line, 1 (2011). http://dx.doi.org/10.5201/ipol.2011.g_lmii.

[8] R. M. HARALICK, *Digital step edges from zero crossing of the second directional derivatives*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 6 (1984), pp. 58–68. http://dx.doi.org/10.1109/TPAMI.1984.4767475.

[9] D. MARR; E. C. HILDRETH., *Theory of edge detection*, Proceedings of the Royal Society, London B, 207 (1980), pp. 187–217.

[10] W. E. LORENSEN AND H. E. CLINE, *Marching cubes: A high resolution 3D surface construction algorithm*, in Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, vol. 21, New York, NY, USA, Aug. 1987, ACM, pp. 163–169. http://dx.doi.org/10.1145/37402.37422.

[11] D. MARTIN, C. FOWLKES, D. TAL, AND J. MALIK, *A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics*, Proceedings of 8th International Conference on Computer Vision, (2001), pp. 416–423.

[12] J. M. MOREL AND S. SOLIMINI, *Variational Methods in Image Segmentation*, Birkhäuser, 1995. ISBN 0-8176-3720-6.

[13] L. NAJMAN AND M. SCHMITT, *Geodesic saliency of watershed contours and hierarchical segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 18 (1996), pp. 1163 –1173. http://dx.doi.org/10.1109/34.546254.

[14] J.M.S. PREWITT, *Object enhancement and extraction*, in Picture Processing and Psychopictorics, B. S. Lipkin and A. Rosenfeld, eds., Academic Press, June 1979, pp. 75–149.

[15] L.G. ROBERTS, *Machine perception of three-dimensional solids*, Phd. Thesis. Massachusetts Institute of Technology, (1963).

[16] R. C. GONZALEZ; R. E. WOODS, *Digital Image Processing*, Prentice Hall, Upper Saddle River, NJ., 2002. ISBN 013168728X.