Published in Image Processing On Line on 2012–06–06.
ISSN 2105–1232 © 2012 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials, software, datasets and online demo at
http://dx.doi.org/10.5201/ipol.2012.XXXXXXXX

A Review of Classic Edge Detectors

Haldo Spontón[1], Juan Cardelino[2]

[1] IIE, UdelaR, Uruguay (haldos@fing.edu.uy)
[2] IIE, UdelaR, Uruguay (juanc@fing.edu.uy)

Abstract

In this paper some of the classic alternatives for edge detection in digital images are studied. The main idea behind edge detection 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.
Results obtained from a qualitative point of view (perceptual) and from a quantitative point of view (number of operations, execution time) are compared, considering different ways to convolve an image with a kernel (step required in some of the algorithms).

Source Code

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

1 Introduction

The basic idea of edge detection algorithms 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 based on using small operators (such as Sobel masks), attempting to compute an approximation of the first derivative of the image [1]. Next section describes such algorithms and serves as an introduction to a more sophisticated analysis of the edge detection process.

In 1980 Marr and Hildreth [2] 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 trough) in the first derivative or, equivalently, as a zero crossing in the second derivative. This algorithm is presented in Section 3. Haralick's algorithm [3] is an alternative approach based on the second derivative which is also reviewed in Section 3. This algorithm has the particularity of proposing a model to locally approximate the image 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.

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.

[1]http://dev.ipol.im/~haldos/ipol_demo/xxx_edges/

2 Algorithms based on first derivative

Algorithms based on first derivative studied in this paper have a common scheme, with the only difference in the type of filtering. Figure 1 shows a block diagram of these algorithms.

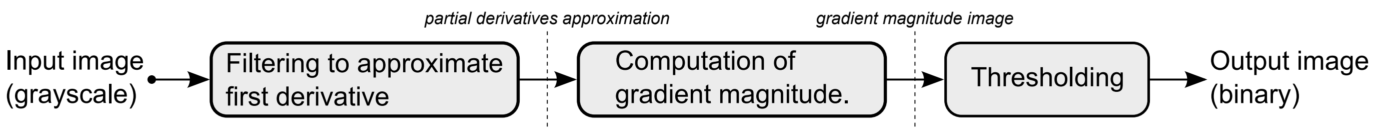

partial derivatives approximation
gradient magnitude image
Input image (grayscale)
Filtering to approximate first derivative
Computation of gradient magnitude.
Thresholding
Output image (binary)

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

The usual tool to find the amplitude and direction of changes in intensity 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 in the image. When dealing with digital images, numerical approximations of the partial derivatives are required, calculated 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 most usual approach to approximate the first derivative is to take the Taylor expansion of first order with a small h. Thus $f'(x)$ is computed as

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

The image is composed by discrete points of coordinates (i, j), so taking $x = ih$ and $y = jh$, the first derivative of the image is approximated, based on the intensity values at points of the image, as[2]:

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

and

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

[2]This approximation considerates $h = 1$, so the partial derivatives can be computed as intensity difference between neighbor pixels.

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 in Figure 2. 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 diferences 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 top-down. It is also usual to scale the mask values, in order to have unit derivative when unitary steps in intensity occurs.

-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.

Figure 2 shows the simplest way to compute derivatives, but Roberts [5] proposes to compute them as shown in Figure 3. 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, what 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 3x3. 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 to the partial derivatives using masks of size 3×3 are 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 in Figure 4. These masks are called *Prewitt operators* [6].

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

These masks can be obtained (beyond normalization) as the convolution of a horizontal derivation mask $[-1\ 0\ 1]$ with a vertically moving average $[1\ 1\ 1]^T$, and vice versa. Hence, these operators have smoothing properties in the direction orthogonal to the gradient.

2.3 The *Sobel* operators

A slight variation of the Prewitt operators uses more weight on the central coefficients of the difference. It can be shown that using double weight in the center location provides better image smoothing. This variation is implemented using the masks in Figure 5. These operators are called *Sobel operators.*

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

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

The computational difference between Prewitt and Sobel masks is not important, thus It is preferable to use Sobel operators, because edges are better localized, and their filter are also less aliased, because of the weighted shape of $[1\ 2\ 1]$.

2.4 Computation of the edges

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

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. Gradient magnitude image M is calculated using the equation 2.2 (Figure 6(c)).

Finally, the edges image is obtained by thresholding the gradient magnitude image, i.e. for pixel at position i in the image compute

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

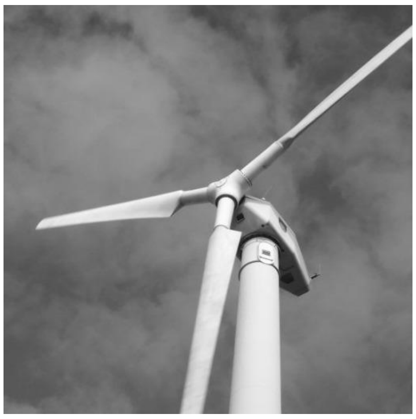
approximate where th is the threshold. A black and white image is obtained, in which edge points are indicated in white (see Figure 7).

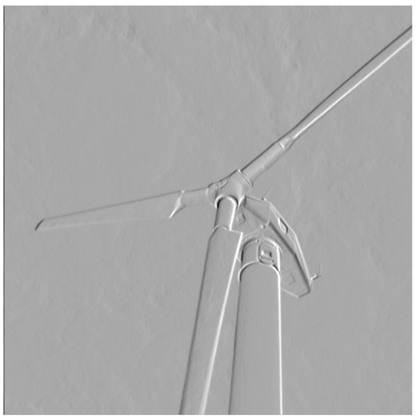
Note that different thresholds lead to different results. More and thicker edges for small thresholds values, and the opposite for large thresholds values. This threshold, and the others mentioned below, are defined as a percentage of the maximum value of the gradient image[3]. The advantage of this 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

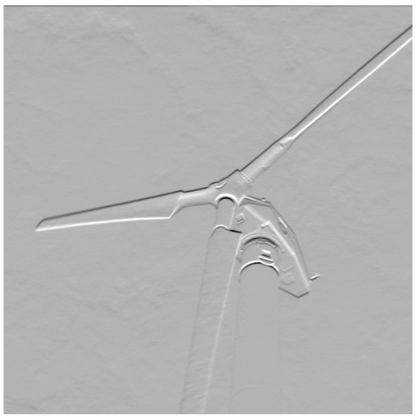
2.5 Pseudo-code

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

[3]This means no loss of generality in applying the threshold, and makes the threshold adapt to the behavior of different images.


(a) Grayscale image.


(b) Horizontal derivative f_x.


(c) Vertical derivative f_y.

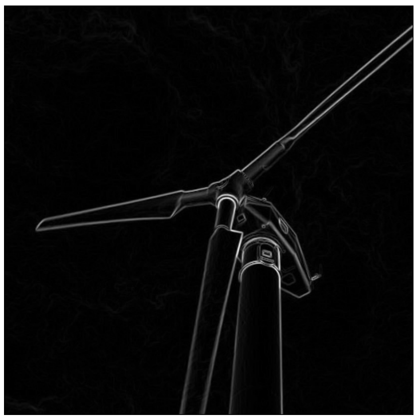
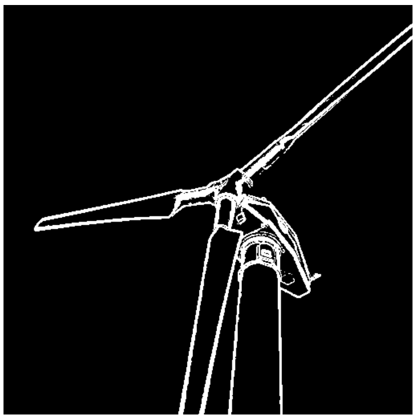
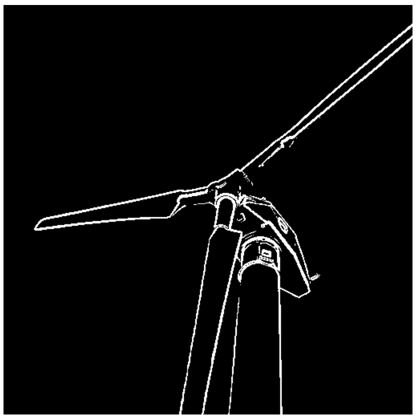

(d) Gradient module image $M = \sqrt{f_x^2 + f_y^2}$.

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


(a) $th = 0.1 * \max(M)$.


(b) $th = 0.2 * \max(M)$.

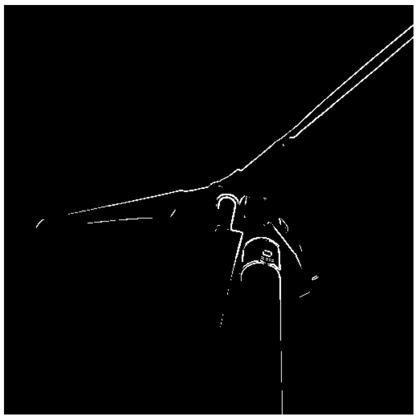

(c) $th = 0.4 * \max(M)$.

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

Algorithm 1: First derivative edge detection algorithms.

```
Require: input image, threshold $th$.
1:  $im \leftarrow$ input image
2:  Define $operator_x$ and $operator_y$ {Roberts, Prewitt or Sobel.}
3:  $g_x \leftarrow$ convolution($im$,$operator_x$)
4:  $g_y \leftarrow$ convolution($im$,$operator_y$)
5:  $max_M \leftarrow 0$
6:  for all pixel $i$ in image do
7:      $M[i] \leftarrow \sqrt{g_x^2 + g_y^2}$ {Gradient magnitude.}
8:      if $M[i] > max_M$ then
9:          $max_M \leftarrow M[i]$
10:     end if
11: end for
12: for all pixel $i$ in image do
13:     if $M[i] \geq th \times max_M$ then
14:         $im_{OUT}[i] \leftarrow 255$
15:     else
16:         $im_{OUT}[i] \leftarrow 0$
17:     end if
18: end for
19: return output image $\leftarrow im_{OUT}$
```

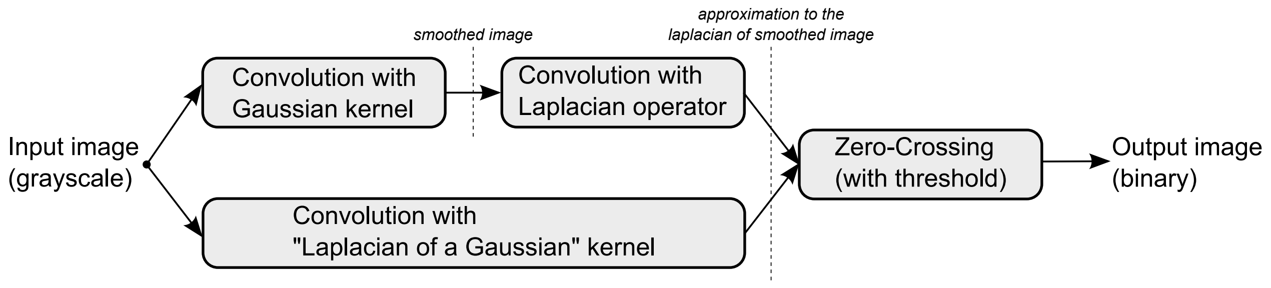
3 Algorithms based on second derivative

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 noise in the image.

The two algorithms presented in this section (Marr-Hildreth [2] and Haralick [3]) are based on the second derivative of the image, and both 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 can be done in several ways. Two different ways of doing this are implemented in this work (see block diagram in Figure 8): convolving the image with a Gaussian kernel and then approximating the second derivative (Laplacian) with a 3x3 kernel, or convolving the image with a kernel calculated as the Laplacian of a Gaussian function. There are more ways to do so, for example, using recursive Gaussian filters [7].



smoothed image
approximation to the laplacian of smoothed image
Convolution with Gaussian kernel
Convolution with Laplacian operator
Input image (grayscale)
Zero-Crossing (with threshold)
Output image (binary)
Convolution with "Laplacian of a Gaussian" kernel

Figure 8: Block diagram of Marr-Hildreth algorithm.

The algorithm is divided in two steps, each one 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.

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, with different boundary conditions. These operations will be discussed in detail in section 4.

3.1.1 Gaussian and LoG kernels

The Marr-Hildreth algorithm consists on convolving 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[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. Compute the Laplacian of the image resulting from step 1, using, for example, the 3×3 mask[5]:

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

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 a threshold is used, so that not only the signs of the opposite pixels must differ, also 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 (both Gaussian and LoG kernels). Each pixel p has eight neighbors, named according to their position as follows.

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

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

$$\begin{tikzcd}
p_{up,left} & p_{up,middle} & p_{up,right} \\
p_{middle,left} & p \arrow[ul] \arrow[u] \arrow[ur] \arrow[l] \arrow[r] \arrow[dl] \arrow[d] \arrow[dr] & p_{middle,right} \\
p_{down,left} & p_{down,middle} & p_{down,right}.
\end{tikzcd}$$

Then a pixel p is considered as 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 is a technique that yields good results, but if more precision in finding the zero crossings is needed, more advanced methods for finding zero crossings with subpixel accuracy could be used (e.g. marching squares [8]).

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 the original work of Haralick [3] on edge detection is presented in detail. The main idea behind 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.

3.2.1 Bi-cubic polynomial fitting

Here, the interpolation method used in the original work of Haralick is presented, although there are other options to do this [9].

The surrounding neighborhood of a point (x, y) in the image f is approximated using the following bi-cubic polynomial

$$f(x,y) = k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 + k_7x^3 + k_8x^2y + k_9xy^2 + k_{10}y^3. \tag{3.3}$$

where (x, y) are the offsets of each neighborhood point relative to the center point (e.g. in a neighborhood of size 5×5, x and y take values between -2 and 2).

To solve this 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 data fitting technique. In Haralick's work, this approximation is computed using least squares.

Algorithm 2: Marr-Hildreth edge detection algorithm.

```
Require: input image, standard deviation $\sigma$, kernel size $n$ and zero-crossing threshold $tzc$.
 1: $im \leftarrow$ input image
 2: $kernel \leftarrow$ generate_kernel($n$,$\sigma$) {Generated Gaussian or LoG kernel}
 3: $im_{SMOOTHED} \leftarrow$ convolution($im$,$kernel$)
 4: if Gaussian kernel then
 5:    Define Laplacian operator $laplacian$
 6:    $im_{LAPL} \leftarrow$ convolution($im$,$laplacian$)
 7: else
 8:    $im_{LAPL} \leftarrow im_{SMOOTHED}$
 9: end if
10: $max_L \leftarrow 0$
11: for all pixel $i$ in image $im_{LAPL}$ do
12:    if $im_{LAPL}[i] > max_L$ then
13:       $max_L \leftarrow im_{LAPL}[i]$
14:    end if
15: end for
16: for all pixel $i$ in image $im_{LAPL}$, except borders do
17:    for all pair $(p_1, p_2)$ of opposite neighbors of $p$ in $im_{LAPL}$ do
18:       if (sign($im_{LAPL}[p_1]$) $\neq$ sign($im_{LAPL}[p_2]$)) and ($|im_{LAPL}[p_1] - im_{LAPL}[p_2]| > th_{ZC}$) then
19:          $im_{OUT}[i] \leftarrow 255$
20:       else
21:          $im_{OUT}[i] \leftarrow 0$
22:       end if
23:    end for
24: end for
25: return output image $\leftarrow im_{OUT}$
```

Consider 25 points in a small neighborhood of a point (x, y) in the image. This gives us an equal number of equations to find 10 coefficients. As mentioned before this leads to an overdetermied system which can be solved by least squares. By substituting the 25 data points into the polynomial equation (3.3), the following system of equations is obtained,

$$\begin{aligned}
&f_1 = f(x_1, y_1) = k_1 + k_2x_1 + k_3y_1 + k_4x_1^2 + k_5x_1y_1 + k_6y_1^2 + k_7x_1^3 + k_8x_1^2y_1 + k_9x_1y_1^2 + k_{10}y_1^3\\
&f_2 = f(x_2, y_2) = k_1 + k_2x_2 + k_3y_2 + k_4x_2^2 + k_5x_2y_2 + k_6y_2^2 + k_7x_2^3 + k_8x_2^2y_2 + k_9x_2y_2^2 + k_{10}y_2^3\\
&\vdots\\
&f_{25} = f(x_{25}, y_{25}) = k_1 + k_2x_{25} + k_3y_{25} + k_4x_{25}^2 + k_5x_{25}y_{25} + k_6y_{25}^2 + k_7x_{25}^3 + k_8x_{25}^2y_{25} + k_9x_{25}y_{25}^2 + k_{10}y_{25}^3.
\end{aligned}$$

Using matrix notation, the system can be rewritten as

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{25} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 & \dots & y_1^3 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 & \dots & y_2^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{25} & y_{25} & x_{25}^2 & x_{25}y_{25} & y_{25}^2 & \dots & y_{25}^3 \end{bmatrix} \times \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_{10} \end{bmatrix} \Rightarrow \mathbf{f} = \mathbf{Ak}.$$

Then, the least squares problem can be approximately solved by computing the pseudo inverse of $\mathbf{A}$,

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

B is a 10×25 matrix,

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

For each coefficient (i from 1 to 10),

$$k_i = b_{i,1} f_1 + b_{i,2} f_2 + b_{i,3} f_3 + \cdots + b_{i,25} f_{25} \tag{3.4}$$

where[6]

$$\mathbf{b_i} = \begin{bmatrix} b_{i,1} & b_{i,2} & \cdots & b_{i,5} \\ b_{i,6} & b_{i,7} & \cdots & b_{i,10} \\ \vdots & \vdots & \ddots & \vdots \\ b_{i,21} & b_{i,22} & \cdots & b_{i,25} \end{bmatrix}. \tag{3.5}$$

Using equation 3.4 and the masks given in Table 3.1, it is possible to compute the coefficients $k_1 \dots k_{10}$ for all the points in the image. The elements of the mask $\mathbf{b_i}$ are the elements of the i-th row of $\mathbf{B} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$.

In order to speed up the calculations, the solution is computed by convolving the image with some precomputed masks in Table 3.1, to find the coefficients $k_1 \dots k_{10}$ at each point (x, y).

3.2.2 Analytical calculation of the second derivative

The neighborhood of each point of the image is approximated using the bi-cubic polynomial expression in equation 3.3. If just the first order terms of this polynomial are taken, the gradient angle θ, defined with negative x-axis, can be approximated as

$$\begin{aligned} \sin(\theta) &= -\frac{k_2}{\sqrt{k_2^2 + k_3^2}} \\ \cos(\theta) &= -\frac{k_3}{\sqrt{k_2^2 + k_3^2}}. \end{aligned} \tag{3.6}$$

Now substituting the variables x and y in polar form as

$$x = \rho \cos\theta \text{ and } y = \rho \sin\theta$$

in the bi-cubic polynomial, we obtain

$$f_\theta(\rho) = C_0 + C_1\rho + C_2\rho^2 + C_3\rho^3, \tag{3.7}$$

where

$$\begin{aligned} C_0 &= k_1 \\ C_1 &= k_2 \sin(\theta) + k_3 \cos(\theta) \\ C_2 &= k_4 \sin^2(\theta) + k_5 \sin(\theta)\cos(\theta) + k_6 \cos^2(\theta) \\ C_3 &= k_7 \sin^3(\theta) + k_8 \sin^2(\theta)\cos(\theta) + k_9 \sin(\theta)\cos^2(\theta) + k_{10} \cos^3(\theta). \end{aligned} \tag{3.8}$$

The derivatives are obtained as follows:

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

[6] In this matrix the coefficients b_i are simply spatially arranged with the same topology as the pixels they multiply.

(a) $\mathbf{b_1}$.

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) $\mathbf{b_2}$.

-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) $\mathbf{b_3}$.

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) $\mathbf{b_4}$.

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) $\mathbf{b_5}$.

-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) $\mathbf{b_6}$.

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) $\mathbf{b_7}$.

-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) $\mathbf{b_8}$.

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) $\mathbf{b_9}$.

-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) $\mathbf{b_{10}}$.

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 3.1: Masks to compute the coefficients of the bicubic fit.

Then, the condition that the second derivative is equal to zero becomes

$$f''_\theta(\rho) = 2C_2 + 6C_3\rho = 0 \quad \Rightarrow \quad \left|\frac{C_2}{3C_3}\right| < \rho_0, \tag{3.9}$$

where ρ_0 is a fixed threshold, passed as an argument to the algorithm. Ideally, the polynomial function should be zero at $\rho = 0$, but this condition is relaxed[7], allowing the polynomial to become zero in a neighborhood of $\rho = 0$ ($\rho \leq \rho_0$). This parameter must be greater than zero and and less than $\frac{1}{\sqrt{2}}$. Its optimal value is between 0.4 and 0.6.

The other condition required is that the third derivative to be negative, i.e.

$$f'''_\theta(\rho) = 6C_3 < 0 \quad \Rightarrow \quad C_3 < 0. \tag{3.10}$$

Given the direction defined by θ, with $\rho > 0$, an edge will always be an ascending step. This indicates that the first derivative in this direction has a maximum and that the second derivative has a zero crossing with negative slope. That is, the third derivative less than zero.

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 3.6).
3. Compute C_2 and C_3 (equations 3.8).
4. If $\left|\frac{C_2}{3C_3}\right| < \rho_0$ and $C_3 < 0$, then that point is an edge point.

3.2.4 Pseudo-code

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

4 Common Mathematical Operations

All algorithms implemented 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.

[7]This polynomial being zero at $\rho = 0$ means that the edge passes right through the exact coordinates of the pixel. Since the edge may be slightly displaced, other ρ values are permitted as long as the edge is inside the pixel, and for this, the maximum distance from the center of the pixel may be $1/\sqrt{2}$.

Algorithm 3: Haralick edge detection algorithm.

```
Require: input image (width $w$, height $h$), edge point condition threshold $\rho_0$.BibTex — RIS — RefWorks
 1: $im \leftarrow$ input image
 2: Define the ten $5x5$ masks ($mask_1 \dots mask_{10}$) used to determine coefficients $k_1$ to $k_{10}$ {See Table 3.1.}
 3: for all pixel $i$ in image $im$ do
 4:    $neighbors \leftarrow 5 \times 5$ neighborhood of pixel $i$ in image $im$
 5:    for $j = 1$ to 10 do
 6:       $k_j \leftarrow \sum_{n,m} neighbors[n,m] mask_j[n,m]$
 7:    end for
 8:    $C_2 \leftarrow \frac{k_2^2 k_4 + k_2 k_3 k_5 + k_3^2 k_6}{k_2^2 + k_3^2}$
 9:    $C_3 \leftarrow \frac{k_2^3 k_7 + k_2^2 k_3 k_8 + k_2 k_3^2 k_9 + k_3^3 k_{10}}{(\sqrt{k_2^2 + k_3^2})^3}$
10:    if $|C_2/3C_3| \leq \rho_0$ and $C_3 < 0$ then
11:       $im_{OUT}[i] \leftarrow 255$
12:    else
13:       $im_{OUT}[i] \leftarrow 0$
14:    end if
15: end for
16: return output image $\leftarrow im_{OUT}$
```

4.1.1 Gaussian kernel

Gaussian kernel 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. A value of n large enough is needed 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 $6\sigma = 24$).

4.1.2 LoG kernel

The Laplacian of Gaussian $\nabla^2 G(x,y)$ can be obtained analyically 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 case of Gaussian kernel, 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

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

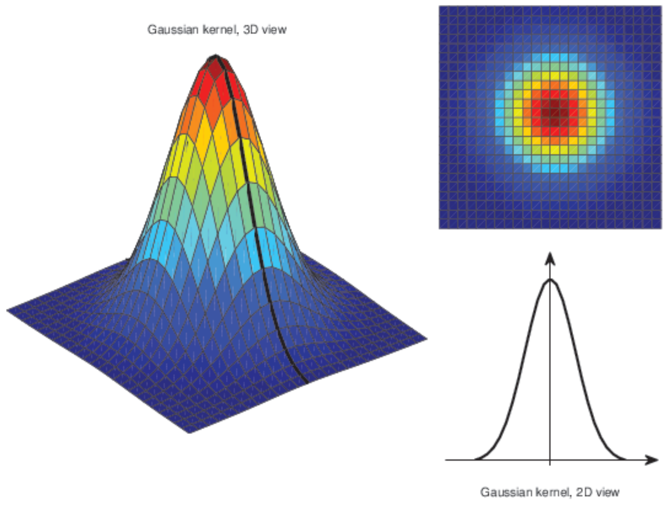


Figure 9: Gaussian kernel, $\sigma = 4$, $n = 25$. Is easy to see that the selected value of n is large enough to have a good approximation of the Gaussian function in the kernel.

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

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

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

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 $n = 31$.

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 of each one. This is because the function LoG has wider support than the Gaussian function (i.e. LoG function has a slower decay), so a greater value of n is needed to

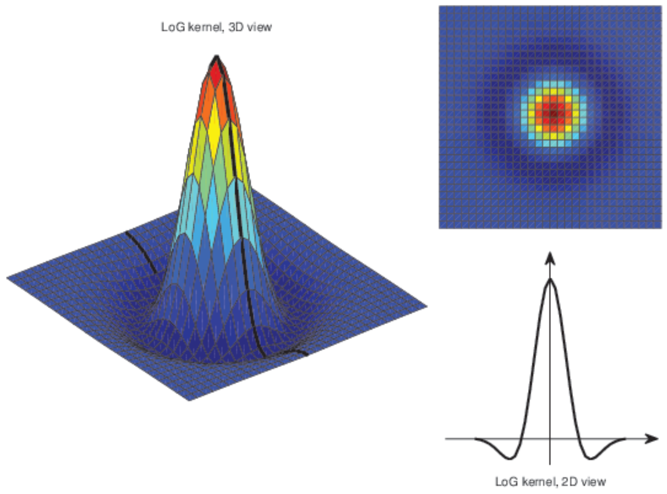


Figure 10: Laplacian of a Gaussian kernel, $\sigma = 4$, $n = 31$. The selected value of n is sufficient to have a good approximation of the LoG function in the kernel, but is greater than in the case of Gaussian kernel.

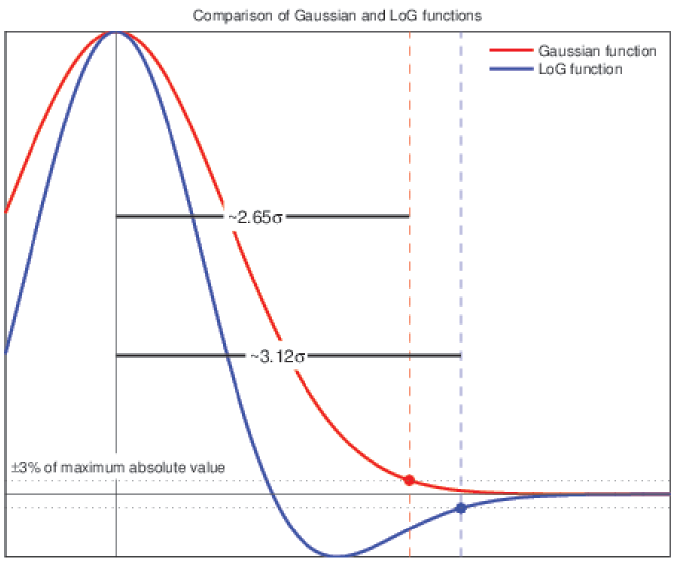


Figure 11: Comparison of the Gaussian and LoG functions.

generate a correctly sampled LoG kernel than in the case of the Gaussian kernel, with the same σ.

Figure 11 shows both functions generated with the same value of σ. Is clearly required a larger kernel size in the case of the LoG function (approximately 18% more). 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 making a convolution, it is necessary to define the boundary conditions used to calculate such convolution around the edges of the image, and to get a valid output the same size as input image. In this paper, two methods were implemented: zero-padding and boundary reflection. Zero-padding implies complete the borders of the image with many zero valued pixels as needed (depending on the size of the convolution kernel). Reflection involves completing 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.

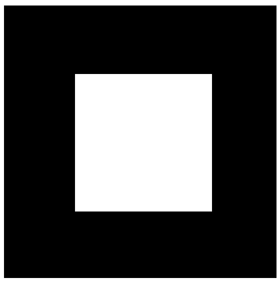
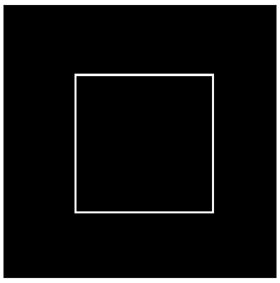
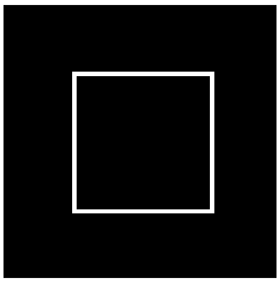
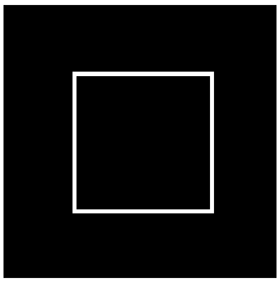


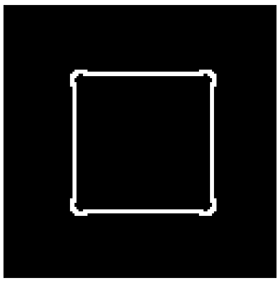
Figure 12: Test image 1: white square on black background (127×127 pixels).

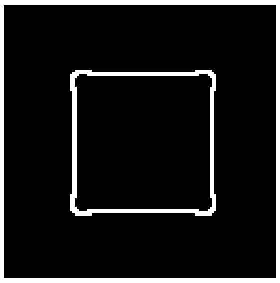
Figures 13(a) to 13(f) show the output of each algorithm (including execution times[8]).


(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 : $0.05s$.

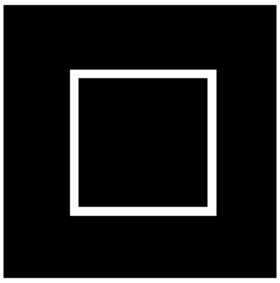

(f) Haralick. $\rho = 0.4$. Exec-time : $0.04s$.

Figure 13: Results of the algorithms using the 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 Gaussian blur.

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.

Now the performance of each algorithm is analyzed using a natural image shown in Figure 14. First the results of the first derivative edge detection algorithms are shown Figures 15(a) to 15(c). Figures 15(d) and 15(e) show the results obtained using the Marr-Hildreth algorithm (both Gaussian and LoG kernels), and Figure 15(f) shows the results obtained using the Haralick algorithm. Table 5.1 shows the execution time for each of the algorithms.

In this example the difference in performance between the first derivative edge detection algorithms and the second derivative ones is more significative. It can be seen in Figure 16 an area of interest of the image, enlarged to have a better view of the detail. 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.

[8]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.

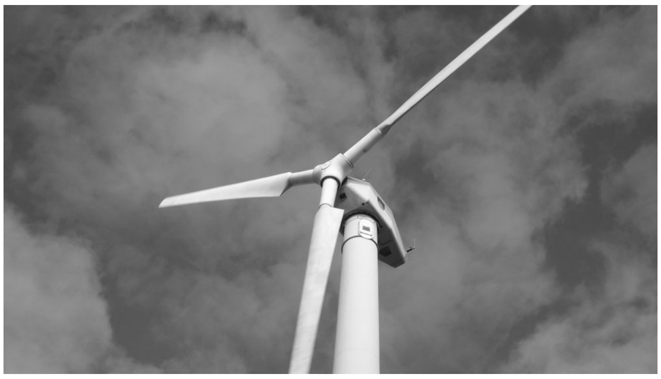
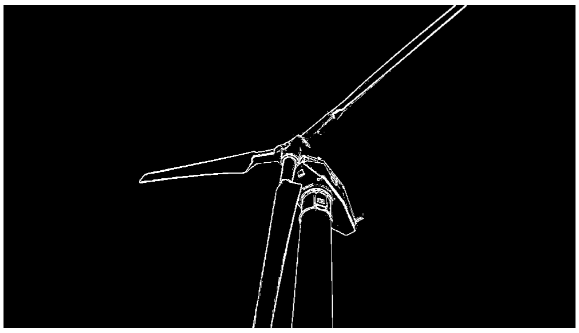
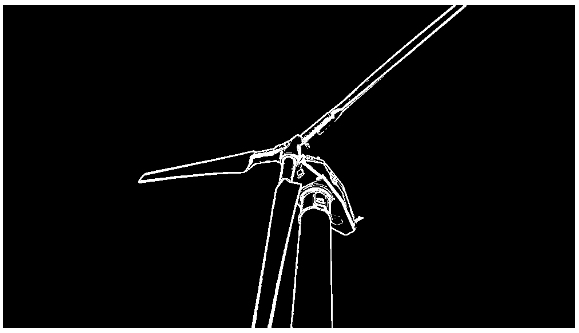


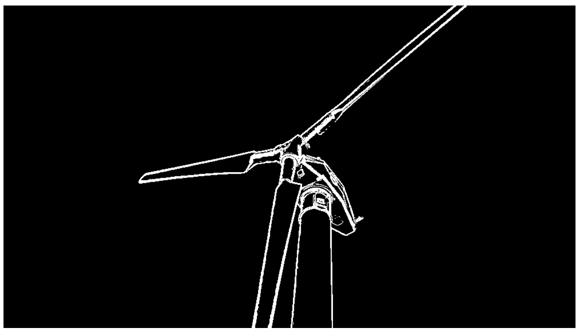
Figure 14: Test image 2: windmill (1000×563 pixels).



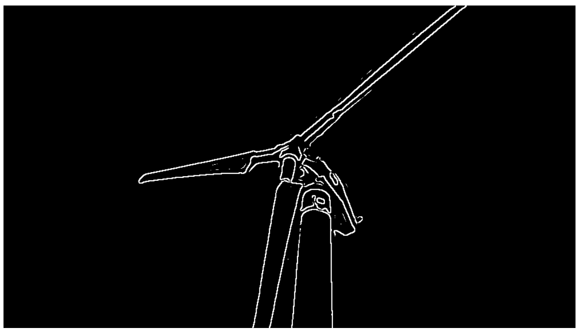
(a) Roberts. $th = 0.1$. Exec-time : $0.55s$.



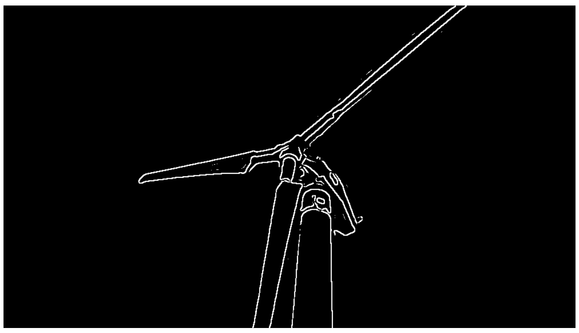
(b) Prewitt. $th = 0.1$. Exec-time : $0.55s$.



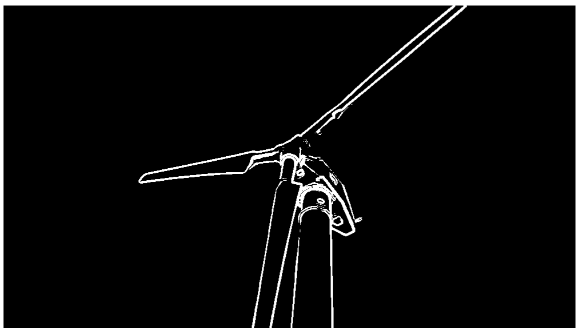
(c) Sobel. $th = 0.1$. Exec-time : $0.55s$.



(d) Marr-Hildreth (Gaussian). $\sigma = 3$, $n = 25$, $th_{ZC} = 0.07$. Exec-time : $1.05s$.



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



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

Figure 15: Results of the first derivative, Marr-Hildreth and Haralick's algorithms using Figure 14 as input image.

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 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 17 and 18. More examples can be found in the online demo[9].

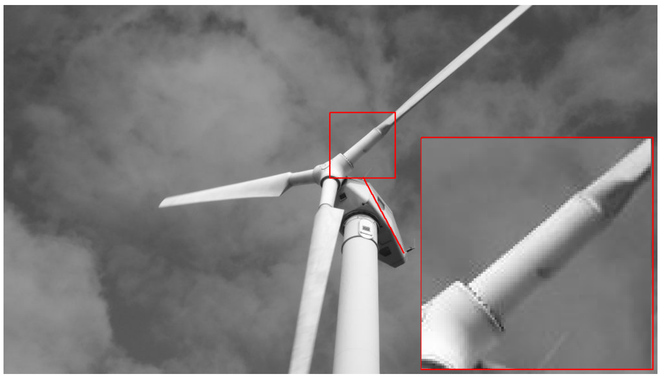
Some observations:

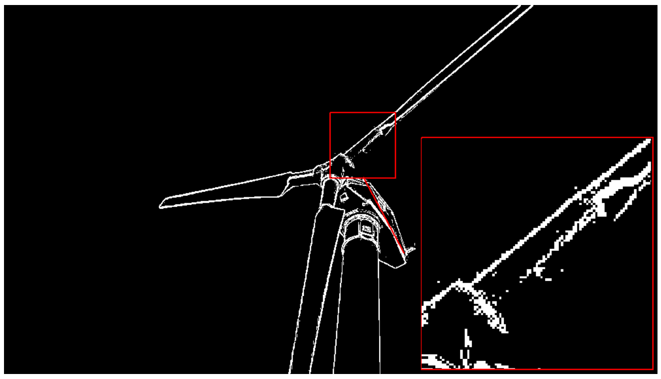
- The first derivative algorithms, although they work properly and quickly, have some problems like discontinuity of the edges. They are also very affected by noise in images, appearing unwanted isolated edges (More sophisticated methods help to avoid and improve this, e.g. Canny edge detector [10]).
- The Marr-Hildreth algorithm (in its two versions) achieved interesting results in the edge detail. 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 17), or inside the oranges (example 18).
- As mentioned earlier, Haralick's algorithm is the first to assume higher degree of regularity 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 18. 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.

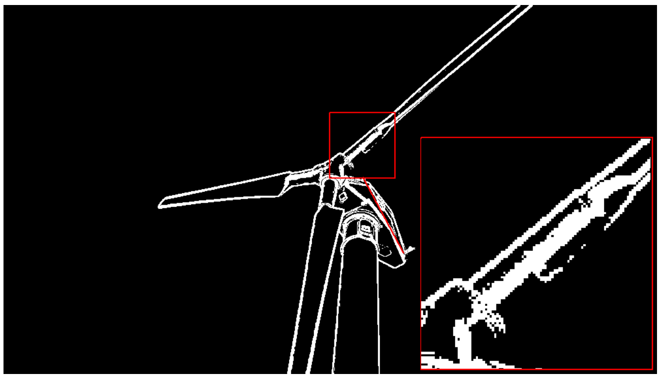
5.2 Video

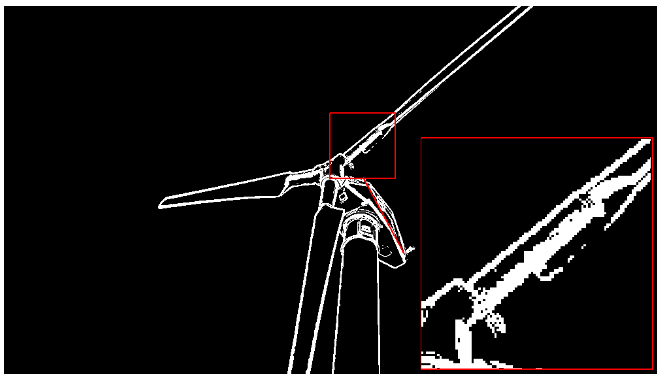
As supplementary material, there are videos testing the frame by frame application of these algorithms.

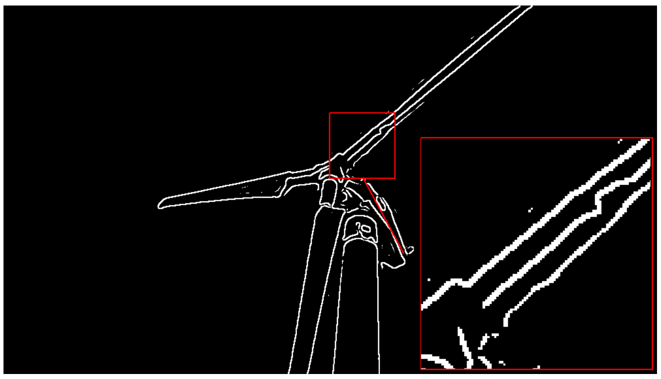
[9]`http://dev.ipol.im/~haldos/ipol_demo/xxx_edges/`

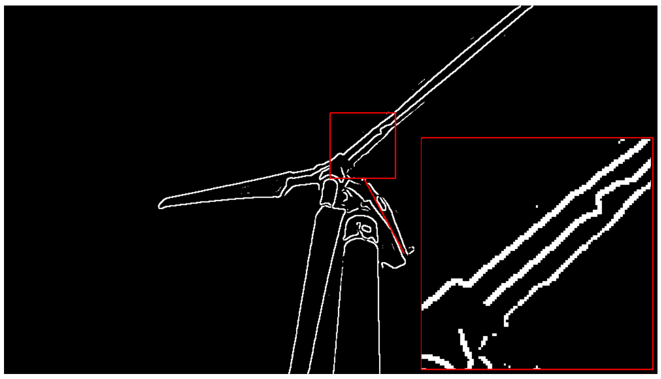

(a) Original.


(b) Roberts.


(c) Prewitt.


(d) Sobel.


(e) Marr-Hildreth (Gaussian).


(f) Marr-Hildreth (LoG).

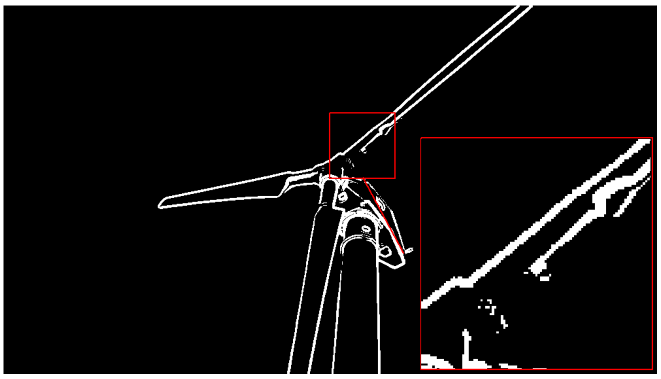
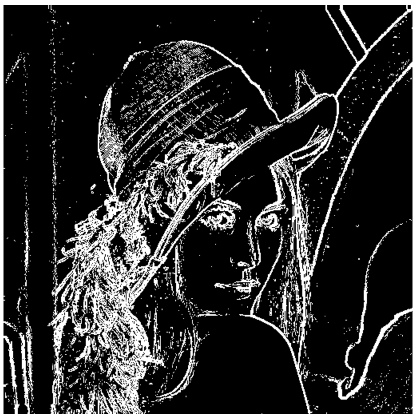
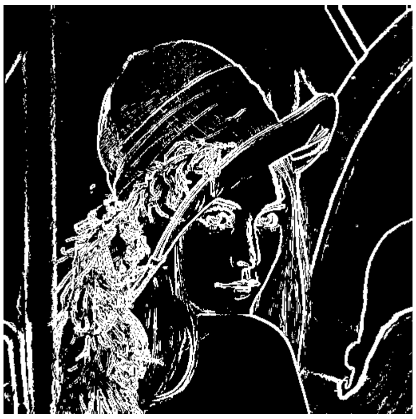

(g) Haralick.

Figure 16: Results of the algorithms using Figure 14 as input image, enlarging an interesting area.

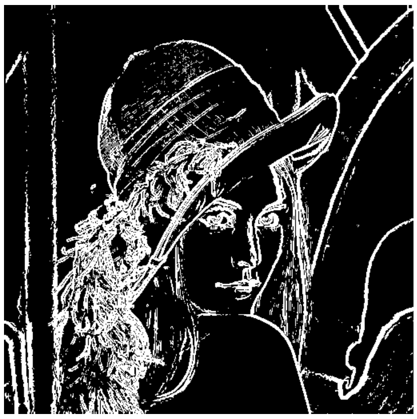
(a) Original.



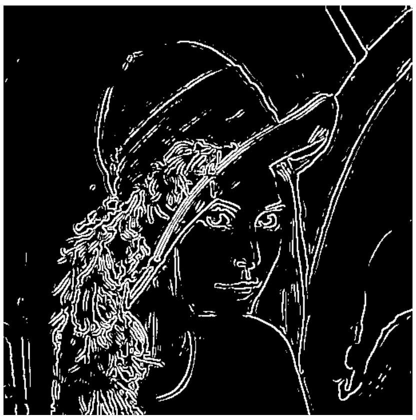
(b) Roberts. $th = 0.1$.



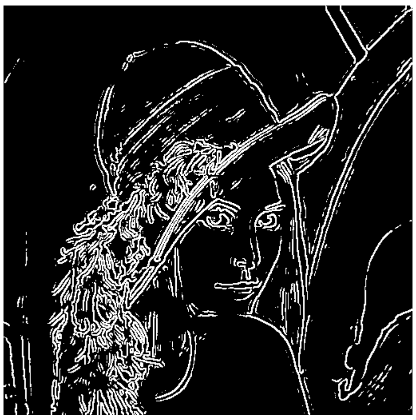
(c) Prewitt. $th = 0.1$.



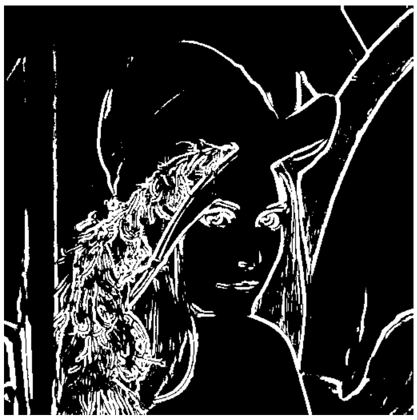
(d) Sobel. $th = 0.1$.



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

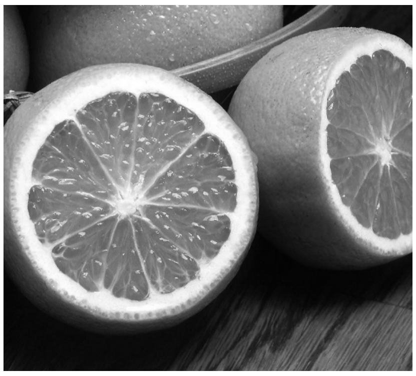


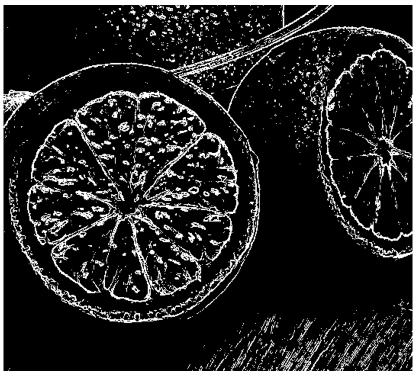
(f) Marr-Hildreth (LoG). $\sigma = 2$, $n = 25$, $th_{ZC} = 0.15$.

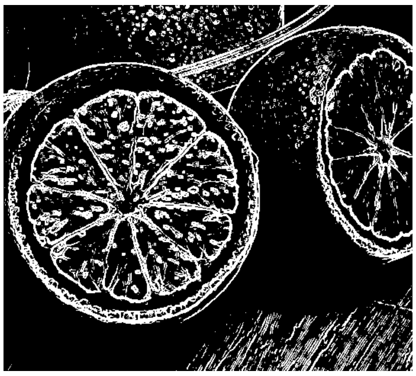


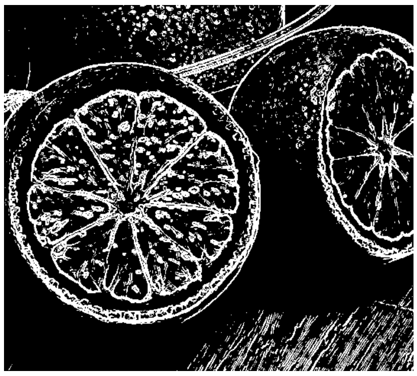
(g) Haralick. $\rho = 0.35$.

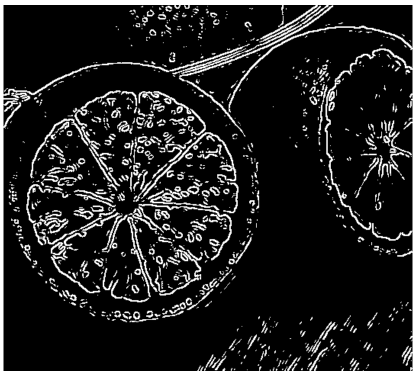
Figure 17: Example: Lena (512×512 pixels).

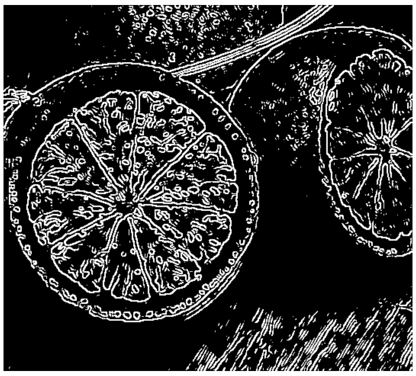

(a) Original.


(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). $\sigma = 2$, $n = 25$, $th_{ZC} = 0.15$.

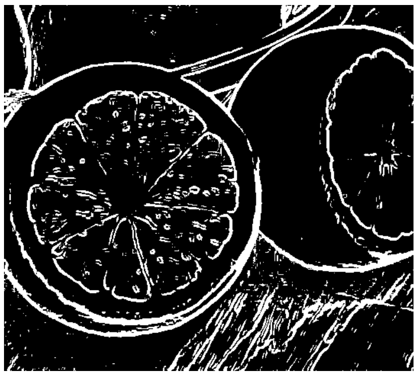

(g) Haralick. $\rho = 0.4$.

Figure 18: Example: Oranges (536×480 pixels).

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. Also they run extremely fast, because they only consists of 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 more fair the comparison with the second derivative methods.

The second derivative algoritms (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 significative 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 polinomial 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, 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, none of the implemented methods ensure edge connectivity, as the Canny [10] algorithm does.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Prentice Hall, 3 ed., Aug. 2007.

[2] D. Marr and E. Hildreth, “Theory of edge detection,” Tech. Rep. AIM-518, MIT Artificial Intelligence Laboratory, Apr. 6 1979.

[3] R. M. Haralick, “Digital step edges from zero-crossings of second directional derivatives,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 6, pp. 58–68, Jan. 1984.

[4] “Image Processing On Line.” `http://www.ipol.im/`. ISSN:2105-1232, DOI:10.5201/ipol[10].

[5] L. G. Roberts, “Machine perception of three-dimensional solids,” tech. rep., DTIC Document, 1963.

[6] J. M. Prewitt, “Object enhancement and extraction,” *B.S.Lipkin and A.Rosenfeld (eds.) Picture processing and psychopictorics*, pp. 75–149, 1970.

[7] R. Deriche, “Recursively implementing the Gaussian and its derivatives,” Tech. Rep. 1893, INRIA, May 1993.

[8] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” *SIGGRAPH Comput. Graph.*, vol. 21, pp. 163–169, Aug. 1987.

[9] P. Getreuer, “Linear Methods for Image Interpolation.” `http://www.ipol.im/pub/algo/g_linear_methods_for_image_interpolation/`, IPOL 2011. ISSN:2105-1232, DOI:.[11]

[10] J. Canny, “A Computational Approach to Edge Detection,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, pp. 679–698, Nov. 1986.

[10] `http://dx.doi.org/10.5201/ipol`

[11] `http://dx.doi.org/10.5201/ipol.2011.g_lmii`