Edge detection

Authors : Peter Bock*, Cécilia Ostertag, Ophélie Thierry

Introduction

Image processing is one of the most important fields in the domain of computer vision[1]. Most scientific domains use information extracted from images in one way or another. For a computer to make sense of these images, and be able to extract meaningful data from them, it needs to be able to interprete and understand them. That is where Image Processing comes in, allowing a computer to process an image and detect its major features, and to perform higher-level vision tasks like face recognition. In our project, we will examine one specific field of image processing called edge detection.

The physical notion of edge comes from the shape of three dimensional objects or from their material properties. But, seeing as the acquisition process translates 3D scenes to 2D representations, this definition does not apply to image processing. In this report we will use the following definition by Bovik (2009): "An edge can generally be defined as a boundary or contour that separates adjacent image regions having relatively distinct characteristics according to some features of interest. Most often this feature is gray level or luminance"[2]. According to this definition, the pixels of an image belonging to an edge are the pixels located in regions of abrupt gray level changes. Moreover, to avoid counting noise pixels as edges, the pixels have to be part of a contour-like structure. Edge detection is the process of finding the pixels belonging to the edges in an image, and producing a binary image showing the locations of the edge pixels.

In our project, we will begin by documenting the 3 main linear edge detection approaches and algorithms, and their implementation in the image processing software ImageJ[3]:

- Convolution with edge templates (Prewit, Sobel, Kirsh)[4] [5] [6]
- Zero-crossings of Laplacian of Gaussian convolution[7]
- Zero-crossings of directional derivatives of smoothed images (Canny)[8]

We will then perform a benchmark on the ImageJ plugins, in order to compare them by measuring their execution time and the memory load for the Java Virtual Machine (JVM).

The link to our github repository containing our report in markdown format, the images, and the code for the benchmark is : https://github.com/bockp/Edge-Detection-project .

[1] Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

[2] Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

[3] Schindelin J, Rueden CT, Hiner MC, Eliceiri KW. The ImageJ ecosystem: An open platform for biomedical image analysis. Molecular reproduction and development. 2015 Jul 1;82(7-8):518-29.

[4] Prewitt JM. Object enhancement and extraction. Picture processing and Psychopictorics. 1970 Jan 1;10(1):15-9.

[5] Sobel I. An isotropic 3× 3 image gradient operator, presentation at Stanford Artificial Intelligence Project (SAIL).

[6] Kirsch RA. Computer determination of the constituent structure of biological images. Computers and biomedical research. 1971 Jun 1;4(3):315-28.

[7] Marr D, Hildreth E. Theory of edge detection. Proceedings of the Royal Society of London B: Biological Sciences. 1980 Feb 29;207(1167):187-217.

[8] Canny J. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence. 1986 Nov(6):679-98.

Material & Methods

Edge-detection theory

The derivative or the gradient of the grey level intensity can be used to detect edges, as abrupt intensity changes translate to local extrema in the 1st derivative (Sobel approach), and to a zero-crossing in the 2nd derivative[9] [10] [Fig.1] (Laplacian approach).

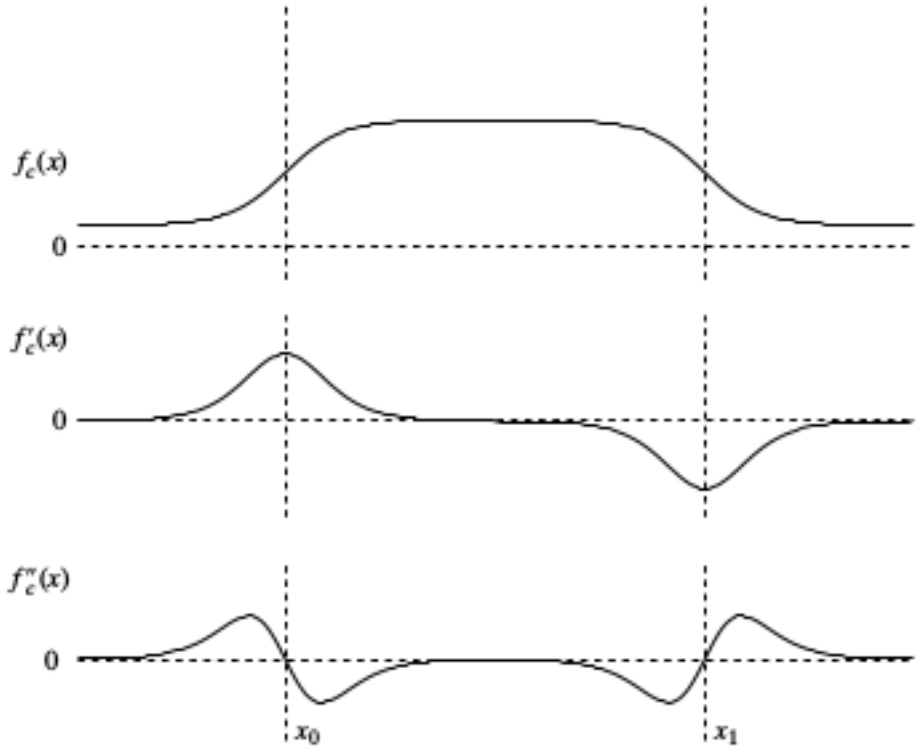

Fig.1: Edge detection in a 1D continuous space : fc(x) is the gray level intensity function, fc'(x) is the 1st derivative, and fc"(x) is the 2nd derivative. The vertical dotted lines represent the edge locations[11]

Edge detectors based on the derivative are sensitive to noise, which are pixels of aberrant intensity. This leads to the development of several algorithms to find the most relevant edges in an image. Most of them use a filter to reduce noise before detecting edges in the image[12]. These algorithms usually have three main steps:

[9] Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

[10] Chaabane SB, Fnaiech F. Color edges extraction using statistical features and automatic threshold technique: application to the breast cancer cells. Biomedical engineering online. 2014 Jan 23;13(1):4.

[11] Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

[12] Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

- smoothing: use of a filter to suppress the noise.
- differentiation: amplification of the edges in the image
- decision: detection of edges using 1st or 2nd derivatives, usually combined with thresholding.

Errors in edge detection can either be false positives (classification of non-edge pixels as edge pixels) or false negatives (classification of edge pixels as non-edge pixels). There is also a conflict between the correct detection of edges and the precise localization of their positions.

Robert's Cross Operator

The Robert's Cross Operator, first described in 1975 by Davis L.S.[13], performs a simple, efficient, computationally cheap 2D spatial gradient measurement on an image, which is based on the 2D measure of the 1st derivative[Eq.1]

$$\nabla f_c(x,y) = \frac{\partial f_c(x,y)}{\partial x}\mathbf{i}_x + \frac{\partial f_c(x,y)}{\partial y}\mathbf{i}_y,$$

Eq.1: Gradient of a continuous gray level intensity function fc(x,y), where ix and iy are the unit vectors in the x and y directions[14]

The operator consists of a pair of 2X2 convolution kernels[Fig.2], designed to respond maximally to edges running at a 45° angle[15].

+1	0
0	-1

Gx

0	+1
-1	0

Gy

Fig.2: Robert's Cross operator's horizontal and vertical convolution masks[16]

Sobel Operator

The Sobel Operator, introduced in a presentation at the Standford A.I Project in 1968 by Irwin Sobel[17], is the default algorithm implemented in ImageJ for the Find Edges function, and is considered one of the simplest functional Edge Detection algorithms.

[13]Davis LS. A survey of edge detection techniques. Computer graphics and image processing. 1975 Sep 1;4(3):248-70.
[14]Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.
[15]Maini R, Aggarwal H. Study and comparison of various image edge detection techniques. International journal of image processing (IJIP). 2009 Jan;3(1):1-1.
[16]Maini R, Aggarwal H. Study and comparison of various image edge detection techniques. International journal of image processing (IJIP). 2009 Jan;3(1):1-1.
[17]Sobel I. An isotropic 3× 3 image gradient operator, presentation at Stanford Artificial Intelligence Project (SAIL).

It is based on the gradient of the gray level intensity function[Eq.1].

After finding all the local extrema of the gradient magnitude, a thresholding step is applied and the points where the magnitude is superior to a given threshold are classified as candidate edge points [Eq.2].

$$\left|\nabla f_c(x,y)\right| \geq T.$$

Eq.2: Thresholding of the gradient magnitude, where T is the threshold[18]

Finally, to obtain edges as zero-width segments, a thinning step is required : if the gradient magnitude is not a local maximum along the gradient direction, the point is suppressed from the edge candidates.

In practice, this algorithm works by using two masks, one horizontal and one vertical[19] Fig.3, these masks are designed to respond maximally to edges in the horizontal and vertical directions, respectively, and also smoothen out the gaussian noise in advance to reduce the noise sensitivity of the algorithm.

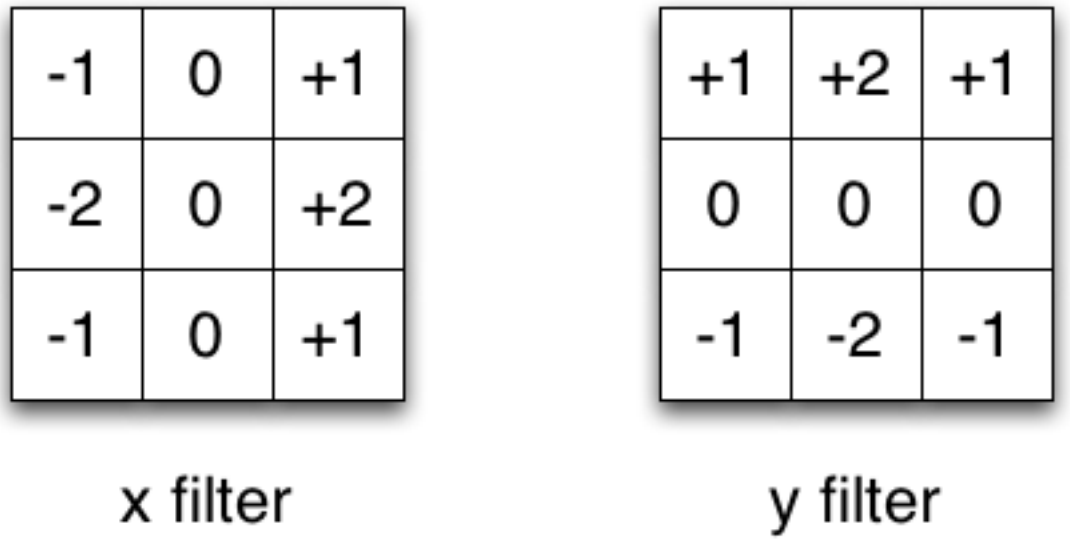

Fig.3: Sobel operator's horizontal and vertical convolution masks[20]

The two resulting images are then combined to get an image representing the approximate absolute gradient magnitude of the original image[Fig.4].

[18]Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

[19]Vincent OR, Folorunso O. A descriptive algorithm for sobel image edge detection. InProceedings of Informing Science & IT Education Conference (InSITE) 2009 Jun 12 (Vol. 40, pp. 97-107).

[20]Vincent OR, Folorunso O. A descriptive algorithm for sobel image edge detection. InProceedings of Informing Science & IT Education Conference (InSITE) 2009 Jun 12 (Vol. 40, pp. 97-107).

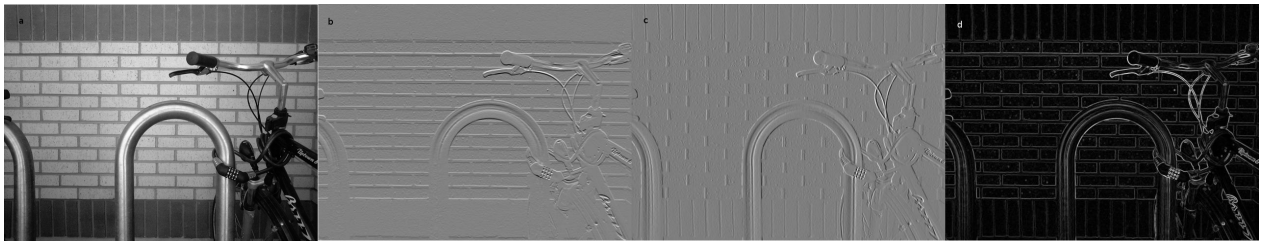

Fig.4: Result of Sobel filtering. a: original image, b: Sobel Y-gradient image, c: Sobel X-gradient image, d: absolute gradient magnitude image

Prewitt Operator

The Prewitt operator, developed by Judith M. S. Prewitt[21], is also based on the gradient of the gray level intensity function[Equation.1], and functions in a similar way to the Sobel algorithm, though using different convolution masks[Fig.5]

-1	0	+1
-1	0	+1
-1	0	+1

Gx

+1	+1	+1
0	0	0
-1	-1	-1

Gy

Fig.5: Prewitt operator's horizontal and vertical convolution masks[22]

Kirsch Operator

The Kirsch operator, named after the computer scientist Russell A. Kirsch[23], uses the same gray level gradient based approach as the Sobel, Prezitt and Robert's Cross operators, but with a more complex kernel convolution.

The Kirsch Operator possesses a single convolution kernel, but the kernel in question is applied on every part of the image in 8 different configurations (created by turning the default kernel by 45° segments[Fig.6]), and the edge magnitude is calculated as being the maximum magnitude found across the 8 different configurations, while the edge direction is determined by the kernel configuration possessing that maximum gradient value.

[21]Prewitt JM. Object enhancement and extraction. Picture processing and Psychopictorics. 1970 Jan 1;10(1):15-9.

[22]Maini R, Aggarwal H. Study and comparison of various image edge detection techniques. International journal of image processing (IJIP). 2009 Jan;3(1):1-1.

[23]Kirsch RA. Computer determination of the constituent structure of biological images. Computers and biomedical research. 1971 Jun 1;4(3):315-28.

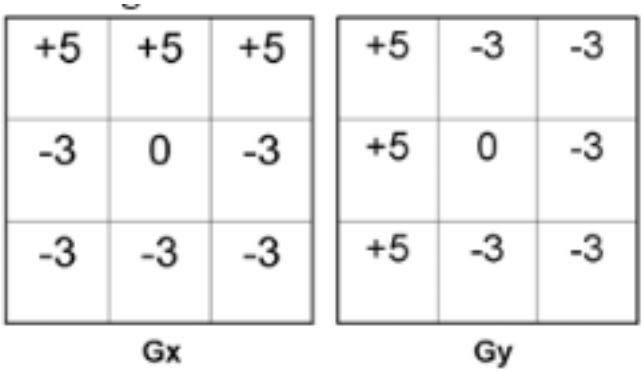

Fig.6: Kirsch kernel, in the original configuration (left) and configuration 7 (6 45° rotations to the right performed on the original configuration)[24]

Laplacian based methods:

The Laplacian[25] is a 2D isotropic measure of the 2nd spatial derivative[Eq.3]. It is used to detect regions of rapid intensity change in an image : - In the regions of constant intensity (intensity gradient equal to zero), the Laplacian is equal to zero. - In regions where there is a change in intensity, the Laplacian is positive on the darker side, and negative on the lighter side.

$$\nabla^2 f_c(x,y) = \nabla \cdot \nabla f_c(x,y) = \frac{\partial^2 f_c(x,y)}{\partial x^2} + \frac{\partial^2 f_c(x,y)}{\partial y^2}.$$

Eq.3: Laplacian of a continuous gray level intensity function fc(x,y)[26]

This has the effect of highlighting the edges in the image, and can be used as an enhancement technique, by adding the filtered image to the original image.

The Laplacian can be estimated by designing a pair of 1D 2nd derivative filters and combining them into a 2D filter [Eq.4]

$$\begin{aligned}\nabla^2 f_c(x,y) \rightarrow \widehat{\nabla}^2 f(n_1,n_2) &= f_{xx}(n_1,n_2) + f_{yy}(n_1,n_2) \\ &= f(n_1+1,n_2) + f(n_1-1,n_2) + f(n_1,n_2+1) \\ &\quad + f(n_1,n_2-1) - 4f(n_1,n_2) \\ &= \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.\end{aligned}$$

Eq.4: Discrete Laplacian estimate for an image f(n1,n2)[27]

[24]Kekre HB, Gharge SM. Image segmentation using extended edge operator for mammographic images. International journal on computer science and Engineering. 2010;2(4):1086-91.

[25]Marr D, Hildreth E. Theory of edge detection. Proceedings of the Royal Society of London B: Biological Sciences. 1980 Feb 29;207(1167):187-217.

[26]Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

[27]Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

Other 3x3 kernels are :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \qquad \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}.$$

Fig.7: Laplacian operator kernels)[28]

The Laplacian opeator is usually used on gray level images, previously smoothed with a Gaussian filter to reduce noise. It is also possible to convolve the Gaussian smoothing filter with the Laplacian filter, before convolving this Laplacian of Gaussian (LoG)[Eq.5] with the image.

$$\begin{aligned} h_c(x,y) &= \nabla^2 g_c(x,y) \\ &= \frac{x^2+y^2-2\sigma^2}{\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right). \end{aligned}$$

Eq.5: Laplacian of Gaussian function gc(x,y), where sigma is the standard deviation of the Gaussian function[29]

To implement a discrete form, a filter can be constructed by sampling this equation after choosing a value for sigma. The Gaussian and Laplacian kernels are both small so it requires fewer operations than using both filters on the image. Another advantage of the LoG is that it can be calculated in advance as it is independent of the image being processed. It is important to note that the result of these edge detectors is highly influenced by the standard deviation used for the Gaussian filter chosen for the smoothing step. The LoG can also be approximated by the Difference of Gaussian (DoG).

It is not possible to directly extract the edge orientation information from the Laplacian output. To extract the edges, we need to detect the zero-crossings in the output of the Laplacian (or the LoG), i.e. the regions of the image where the Laplacian passes through zero. However this can also happen in regions that are features other than edges in the image and can be the cause of false positives. The input of the zero-crossing detector is the LoG filtered image, and the output is a binary image with lines representing the positions of all the zero-crossing points. Each pixel of the image is compared to its eight immediate neighbors, and a pixel is classified as a zero-crossing if its sign is different than the sign of its neighbors.[Eq.6]

$$\left|\nabla^2 f(\mathbf{p})\right| \leq \left|\nabla^2 f(\mathbf{q})\right|.$$

Eq.6: Zero-crossing classification of a pixel p[30]

[28]Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.
[29]Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.
[30]Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

All of the contour lines are closed lines because the strength of the edge is not considered, so even gradual intensity transitions result in a zero-crossing. As previously indicated, local minima of the gradient magnitude can cause false edges, that can be eliminated by using a threshold for edge strength, causing breaks in the closed contours.

In ImageJ, two plugins provide an implementation of the LoG operator. The first one is the Laplacian plugin included in the FeatureJ package created by Erik Meijering. This plugin is based on ImageScience, a java library for image processing, which provides tools for computing the LoG of an image and detecting the zero-crossings. The only parameter accessible to the user is the laplacian smoothing scale, meaning the standard deviation used for the Gaussian kernel. The second one is the Log_Filter plugin, by Lokesh Taxali and Jesse Jin. This plugin is composed of a unique class file and provides more parameters to the user : sigma (standard deviation for the Gaussian filter), filter width (size of the LoG kernel), threshold for 1D DoG filtering, and delta (level for adjusting zero-crossings). Unlike the previous plugin, this one involves thresholding of the LoG output with a Difference of Gaussians. Moreover, its use is limited to 8-bit images.

Canny Operator

The Canny operator is often used and known as less sensitive to noise than Sobel's and Laplace's[31] [32], despite its time consuming calculations[33] and its sensibility to textured regions on image which lead to define false edges and discontinuous edges[34].

The implementation of the Canny model is relatively simple. First a Gaussian filter is applied to the image, then the gradient magnitude of the signal has to be defined. For this step, the aim is to discriminate edges defined by local maxima by comparing the value of the local maxima to its two closest neighbors. These firsts steps lead to an initial edge map. A thresholding step is then applied to the map by defining a low and high threshold value which will define major and minor pixels used for a thresholding final step hysteresis. All the edge pixels whose values are above the higher threshold will be kept on the map, but the pixels whose values are under the lower threshold will be removed. This last step, though, can lead to disrupted edges[35] [36] [37] [38] [39].

In the original work by Canny in 1986, the author defines the signal[Fig.1] as the combination of a noise function and an edge function. So the edge is detected by the convolution of the signal with a specific filter. The aim of the author was to find a mathematical function which would define edges using three distinct steps.

[31] Zhao J, Zheng W, Zhang L, Tian H. Segmentation of ultrasound images of thyroid nodule for assisting fine needle aspiration cytology. Health information science and systems. 2013 Dec 1;1(1):5.

[32] Abdelsamea MM, Gnecco G, Gaber MM, Elyan E. On the relationship between variational level set-based and som-based active contours. Computational intelligence and neuroscience. 2015 Jan 1;2015:34.

[33] Chaabane SB, Fnaiech F. Color edges extraction using statistical features and automatic threshold technique: application to the breast cancer cells. Biomedical engineering online. 2014 Jan 23;13(1):4.

[34] Canny J. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence. 1986 Nov(6):679-98.

[35] Canny J. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence. 1986 Nov(6):679-98.

[36] Deriche R. Using Canny's criteria to derive a recursively implemented optimal edge detector. International journal of computer vision. 1987 Jun 1;1(2):167-87.

[37] Ding L, Goshtasby A. On the Canny edge detector. Pattern Recognition. 2001 Mar 31;34(3):721-5.

[38] Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

[39] Abdelsamea MM, Gnecco G, Gaber MM, Elyan E. On the relationship between variational level set-based and som-based active contours. Computational intelligence and neuroscience. 2015 Jan 1;2015:34.

The first is the actual edge detection, then defining the edge as close as possible to the real localization of the edge on the picture, and finally to assure the unambiguity of the signal. For this last point, often the signal is not a smooth curve[Fig.1] but a series of small maxima close to each other, which can lead to the detection of two edges instead of a single one.

The first step affects the number of false-negatives and the two others the number of false positives[40]. Therefore, they defined that the best edge function will be the one to maximize the following equation[Eq.7].

$$\text{SNR}(f) * \text{Localization}\ (f) - \sum \mu_i P_i(f)$$

Eq.7: Equation used to define the best function to find edges from a grey-level signal, with f as the function which is used to define edges, and Pi the constrains i modulated by its associated factor μiEquation used to define the best function to find edges from a grey-level signal, with f as the function which is used to define edges, and Pi the constrains i modulated by its associated factor μi[41]

The SNR is the signal-to-noise ratio, which has to be as high as possible, but wihtout affecting the localization precision. That is why the author uses the product of these two measurements, so as to maximize both. The third part concerns the sum of the additional constraints that the function has to take into account, as the reduction of edges due to local maxima, and will be defined as the sum of the series of penalty functions. For this, they defined an expression representing the distance between adjacent noise peaks in a defined space, using the Rice noise studies on the response of a function to the application of Gaussian noise[42]. This solution involves the first and the second derivatives of the answer function and leads to the creation of a factor which will define the number of maxima in a specific width, which can lead to false answers. This factor corresponds to a fraction of the defined interval of the beginning.

The SNR assesses the quality of a signal according to a model, by using the model as numerator and the noise definition as the denominator. In this case, both parameters are defined as integral, the absolute value of a convolution integral and the root-mean-square response to the noise function, respectively, which will represent the area of dispersion from the model. Each additional constraint will be modulated by a specific value which will define the importance of the associated constraint in the model. As edges are defined as local maxima, the addition of the firsts derivatives of the edge function and noise function is equal to zero[Fig.1], so both of them are opposite and can be used to define the localization parameter. It will be defined as the first derivative of the SNR function.

After several demonstrations, Canny demonstrated that the Gaussian operator is the most efficient way to maximize this equation [Eq.8], and it has the advantage to be easy to implement for a two dimensional model.

$$g_G(x,y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

[40]Canny J. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence. 1986 Nov(6):679-98.

[41]Canny J. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence. 1986 Nov(6):679-98.

[42]Rice SO. Mathematical analysis of random noise. The Bell System Technical Journal. 1945 Jan;24(1):46-156.

Eq.8: Gaussian filter[43]

Several modifications were made to the model in order to improve its efficiency, such as the Deriche modification, allowing the model to use higher threshold values. This lead to a change to the efficiency of the Canny method, according to the values considered after a Fourier transformation, leading to the development of a new fonction with only one constant parameter [44] [Eq.9], also called the Ding modification, able to take into account pixels under the low threshold value in order to correct edge disruptions[45].

$$g(x) = -c \cdot x \, e^{-\alpha \cdot |x|}$$

Eq.9: Gaussian filter[46]

Currently, several plugins using this method have been developed for ImageJ: the Edge Detection, using Canny-Deriche filtering, by Thomas Boudier, the Edge Detector by Carmelo Pulvirenti, able to use other operators (LoG, DoG) and FeatureJ Edges by Erik Meijering.

All of them require from the user to specify the standard deviaiton for a Gaussian kernel, which will be involved in the initial processing step by the Gaussian filter, and will define the width of the neighborhood in which only a single peak will be identified. The two other parameters are the low and high threshold value for non-maximum suppression.

As we haven't managed to make Carmelo Pulvirenti's plugin work, it was not used in this analysis. Also, the latest review on this plugin was in July 2007, whereas it was in April 2015 for Thomas Boudier's plugin and December 2015 for the Erik Meijering's plugin according to their github repository.

Approaches for color images

Unlike greyscale, images encoded in a color space are composed of three channels, which makes the computation trickier. Algorithms for color images usually only take the luminance component into account. This is a cost effective method because according to the color space either the luminance is one of the channels or it can be computed directly, for example with an RGB image. However, all edges are not necessarily best described by the luminance component, which results in a number of false negative edge pixels candidates. Another approach is to construct a cumulative edge map from each component of the image (for example after calculating the sum of the gradient magnitude of the three channels of an RGB image), but the results will be biased depending of the chosen color space[47].

Some algorithms developed for edge detection in color images, based on vector approaches, are described in

[43] Canny J. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence. 1986 Nov(6):679-98.

[44] Deriche R. Using Canny's criteria to derive a recursively implemented optimal edge detector. International journal of computer vision. 1987 Jun 1;1(2):167-87.

[45] Ding L, Goshtasby A. On the Canny edge detector. Pattern Recognition. 2001 Mar 31;34(3):721-5.

[46] Deriche R. Using Canny's criteria to derive a recursively implemented optimal edge detector. International journal of computer vision. 1987 Jun 1;1(2):167-87.

[47] Bovik AC, editor. The essential guide to image processing. Academic Press; 2009 Jul 8.

papers by Trahanias and Venetsanopoulos[48] and Scharcanski and Venetsanopoulos[49].

Benchmarking process

The performance and the efficiency of each edge detection function can be assessed through several parameters: the execution time necessary for the processing of an input image and the memory load corresponding to this opperation.

To perform this benchmark, we implemented a small JavaScript plugin : *benchmark.js*. This script measures, on one hand the time elapsed between the start and the end of a given ImageJ or plugin function, and on the other hand the memory used by ImageJ JVM at the end of this function. Java uses a garbage collector to handle memory allocation so our results have to be treated cautiously,even if we forced the garbage collector to run before the execution of the function. The image used for this benchmark is the default picture Lena, 8-bit, 256x256 pixels.

For both measurements we ran the operation 100 times, after a front loading step consisting of running each function five times without recording the results. This was done to avoid outliers in our data, because the first executions of a function are usually slower because of internal allocations and loading of images in RAM or cache.

As the Canny plugin is also able to process RGB pictures, a new benchmark is done with the Lena default picture 256x256 pixels in 8 bits and RGB colors.

The benchmark was done using a computer with an Intel core I7 @4.0 Ghz, on Linux Ubuntu 16.04 64 bits with a kernel 4.10. The version of ImageJ is the 1.51q, using Java 1.8.0_112 (64 bits). We fixed the processor fequency with acpi-cpufreq module to avoid a change of frequency during the benchmark, fixed the choice of processor with the taskset command to avoid a sharing of the processor load, and finally we fixed the ImageJ process with a high priority to avoid preemption.

Results

Implementations of Sobel algorithm

The output of ImageJ Find Edges function is an 8-bit image in which the contours are in white[Fig.8]. Edges are founded but some of them seem to have a width define by several pixels or be made of multiple edges with a one pixel width side by side. It is also difficult to visualy define a precise enclosed contour-like structure.

[48]Trahanias P.E and Venetsanopoulos A.N. Color edge detection using vector order statistics. IEEE Trans. Image Process., 2(2):259–264, 1993.

[49]Scharcanski J and Venetsanopoulos A.N. Edge detection of color images using directional operators. IEEE Trans. Circuits Syst. Video Technol., 7(2):397–401, 1997.

Fig.8: Result of Find Edges function. 1: Input image, 2: Output image

Implementations of Laplacian of Gaussian algorithm

The FeatureJ Laplacian only provides the display of the output of the LoG and the zero-crossing detector [Fig.9]. The contours are also of single-pixel width and in absence of a thresholding all contours are closed lines.

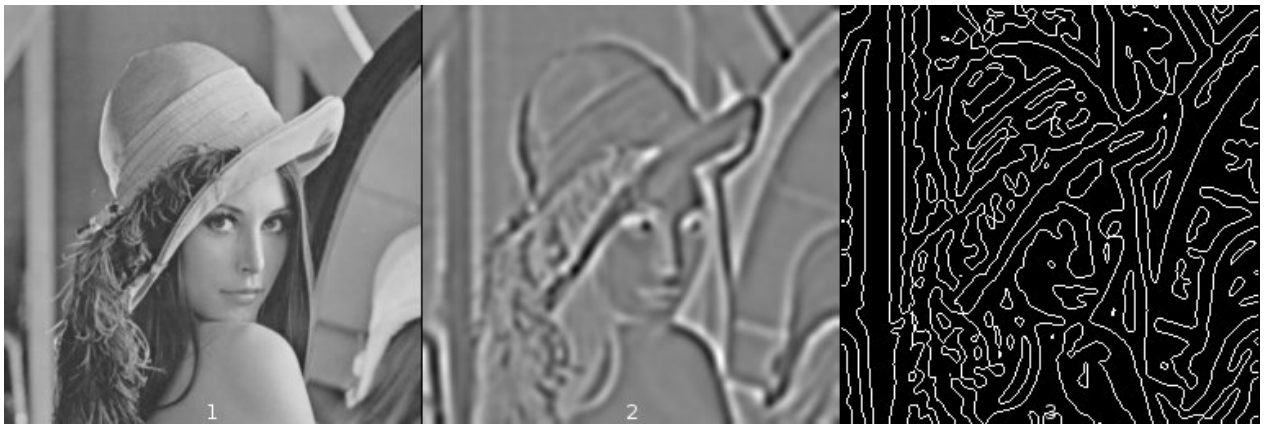

Fig.9: Result of FeatureJ Laplacian plugin, with smoothing scale=3 1: Input image, 2: LoG output, 3: Zero-crossings

Choosing a higher standard deviation for the gaussian filtering reduces the number of false positives but also

reduces the precision for true positive edges [Fig.10].

Fig.10: Result of FeatureJ Laplacian plugin, with various smoothing scales. 1: Input image, 2: smoothing scale=1, 3: smoothing scale=3, 4: smoothing scale=5

With the Log_Filter plugin, we can compute the output of the LoG filtering (in a 0 to 255 range), the absolute value of filtering, the results representing the LoG values -1, 0 or 1, the zero-crossings overlaid to the input image, and finally the output of the zero-crosing detector. The final output is a binary image where the edge pixels are in white[Fig.11]. All the contour lines have a single-pixel width, and due to the DoG filtering the final output is the edges of the contour zones detected by the LoG.

Fig.11: Result of Log_Filter plugin, with sigma=3, filter width=2, DoG threshold=0 delta=0. 1: Input image, 2: LoG output, 3: Absolute value of filtering, 4: Results representing values -1,0 or 1, 5: Zero-crossings overlaid to input image, 6: Zero-crossings

While the result is different from the one given by FeatureJ, we can see that the influence of the standard deviation is similar : a higher value for the smoothing step leads to imprecise contours and the loss of the objects' shapes [Fig.12]. Here the thresholding step prevents us from seeing the impact of the noise removal on the number of false positive edge pixels.

Fig.12: Result of FeatureJ Laplacian plugin, with various smoothing scales. 1: Input image, 2:

smoothing scale=3, 3: smoothing scale=5, 4: smoothing scale=9

Implementations of Canny algorithm

The outputs of Canny Edge Detector and FeatureJ Edges are binary images where the edge pixels are white[Fig.13]. A visual comparision of the outputs of the two Canny implementations with the same initial parameters show that the results are similar.

Fig.13: Result of Canny algorithm plugins, with gaussian kernel radius=2, low threshold=2.5, high threshold=7.5. 1: Input image, 2: Output of Canny Edge Detector, 3: Output of FeatureJ Edges

Knowing this, we decided to test the influence of each parameter only for the Canny Edge Detector plugin, assuming that the results given by FeatureJ Edge would be the same. A variation of the standard deviation of the Gaussian function[Fig.14] has an influence of the number of pixels classified as edges by the function : with a standard deviation of 1.0 the contours of the eyes, nose, mouth and feathers are well defined, but some contours are created which have no physical sense like the line below the right eye. Raising this value to 2.0 and then 3.0 makes weaker edges disappear from the results, until only the outlines remain.

Fig.14: Result of Canny Edge Detector plugin, with various gaussian kernel radius, low threshold=2.5, high threshold=7.5. 1: Input image, 2: Radius=1.0, 3: Radius=2.0, 4: Radius=3.0

We then changed the value of either the low or the high threshold values[Fig.12]. A reduction of the low

threshold does not seem to have a significant impact on the result, but an augmentation of the high threshold reduces the number of edges detected and creates gaps in continuous edges.

Fig.15: Result of Canny Edge Detector plugin, with gaussian kernel radius=2.0 and various low and high thresholds. 1: Input image, 2: Low threshold=2.5 and high threshold=7.5, 3: Low threshold=0.1 and high threshold=7.5, 4: Low threshold=2.5 and high threshold=10

Canny edge detector is the only plugin working on RGB images, and gives an output similar to the one obtained with the 8-bit image[Fig.16].

Fig.16: Result of Canny Edge Detector plugin, with gaussian kernel radius=2, low threshold=2.5, high threshold=7.5. 1:8-bit input image, 2:8-bit output image, 3:RGB input image, 4:RGB output image

Benchmark results

The results of the benchmark for the execution time[Fig.17 and Fig.19] show that the Find Edges function is the quickest to run on this machine, with a mean of 0.88 ms, followed by Log_Filter, FeatureJ Laplacian and FeatureJ Edges which do not have a mean execution time superior to 50 ms. However the Canny Edge Detector plugin has an average exectution time of 205.8 ms.

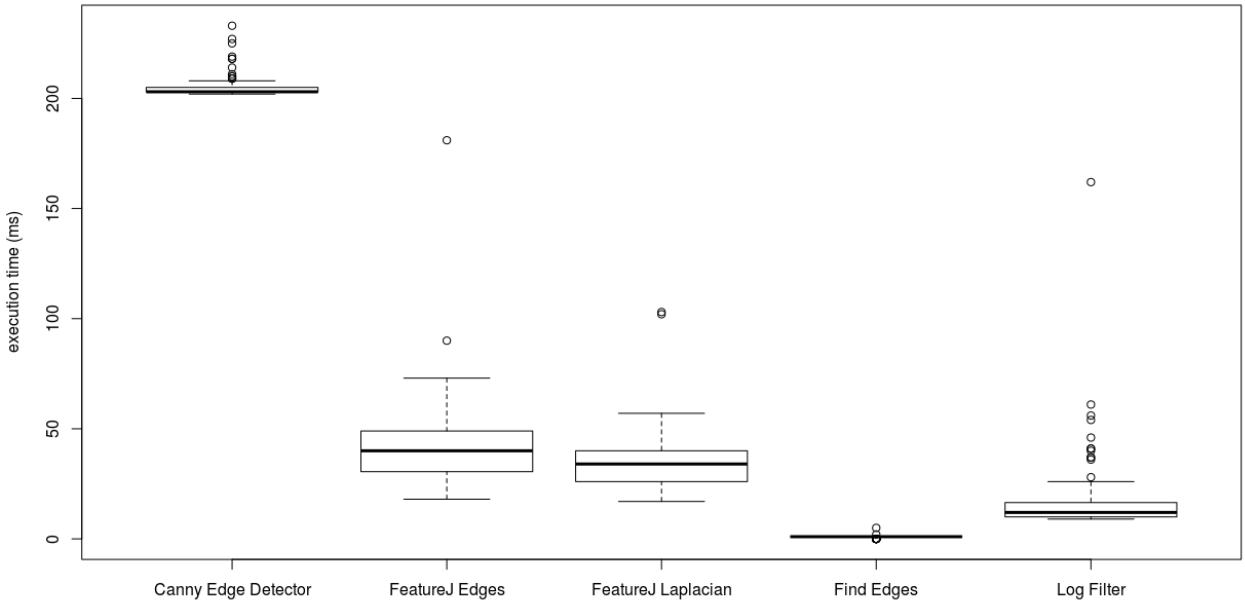

Fig.17: Result of the benchmark for the execution time of ImageJ edge detection functions. The Canny Edge Detector and FeatureJ Edges plugin use the Canny algorithm, FeatureJ Laplacian and Log Filter use the LoG, and Find Edges the Sobel

For the JVM memory load [Fig.18 and Fig.19], we can see that Find Edges uses the least memory, with a mean of 27.4 MegaBytes. Then the three functions Canny Edge Detector, FeatureJ Edges and Log_Filter have an average of about 50 MB. And finally the most memory expensive function is FeatureJ Laplacian with a mean of 55 MB.

The data are not normaly distributed.

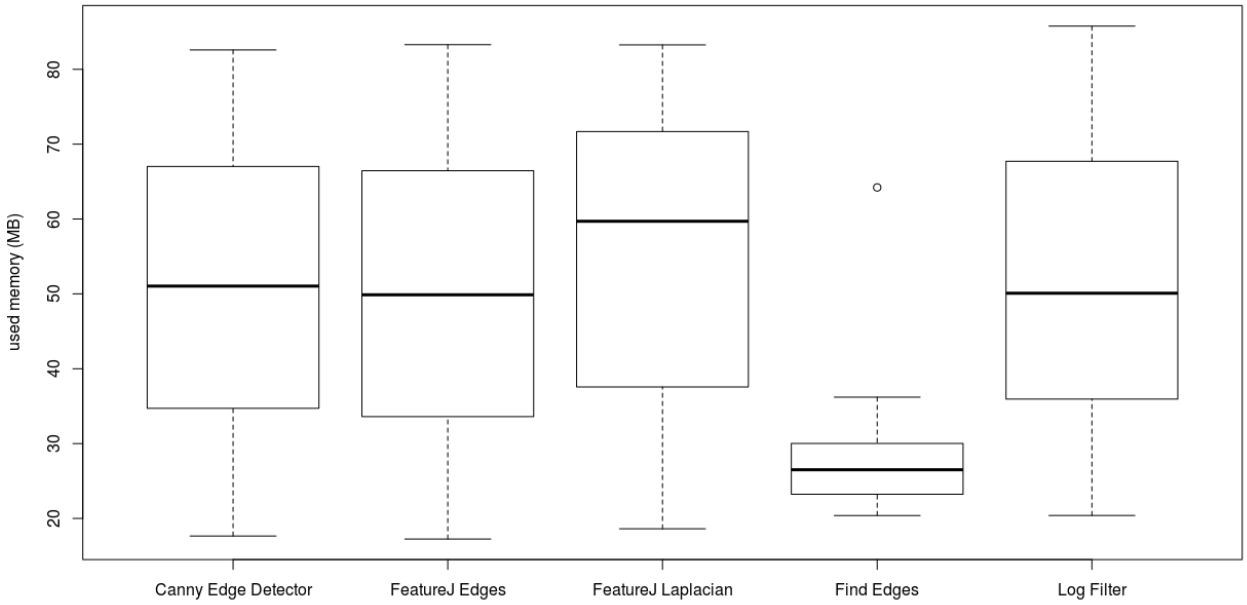

Fig.18: Result of the benchmark for the memory load of ImageJ edge detection functions. The Canny Edge Detector and FeatureJ Edges plugin use the Canny algorithm, FeatureJ Laplacian and Log Filter use the LoG, and Find Edges the Sobel

The data is not normaly distributed. Visually, the Find Edges algorithm have different results from the others for the time and the memory consuption. A Kruskal-Wallis test between its result and the results of the other algorithms shows a significant difference for both factors with the p-values associated being of less than 2e-06 and 2e-16 for respectively the memory and time consumption.

For the plugins using a Laplacian approach, FeatureJ Laplacian and Log_Filter, the means obtained for the time and the memory consumption are close to each other. A Kruskal-Wallis test between them shows that they are significantly different for the time consuption, with a p-value of 7.385e-12 but not for the memory load, which generate a p-value of 0,2282.

In the same way, graphically, the difference between the results for the two Canny implementations, Canny Edge Detector and FeatureJ Edges, is difficult to assess for the memory consuption. Another Kruskal-Wallis test shows a significant difference between the mean of execution time, with a p-value of 2,2e-16, but not for the memory used, with a p-value of 0,4667.

Function	Canny Edge Detector	FeatureJ Edges	FeatureJ Laplacian	Find Edges	Log Filter
Avg time (ms)	205.8	41.8	35.4	0.88	17.6
Avg memory (MB)	50.6	50.0	55.0	27.4	51.6

Fig.19: Average execution time and used memory for ImageJ edge detection functions. The Canny Edge Detector and FeatureJ Edges plugin use the Canny algorithm, FeatureJ Laplacian and Log Filter use the LoG, and Find Edges the Sobel

Given that the Canny Edge Detector plugin can also be used on RGB images, we also ran a benchmark comparing the execution time and memory load difference for this plugin on RGB and 8-bit images[Fig.20].

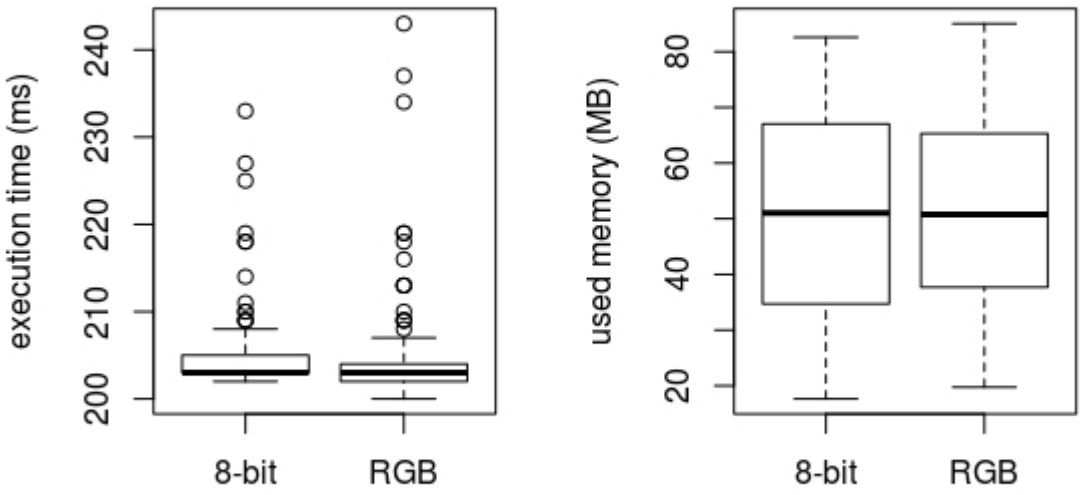

Fig.20: Result of the benchmark of Canny Edge Detector plugin on 8-bit and RGB images, for both execution time and memory load

The mean obtained by the benchmark for the JVM memory load and the time consumption are the same

for the RGB and the 8-bits picture with respectively 51 MB and 205 ms, but the distribution of the data is not spread in the same way between the two sides of the median for the execution time. The data does not follow the Normal Distribution, and the Kruskal-Wallis test shows a significant difference only for the time consumption, with a p-value of 1e-04 against 0,49.

Discussion

Qualitative Comparison

As can be seen in the above examples [Fig.8-15], the quality of the edge detection process depends a lot on which algorithm is used, and with what parameters. The outputs of all functions show us that the strong contours, corresponding to the hat, the face and the hair are well identified, while the contours of the details, like the hat's feathers are often not detected (at least not with the range of parameter values used here).

For the Sobel algorithm, the edges are outlined correctly, though it also outlines noise (shadows, changes in color) as edges.

The Laplacian based algorithms, on the other hand, give an output with a lower amount of misidentified noise if configured properly, although they do not identify all of the edges. As can be seen in the difference of output between the Log_filter plugin[Fig.11 Image 5-6], which detects most edges without noise in the result, but misses out on edges which were blurred in the original image and the FeatureJ implementation [Fig.9 Image 3], that creates an output image that identifies edges everywhere in the image, leading to an output image that is hard to compare with the original image. This error on the part of the FeatureJ implementation is most likely due the absence of thresholding, leading to the identification of very small variations in pixel values as indicative of an edge, whereas they are probably due to noise or color and light variations.

We must note that comparing these two results is difficult because the parameters that the user can choose for the computation are not the same, except for the gaussian standard deviation. The results optimization is also different : while the FeatureJ plugin stops at the detection of the zero-crossings, the Log_Filter uses an additional DoG filtering as a threshold on the LoG output.

The two Canny implementations[Fig.13] give a better result than both the Sobel implementation, because they are less sensible to noise, and the LoG implementation, because they detect the real edges more accurately. They are not perfect, and miss edges where the pixel values do not vary sharply on each side, create edges in the presence of differences due to lighting, and create disrupted edges. All in all, the 2 Canny implementations give nearly identical results, though the FeatureJ implementation detects more continuous edges than the Canny Edge Detector. However Canny Edge Detector can process RGB images contrarly to the other plugins.

Performance Comparison

As can be seen in our benchmark results[Fig.17-19], the Sobel algorithm is the fastest Edge Detection implementation in ImageJ, followed by the Laplacian implementations and then the FeatureJ Edge implementation of the Canny algorithm. The Canny Edge Detector implementation is up to three times slower than it's FeatureJ implementation, making it the slowest of all the functions.

As far as memory load goes, most of the algorithms use an average of around 50 MB, with the FeatureJ Laplacian being slightly more voracious (55MB), and the Find Edges (Sobel) algorithm using only 27MB

of memory on average. Sobel being the simplest algorithm, we expected it would also be the least memory intensive, since it only uses first derivatives to determine edges, while the other alogrithms have to compute second derivatives.

The two Laplacian implementations are difficult to compare, as they do not offer the same customizable parameteres, which might skew the results. The Canny algorithms, though, do offer the same customizable parameters, and can therefore be compared without the danger that a changed parameter is the cause of the differences observed, instead of the implementation itself.

The enormous difference in speed between the Canny Edge Detector and the FeatureJ Edges plugin can only be explained by the choices made by their developpers during the implementation for ImageJ, as well as the optimizations they added to the original algorithm. Canny Edge Detector is slower than the other plugin but it is also the only one able to process RGB images. Moreover,if we compare the speed and memory load of running the Canny Edge Detector on an RGB image versus on an 8-bit version of that same image, we see no differences in the means for the speed or memory load. The significant difference shown by the statistical test seems to come from the outlier values[Fig.20].

An RGB image being more complex because of its 3 channels, we would have expected higher processing time and memory load when running Canny Edge Detector plugin on the RGB version of our image. This can be explained by the fact that one of the first steps of this function is a conversion of the image to an 8-bit format, meaning that all the folowing steps are the same for all types of images.

Conclusion

Edge detection stays one of the most important steps in image processing in various domains (use of Canny, LoG and Sobel in electron microscopy images to identify neuron structure[50], use of the Canny algorithm in the identification of weapons on CCTV camera pictures[51], use of edge detection method in order to identify organs images[52]). In the biological field, it allows to give a biological meaning to a picture, define structures which will be analysed thereafter. The two main approaches consist to use either the gradient or the derivative of the greylevel intensity, and it has lead to the development of several algoritms more than twenty years ago[53] [54] [55] [56] still reviewed and updated in order to reduce their sensibility to the different noises and keep

[50]Zhu F, Liu Q, Fu Y, Shen B. Segmentation of Neuronal Structures Using SARSA (lambda)-Based Boundary Amendment with Reinforced Gradient-Descent Curve Shape Fitting. PLoS One, 9(3):1–19, 2014.

[51]Grega M, Matiolanski A, Leszczuk M. Automated Detection of Firearms and Knives in a CCTV Image. Sensors 2016, 16, 47; doi:10.3390/s16010047.

[52]Jalalian A, Mashohor S, Mahmud R, Karasfi B, Saripan MIB, Ramli ARB. Foundation and Methodologies in computer-aided diagnosis systems for breast cancer detection. EXCLI Journal, 16:113-137, 2017.

[53]Sobel I. An isotropic 3× 3 image gradient operator, presentation at Stanford Artificial Intelligence Project (SAIL).

[54]Prewitt JM. Object enhancement and extraction. Picture processing and Psychopictorics. 1970 Jan 1;10(1):15-9.

[55]Kirsch RA. Computer determination of the constituent structure of biological images. Computers and biomedical research. 1971 Jun 1;4(3):315-28.

[56]Canny J. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence. 1986 Nov(6):679-98.

the specificity of each approch[57] [58] [59] [60]. Using theses algorithms, several tools adapted for a specific targuet are developed[61].

Of all the functions in ImageJ, the fastest and least memory intensive is the Sobel implementation Find Edges, though the function giving the best results (least amount of false positives and false negatives) is the FeatureJ implementation of the Canny Edge Detection algorithm. We can't dismiss the Canny Edge Detector implementation entirely, though, as it is the only one studied capable of processing RGB images. The study of these five examples of edge detection functions showed that there is not a unique ideal choice of algorithm for edge detection. Each of them implements a specific approach of edge detection, offering to the user a customized analysis according to its needs. The Sobel one provides a very fast and memory effective processing, the Laplacian approach from FeatureJ allows the obtention of closed contours structures, and the Canny approaches define the most significant edges, with a specificity for the Canny Edge Detector function able to process RGB pictures.

Our conclusion is that the choice of the algorithm have to take into account the time and memory limitations of the users, as well as the type of images they are working with.

[57] Ding L, Goshtasby A. On the Canny edge detector. Pattern Recognition. 2001 Mar 31;34(3):721-5.

[58] Treloas KK, Simpson MJ, Kabla AJ. Sensitivity of Edge Detection Methods for Quantifying Cell Migration Assays. PLoS One, 8(6):e67389, 2013.

[59] Haq I, Anwar S, Shah K, Khan MT, Shah SA. Fuzzy Logic Based Edge Detection in Smooth and Noisy Clinical Images. PLoS One. 10(9):e0138712, 2015.

[60] Luo S, Yang J, Gao Q, Zhou S, Zhan CA. The Edge Detectors Suitable for Retinal OCT Image Segmentation. Journal of Healthcare Engineering 2017; 2017: 3978410.

[61] Choudhry P. High-Throughput Method for Automated Colony and Cell Counting by Digital Image Analysis Based on Edge Detection. PLoS One. 2016; 11(2): e0148469.