Edge Detection Algorithms in OpenCV

Daniel Silva - 51908 | João Cravo - 63784

Resumo **- Este artigo pretende ilustrar e comparar o funcionamento de diferentes algoritmos de detecção de arestas, ilustrando o efeito dos diferentes algoritmos e parâmetros escolhidos.**

Abstract **- This project consists in illustrate and compare the *modus operandi* of some edge detecting algorithms, in order to illustrate the effect of different chosen algorithms and parameters.**

I. Introduction

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real time computing vision.

It was originally developed by Intel research center in Nizhny Novgorod (Russia), later supported by Willow Garage and now maintained by Itseez.

In this project the user can chose one image, at a time, to be modified by the algorithms. Only need to write the name on console page. After that, many images will be displayed on the screen. Finally, in the console will appear information about the image, depending on the algorithm.

II. User Interface

A. Console

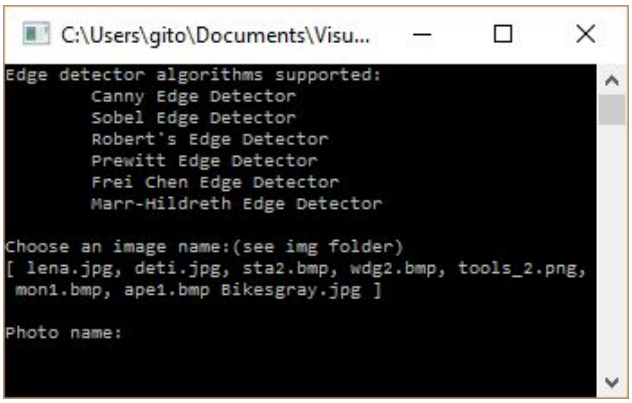

Fig. 1 - Console.

When the user open the application, will appear a console where you can chose a image to be modified.

The user just need to write the name of the image. For example:

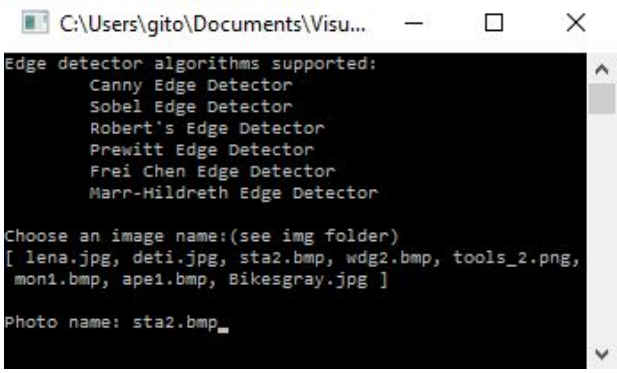

Fig. 2 - Image Chose.

First, the original image will appear.

Fig. 3 - Original Image.

For each algorithm, a image will be showed.
In this paper we will see each image modified in the various algorithms used.
As long as the images appear, the information will be printed on console.

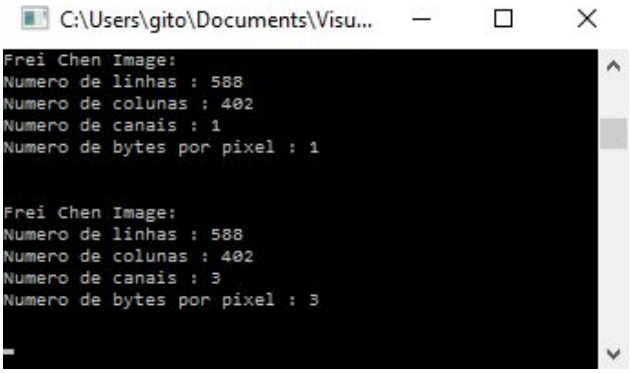

Fig. 4 - Information.

III. Algorithms

A. Canny Edge Detector

The Canny Edge detector was developed by John F. Canny in 1986. Also known to many as the optimal detector, Canny algorithm aims to satisfy three main criteria:

- **Low error rate:** Meaning a good detection of only existent edges.
- **Good localization:** The distance between edge pixels detected and real edge pixels have to be minimized.
- **Minimal response:** Only one detector response per edge.

The steps for apply this algorithm are:

1. Filter out any noise. The Gaussian filter is used for this purpose.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2. Find the intensity gradient of the image.
   a. Apply a pair of convolution masks (in X and Y directions:
   b. Find the gradient strength and direction with:

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

3. Non-maximum suppression is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.
4. Hysteresis: The final step. Canny does use two thresholds (upper and lower).

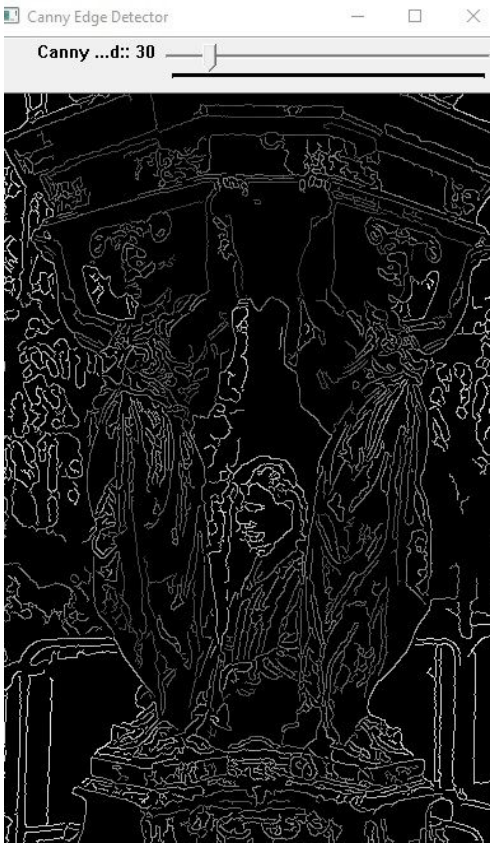

Fig. 5 - Canny Image.

In this case, and in the Marr-Hildreth, the user can chose the threshold value.
This is useful for change some parameters that will change the image as well.

B. Sobel Edge Detector (Sobel Derivatives)

This is how we calculate gradients and use them to detect edges.

Its possible to deduce that a method to detect edges in an image can be performed by locating pixel locations where the gradient is higher than its neighbors (or to generalize, higher than a threshold).

1. We calculate two derivatives:
    a. **Horizontal changes:** This is computed by convolving **I** with a kernel **Gx** with odd size. For example for a kernel size of 3, **Gx** would be computed as:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

    b. **Vertical changes:** This is computed by convolving **I** with a kernel **Gy** with odd size. For example for a kernel size of 3, **Gy** would be computed as:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

2. At each point of the image we calculate an approximation of the gradient in that point by combining both results above:

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

Fig. 6 - Sobel Edge Image.

C. Robert's Edge Detector

The Roberts operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. It thus highlights regions of high spatial frequency which often correspond to edges. In its most common usage, the input to the operator is a gray-scale image, as is the output. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point.

In theory, the operator consists of a pair of 2×2 convolution kernels as shown in Figure 1. One kernel is simply the other rotated by 90°. This is very similar to the Sobel operator.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

These kernels are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these **Gx** and **Gy**). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

Although typically, an approximate magnitude is computed using:

$$G = abs(Gx) + abs(Gy)$$

The angle of orientation of the edge giving rise to the spatial gradient (relative to the pixel grid orientation) is given by:

$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Fig. 7 - Robert's Image.

D. Prewitt's Edge Detector

The operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how "abruptly" or "smoothly" the image changes at that point, and therefore how likely it is that part of the image represents an edge, as well as how that edge is likely to be oriented. The Prewitt kernels are given by:

$$h_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad h_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

How do we combine the results of convolution with these two kernels to give a single measure of the presence of an edge?

The two gradients computed at each pixel (**Gx** and **Gy**) by convolving with above two kernels can be regarded as the x and y components of gradient vector. This vector is oriented along the direction of change, normal to the direction in which the edge runs. Gradient magnitude is given by:

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

Although typically, an approximate magnitude is computed using:

$$G = abs(Gx) + abs(Gy)$$

Gradient direction is given by:

$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

1. Loop through every pixel in the Laplacian of the smoothed image and look for sign changes. If there is a sign change and the slope across this sign change is greater than some threshold, mark this pixel as an edge. Alternatively, you can run these changes in slope through a hysteresis (described in the canny edge detector) rather than using a simple threshold.

Fig. 8 - Prewitt's Image.

E. Frei-Chen Edge Detector

The Frei and Chen module is used to detected edges with an image. The Frei and Chen edge detection technique is less sensitive to noise than other edge operators.

The final pixel value is the ratio between the multiplication of four kernels with the surrounding pixel neighborhood and the square sum of all pixels. The following are the kernels used to multiply the pixel neighborhood:

+2	+3	+4
0	0	0
-2	-3	-2

+2	0	-2
0	-2	+3
+3	-2	0

+3	0	-3
+2	0	-2
-2	0	+2

+2	0	-2
-3	+2	0
0	+2	-3

When we are using the Frei-Chen masks for edge detection we are searching for the cosine defined above and we use the masks as the elements of importance.

After this step, we just need to compute the gradients just like on Robert's or Prewitt's algorithms.

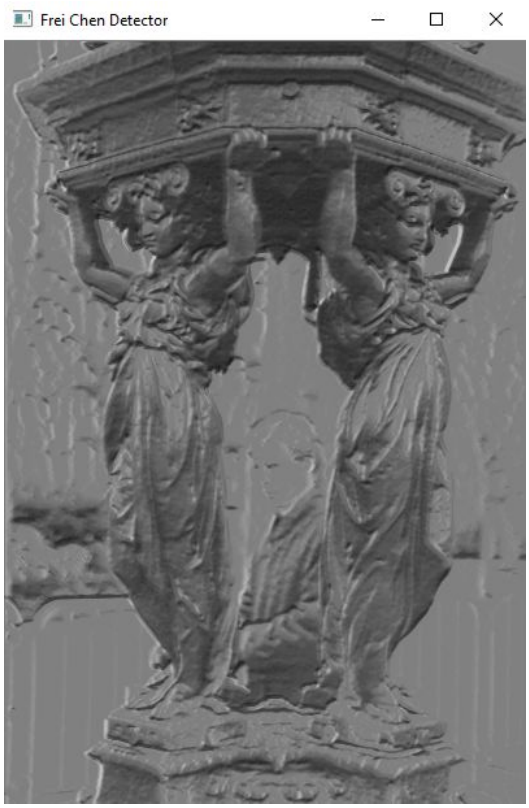

Fig. 9 - Frei-Chen's Image.

F. Marr-Hildreth Edge Detector

The Marr-Hildreth edge detector was a very popular edge operator before Canny released his paper. It is a gradient based operator which uses the Laplacian to take the second derivative of an image. The idea is that if there is a step difference in the intensity of the image, it will be represented by in the second derivative by a zero crossing.

So the general algorithm for the Marr-Hildreth edge detector is as follows:

1. Smooth the image using a Gaussian. This smoothing reduces the amount of error found due to noise.
2. Apply a two dimensional Laplacian to the image. This Laplacian will be rotation invariant and is often called the "Mexican Hat operator" because of its shape: This operation is the equivalent of taking the second derivative of the image.

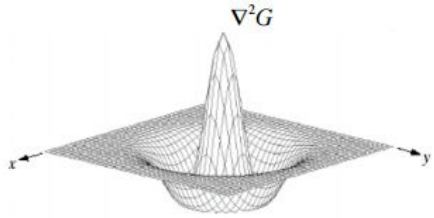

Fig. 10 Marr-Hildreth's Image

IV. Conclusion

This project gave us the opportunity to learn more about OpenCV, which for us was something interesting, despite our difficulties in the development of it.

After this project, we have the opinion that OpenCV is a great tool to edit images.

We used 6 different algorithms which gave us a good feedback about the potentiality of this tool..

To conclude, we are pleased with the result. It was a productive and instructive project for us.

References

[1] Repository
https://github.com/danifss/cv2015-OpenCV

Algorithms:
http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_Chapter5.pdf

[2] Canny Edge Detector
http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

[3] Sobel Edge Detector (Sobel Derivatives)
http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html
http://opencvexamples.blogspot.com/2013/10/sobel-edge-detection.html

[4] Robert's Edge Detector
http://homepages.inf.ed.ac.uk/rbf/HIPR2/roberts.htm

[5] Prewitt's Edge Detector
http://www.programming-techniques.com/2013/03/sobel-and-prewitt-edge-detector-in-c.html

[6] Frei-Chen Edge Detector
http://www.roborealm.com/help/Frei_Chen.php
http://rastergrid.com/blog/2011/01/frei-chen-edge-detector/

[7] Marr-Hildreth Edge Detector
http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/laplace_operator/laplace_operator.html
http://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html