Visual Computing and Multimedia

Abel J. P. Gomes

LAB. 3

EDGE DETECTORS

Based on: www.uweb.ucsb.edu/~shahnam/**AfED.doc**

1. Objectives
2. Edge Detection: an Introduction
3. Edge Detection Techniques
4. Exercises
5. Homework

Departamento de Informática
Universidade da Beira Interior
Portugal
2011

Lab. 3

IMAGE SEGMENTATION

1. Objectives

1.1. General Objectives

In general terms, the idea of this labwork is to learn the fundamental image segmentation techniques and algorithms, in particular those related to edge detectors:

- Gradient-based detectors (e.g., Sobel, Prewitt, etc).
- Laplacian-based detectors (e.g., LoG, Canny's, etc).

2. Edge Detection: An Introduction

Edge detection refers to the process of identifying and locating sharp discontinuities in an image. The discontinuities are abrupt changes in pixel intensity. Such discontinuities characterize boundaries of objects in a scene. Classical methods of edge detection involve convolving the image with an operator (a 2D filter), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions. There is a series of edge detection operators, each designed to be sensitive to certain types of edges. Choosing an edge detection operator involves various factors, namely:

- **Edge orientation:** The geometry of the operator determines a characteristic direction in which it is most sensitive to edges. Operators can be optimized to look for horizontal, vertical, or diagonal edges.
- **Noise environment:** Edge detection is difficult in noisy images, since both the noise and the edges contain high-frequency content. Attempts to reduce the noise result in blurred and distorted edges. Operators used on noisy images are typically larger in scope, so they can average enough data to discount localized noisy pixels. This results in a less accurate localization of the detected edges.
- **Edge structure:** Not all edges involve a step change in intensity. Effects such as refraction or poor focus can result in objects with boundaries defined by a gradual change in intensity. The operator needs to be chosen to be responsive to such a gradual change in those cases. Newer wavelet-based techniques actually characterize the nature of the transition for each edge in order to distinguish, for example, edges associated with hair from edges associated with a face.

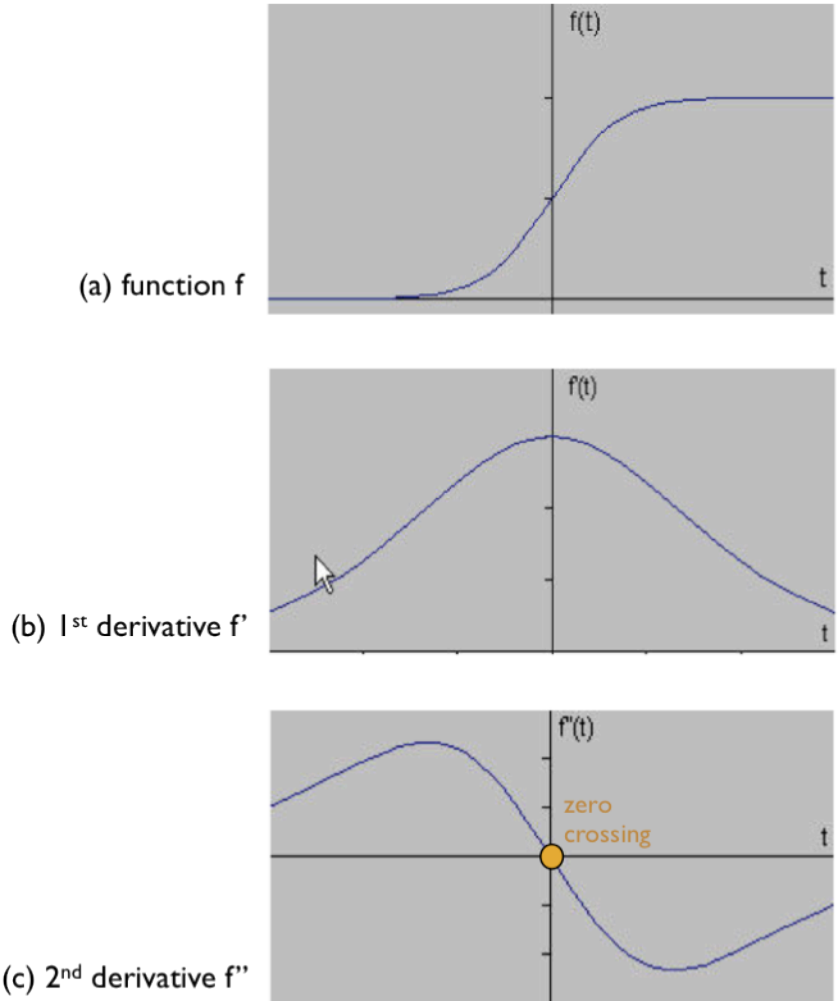
There are many ways to perform edge detection. However, the majority of different methods may be grouped into two categories:

- **Gradient:** The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image.

– **Laplacian:** The Laplacian method searches for zero crossings in the second derivative of the image to find edges. An edge has the 1-D shape of a ramp and calculating the derivative of the image can highlight its location.

Suppose we have the signal shown in Fig. 1(a), with an edge shown by the jump in intensity function f. If we take the gradient of this signal (which, in one dimension, is just the first derivative with respect to t) we get the curve of the derivative f' shown in Fig. 1(b). Clearly, the derivative shows a maximum located at the center of the edge in the original signal.

This method of locating an edge is characteristic of the "gradient filter" family of edge detection filters and includes the Sobel method. A pixel location is declared an edge location if the value of the gradient exceeds some threshold. As mentioned before, edges will have higher pixel intensity values than those surrounding it. So once a threshold is set, you can compare the gradient value to the threshold value and detect an edge whenever the threshold is exceeded. Furthermore, when the first derivative is at a maximum, the second derivative is zero. As a result, another alternative to finding the location of an edge is to locate the zeros in the second derivative. This method is known as the Laplacian and the second derivative of the signal is shown in Fig.1(c).



(a) function f

(b) 1st derivative f'

(c) 2nd derivative f''

Fig. 1: An intensity signal described by a function f and its first and second derivatives.

3. Edge Detection Techniques

3.1. Sobel operator

The operator consists of a pair of 3×3 convolution kernels as shown in Fig. 2. One kernel is simply the other rotated by 90°.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

(a) G_X (b) G_Y

Fig. 2: (a) x-direction gradient kernel; (b)y-direction gradient kernel.

These kernels are designed to respond maximally to edges running *vertically* and *horizontally* relative 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 G_X and G_Y). 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:

$$|G| = \sqrt{G_X^2 + G_Y^2}$$

but, usually, the following approximate magnitude

$$|G| = |G_X| + |G_Y|$$

is calculated instead because it is much faster to compute.

The angle of orientation of an edge (relative to the pixel grid) is given by:

$$\theta = \tan^{-1}\left(\frac{G_Y}{G_X}\right)$$

3.2. Roberts cross operator

The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point.

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

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

(a) G_X (b) G_Y

Fig. 3: (a) x-direction gradient kernel; (b)y-direction gradient kernel.

These kernels 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 G_X and G_Y). 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 also given by:

$$|G| = \sqrt{G_X^2 + G_Y^2}$$

although typically, an approximate magnitude is computed using:

$$|G| = |G_X| + |G_Y|$$

which is much faster to compute.

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

$$\theta = \tan^{-1}\left(\frac{G_Y}{G_X}\right) - \frac{3\pi}{4}$$

3.3. Laplacian of Gaussian (LoG)

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian Smoothing filter in order to reduce its sensitivity to noise. The operator normally takes a single gray-level image as input and produces another gray-level image as output.

The Laplacian L(x,y) of an image with pixel intensity values f(x,y) is given by:

$$L(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Three commonly used small kernels are shown in Fig. 4.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

Fig. 4: Three commonly used discrete approximations to the Laplacian filter.

These kernels are very sensitive to noise because they approximate the second derivative. To overcome this noise problem, the image is often Gaussian-smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

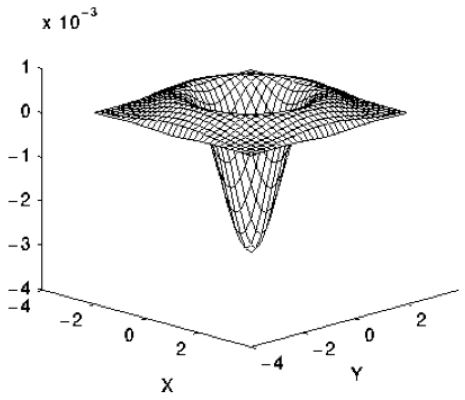
In fact, since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter first of all, and then convolve this hybrid filter with the image to achieve the required result. Doing so in this manner has two advantages:

- Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations.
- The LoG (Laplacian of Gaussian) kernel can be pre-calculated in advance so only one convolution needs to be performed on the image in runtime.

The 2-D LoG function centered at zero and with Gaussian standard deviation σ has the form:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

and is shown in Fig. 5.



$$\begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -24 & -40 & -24 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \end{bmatrix}$$

Fig. 5: (left) LoG function; (right) discrete approximation to LoG function with Gaussian deviation σ=1.4.

Note that as the Gaussian is made increasingly narrow, the LoG kernel becomes the same as the simple Laplacian kernels shown in Fig. 4. This is because smoothing with a very narrow Gaussian (σ< 0.5 pixels) on a discrete grid has no effect. Hence on a discrete grid, the simple Laplacian can be seen as a limiting case of the LoG for narrow Gaussians.

Web interaction:

You can interactively experiment with the Laplacian operator by clicking on:

http://homepages.inf.ed.ac.uk/rbf/HIPR2/laplaciandemo.htm

You can interactively experiment with the LoG operator by clicking on:

http://homepages.inf.ed.ac.uk/rbf/HIPR2/logdemo.htm

3.4. Canny's Edge Detector Algorithm

The Canny edge detection algorithm is known as the optimal edge detector. Canny's intentions were to enhance the many edge detectors already out at the time he started his work. He was very successful in achieving his goal and his ideas and methods can be found in his paper, "*A Computational Approach to Edge Detection*". In his paper, he followed a list of criteria to improve current methods of edge detection.

The first and most obvious criterion is low error rate. It is important that edges occurring in images should not be missed and that there be NO responses to non-edges. The second criterion is that the edge points be well localized. In other words, the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum. A third criterion is to have only one response to a single edge. This was implemented because the first two criteria were not enough to completely eliminate the possibility of multiple responses to an edge.

Based on these three criteria, the Canny edge detector first smoothes the image to eliminate and noise. It then finds the image gradient to highlight regions with high spatial derivatives. The algorithm then tracks along these regions and suppresses any pixel that is not at the maximum (non-maximum suppression). The gradient array is now further reduced by hysteresis. Hysteresis is used to track along the remaining pixels that have not been suppressed. Hysteresis uses two thresholds and if the magnitude is below the first threshold, it is set to zero (made a non-edge). If the magnitude is above the high threshold, it is made an edge. And if the magnitude is between the two thresholds, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above the high threshold.

STEP 1

In order to implement the Canny edge detector algorithm, a series of steps must be followed. The first step is to filter out any noise in the original image before trying to locate and detect any edges, what is done using a Gaussian filter. Once a suitable mask has been found, the Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The larger the width of

the Gaussian mask, the lower is the detector's sensitivity to noise. The localization error in the detected edges also increases slightly as the Gaussian width is increased. A Gaussian mask is shown in Fig. 6.

$$\frac{1}{115}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Fig. 6: Discrete approximation to Gaussian function with σ=1.4.

STEP 2

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows), as shown in Fig. 2. The magnitude (or edge strength) of the gradient is approximated by

$$|G| = |G_X| + |G_Y|$$

STEP 3

The direction of the edge is computed using the gradient in the x and y directions. However, an error will be generated when sumX is equal to zero. So in the code there has to be a restriction set whenever this takes place. Whenever the gradient in the x direction is equal to zero, the edge direction has to be equal to 90 degrees or 0 degrees, depending on what the value of the gradient in the y-direction is equal to. If GY has a value of zero, the edge direction will equal 0 degrees. Otherwise the edge direction will equal 90 degrees. The formula for finding the edge direction is just:

$$\theta = \tan^{-1}\left(\frac{G_Y}{G_X}\right)$$

STEP 4

Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced in an image. So if the pixels of a 5x5 image are aligned as follows:

x x x x x
x x x x x
x x a x x
x x x x x
x x x x x

Fig. 7: Example of a 5x5 image.

Then, it can be seen by looking at pixel "a", there are only four possible directions when describing the surrounding pixels - **0 degrees** (in the horizontal direction), **45 degrees** (along the positive diagonal), **90 degrees** (in the vertical direction), or **135 degrees** (along the negative diagonal). So now the edge orientation has to be resolved into one of these four directions depending on which direction it is closest to (e.g. if the orientation angle is found to be 3 degrees, make it zero degrees). Think of this as taking a semicircle and dividing it into 5 regions.

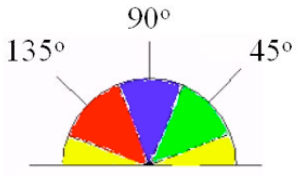


Fig. 8: The four directions centered at a given pixel of an image array.

Therefore, any edge direction falling within the **yellow range** (0 to 22.5 & 157.5 to 180 degrees) is set to 0 degrees. Any edge direction falling in the **green range** (22.5 to 67.5 degrees) is set to 45 degrees. Any edge direction falling in the **blue range** (67.5 to 112.5 degrees) is set to 90 degrees. And finally, any edge direction falling within the **red range** (112.5 to 157.5 degrees) is set to 135 degrees.

STEP 5

After the edge directions are known, non-maximum suppression now has to be applied. Non-maximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (sets it equal to 0) that is not considered to be an edge. This will give a thin line in the output image.

STEP 6

Finally, hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold. If a single threshold, T is applied to an image, and an edge has an average strength equal to T, then due to noise, there will be instances where the edge dips below the threshold. Equally it will also extend above the threshold making an edge look like a dashed line.

To avoid this, hysteresis uses 2 thresholds, a high T_H and a low T_L. Any pixel in the image that has a value greater than T_L is presumed to be an edge pixel, and is marked as such immediately. Then, any pixels that are connected to this edge pixel and that have a value greater than T_H are also selected as edge pixels. If you think of following an edge, you need a gradient of T_H to start but you don't stop till you hit a gradient below T_L.

3.5. References

RC **Gonzalez** and RE **Woods**, Prentice-Hall, 2002

http://zone.ni.com/devzone/conceptd.nsf/webmain/37BF3B76F646EF5286256802007BA00F

http://www.pages.drexel.edu/~weg22/edge.html

http://people.westminstercollege.edu/students/djb0331/CMPT300/Project/ProjectDiscussion.htm

http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

http://www.redbrick.dcu.ie/~bolsh/thesis/node30.html

http://www.pages.drexel.edu/~weg22/can_tut.html

http://www.personal.psu.edu/users/b/r/brv106/cse485/project2.htm

http://www.altera.com/literature/cp/gspx/edge-detection.pdf

4. Exercises

1) Implement the gradient edge detector.
2) Implement the Roberts cross edge detector.
3) Implement the Sobel edge detector.
4) Implement the LoG detector.
5) Implement the Canny edge detector.

5. Homework

Gaussian smoothing (http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm):

1. Starting from the Gaussian noise (mean 0, σ=13) corrupted image compute both mean filter and Gaussian filter smoothing at various scales, and compare each in terms of noise removal versus loss of detail.
2. At how many standard deviations from the mean does a Gaussian fall to 5% of its peak value? On the basis of this suggest a suitable square kernel size for a Gaussian filter with $\sigma=s$.
3. How does the time taken to smooth with a Gaussian filter compare with the time taken to smooth with a mean filter *for a kernel of the same size?* Notice that in both cases the convolution can be speeded up considerably by exploiting certain features of the kernel.

LOG filter (http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm):

1. Try the effect of LoG filters using different width Gaussians on an image of your choice. What is the general effect of increasing the Gaussian width? Notice particularly the effect on features of different sizes and thicknesses.
2. Construct a LoG filter where the kernel size is much too small for the chosen Gaussian width (*i.e.* the LoG becomes truncated). What is the effect on the output? In particular what do you notice about the LoG output in different regions each of uniform but different intensities?
3. Devise a rule to determine how big an LoG kernel should be made in relation to the of the underlying Gaussian if severe truncation is to be avoided.
4. If you were asked to construct an edge detector that simply looked for peaks (both positive and negative) in the output from a LoG filter, what would such a detector produce as output from a single step edge?

Canny edge detector (http://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm):

1. Adjust the parameters of the Canny operator so that you can detect the edges of an image while removing *all* of the noise.
2. What effect does increasing the Gaussian kernel size have on the magnitudes of the gradient maxima at edges? What change does this imply has to be made to the tracker thresholds when the kernel size is increased?
3. It is sometimes easier to evaluate edge detector performance after thresholding the edge detector output at some low gray scale value so that all detected edges are marked by bright white pixels. Try this out on the third and fourth example images of the clown mentioned above. Comment on the differences between the two images.
4. How does the Canny operator compare with the Roberts Cross and Sobel edge detectors in terms of speed? What do you think is the slowest stage of the process?
5. How does the Canny operator compare in terms of noise rejection and edge detection with other operators such as the Roberts Cross and Sobel operators?
6. How does the Canny operator compare with other edge detectors on simple artificial 2-D scenes? And on more complicated natural scenes?
7. Under what situations might you choose to use the Canny operator rather than the Roberts Cross or Sobel operators? In what situations would you definitely not choose it?