INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
ROUEN NORMANDIE

Mehdi ABOUZAID
Damien TOOMEY

–

À l'attention de
M. Bensrhair

Canny-Deriche
Détecteur de Contours
Projet TIM

TIM - ASI - 2018-2019

Table des matières

1 Introduction

L'intérêt majeur de la détection de contours est la réduction d'information. En effet, l'information d'une image peut être résumée par les contours des différents objets qu'elle contient car les contours contiennent les parties les plus informatives de l'image.

Pour certaines applications (reconnaissance de formes, imagerie médicale, cartographie,...), les contours de l'image suffisent pour l'exploiter. La détection de contours permet ainsi d'éviter de stocker des informations lourdes inutilement.

Dans ce rapport, nous allons nous intéresser au détecteur de contours de Canny-Deriche.

Le filtre de Deriche est un opérateur de détection de contours développé par Rachid Deriche en 1987. C'est un algorithme permettant la détection de contours dans une image à deux dimensions. Cet algorithme est une évolution du filtre de Canny développé par John Canny en 1986.

2 Description du filtre

2.1 Etapes de la détection de contours

Comme le détecteur de contours de Canny, celui de Deriche s'effectue en quatre étapes :

1 Gradients et lissage (réduction du bruit)
2 Norme du gradient et direction du gradient
3 Suppression des non-maxima locaux
4 Seuillage par hystérésis

2.2 Critères du filtre

Dans le filtre de Deriche, on retrouve les mêmes critères de détection optimale des contours du filtre de Canny :

A Bonne détection : faible taux d'erreur dans la signalisation des contours. Il ne faut manquer aucun contour, ni en rajouter (faux positifs)
B Bonne localisation : les points détectés comme contours doivent être aussi proches que possible des points de contours réels
C Clarté de la réponse : un point du contour ne doit être détecté qu'une seule fois par le filtre. Ce critère est inclus implicitement dans le premier critère puisque si l'on détecte deux contours là où il n'y en a qu'un, une des deux réponses doit être considérée comme fausse.

C'est pourquoi ce filtre est communément appelé opérateur de Canny-Deriche.

2.3 Améliorations du filtre de Canny

La principale différence se trouve dans l'implémentation des deux premières étapes de l'algorithme (**1**,**2**). Contrairement au détecteur de contours de Canny, celui de Deriche utilise un filtre RII [1] et non RIF [2].

A chaque critère (**A**,**B**,**C**) est associé une formule mathématique. La maximisation de ces trois critères conduit à la résolution de l'équation différentielle (1) dont la solution est l'opérateur f.

Equation différentielle

$$2 \cdot f(x) - 2 \cdot \lambda_1 \cdot f''(x) + 2 \cdot \lambda_2 \cdot f''''(x) + \lambda_3 = 0 \qquad (1)$$

1. Réponse Impulsionnelle Infinie
2. Réponse Impulsionnelle Finie

2.3.1 Canny : filtre RIF

Conditions initiales

$$f(0) = 0 \quad f(W) = 0 \quad f'(0) = S \quad f'(W) = 0$$

Solution générale sur $[-W, W]$

$$f(x) = a_1 \cdot e^{\alpha \cdot x} \cdot sin(\omega \cdot x) + a_2 \cdot e^{\alpha \cdot x} \cdot cos(\omega \cdot x) + a_3 \cdot e^{-\alpha \cdot x} \cdot sin(\omega \cdot x) + a_4 \cdot e^{-\alpha \cdot x} \cdot cos(\omega \cdot x) + C$$

2.3.2 Deriche : filtre RII

Conditions initiales

$$f(0) = 0 \quad f(+\infty) = 0 \quad f'(0) = S \quad f'(+\infty) = 0$$

Solution générale sur $\mathbb{R}$

$$f(x) = -c \cdot e^{-\alpha \cdot |x|} \cdot sin(\omega \cdot x)$$

L'avantage du filtre RII de Deriche est qu'il peut être adapté aux caractéristiques de l'image en modifiant seulement deux paramètres $(\alpha, \omega) \in \mathbb{R}^+$, sans impacter le temps d'exécution.

- Si la valeur de α est petite (généralement entre 0.25 et 0.5), la détection est meilleure.
- Si la valeur de α est grande (entre 2 et 3), la localisation des contours est meilleure.

C'est pourquoi, dans la plupart des cas, il est recommandé de fixer la valeur de α à 1.

3 Modélisation mathématique

3.1 Cas mono-dimensionnel

L'article présente deux détecteurs de contours optimaux :

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

L'expression de $g(x)$ est une forme simplifiée de l'expression de $f(x)$ quand ω tend vers 0. En effet, quand $\omega \cdot x$ est petit, $sin(\omega \cdot x) \approx \omega \cdot x$.

Dans cet article, seule l'implémentation récursive de l'opérateur $f(x)$ est proposée. Nous nous focaliserons donc uniquement sur l'opérateur $f(x)$. L'auteur dit que l'implémentation de l'opérateur $g(x)$ sera proposée dans un autre article que nous n'étudions pas dans le cadre de ce projet.

Par ailleurs, les résultats de détection de contours sont améliorés en utilisant $g(x)$ au lieu de $f(x)$. De plus, avec $g(x)$, le filtre ne dépend que de α et non plus de ω.

3.1.1 Fonction de lissage $h(n)$

$h(n)$ correspond à l'intégrale $h(x)$ de $f(x)$ avec $n \in \mathbb{Z},\ x \in \mathbb{R}$

$$h(n) = (c_1 \cdot sin(\omega \cdot |n|) + c_2 \cdot cos(\omega \cdot |n|)) \cdot e^{-\alpha \cdot |n|}$$

3.1.2 Coefficients du filtre

La forme du filtre est déterminée par $(\alpha, \omega) \in \mathbb{R}^+$. On choisit ces deux paramètres en fonction de nos données. Des petites valeurs de α donnent des filtres larges.

$$\begin{cases} c = \frac{1-2\cdot e^{-\alpha}\cdot cos(\omega)+e^{-2\cdot\alpha}}{e^{-\alpha}\cdot sin(\omega)} \\ c_1 = \frac{k\cdot\alpha}{\alpha^2+\omega^2} \\ c_2 = \frac{k\cdot\omega}{\alpha^2+\omega^2} \end{cases} \qquad \begin{cases} b_1 = -2 \cdot e^{-\alpha} \cdot cos(\omega) \\ b_2 = e^{-2\cdot\alpha} \end{cases} \qquad \begin{cases} a = -c \cdot e^{-\alpha} \cdot sin(\omega) \\ a_0 = c_2 \\ a_1 = (-c_2 \cdot cos(\omega) + c_1 \cdot sin(\omega)) \cdot e^{-\alpha} \\ a_2 = a_1 - c_2 \cdot b_1 \\ a_3 = -c_2 \cdot b_2 \end{cases}$$

$$k = \frac{(1 - 2 \cdot e^{-\alpha} \cdot cos(\omega) + e^{-2\cdot\alpha}) \cdot (\alpha^2 + \omega^2)}{2 \cdot \alpha \cdot e^{-\alpha} \cdot sin(\omega) + \omega - \omega \cdot e^{-2\cdot\alpha}}$$

3.1.3 Implémentation récursive

Dérivation (gradient en 1D)

Le signal de sortie y, ayant comme signal d'entrée x et comme réponse impulsionnelle f, peut être obtenu de manière récursive :

$$y^+(m) = x(m-1) - b_1 \cdot y^+(m-1) - b_2 \cdot y^+(m-2) \quad m = 1, ..., M$$

$$y^-(m) = x(m+1) - b_1 \cdot y^-(m+1) - b_2 \cdot y^-(m+2) \quad m = M, ..., 1$$

$$y(m) = a \cdot (y^+(m) - y^-(m)) \quad m = 1, ..., M$$

Lissage

Le signal de sortie y, ayant comme signal d'entrée x et comme réponse impulsionnelle h, peut également être obtenu de manière récursive :

$$y^+(m) = a_0 \cdot x(m) + a_1 \cdot x(m-1) - b_1 \cdot y^+(m-1) - b_2 \cdot y^+(m-2) \quad m = 1, ..., M$$

$$y^-(m) = a_2 \cdot x(m+1) + a_3 \cdot x(m+2) - b_1 \cdot y^-(m+1) - b_2 \cdot y^-(m+2) \quad m = M, ..., 1$$

$$y(m) = y^+(m) + y^-(m) \quad m = 1, ..., M$$

Cette implémentation récursive du détecteur de contour permet d'éviter de faire des convolutions, ce qui augmente l'efficacité de l'algorithme.

3.2 Cas bi-dimensionnel

$$f(x,y) = -\frac{c \cdot k}{\alpha^2 + \omega^2} \cdot e^{-\alpha \cdot (|x|+|y|)} \cdot [sin(\omega \cdot x) \cdot (\alpha \cdot sin(\omega \cdot |y|) + \omega \cdot cos(\omega \cdot |y|)) + sin(\omega \cdot y) \cdot (\alpha \cdot sin(\omega \cdot |x|) + \omega \cdot cos(\omega \cdot |x|))]$$

$$= \underbrace{f(x) \cdot h(y)}_{X} + \underbrace{h(x) \cdot f(y)}_{Y} \qquad \text{avec } (x,y) \in \mathbb{R}^2$$

Le filtre est donc séparable en deux masques X et Y.

3.2.1 Masques X et Y

$$X(m,n) = f(m) \cdot h(n) = \frac{(-c \cdot e^{-\alpha \cdot |m|} \cdot sin(\omega \cdot m)) \cdot (k \cdot (\alpha \cdot sin(\omega \cdot |n|) + \omega \cdot cos(\omega \cdot |n|)) \cdot e^{-\alpha \cdot |n|})}{\alpha^2 + \omega^2}$$

$$Y(m,n) = h(m) \cdot f(n) = \frac{(k \cdot (\alpha \cdot sin(\omega \cdot |m|) + \omega \cdot cos(\omega \cdot |m|)) \cdot e^{-\alpha \cdot |m|}) \cdot (-c \cdot e^{-\alpha \cdot |n|} \cdot sin(\omega \cdot n))}{\alpha^2 + \omega^2}$$

3.2.2 Implémentation récursive (sans convolution)

Pour éviter la convolution, on applique l'algorithme récursif suivant sur l'image originale, ce qui permet d'obtenir le gradient selon x.

Gradient

$$Y^+(i,j) = I(i,j-1) - b_1 \cdot Y^+(i,j-1) - b_2 \cdot Y^+(i,j-2) \quad j = 1,...,NCL \quad i = 1,...,NLG$$

$$Y^-(i,j) = I(i,j+1) - b_1 \cdot Y^-(i,j+1) - b_2 \cdot Y^-(i,j+2) \quad j = NCL,...,1 \quad i = 1,...,NLG$$

$$S(i,j) = a \cdot (Y^+(i,j) - Y^-(i,j)) \quad j = 1,...,NCL \quad i = 1,...,NLG$$

Lissage

$$R^+(i,j) = a_0 \cdot S(i,j) + a_1 \cdot S(i-1,j) - b_1 \cdot R^+(i-1,j) - b_2 \cdot R^+(i-2,j)$$
$$i = 1,...,NLG \quad j = 1,...,NCL$$

$$R^-(i,j) = a_2 \cdot S(i+1,j) + a_3 \cdot S(i+2,j) - b_1 \cdot R^-(i+1,j) - b_2 \cdot R^-(i+2,j)$$
$$i = NLG,...,1 \quad j = 1,...,NCL$$

$$R(i,j) = R^-(i,j) + R^+(i,j) \quad i = 1,...,NLG \quad j = 1,...,NCL$$

En appliquant l'algorithme ci-dessus (gradient et lissage) sur la transposée de l'image originale, on obtient la transposée du gradient selon y.

3.2.3 Implémentation directe (avec convolutions)

Soit I l'image de taille NLG x NCL.

Convoluer le masque X avec l'image I correspond à une convolution de l'image avec l'opérateur $f(n)$ dans la direction **horizontale** (donne le gradient selon x) suivi d'une convolution avec l'opérateur $h(n)$ dans la direction **verticale** (lissage).

Convoluer le masque Y avec l'image I revient à une convolution de l'image avec l'opérateur $f(n)$ dans la direction **verticale** (donne le gradient selon y) suivi d'une convolution avec l'opérateur $h(n)$ dans la direction **horizontale** (lissage). (cela revient à appliquer le masque X sur la transposée de l'image originale).

4 Explication de l'implémentation (cas 2D)

4.1 Etape 1/4 : Gradients selon x et y et lissage

Comme l'implémentation récursive proposée dans l'article est moins proche de la modélisation mathématique, nous avons décidé d'implémenter l'algorithme récursif et l'algorithme direct qui utilise des convolutions.

Pour l'algorithme récursif (sans convolution), nous nous sommes basé sur ce qui a été expliqué à la section **3.2.2**

Pour l'algorithme direct (avec convolutions), nous nous sommes basé sur ce qui a été expliqué à la section **3.2.3**

4.2 Etape 2/4 : Norme du gradient et direction du gradient

4.2.1 Norme du gradient

$$A(m,n) = \sqrt{r(m,n)^2 + s(m,n)^2)}$$

$$\text{avec} \begin{cases} r \text{ gradient selon x} \\ s \text{ gradient selon y} \end{cases}$$

4.2.2 Direction du gradient

$$arg = arctan(\frac{s(m,n)}{r(m,n)})$$

Dans le programme que nous avons écrit, on calcule l'arc tangente entre r et s avec une fonction du langage qui retourne une valeur en degré entre -180° et 180°, ce qui correspond à l'arc tangente dans les quatre quadrants du plan.

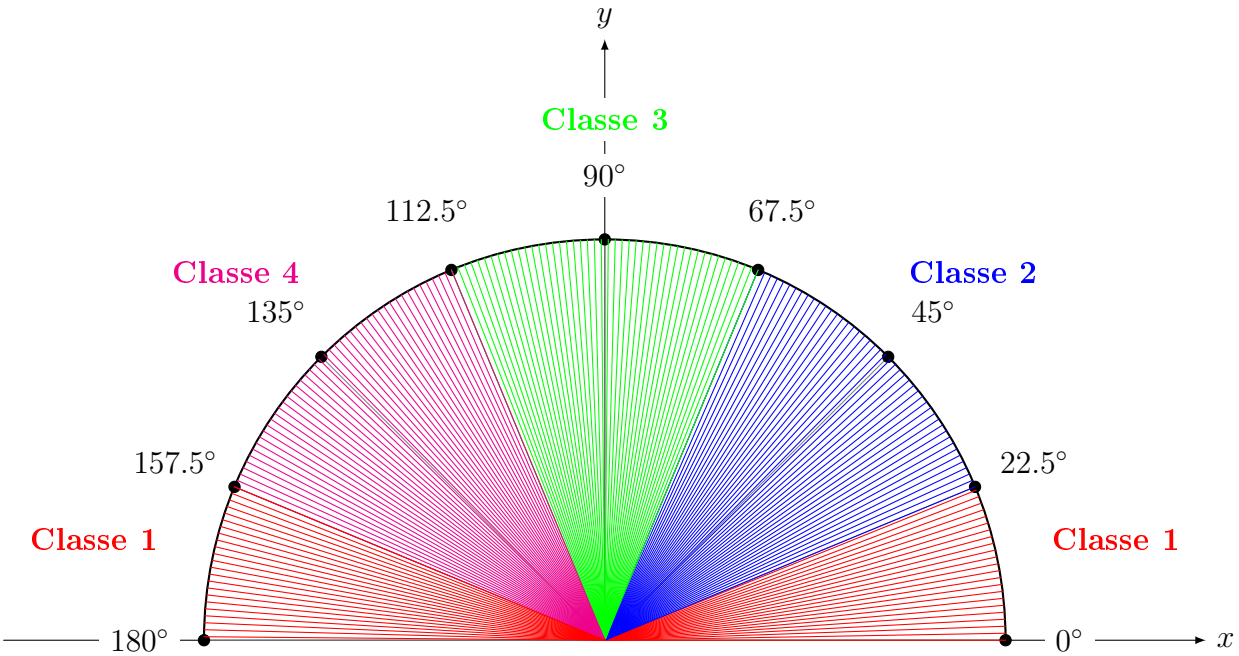
4.3 Etape 3/4 : Suppression des non-maxima locaux

La suppression des non-maxima locaux est une méthode appliquée sur la norme du gradient qui permet d'affiner les contours, c'est-à-dire mettre à 0 les pixels qui ne constituent pas un contour. Un pixel de la norme du gradient est considéré comme contour s'il forme un maximum local dans la direction du gradient.

Soit arg la direction du gradient en degré avec $arg \in [-180; 180]$. On ramène toutes les valeurs entre 0 ° et 180 ° en ajoutant 180 ° aux valeurs négatives de arg. Pour la suppression de non maxima, on ramène l'intervalle $[0; 180]$ à quatre classes : 1, 2, 3 et 4.

On choisit les règles suivantes :

- si $arg \in [0; 22.5[$ ou $arg \in [157.5; 180]$ alors $arg \in$ **Classe 1**
- si $arg \in [22.5; 67.5[$ alors $arg \in$ **Classe 2**
- si $arg \in [67.5; 112.5[$ alors $arg \in$ **Classe 3**
- si $arg \in [112.5; 157.5[$ alors $arg \in$ **Classe 4**



On traite ensuite les cas suivants :

Classe 1 (le contour est dans la direction Nord – Sud) : le pixel est un contour si la norme du gradient en ce point est supérieure au maximum de la norme du gradient des pixels Est et Ouest

Classe 2 (le contour est dans la direction Nord-Ouest – Sud-Est) : le pixel est un contour si la norme du gradient en ce point est supérieure au maximum de la norme du gradient des pixels Nord-Est et Sud-Ouest

Classe 3 (le contour est dans la direction Est – Ouest) : le pixel est un contour si la norme du

gradient en ce point est supérieure au maximum de la norme du gradient des pixels Nord et Sud

Classe 4 (le contour est dans la direction Nord-Est – Sud-Ouest) : le pixel est un contour si la norme du gradient en ce point est supérieure au maximum de la norme du gradient des pixels Nord-Ouest et Sud-Est

Après cette étape, l'image n'est pas encore binarisée mais on a mis à 0 les pixels qui ne constituaient pas un contour.

4.4 Etape 4/4 : Seuillage par hystérésis

Le seuillage par hystérésis permet de retirer les faux contours.

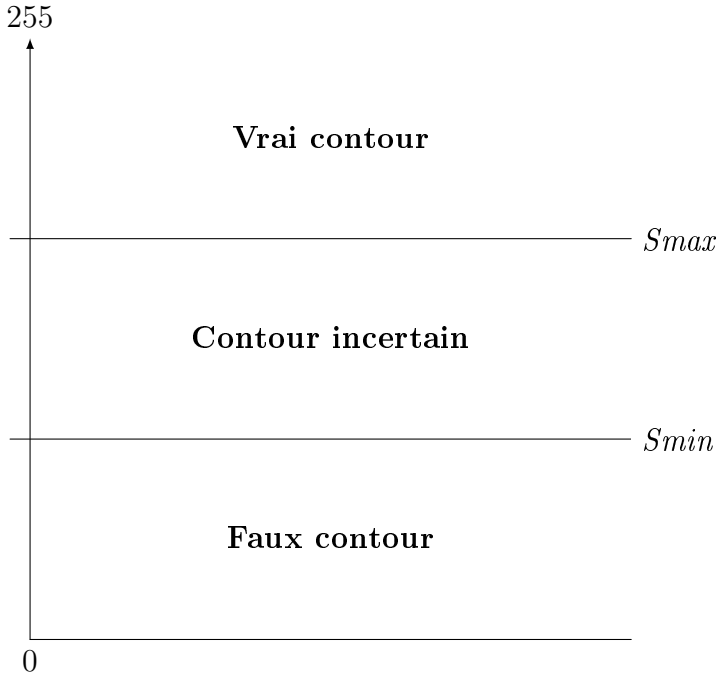
Pour cela, on définit deux seuils :

- *Smin* : seuil minimum
- *Smax* : seuil maximum

Pour chaque pixel de la norme du gradient obtenu après la suppression des non-maxima locaux, on regarde sa valeur.

Cas 1 : si la valeur du pixel est supérieure au seuil maximum alors ce pixel est un vrai contour donc on met sa valeur à 1
Cas 2 : si la valeur du pixel est inférieure au seuil minimum alors ce pixel est un faux contour donc on met sa valeur à 0
Cas 3 : si la valeur du pixel est entre le seuil minimum et le seuil maximum alors on regarde les valeurs des huit pixels voisins et on regarde s'il existe au moins un pixel voisin qui a une valeur supérieure au seuil maximum. Si c'est le cas, la valeur du pixel initial devient un 1 sinon 0

Pour le **Cas 3**, dans le code, on prend une fenêtre 3x3 pour obtenir facilement les voisins du pixel au centre de cette fenêtre.



L'image obtenue est maintenant binarisée. Elle contient les contours de l'image initiale et un peu de bruit.

5 Bibliographie

- **Article**

Titre Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector
Auteur RACHID DERICHE

- Deriche edge detector
  https://en.wikipedia.org/wiki/Deriche_edge_detector
- Can't get clean output in my MATLAB implementation of Canny-Deriche
  https://stackoverflow.com/questions/14132356/cant-get-clean-output-in-my-matlab-in
- Canny edge detector
  https://en.wikipedia.org/wiki/Canny_edge_detector
- Canny Edge Detection Step by step
  https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html
- Détection de contours : les opérateurs de Canny-Deriche
  https://perso.esiee.fr/~coupriem/Algo/algoima.html
- Extraction de contours et son extension du contour actif
  http://ninebill.free.fr/ExtractionContours/detection/canny.html
- JAVA demo of Canny/Deriche-like filter (EPFL Biomedical Imaging Group)
  http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/
- Segmentator – Fix gramag export error with deriche filter (3D python)
  https://github.com/ofgulban/segmentator
- Edge Detection with MATLAB
  https://fr.mathworks.com/videos/edge-detection-with-matlab-119353.html
- Gaël Deest (pdf)
  www.theses.fr/2017REN1S102/abes
- Canny en python (OpenCV)
  https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html
- Convolution in Two Dimensions
  http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/space/convol.htm
- Convolution en 2D
  http://web.stanford.edu/group/sequoia/cgi-bin/node/185
- EECS 442 – Computer vision
  http://vhosts.eecs.umich.edu/vision//teaching/EECS442_2012/lectures/lecture14.pdf
- DETECTION DE CONTOURS

http://www.lirmm.fr/~strauss/PageImage3/EdgeDetection.pdf

- Techniques d'extraction de contours
  ftp://ftp-sop.inria.fr/athena/Team/Rachid.Deriche/Lectures/Master-Stic-IGMMV/techniques_contours.pdf
- Détection de contours
  http://devernay.free.fr/cours/vision/pdf/c3.pdf
- Bases du traitement des images - Détection de contours
  http://webia.lip6.fr/~thomen/Teaching/BIMA/cours/contours.pdf
- La détection decontours dans des images à niveaux de gris : mise en œuvre et sélection de détecteurs
  http://docnum.univ-lorraine.fr/public/INPL_T_1991_ZIOU_D.pdf
- Detection de contours
  http://dept-info.labri.fr/~achille/enseignement/TI/TI-ch4-notes.pdf
- Segmentation d'image : Contours
  http://www.lgi2p.mines-ales.fr/~montesin/CoursPDF/segmentation_contours.pdf
- Filtre de Deriche
  http://www.tsi.enst.fr/pages/enseignement/ressources/mti/Shen_ou_Deriche/node4.html