Détection de contour dans une image (***)

Professeur responsable : M. Pazat - nicolas.pazat@gmail.com

1 Présentation

L'objectif est de programmer un algorithme, appelé filtre de Canny, permettant de détecter automatiquement les contours dans une image. Plus précisément, à partir d'une image en niveaux de gris, l'algorithme renvoie une image correspondante en noir ou blanc faisant apparaître les contours de l'image initiale, comme sur l'exemple suivant (cette image n'a pas été choisie comme appât pour inciter certains à prendre ce sujet, elle a longtemps servi de référence en traitement des images, domaine très sérieux contrairement aux apparences. . .).

Image initiale

Image de contours

L'extraction d'une image de contours n'est généralement qu'une première étape dans une chaîne de traitement allant jusqu'à la reconnaissance automatique d'objets dans l'image grâce à des techniques d'intelligence artificielle. De nombreuses applications pratiques existent, notamment en imagerie médicale pour l'interprétation automatique d'images radio par exemple.

La consultation des sites suivants (et d'autres) permettra d'enrichir la compréhension des outils présentés dans la suite du document :

http://fr.wikipedia.org/wiki/Détection_de_contours
http://ninebill.free.fr/ExtractionContours/
http://edgestracing.aurelie-fruitiere.fr/canny.html
http://www.optique-ingenieur.org/fr/cours/OPI_fr_M04_C05/co/Grain_OPI_fr_M04_C05.html

2 D'un fichier image à un tableau NumPy et réciproquement

Pour lire sous Python une image **en noir et blanc** stockée dans un fichier **au format png** (Portable Network Graphics) nous allons utiliser le module **matplotlib.pyplot** qui va nous permettre de stocker cette image sous la forme d'un tableau bidimensionnel du module **numpy** dont chaque case contient un flottant compris entre 0 et 1 mesurant le niveau de gris du pixel correspondant de l'image, le 0 correspondant au noir absolu et le 1 au blanc absolu. La syntaxe est la suivante, le tableau étant finalement stocké dans la variable **Im**.

```
import numpy as np
import matplotlib.pyplot as plt

temp=plt.imread('nom_fichier_image.png')
Im=np.zeros(np.shape(temp)[0:2])
Im=temp[:,:,0]
```

A partir du tableau **Im**, on peut afficher l'image correspondante dans une fenêtre graphique par la commande :

```
plt.imshow(Im,cmap='gray')
```

On peut enfin sauvegarder l'image contenue dans le tableau **Im** sous la forme d'un fichier **png** par la commande :

```
plt.imsave('nom_fichier_image.png',Im,cmap='gray')
```

3 Filtre de Canny

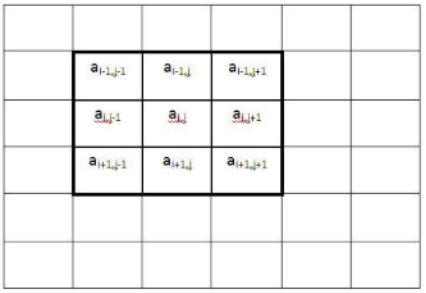
3.1 Masque et convolution

Les deux premières étapes du filtrage de Canny utilisent la notion de convolution, que nous allons expliquer.

Soit A un tableau, dont la case d'indice de ligne i et d'indice de colonne j est notée $a_{i,j}$. Soit M un autre tableau, carré de taille impaire, inférieure à celle de A. On appelle convolution du tableau A par le masque M le tableau C, de même taille que A, et dont le contenu de chaque case est obtenu en centrant le masque M sur la case correspondante du tableau A et en calculant la somme des éléments de A encadrés par M, pondérée par les coefficients correspondants du masque M.

2	-1	3
-4	5	1
0	2	4

Exemple de masque M



Convolution du tableau A par le masque M en position i, j

Par exemple, pour le masque M de taille 3 de la figure précédente, la case d'indice de ligne i et d'indice de colonne j du tableau C vaudra :

$$c_{i,j} = 2a_{i-1,j-1} - a_{i-1,j} + 3a_{i-1,j+1} - 4a_{i,j-1} + 5a_{i,j} + a_{i,j+1} + 2a_{i+1,j} + 4a_{i+1,j+1}.$$

Pour les cases du tableau C situées « trop sur le bord », pour lesquelles le masque « déborde » du tableau A (pour un masque de taille 3, c'est le cas uniquement des cases du bord) on pose simplement :

$$c_{i,j} = a_{i,j}.$$

Nous allons maintenant décrire une à une les différentes étapes du filtrage de Canny.

3.2 Réduction du « bruit »

Certaines imperfections des capteurs utilisés dans les appareils photographiques génèrent l'apparition de valeurs faussées pour certains pixels de l'image, suffisamment peu nombreux pour être pratiquement imperceptibles à l'oeil, mais qui provoquent la détection de « faux contours » lors des étapes suivantes du filtrage.

Pour atténuer ce « bruit » contenu dans l'image, on va effectuer la convolution de celle-ci par un masque, dit Gaussien, défini par la formule :

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

où (x, y) désigne les coordonnées d'un pixel du masque définies par rapport au pixel central, et σ un paramètre du filtre gaussien appelé écart-type.

Ci-dessous, un masque gaussien de taille 3 pour un écart-type $\sigma = 0.8$.

x=-1 y=-1	x=-1 y=0	x=-1 y=1
x=0 y=-1	x=0 y=0	x=0 y=1
x=1 y=-1	x=1 y=0	x=1 y=1

Coordonnées

0.05	0.1	0.05
0.1	0.25	0.1
0.05	0.1	0.05

Valeurs

L'effet de la convolution de l'image par un masque gaussien est de « lisser » les variations trop brutales d'intensité lumineuse, et ce d'autant plus que l'écart-type choisi est grand (à condition de prendre la taille du masque suffisamment grande aussi, sans quoi l'image est globalement assombrie). En contre-partie, l'image devient moins nette (effet de « flouté ») ce qui contribue à atténuer les contours que nous souhaitons justement déceler. Un compromis doit donc être trouvé en pratique entre réduction du bruit et conservation des contours.


Image initiale


taille 3 et $\sigma = 0.8$


taille 21 et $\sigma = 2$

3.3 Construction d'une première image de contour : « l'image de gradient »

Le gradient est une notion jouant pour les fonction des deux variables une rôle similaire à la dérivée pour les fonction d'une seule variable, avec la différence qu'il s'agit d'un vecteur, dont la norme quantifie la vitesse de variation de la fonction, mais dont la direction donne en outre la direction dans laquelle cette variation est maximale.

Un tableau de pixel peut être assimilé à une fonction de deux variables (l'abscisse et l'ordonnée du pixel, la valeur prise par la fonction étant le niveau de gris du pixel). Pour déterminer les contours dans l'image, on va estimer la norme du gradient en chaque point. Le contour d'un objet est généralement caractérisé par une forte variation du niveau de gris (passage du « fond » à l'objet). Or un pixel de l'« image de gradient » sera d'autant plus clair qu'elle correspond à une zone de modification rapide de la luminosité dans l'image initiale, et au contraire d'autant plus sombre que cette image initiale est homogène localement autour de ce point. L'étape suivante pour construire l'image de contour sera donc de ne conserver dans cette image de gradient que les points de forte valeur du gradient, mis à blanc, les autres points étant mis à noir.

Pour estimer les coordonnées en x et en y du vecteur gradient en chaque point de l'image filtrée, obtenue à la première étape, on construit la convolution de ce tableau avec les masques Mx (pour la composante suivant x du gradient) et My (pour la composante suivant y), dits filtres de Sobel, définis par :

-1	-2	-1
0	0	0
1	2	1

Mx

-1	0	1
-2	0	2
-1	0	1

My

Les pixels situés sur le bord de l'image, qui n'ont pas été transformés par la convolution par ces masques de taille 3, seront placés à 0 afin qu'ils ne provoquent pas la détection de faux contours.

Si on note Gx et Gy les tableaux obtenus par convolution de l'image filtrée avec Mx et avec My respectivement, on définit alors l'image de gradient comme étant celle définie par le tableau G suivant :

$$G = \sqrt{Gx^2 + Gy^2}.$$

(Les notations Gx^2 et Gy^2 désignent ici les tableaux dont les éléments sont les carrés de ceux de Gx et Gy respectivement, idem pour la racine carrée.)

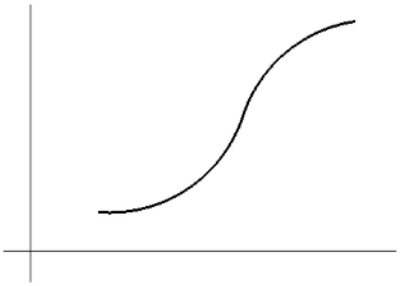
Voici l'image de gradient obtenue à partir de notre image initiale :


Image initiale

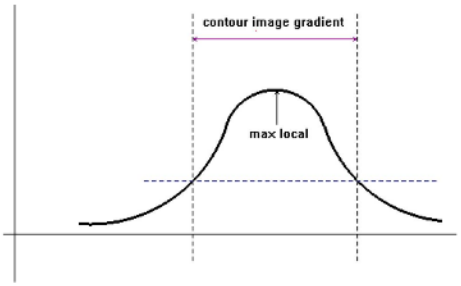

Image de gradient

3.4 Affinage de l'image de gradient par recherche des maximums locaux

L'image de gradient obtenue à l'étape précédente a le défaut de faire apparaître des bordures assez épaisses, dues en particulier à l'effet du filtre gaussien de la première étape qui certes a réduit le bruit dans l'image, mais au prix d'un certain étalement des contours. Pour affiner ces contours apparaissant dans l'image de gradient, nous allons utiliser que le gradient est un vecteur dont la direction donne celle de la plus forte variation locale de l'intensité lumineuse, et ne conserver que les valeurs correspondant à des maximums locaux dans la direction du gradient.

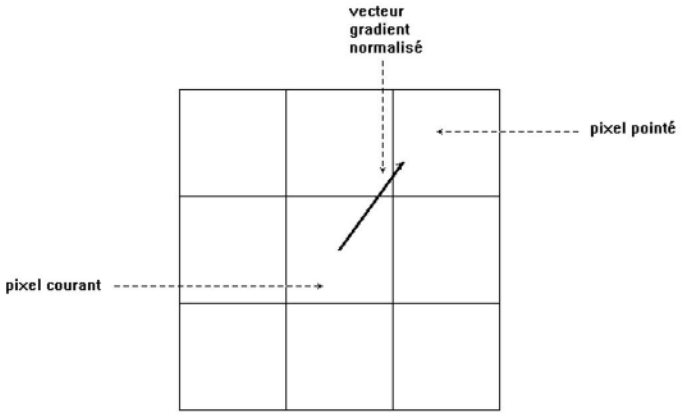


Évolution de l'intensité lumineuse



Évolution de la norme du gradient

Pour cela, nous reprenons les coordonnées Gx et Gy du gradient calculées à l'étape précédente, et déterminons parmi les 8 pixels voisins d'un pixel donné (non sur le bord de l'image), lequel est « pointé » par le vecteur gradient normalisé (ie. le vecteur gradient divisé par sa norme, pour conserver la direction et le sens, mais de norme 1). Si la valeur de la norme du gradient G en ce pixel pointé est strictement supérieure à celle du pixel central, alors ce dernier n'est pas un extremum local, et sa valeur dans G est mise à 0 pour l'éliminer du contour.



Voici le résultat obtenu pour notre image initiale :

Image de gradient

Image des maximums locaux du gradient

3.5 Seuillage

Nous abordons la dernière étape du filtre de Canny, qui consiste à transformer l'image des maximums locaux du gradient, qui est en niveaux de gris, en une image de contours binaire où tous les pixels seront noir (0) sauf ceux du contours qui seront blancs (1). Pour cela, on utilise un procédé appelé seuillage par hystérésis.

Dans un premier temps, on définit un premier seuil $s_1 \in [0, 1]$ dit seuil haut, et un deuxième seuil $s_2 \in [0, s_1[$ dit seuil bas. Le principe du seuillage par hystérésis est alors le suivant. Tous les pixels, de l'image des maximums locaux du

gradient, ayant une valeur supérieure à s_1 sont placés à blanc (1), et tous ceux ayant une valeur inférieure à s_2 sont placés à noir (0). Puis on parcourt un à un chaque pixel de valeur comprise entre de s_2 et s_1 ; s'il possède parmi ses huit voisins immédiats au moins un pixel déjà placé à blanc, alors on place aussi ce pixel à blanc. Sinon, on laisse (pour l'instant) le pixel à son niveau de gis. On répète alors ce traitement jusqu'à ce qu'aucun des pixels de valeur comprise entre s_2 et s_1 restants n'ait pour voisin un pixel blanc. Ces pixels restants sont alors mis à noir (0). Notez que chaque pixel pourra être examiné plusieurs fois, puisque l'état de ses voisins peut varier au cours de l'algorithme.

On peut comprendre le principe de l'algorithme ainsi : on déclare que pour les pixel de niveau supérieur à s_1, on est sûr qu'ils font partie du contours, et que pour ceux de niveau inférieur à s_2 on est sûr qu'ils n'en font pas partie. Pour les autres, on ne les acceptent comme faisant partie du contour que s'ils « prolongent » un pixel ayant déjà été déclaré comme faisant partie du contour, ce prolongement se répétant de proche en proche.

Voici finalement le résultat obtenu sur notre exemple avec $s_1 = 0.15$ et $s_2 = 0.08$:

4 Travail demandé

Écrire un programme Python mettant en oeuvre les différentes étapes du filtre de Canny décrit dans les pages précédente. A chaque étape, le programme lira le fichier de l'image stocké sur l'ordinateur à l'étape précédente, lui appliquera le traitement correspondant à cette étape, puis sauvegardera le fichier résultat dans la mémoire de l'ordinateur. Certaine étapes nécessitent la saisie par l'utilisateur de paramètres (taille et écart-type pour la réduction du bruit, seuils haut et bas pour le seuillage) : à l'issue de ces traitements, le programme demandera à l'utilisateur s'il souhaite recommencer le traitement avec d'autres valeurs des paramètres (après avoir consulté l'image obtenue en mémoire de l'ordinateur) ou bien s'il valide ce choix et passe à l'étape suivante.

Vous ferez fonctionner votre programme sur plusieurs images, téléchargées sur le web ou de votre propre création. Observer en particulier l'effet sur vos images d'une modification des différents paramètres (taille et écart-type du filtre gaussien, seuil haut et bas du seuillage par hystérésis).

5 Contenu du rapport

L'organisation et la présentation du rapport sont laissés à la libre appréciation des étudiants, mais il devra contenir :

- un **raffinage** du programme en français, permettant en particulier de dégager plusieurs **fonctions**, et en justifiant la pertinence des choix faits, en particulier quand plusieurs options auraient été possibles ;
- pour chacune de ces fonctions, donner sa **spécification** précise (nature des entrées et sens de la valeur de sortie), un éventuel sous-raffinage en français si nécessaire, et **un jeu de valeurs sur lesquels cette fonction a été testée** (avec les résultats correspondants) ;
- des **exemples** d'images obtenues à chaque étape du traitement ;
- une **analyse** des difficultés éventuellement rencontrées lors de ce projet, et des tentatives mises en oeuvre pour les résoudre.

Le code complet du programme, commenté sans excès mais de manière pertinente, sera en outre envoyé avec le rapport.