Traitement d'Images
TP N°1 : Filtrage et détection de contours

Master 1
Ghiles Mostafaoui

1 Introduction

1.1 Objectifs du TP

L'objectif de ce premier TP est de vous permettre de faire vos premiers pas en traitement d'images. A travers une application sur la détection de contours, vous appréhenderez les notions suivantes :

- la manipulation des images en niveau de gris
- la manipulation des images couleur de type RGB
- le produit de convolution
- le filtrage passe bas
- le calcul du gradient
- le seuillage
- et enfin la détection de contour

1.2 La bibliothèque NRC(Numerical Recipes in C)

Nous utiliserons dans le cadre des travaux pratiques de ce module la bibliothèque Numerical Recipes (NRC) qui contient un ensemble de fonctions qui permettent :

- de gérer les entrées/sorties : lire et écrire des images de type PGM ou PPM (voir le fichier nrio.c)
- de gérer les allocations mémoire : allouer en mémoire des matrices de différents types et de différentes taille (voir le fichier nralloc.c)
- de réaliser des calculs vectoriels et matriciels (voir le fichier nrarith.c)

Pour utiliser cette bibliothèque il suffit d'inclure dans votre fichier ".c" les différents Headers : "def.h", "nrio.h", "nralloc.h" et "nrarith.h".
Nous utiliseront principalement les fonctions suivantes :

- **byte ** LoadPGM_bmatrix(char *filename, long *nrl, long *nrh, long *ncl, long *nch)** : Lecture d'une image de type PGM
- **rgb8 ** LoadPPM_rgb8matrix(char *filename, long *nrl, long *nrh, long *ncl, long *nch)** : Lecture d'une image de type PPM
- **void SavePGM_bmatrix(byte **m, long nrl, long nrh, long ncl, long nch, char *filename)** : Ecriture d'une image de type PGM
- **void SavePPM_rgb8matrix(rgb8 **m, long nrl, long nrh, long ncl, long nch, char *filename)** : Ecriture d'une image de type PPM
- **byte** bmatrix(long nrl, long nrh, long ncl, long nch)** : Allocation d'une matrice de "byte"
- **rbg8** rgb8matrix(long nrl, long nrh, long ncl, long nch)** : Allocation d'une matrice de "rgb8"
- **int** imatrix(long nrl, long nrh, long ncl, long nch)** : Allocation d'une matrice de "int"

- **void free_bmatrix(byte **m, long nrl, long nrh, long ncl, long nch)** : libération de la mémoire pour une matrice de type "byte"
- **vois free_rgb8matrix(rgb8 **m, long nrl, long nrh, long ncl, long nch)** : libération de la mémoire pour une matrice de type "rgb8"
- **void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch)** : libération de la mémoire pour une matrice de type "int"

2 Prise en main : Lecture et écriture d'une image de type PGM

Ecrivez votre premier programme que vous nommerez tp1.c qui lit l'image "cubesx3.pgm" qui vous sera fourni et qui enregistre la même image sous le nom "imageTest.pgm".
Afin de compiler votre programme il suffit d'inclure tous les .c comme suit :

gcc -o tp1 tp1.c nrio.c nralloc.c nrarith.c

Note : Pensez à bien libérer vos mémoires !

3 Produit de convolution et Filtrage passe bas

Nous voulons ic réaliser un filtrage passe bas (filtre moyenneur) d'une image en niveau de gris. Vous testerez en premier lieu sur l'image "Cubesx3.pgm" avant d'essayer sur d'autres images à choisir parmi les images fournies ou à télécharger sur le net.

Le résultat de notre traitement (filtrage) sera une nouvelle image $g(x, y)$ égale au produit de convolution entre l'image d'origine $f(x, y)$ et la réponse impultionnelle du filtre $h(x, y)$:

$$g(x, y) = f(x, y) * h(x, y) \ .$$

.

pour rappel :

$$g(x, y) = \int_{-\infty}^{\infty} \int_{\infty}^{\infty} f(x - u, y - v) \cdot h(u, v) du dv$$

Vous testerez le produit de convolutions en filtrant l'image à l'aide du masque (représentant la réponse impultionnelle) $3x3$:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

.

Rappels :

- Attention, dans notre cas, notre signal 2D (image) n'est pas infini, il y aura donc un effet de bord à gérer, le produit de convolution ne pourra être calculé en dehors des bornes de notre signal (image)
- Ne pas oublier que vous travaillez sur des matrices de type Byte (valeurs comprises entre 0 et 255), il faudra faire attention aux overflows et underflows.

4 Calcul du gradient

4.1 Gradient horizontal

Convoluez l'image à l'aide du masque de Sobel correspondant au gradient horizontal:

$$\begin{matrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{matrix}$$

Vérifiez le résultat (qu'on notera I_x) en le sauvegardant dans un image de type pgm.

4.2 Gradient vertical

De la même façon, convoluez l'image à l'aide du masque de Sobel correspondant au gradient vertical :

$$\begin{array}{ccc} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{array}$$

Vérifiez le résultat (qu'on notera I_y) en le sauvegardant dans un image de type pgm.

4.3 Norme du gradient

Calculez la norme du gradient $\sqrt{I_x^2 + I_y^2}$.

5 Détection de contours

Les valeurs hautes de la norme du gradient indiquent le présence de zones de contours. Afin de détecter les contours de l'image, il suffit alors de seuiller la norme calculée précédemment; il en résultera une image binaire (valeurs des niveaux de gris à 0 ou 1) représentant les contours de l'image. Faites plusieurs essais avec différents seuils (sur des images réelles) pour apprécier la difficulté de trouver un bon rapport signal sur bruit.

6 Détection de contour dans les images couleur et seuillage par Hysthérésis

Dans un premier temps, détectez les contours sur chacun des plans RGB de votre image couleur (PPM). Gardez le même seuil pour chaque plan et enregistrer le résultat obtenu pour R,G et B dans trois images différentes de type PGM. essayer ensuite de trouver les seuils adéquats pour chaque plan.

vous obtenez alors trois matrices binaires. Le résultat d'une détection de contour doit être une seule image binaire. Afin de pouvoir calculer cette dernière, on peut citer trois choix "intuitifs" :

- un pixel de contour doit être détecté comme tel sur les 3 plans
- un pixel de contour doit être détecté comme tel sur au moins 2 plans
- un pixel de contour doit être détecté comme tel sur au moins un plan

Testez ces différentes combinaisons
Une autre solution, plus fine, consiste à utiliser un hystérésis à deux seuils, un bas (noté S_b) et un haut (noté S_h). On aura les combinaisons suivantes :

- $S_b = 3$ et $S_h = 3$: Uniquement les pixels détectés (valeur à 1) sur 3 plans sont considérés comme des pixels de contour
- $S_b = 2$ et $S_h = 3$: Les pixel détectés sur les 3 plans sont des pixels de contours, les pixels détectés sur au moins 2 plans et connexes à des pixels détectés sur les 3 plans sont aussi des pixels de contour
- $S_b = 1$ et $S_h = 3$: Les pixel détectés sur les 3 plans sont des pixels de contours, les pixels détectés sur au moins 1 plans et connexes à des pixels détectés sur les 3 plans sont aussi des pixels de contour

Cette approche permet de diminuer le bruit (pixels détectés sur 1 ou 2 plans mais non connexes au pixels détectés sur les 3 plans) tout en gardant un maximum d'information utile . Testez cette méthode et évaluez le rapport signal sur bruit comparativement au seuillage classique.