Détecteur de contours de Modestino et Fries par Arnaud Bourge et Michèle Nguyen

Introduction

Le problème traité dans ce projet est la détection des contours dans une image bruitée. L'idée de base qui est utilisée ici est d'exploiter la notion intuitive que les pixels qui se situent au bord de la frontière entre deux objets ou entre deux régions présentent des niveaux de gris très différents. La contrainte pour avoir un bon outil de détection est double. D'un côté, il faut que le détecteur soit relativement insensible au bruit, et de l'autre il faut que l'implémentation soit efficace en terme de complexité. Généralement, ces deux critères entrent en contradiction.

L'approche de Modestino et Fries est relativement originale, mais simple. Ils définissent tout d'abord un modèle stochastique pour fabriquer des images, qui possèdent notamment une orientation privilégiée. Puis, se basant sur les formules issues de ce modèle, ils créent un filtre adapté à la détection des contours dans ces images. Ils constatent ensuite que ce filtre est également satisfaisant sur n'importe quelle autre image du monde réel ne comportant pas de caractéristique particulière.

Nous ne décrirons pas le modèle stochastique utilisé par Modestino et Fries. Notre travail consistera principalement en l'implémentation d'un filtre à deux dimensions de réponse impulsionnelle infinie, ainsi que du critère de décision pour la détection des contours.

Théorie

1. Présentation du filtre utilisé

La première étape du développement du détecteur de contours est l'implémentation d'un filtre de Wiener déduit du modèle stochastique utilisé par Modestino et Fries. En fait, le but de ce filtrage est d'effectuer sur l'image non bruitée une opération de Laplacien (qui sera suivie par la suite d'une détection de zéros alliée à un seuilage). Bien sûr, on n'a accès qu'à l'image bruitée et la solution consiste à multiplier dans le domaine fréquentiel l'opération de Laplacien et le filtre de Wiener adapté à l'image. Le filtre global resultant sera appelé H0. Il est donc déduit du modele qu'on se donne pour l'image traitée. Ce modèle décrit une image à niveaux de gris et ses contours à partir de 3 paramètres simples : λ, ρ et ξ.

λ représente le nombre moyen de contours traversés par unité de distance dans une direction donnée, ρ la corrélation entre niveaux de gris de part et d'autre d'un contour, ξ donne le rapport signal sur bruit dans l'image.

Ainsi :

- plus λ est grand, plus la densité de contours est grande
- ρ est dans]-1;1[: plus ρ est grand, plus deux objets adjacents dans l'image ont des niveaux de gris proches (si ρ est proche de -1, les différents objets seront au contraire très contrastés les uns par rapport aux autres)
- plus ξ est grand, plus l'image est bruitée, ce qui risque d'entraîner la détection erronnée de contours dus uniquement au bruit.

A partir de ce modèle on peut calculer pour λ, ρ et ξ donnés le filtre linéaire optimal correspondant (filtre de Wiener convolué au filtre de Laplacien) qui nous permettra la détection de contours. Sa fonction de transfert admet la forme générale suivante :

$$H0(z1,z2) = \sum_{i=0}^{Mb}\sum_{j=0}^{Nb} b_{ij} z_1^{-i} z_2^{-j} \Bigg/ 1 + \sum_{\substack{i=0 \\ i=j\neq 0}}^{Ma}\sum_{j=0}^{Na} a_{ij} z_1^{-i} z_2^{-j}$$

où les coefficients a_{ij} et b_{ij} sont fonction des paramètres λ, ρ et ξ de l'image à traiter.
Ce filtrage préliminaire étant sensé favoriser la détection des contours dans l'image, on peut s'attendre à ce qu'il se comporte comme un passe-bas afin de réduire les effets néfastes du bruit (ξ) tout en conservant suffisamment de contraste entre objets dans l'image (grâce à la prise en compte de ρ et λ). Si on choisit correctement λ, ρ et ξ pour l'image étudiée, le filtre obtenu doit donc être le compromis optimal entre ces contraintes contradictoires.

Le choix d'un filtrage de réponse impulsionnelle infinie s'impose si l'on veut que la complexité calculatoire ne soit pas trop importante. Les économies de calculs sont surtout dûes à la nature récursive de cette formule. En particulier, la séquence de sortie $\{y_{i,j}\}$ répondant à la séquence

$$f(x) \otimes I(x) = [h(x) \otimes I(x)]'$$

d'entrée $\{x_{i,j}\}$ peut être obtenue récursivement grâce à la formule suivante :

avec m, n ≥ 0

On admettra que le filtre de Wiener calculé est un filtre stable, partant du coin en haut à gauche de l'image. On remarque que pour ce filtre qui débute le traitement en haut à gauche, seuls les pixels situés en bas à droite auront subi assez d'itérations pour que le filtrage se fasse sentir. Il convient donc d'appliquer le même filtre récursif en partant des quatre coins de l'image. La fonction de transfert du filtre correspondant est alors : $H(z_1,z_2) = z_1^{-\frac{1}{2}} z_1^{-\frac{1}{2}} H_0(z_1^{-\frac{1}{2}}, z_2^{-\frac{1}{2}})$

avec $H_1(z_1,z_2) = H_0(z_1,z_2) + z_1 H_0(z_1^{-1},z_2) + z_2 H_0(z_1,z_2^{-1}) + z_1 z_2 H_0(z_1^{-1},z_2^{-1})$

Pour des raisons d'économies de calcul nous nous limiterons à un filtre du premier ordre, qui a alors la forme :

$$H_0(z_1,z_2) = A\left(\frac{1 - \frac{1}{2}(b_{11}+1)(z_1^{-1}+z_1^{-1}) + b_{11} z_1^{-1} z_2^{-1}}{1 + a_{10}(z_1^{-1}+z_2^{-1}) + a_{11} z_1^{-1} z_2^{-1}}\right)$$

On remarque que ce filtre possède une reponse nulle à la fréquence zéro, il elimine donc la composante continue de notre image qui se retrouvera en sortie centrée en 0. Les trois coefficients a_{10}, a_{11}, b_{11} et le gain A seront choisis pour affiner la détection des contours, en fonction des paramètres λ, ρ et ξ. Ces paramètres sont en pratique choisis empiriquement par l'utilisateur en fonction de l'image qu'il a à traiter.

2. Décision et détection des contours

Une fois que l'on a obtenu l'image filtrée, on l'examine par tableaux de 3×3 pixels centrés sur le pixel en cours d'examen. On étudie alors les quatre pairs de pixels opposés, et le point central est déclaré comme un point du contour si:

- les niveaux de gris de n'importe quel couple de pixels sont de signes opposés
- l'amplitude de cette différence est supérieure à un seuil fixé.

Exemple I : BUREAU

Nous avons testé l'algorithme de Modestino & Fries sur l'image "bureau.ima". Elle présente l'avantage de contenir beaucoup d'objets aux formes géométriques simples (rectangles) avec des directions privilégiées et des espacements plus ou moins réguliers entre contours, ce qui en fait un exemple d'image "réelle" (par opposition à "de synthèse") proche du modèle théorique développé afin de trouver le filtre optimal H_0 *(cf. partie "Théorie").*

Après quelques essais, nous avons obtenu un des résultats les plus satisfaisants avec les paramètres λ=0.0125, ξ=10, et ρ=0. Le seuil permettant l'étape de décision à partir de l'image filtrée a été fixé à 2 pour un résultat optimal.

Image originale

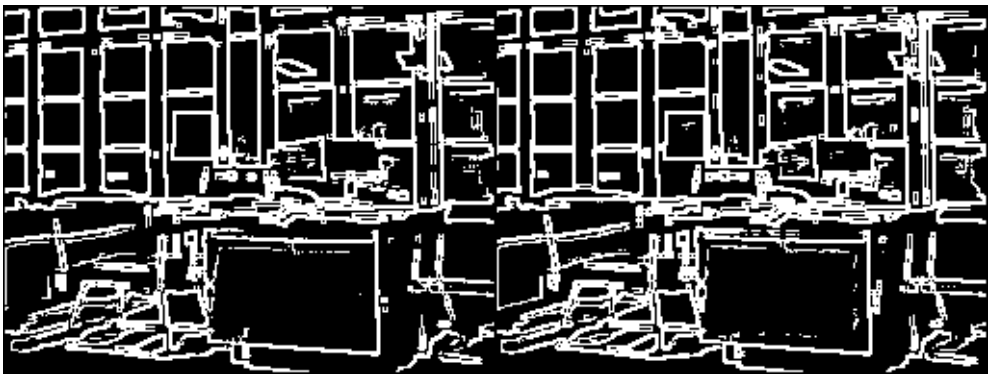
λ=0.0125; ξ=10; ρ=0

Constatons d'abord que la détection de contours pour cette image est d'un très bon niveau : la face avant du bureau ainsi que les carreaux et les bordures des fenêtres sont tous détectés sans problème. Il est vrai que certains parasites difficiles à éviter sont présents mais on peut aussi remarquer que certains détails à peine perceptibles a l'oeil nu ont été trouvés (barreaux de la chaise et structure du meuble situé à gauche).

Ensuite on peut s'intéresser à la répercussion des paramètres utilisés pour trouver les contours des objets de cette image. Nous avons dans la suite modifié à chaque fois un seul de ces trois paramètres (λ, ξ et ρ) par rapport au résultat précédent. Ainsi nous pouvons bien visualiser leurs effets indépendamment les uns des autres, et constater si oui ou non les résultats correspondent aux interprétations de λ, ξ et ρ que nous avions données.

a) Effet du paramètre lambda:

Rappel : le paramètre λ est sensé représenter le nombre de contours par unité de longueur.



λ=0.0250 λ=0.0500

On voit bien que plus λ est grand plus la densité de contours détectés est élevée. En effet le filtre va tenter de conserver suffisamment de contraste dans l'image pour garder un grand nombre de contours.

b) Effet du paramètre ksi:

Rappel : le paramètre ξ représente le rapport signal sur bruit (RSB) dans l'image de départ.

ξ =infini ξ =3dB

ξ infini signifie que l'image est considérée comme non bruitée. Alors le filtre calculé ne tient pas compte d'un éventuel bruit à supprimer dans l'image et va donc allouer des contours là où il ne faut pas : ça n'est que le bruit qui les a engendrés. C'est bien ce qu'on obtient sur l'image de gauche : beaucoup de contours superflus. La présence de bruit dans l'image "bureau" est donc sous-estimée si on retient ξ =inf.

Par contre un ξ de 3dB correspond à une image très bruitée (RSB médiocre). L'étape de filtrage aura donc tendance à vouloir lisser au maximum l'image originale afin d'atténuer la présence du bruit et l'empêcher de fausser ultérieurement la détection des contours. C'est bien ce qui se passe losrqu'on regarde le résultat de droite : l'image "bureau" a été très (trop) lissée lors du filtrage et le bureau se retrouve en partie noyé dans l'espace qui l'environne (certains contours ont disparu et le contour de la face avant n'est plus fermé). Cependant on peut voir que pratiquement tous les reflets dans les carreaux et les artefacts dus à la texture des plintes n'apparaissent plus (ils sont assimilés à du bruit du point de vue de la détection de contours).

c) Effet du paramètre rhô:

Rappel : ρ représente le degré de corrélation entre les niveaux de gris de deux objets adjacents dans l'image. Ainsi plus le contraste est élevé entre deux objets voisins plus ρ sera petit (proche de -1). Au contraire une image faiblement contrastée correspondra à un ρ grand (proche de 1).

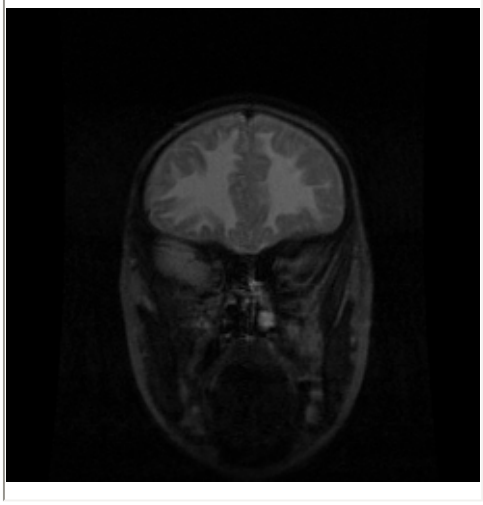
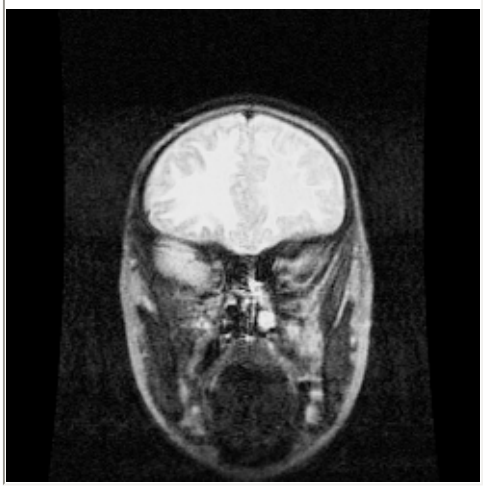
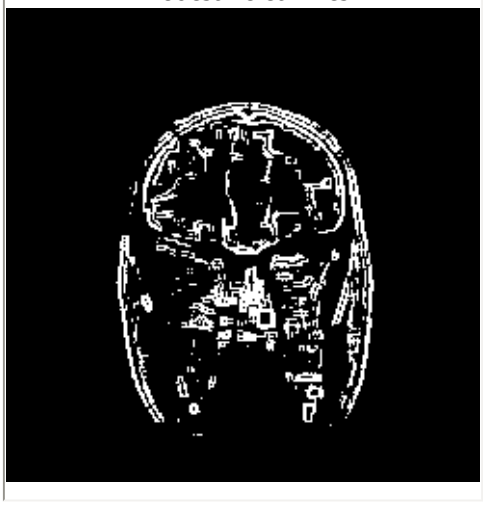
ρ =-0.9 ρ =0.5

Si l'on dit à notre algorithme que l'image est fortement contrastée (ρ =-0.9), alors il ne devrait pas hésiter à la lisser lors de l'opération de filtrage pour atténuer le bruit. En effet la propriété de fort contraste entre objets doit permettre une bonne conservation de la détectabilité des contours même après passage à travers un passe-bas. Au contraire on peut se dire que si l'image possède un faible contraste on va devoir limiter l'opération de filtrage afin de ne pas risquer de confondre deux objets voisins en un seul.

Pourtant, et contrairement à ce que l'on vient de suggérer, c'est l'inverse qui semble se produire sur cet exemple. En effet moins de contours sont détectés pour ρ =0.5, ce qui laisserait entendre que l'effet de lissage de l'opération de filtrage a été plus forte pour ρ élevé.

Exemple II : CERV6_10

Le premier exemple que nous traiterons se fera sur une image IRM du cerveau, que l'on nommera image **(0)**.

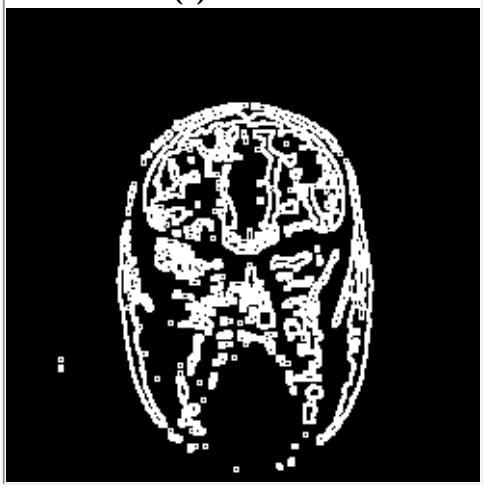
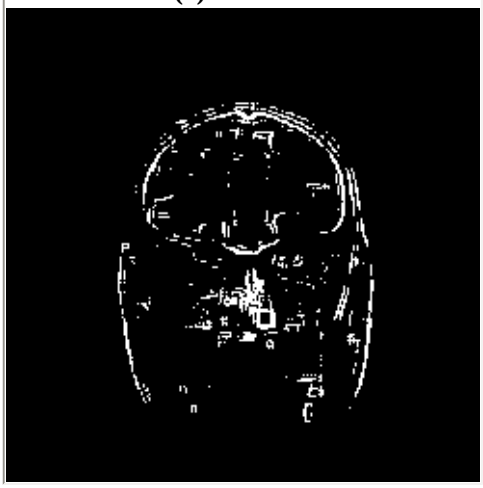
(0) Image initiale	(1) Image dont le contraste a été réhaussé	(2) Détection des contours par Modestino et Fries
		

Nous remarquons que les teintes de gris de cette images sont très proches, ce qui rend les contours difficiles à distinguer. Pour permettre une visualisation plus facile, nous avons utilisé la palette de xv pour augmenter artificiellement le contraste **(1)**. Cependant, il faut noter que tous les traitements que nous avons effectués ont été réalisés sur l'image initiale, et non sur l'image **(1)** qui ne sera utilisée que pour une meilleure compréhension des résultats.

L'image **(2)** est l'image des contours obtenue par l'algorithme de Modestino et Fries. Les valeurs des paramètres sont ici:

- Seuil = 2:

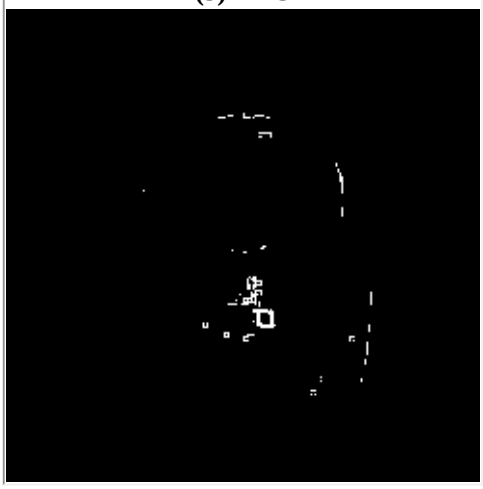
On note que le choix du seuil est très important. En effet, avec un seuil inférieur, comme sur l'image **(3),** les contours sont beaucoup plus nombreux, mais aussi, beaucoup plus épais. Avec un seuil légèrement supérieur comme sur l'image **(4)**, les contours sont au contraires moins bien définis, et notamment, moins bien fermés.

(3) Seuil = 1	(4) Seuil = 3
	

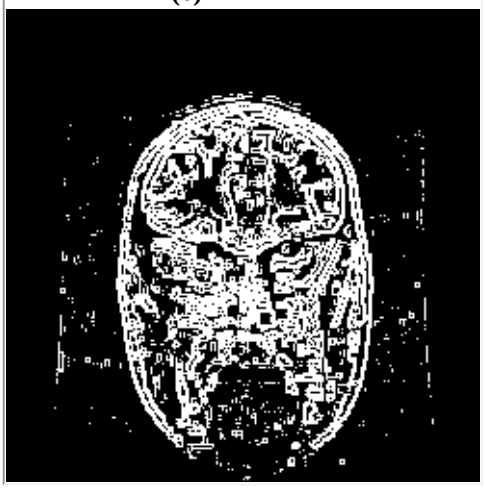
- ξ = 10 :

Sur l'image **(5)**, on a donné a ξ une valeur petite. Dans notre méthode, cela signifie que l'on suppose que le bruit est important dans l'image, et que le filtre devra beaucoup lisser. Par conséquent sur une image comme celle de notre exemple, le lissage est tel que les contours ne sont même plus détectés. Au contraire, sur l'image **(6)**, ξ a une valeur infinie, ce qui se traduit par le fait que le filtre ne prendra pas en compte le bruit. Ainsi, il reste beaucoup de pixels qui ne font pas partie de l'information que l'on cherche à extraire de l'image.

(5) ξ **= 3**



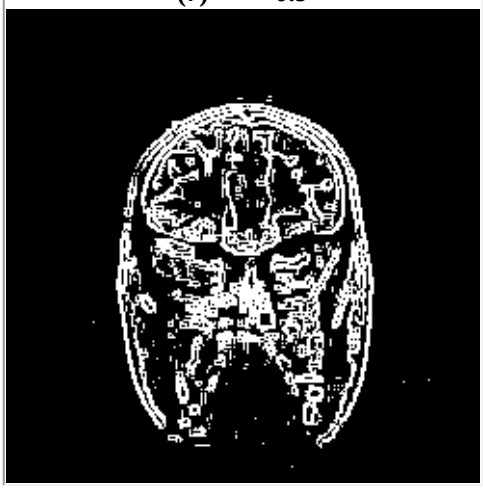
(6) ξ **= infini**



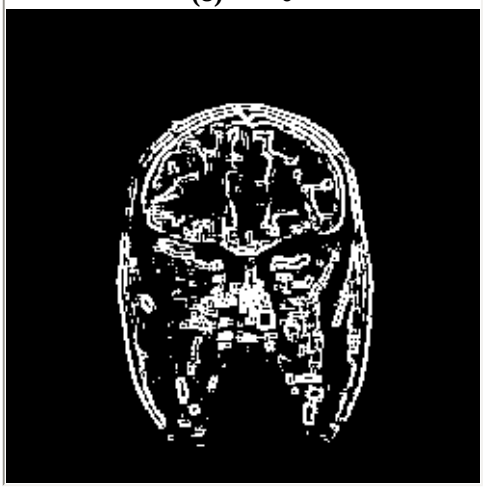
- ρ = 0.05

ρ est le paramètre qui exprime la corrélation entre les niveaux de gris de l'image. Comme nous l'avons vu plus haut, notre image initiale est très peu contrastée, ce qui justifie le fait de choisir une valeur de ρ plutôt proche de 1. Voici les résultats obtenus avec les autres valeurs de ρ .

(7) ρ **= -0.9**

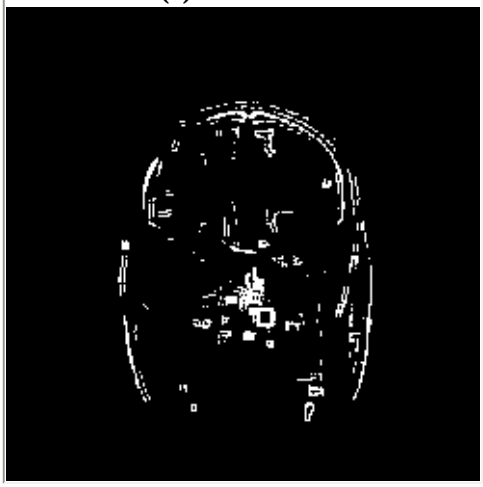
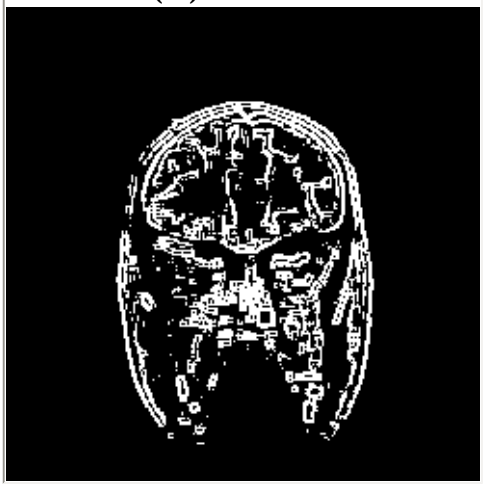


(8) ρ **= 0**



- λ = 0.0250

Ce paramètre doit être réglé en fonction du nombre de transitions qu'il existe entre les contours. Etant donnée notre image initiale, nous avons fixé ce paramètre à une valeur moyenne, ce qui se révèle être le bon choix, si on regarde les deux images qui suivent. Pour un λ trop petit (image **(9)**), il n'y a pas assez de contours détectés, et pour une valeur supérieure (image **(10)**), on détecte plus de contours mais qui ne sont pas forcément très informatifs.

(9) λ = 0.0125	(10) λ = 0.0500
	

Remarque : choix du seuil

Nous avons étudié les performances de l'algorithme de Modestino & Fries sur deux exemples, en montrant à chaque fois le rôle décisif du choix des paramètres qui définissent le filtre utilisé (λ, ξ et ρ).

Cependant une fois l'image filtrée il reste l'étape de décision et celle-ci requiert la définition d'un dernier paramètre : le seuil, qui entre en compte dans l'allocation d'un point de contour pour un pixel donné (cf. partie "Théorie). Plus le seuil choisi est grand et plus on sera exigeant sur la différence de niveau de gris entre deux points pour mettre un pixel en blanc.

En pratique il est difficile de savoir a priori quel seuil utiliser. En effet pour λ, ξ et ρ on peut avoir avec un peu d'expérience une idée, pour une image donnée, des valeurs à prendre car on en a une interprétation précise. En revanche le choix du seuil va dépendre de l'image en sortie du filtre et on ne sait pas vraiment ce qu'elle sera, et surtout quel sera son degré de contraste ou même son étendue en terme de niveaux de gris.

Ceci fait que l'on tâtonne pour trouver le seuil adéquat une fois qu'on a la bonne image filtrée.

L'exemple suivant illustre l'importance du choix du seuil en donnant deux résultats très différents pour des valeurs de λ, ξ et ρ identiques par ailleurs.

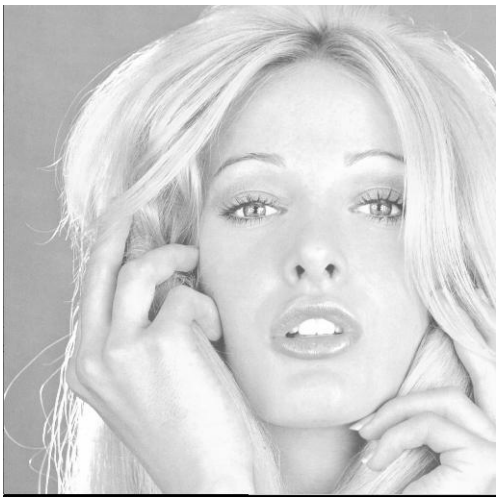


Image originale


seuil=2


seuil=5

Conclusion

Nous avons testé notre programme sur plusieurs images ayant des configurations géométriques et des niveaux de gris très différents. Les résultats obtenus sont dans l'ensemble plutôt satisfaisants.

L'inconvénient de cette méthode reste cependant le nombre important de paramètres à régler pour obtenir une image de contours qui soit bonne. C'est par tâtonnments successifs que l'on peut véritablement affiner les 3 paramètres qui interviennnent dans la définition du filtre, et le seuil qui détermine le critère de décision.

Par contre, pour remédier à ce problème, on peut imaginer une interface qui permettrait de réaliser les tests plus facilement. On pourrait par exemple fixer les paramètres du filtre à l'aide de menus déroulants qui proposeraient pour chacun les valeurs possibles. Ensuite, une fois le filtre calculé pour ces valeurs, il suffirait d'avoir un curseur définissant le seuil, que l'on ferait varier, tout en affichant les images de contours obtenues pour ces valeurs. On pourrait ainsi trouver facilement et presque en temps réel le seuil qui convient le mieux, car on visualiserait le résultat instantanément.

Le programme

```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>


void charger_dim (char* filename, int* larg, int* haut)
{
   char *file_dim;
   char *dimext;
   FILE *f;
   int t;

   t=strlen(filename);

   dimext=".dim";
   file_dim=(char*)malloc(t*sizeof(char));
   strcpy(file_dim,filename);
   strcat(file_dim, dimext);

   f=fopen (file_dim, "r");
   fscanf(f,"%d %d", larg, haut);
   fclose(f);
```

```
}

void charger_ima (char* filename, int* larg, int *haut, int* image)
{
   char *file_ima;
   char *imaext;
   FILE *g;
   unsigned char *pixel;
   int i,j,t;

   t=strlen(filename);

   imaext=".ima";
   file_ima=(char*)malloc(t*sizeof(char));
   strcpy(file_ima,filename);
   strcat(file_ima, imaext);

   pixel = (unsigned char*)malloc((*larg)*(*haut)*sizeof(char));

   g = fopen (file_ima, "r");
   fread(pixel, 1,(*larg)*(*haut), g);
   for (i=0; i<(*haut); i++)
     for (j=0; j<(*larg); j++)
 {
  image[i*(*larg)+j] = (int)(pixel[i*(*larg)+j]);
 }
   fclose(g);
}


/* coefficients a et b du filtre de Wiener d'apres les parametres lambda, ro, E */

void coef (float ro,float lambda, float E, float *A,float *a10,float *a11,float *b11)

{

if(E==99)
  {
   if(abs(ro)>=1)
     /* error */
 {
    printf("ro doit etre de module plus petit que 1");
    exit(1);
 }
   else
     {
 *b11=-0.8256;
       *a10=-0.2149;
       *a11=0.0098;
       *A=0.115;
     }
  }

if(E==10)
  {
   if(ro==-0.9f)
     {
 if(lambda==0.0125f)
   {
    *b11=-0.8924;
    *a10=-0.2565;
    *a11=-0.2189;
    *A=0.053;
   }
 if(lambda==0.0250f)
```

```
  {
    *b11=-0.9396;
    *a10=-0.1989;
    *a11=-0.2490;
    *A=0.07;
  }
    if(lambda==0.0500f)
  {
    *b11=-0.9244;
    *a10=-0.1944;
    *a11=-0.1929;
    *A=0.084;
  }
     }
   if(ro==0.0f)
     {
if(lambda==0.0125f)
  {
    *b11=-0.8252;
    *a10=-0.3217;
    *a11=-0.1579;
    *A=0.038;
  }
if(lambda==0.0250f)
  {
    *b11=-0.8460;
    *a10=-0.2849;
    *a11=-0.1687;
    *A=0.053;
  }
    if(lambda==0.0500f)
  {
    *b11=-0.7980;
    *a10=-0.2925;
    *a11=-0.0884;
    *A=0.067;
  }
     }
   if(ro==0.5f)
     {
if(lambda==0.0125f)
  {
    *b11=-0.5736;
    *a10=-0.4654;
    *a11=0.0449;
    *A=0.025;
  }
if(lambda==0.0250f)
  {
    *b11=-0.8228;
    *a10=-0.3106;
    *a11=-0.1671;
    *A=0.04;
  }
    if(lambda==0.0500f)
  {
    *b11=-0.8480;
    *a10=-0.2821;
    *a11=-0.1696;
    *A=0.054;
  }
     }
  }

if(E==3)
```

```
 {
   if(ro==-0.9f)
     {
if(lambda==0.0125f)
  {
    *b11=-0.8264;
    *a10=-0.3602;
    *a11=-0.1540;
    *A=0.022;
  }
if(lambda==0.0250f)
  {
    *b11=-0.8664;
    *a10=-0.3075;
    *a11=-0.2040;
    *A=0.034;
  }
    if(lambda==0.0500f)
  {
    *b11=-0.8896;
    *a10=-0.2665;
    *a11=-0.2213;
    *A=0.048;
  }
     }
   if(ro==0.0f)
     {
if(lambda==0.0125f)
  {
    *b11=-0.8256;
    *a10=-0.3840;
    *a11=-0.1461;
    *A=0.015;
  }
if(lambda==0.0250f)
  {
    *b11=-0.8120;
    *a10=-0.3686;
    *a11=-0.1350;
    *A=0.023;
  }
    if(lambda==0.0500f)
  {
    *b11=-0.8228;
    *a10=-0.3326;
    *a11=-0.1575;
    *A=0.034;
  }
     }
   if(ro==0.5f)
     {
if(lambda==0.0125f)
  {
    *b11=-0.7556;
    *a10=-0.4815;
    *a11=0.0102;
    *A=0.0088;
  }
if(lambda==0.0250f)
  {
    *b11=-0.8260;
    *a10=-0.3782;
    *a11=-0.1534;
    *A=0.015;
  }
    if(lambda==0.0500f)
```

```
   {
     *b11=-0.7636;
     *a10=-0.3929;
     *a11=-0.0895;
     *A=0.023;
   }
      }
  }
}
```

```
/* filtre de Wiener en 2 dimension approxime au premier ordre */

void filtre (int* resultat, int* res_image, int *image, int haut, int larg, float a10, float a11, float b11, float A)

{
float b01, b10=-(1+b11)/2;     /* ce coefficient du filtre sont deduits de b11
       et on a b01 = b10 (et aussi a01=a10)*/
int hg[haut][larg]; /* resultat du filtre qui part en haut a gauche */
int bg[haut][larg]; /* resultat du filtre qui part en bas a gauche */
int hd[haut][larg]; /* resultat du filtre qui part en haut a droite */
int bd[haut][larg]; /* resultat du filtre qui part en bas a droite */
/*resultat est la somme des 4 resultats precedents */
int max, min;
int i,j;
float moyenne=0;

b01=b10;

for(i=0;i<haut;i++)
   for(j=0;j<larg;j++)
 {
 hg[i][j]=image[i*larg+j];
 bg[i][j]=image[i*larg+j];
 hd[i][j]=image[i*larg+j];
 bd[i][j]=image[i*larg+j];
 }

fprintf(stderr,"les 4 filtres intermediaires sont initialises : %d * %d\n",larg,haut);

/* filtre H0 */

for(i=1;i<haut;i++)
   for(j=1;j<larg;j++)
 {
 hg[i][j]=(int)(image[i*larg+j]+b10*(float)image[(i-1)*larg+j]+b01*(float)image[i*larg+j-1]+b11*(float)image[(i-1)*larg+j-1]-a10*(hg[i-
1][j]+hg[i][j-1])-a11*hg[i-1][j-1]);
 }
fprintf(stderr,"hg calcule ; ");

/* filtre z1*H0(1/z1,z2) */

for(i=haut-3;i>=0;i--)
   for(j=1;j<larg;j++)
 {
 bg[i][j]=(int)(image[(i+1)*larg+j]+b10*(float)image[(i+2)*larg+j]+b01*(float)image[(i+1)*larg+j-1]+b11*(float)image[(i+2)*larg+j-1]-
a10*(bg[i+2][j]+bg[i+1][j-1])-a11*bg[i+2][j-1]);
 }
fprintf(stderr,"bg calcule ; ");
```

```
/* filtre z2*H0(z1,1/z2) */

for(i=1;i<haut;i++)
   for(j=larg-3;j>=0;j--)
 {
 hd[i][j]=(int)(image[i*larg+j+1]+b10*(float)image[(i-1)*larg+j+1]+b01*(float)image[i*larg+j+2]+b11*(float)image[(i-1)*larg+j+2]-a10*(hd[i-
1][j+1]+hd[i][j+2])-a11*hd[i-1][j+2]);
 }
fprintf(stderr,"hd calcule ; ");

/* filtre z1*z2*H0(1/z1,1/z2) */

for(i=haut-3;i>=0;i--)
   for(j=larg-3;j>=0;j--)
 {
 bd[i][j]=(int)(image[(i+1)*larg+j+1]+b10*(float)image[(i+2)*larg+j+1]+b01*(float)image[(i+1)*larg+j+2]+b11*(float)image[(i+2)*larg+j+2]-
a10*(bd[i+2][j+1]+bd[i+1][j+2])-a11*bd[i+2][j+2]);
 }
fprintf(stderr,"bd calcule.\n");

/* sommation des 4 filtres intermediaires */

max=-1000000000.0;
min=1000000000.0;
for(i=0;i<haut;i++)
  {
  for(j=0;j<larg;j++)
    {
     resultat[i*larg+j]=(int)(A*(float)(hg[i][j]+bg[i][j]+hd[i][j]+bd[i][j]));
     moyenne=moyenne+resultat[i*larg+j];
     if (resultat[i*larg+j]>max)
 max=resultat[i*larg+j];
     if (resultat[i*larg+j]<min)
 min=resultat[i*larg+j];

    }
  }

moyenne=moyenne/(larg*haut);
fprintf(stderr,"moyenne = %d\n", moyenne);
fprintf(stderr,"min=%d ; max=%d\n",min,max);

/* on ramene les valeurs entre 0 et 255 pour afficher le resultat */

for(i=0;i<haut;i++)
  {
  for(j=0;j<larg;j++)
    {
     res_image[i*larg+j]=(int)((resultat[i*larg+j]-min)*255/(max-min));
     //fprintf(stderr,"%d ",resultat[i*larg+j]);
    }
  }

 printf("somme des 4 intermediaires effectuee\n");
  }




/* Determination des contours*/

int signe (int pix1, int pix2, int central)
{
```

```
 if ((pix1>central && pix2<=central) || (pix1<=central && pix2>central))
   return 1;
 return 0;
}

void decision (int seuil, int* image, int* contour, int larg, int haut)

{
 int i, j;
 for (i=0; i<haut; i++)
   for (j=0; j<larg; j++)
     contour[i*larg+j]=0;
 for (i=1; i<haut-1; i++)
   for (j=1; j<larg-1; j++)
     {
int detect = 0;
if (signe(image[(i-1)*larg+j-1], image[(i+1)*larg+j+1], 0))
  if (abs(image[(i-1)*larg+j-1]- image[(i+1)*larg+j+1])>=seuil)
    {
      contour[i*larg+j]=255;
      detect = 1;
    }
if (!detect && signe(image[(i-1)*larg+j], image[(i+1)*larg+j], 0))
  if (abs(image[(i-1)*larg+j]- image[(i+1)*larg+j])>=seuil)
    {
      contour[i*larg+j]=255;
      detect = 1;
    }
if (!detect && signe(image[(i-1)*larg+j+1], image[(i+1)*larg+j-1], 0))
  if (abs(image[(i-1)*larg+j+1]- image[(i+1)*larg+j-1])>=seuil)
    {
      contour[i*larg+j]=255;
      detect = 1;
    }
if (!detect && signe(image[i*larg+j-1], image[i*larg+j+1], 0))
  if (abs(image[i*larg+j-1]- image[i*larg+j+1])>=seuil)
    {
      contour[i*larg+j]=255;
      detect = 1;
    }
     }

}

/*Sauver au format .ima*/

void sauver(int* image, char* image_finale, int larg, int haut)
{
   int i, t, a, compteur=0;
   unsigned char aux;
   FILE *f;
   char* file_ima, *file_dim, *imaext, *dimext;

   t=strlen(image_finale);

   imaext=".ima";
   file_ima=(char*)malloc(t*sizeof(char));
   strcpy(file_ima,image_finale);
   strcat(file_ima, imaext);

   fprintf(stderr,"Dans sauver : %d * %d\n", larg, haut);
   f=fopen(file_ima, "w");
   for (i=0;i<larg*haut; i++)
     {
aux=(unsigned char)image[i];
```

```
 a=fwrite(&aux, 1, 1, f);
 compteur=compteur+a;
      }
   fclose(f);

   //fprintf(stderr,"compteur=%d\n",compteur);
   dimext=".dim";
   file_dim=(char*)malloc(t*sizeof(char));
   strcpy(file_dim,image_finale);
   strcat(file_dim, dimext);

   f=fopen(file_dim, "w");
   fprintf(f,"%d %d", larg, haut);
   fclose(f);

}


/*Programme principal*/
////////////////////////////////////////////////////////////////////

void main (int argc, char* argv[])
{

   int seuil;
   int *larg, *haut;
   int *image_init, *image_filtre, *filtre_sauve, *image_contour;
   char *filtre_file;
   float ro, E, lambda;
   float *a10, *a11, *b11, *A;
   int aux;

   filtre_file=(char*)malloc(sizeof(char));
   larg = (int*)malloc(sizeof(int));
   haut = (int*)malloc(sizeof(int));
   a10 = (float*)malloc(sizeof(float));
   a11 = (float*)malloc(sizeof(float));
   b11 = (float*)malloc(sizeof(float));
   A = (float*)malloc(sizeof(float));
   printf("lecture des parametres\n");

   if (argc > 6)
 {
    // On charge l'image
    charger_dim (argv[1], larg, haut);
    image_init = (int*)malloc((*larg)*(*haut)*sizeof(int));
    charger_ima (argv[1], larg, haut, image_init);
    // On convertit les coefficients donnes en entree
    ro = atof(argv[2]);
    E = atof (argv[3]);
    lambda = atof (argv[4]);
       seuil = atoi(argv[5]);
 }

   else
 // On choisit un exemple qui marche bien
 {
    charger_dim ("fillette", larg, haut);
    image_init = (int*)malloc((*larg)*(*haut)*sizeof(int));
    charger_ima ("fillette", larg, haut, image_init);
    ro = 0.5;
    E = 10.0;
    lambda = 0.0125;
    seuil = 20;
```

```
}

   fprintf(stderr,"image chargee\n");
   // On alloue les images
   image_filtre = (int*)malloc((*larg)*(*haut)*sizeof(int));
   filtre_sauve = (int*)malloc((*larg)*(*haut)*sizeof(int));

   // On calcule les coef du filtre;
    coef (ro, lambda, E, A, a10, a11, b11);
    fprintf(stderr,"coefficients calcules : %f, %f, %f, %f\n", *A, *a10, *a11, *b11);
    // On filtre
    filtre(image_filtre, filtre_sauve, image_init, *haut, *larg, *a10, *a11, *b11, *A);
    fprintf(stderr,"image de depart filtree\n");
    strcpy(filtre_file,argv[1]);
    strcat(filtre_file,"F");
    sauver(filtre_sauve, filtre_file,*larg,*haut);
    fprintf(stderr,"image filtree sauvee\n");
    free(filtre_sauve);
    fprintf(stderr,"larg= %d; haut=%d\n",*larg,*haut);
    // On cree l'image des contour
    image_contour = (int*)malloc((*larg)*(*haut)*sizeof(int));
    decision (seuil, image_filtre, image_contour, *larg, *haut);
    fprintf(stderr,"contours places\n");
    // On ecrit en .ima
    if (argc>6)
 sauver(image_contour, argv[6], *larg, *haut);
    else
 sauver(image_contour, "toto", *larg, *haut);

    printf("contours sauves\n");



    // Liberer la memoire!!
    free(image_init);
    free(image_filtre);
    free(image_contour);
    free(filtre_file);
    free(a10);
    free(a11);
    free(b11);
    free(A);
    free(haut);
    free(larg);

}
```
