Thème Image - TP4 - Détection de contours

Ce TP est noté et constitue la note de CC2 de MAP2

— Vous avez deux séances pour faire le TP.
— Les travaux doivent être remis au chargé de TP au plus tard une semaine après la dernière séance de TP.
— Le travail comprend : un compte-rendu au format pdf de la totalité du TP, ainsi que les codes scilab.
— Vous rendrez UN UNIQUE FICHIER au format zip qui contiendra le compte rendu et les codes (vous pourrez recopier des portions de code dans le compte-rendu si cela vous semble opportun.). Le nom du fichier doit être de la forme : `NOM1_NOM2_groupe_XX_TP4.zip` . Soyez attentifs à correctement commenter et indenter le code pour faciliter la lecture. De manière générale, soyez attentifs à la présentation des résultats et à la rédaction. Le thème de ce TP est l'image, il est donc obligatoire d'illustrer votre propos avec les images produites.

1 - Préambule

1.1 - Utilisation des filtres avec `scilab`

A partir de maintenant vous utiliserez la routine `conv2(im, H)` qui est intégrée à `scilab` pour appliquer un filtre `H` à une image `im`. Par rapport à la routine `moyenne2D_W` que vous avez programmée, `conv2` s'exécute plus rapidement. Exemple d'utilisation :

```
exec('init_tp_image.sce');
im1 = lire_imageBMPgris('papillon.bmp');
// filtre moyenne
M = 1/9*ones(3, 3);
im2 = conv2(im1, M, "same");
```

L'option `"same"` est importante pour que `im2` ait la même taille que `im1`.

1.2 - Filtres gaussiens

Le filtrage gaussien permet d'améliorer la qualité des contours dans de nombreux cas. Vous en aurez besoin par la suite. La fonction `W_gauss_2D` vous est fournie dans le fichier `filtres2D.sci`.

2 - Dérivées de l'image

Image continue. On munit le plan du système de coordonnées habituel (x, y). Une image continue est vue comme une fonction

$$I : \begin{array}{ccc} \mathbb{R}^2 & \to & \mathbb{R} \\ (x,y) & \mapsto & I(x,y), \end{array}$$

où $I(x, y)$ correspond à l'intensité lumineuse au point (x, y). On note $\frac{\partial I}{\partial x}$ et $\frac{\partial I}{\partial y}$ les dérivées partielles (qu'on suppose définies partout) et grad $I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ le gradient de I. La norme du gradient

$$\|\text{grad } I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

est donc une fonction de $\mathbb{R}^2$ dans $\mathbb{R}$.

Image numérique. Soit `im` une image numérique donnée sous la forme d'un tableau à deux dimensions contenant des entiers de 0 à 255. L'équivalent numérique des dérivées partielles de l'image est obtenu par convolution avec les filtres `Dx = 0.5*[-1, 0, 1]` pour la direction x et `Dy = 0.5*[1; 0; -1]` pour la direction y. Notez bien que `Dx` est un vecteur ligne et `Dy` un vecteur colonne. On obtient ainsi deux tableaux de nombres :

```
imx = conv2(im, Dx, "same");
imy = conv2(im, Dy, "same");
```

Pour chaque pixel `(u,v)`, le norme du gradient est définie par

$$\texttt{imn}(\texttt{u},\texttt{v}) = \sqrt{\texttt{imx}(\texttt{u},\texttt{v})^2 + \texttt{imy}(\texttt{u},\texttt{v})^2}. \tag{1}$$

Exercice 1 :

- Dans le fichier `contours.sci`, complétez la fonction `norme_gradient(im)` pour qu'elle renvoie la norme `imn` du gradient de l'image `im`, selon la formule (1).
- Pour tester le résultat, visualisez la norme du gradient :

```
exec('init_tp_image.sce');
exec('contours.sci');
im = lire_imageBMPgris('sweets.bmp');
imn = norme_gradient(im);
afficher_image(int(imn));
```

- Améliorez la visualisation en ramenant les valeurs de `imn` dans l'intervalle $\{0, \ldots, 255\}$. Autrement dit, corrigez le contraste de l'image `imn`.

3 - Détection des contours par seuillage

Une première méthode de détection des contours consiste à les définir comme les points de l'image où la norme du gradient dépasse une certaine valeur `seuil`. L'image `imc` contenant les contours est définie par

$$\forall(\mathtt{u},\mathtt{v}), \mathtt{imc}(\mathtt{u},\mathtt{v}) = \begin{cases} 0 & \mathtt{si} \quad \mathtt{imn}(\mathtt{u},\mathtt{v}) < \mathtt{seuil}, \\ 255 & \mathtt{sinon}, \end{cases}$$

où `imn` est l'image contenant la norme du gradient de l'image `im`.

Exercice 2 :

- Dans le fichier `contours.sci`, complétez la fonction `contours_seuil(im, seuil)` de manière à ce qu'elle renvoie l'image `imc` telle que définie ci-dessus.
- Testez la fonction :

```
exec('contours.sci', -1);
im = lire_imageBMPgris('barque.bmp');
seuil = 20;
imc = contours_seuil(im, seuil);
afficher_image(imc);
```

  Vous pouvez faire varier la valeur du paramètre `seuil` pour évaluer son influence sur le résultat.

Le seuil peut varier selon l'image, mais on peut automatiser la recherche de ce seuil en fonction de la répartition des valeurs de `imn`. Etant donné que l'on veut conserver les valeurs les plus élevées de la norme du gradient, on peut fixer le seuil de manière à ce qu'un certain pourcentage `p` des pixels les plus sombres soit éliminé. Soit `Hc` le tableau contenant l'histogramme cumulé de `imn` ; on rappelle que `Hc(v)` représente le nombre de pixels dont la valeur est inférieure ou égale à `v-1`, et que `Hc(256)` représente le nombre total de pixels de l'image. Ainsi, `Hc(v)/Hc(256)` représente le pourcentage de pixels de l'image dont la valeur est inférieure à `v-1`. Le seuil est donc choisi comme la plus petite valeur `v` telle que `Hc(v)/Hc(256) > p`.

Exercice 3 :

- Dans le fichier `contours.sci` complétez la fonction `trouver_seuil(im, p)` pour qu'elle mette en oeuvre la recherche automatique de seuil en fonction d'un pourcentage `p` et renvoie sa valeur. Pour la tester :

```
exec('contours.sci', -1);
p = 0.8;
im = lire_imageBMPgris('barque.bmp');
seuil = trouver_seuil(im, p);
disp(seuil);
```

  Faites varier `p` et vérifiez que `seuil` varie de manière cohérente.
  Il y a une fonction `hist_cumul(im)` *dans le fichier* `contours.sci`, *vous pouvez l'utiliser pour cette question.*
- Vous pouvez ensuite tester le résultat en terme de contours avec la fonction `contours_p(im, p)` qui vous est fournie. Elle détermine les contours à partir d'un pourcentage `p` en se basant sur les fonctions `contours_seuil` et `trouver_seuil` :

```
p = 0.8;
afficher_image(contours_p(im, p));
```

  Puis vous pouvez tester avec d'autres valeurs de `p` et différentes images.

4 - Lissage et contours

Il est souvent intéressant d'appliquer un filtre de lissage avant de calculer la norme du gradient, de manière à ne garder que les variations les plus significatives.

Exercice 4 :

Pour les images du tableau ci-dessous, testez différentes valeurs du seuil `p` et du paramètre `sigma` de la gaussienne ; vous pouvez faire ces opérations et afficher le résultat en une ligne :

```
afficher_image(contours_p(conv2(im, W_gauss_2D(sigma), "same"), p));
```

Pour chacune des images, indiquez dans le tableau les valeurs testées et mettez en gras celle qui vous paraît la mieux adaptée.

	barque.bmp	barque_bruit.bmp	plage.bmp	sweets.bmp
sigma				
p				

N'oubliez pas d'illustrer votre compte-rendu avec les images obtenues !

Exercice 5 :

Comparez la detection de contours après filtrage Gaussien (faite dans l'exercice précédent) et celle après filtre de moyennage uniforme. Testez différentes tailles de filtre.

```
// filtre moyenne
M = 1/9*ones(3, 3);
afficher_image(contours_p(conv2(im, M, "same"), p));
```

5 - Mise en évidence du problème

L'opération de seuillage décrite plus haut impose de choisir un seuil sur la norme du gradient, au-delà duquel un pixel donné est interprété comme un contour. Mais les pics de la norme du gradient n'ont pas tous la même hauteur, et il faut choisir un seuil suffisamment faible pour capter tous les contours significatifs. De ce fait, l'épaisseur des contours n'est pas constante et la méthode peut aboutir à des résultats qui ne sont pas satisfaisants. L'exercice suivant illustre ce problème.

Exercice 6 :

On considère l'image `im` obtenue de la manière suivante :

```
// image intialement noire
N = 40;
M = 2*N
im = zeros(N,M);

// l'image est composée de trois bandes de taille n:
n = floor(M/3)

// le premier tiers reste noir

// le deuxième tiers est un dégradé avec un fort gradient
// de 30 à 255
for j = n:2*n
    im(:, j) = 30 + ((j - n)/n)^4*225;
end

// le dernier tiers est à 255
im(:, 2*n+1:M) = 255;

// lissage gaussien
G = W_gauss(0.5);
im = conv2(im, G);
```

Vous pouvez visualiser cette image avec la fonction `afficher_image`. Vous pouvez également tracer une coupe horizontale avec la commande suivante :

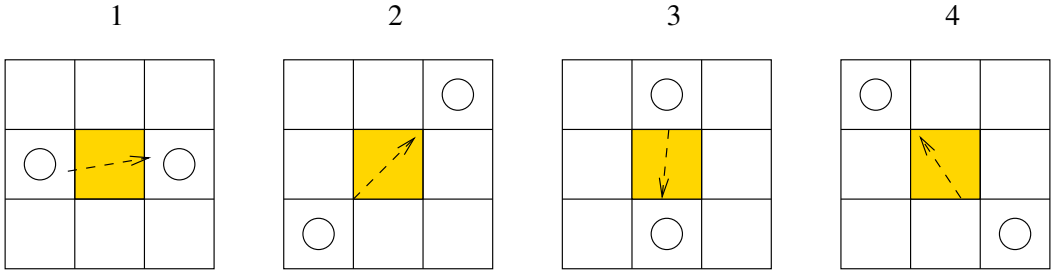
```
scf() ; plot(im(N/2, :))
```

Comme vous pouvez le constater, cette image contient deux variations importantes en les colonnes `j = M/3` et `j = 2M/3` qu'on aimerait pouvoir identifier comme des contours.

1. Essayez d'extraire les contours avec la fonction `contours_p`, en faisant varier le paramètre `p`. Que pensez-vous du résultat (sans se préoccuper des bords) ? Pouvez-vous trouver une valeur de `p` qui permette d'obtenir simultanément les contours en `M/3` et `2M/3` avec une épaisseur comparable ?
2. Proposez une explication. *Indication :* visualiser la norme du gradient.

6 - Elimination des non-maxima locaux

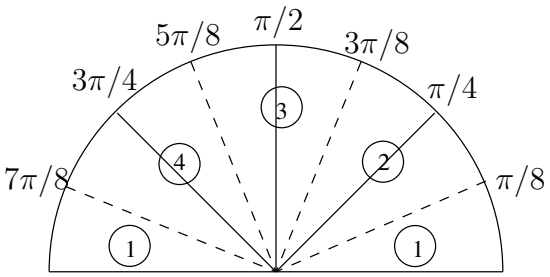
Avec la méthode précédente, on identifie trop de points en tant que contours. Afin de pallier à ce problème on réexamine chaque pixel du contour et on essaie d'identifier s'il correspond à un maximum de la norme du gradient dans la direction donnée par le vecteur gradient (qui est la direction de plus grande variation locale). Ainsi, pour un pixel donné on compare sa valeur à celle de deux de ses voisins, choisis de manière à être les plus proches possibles de la direction du gradient. Il y a quatre cas possibles à choisir en fonction de la direction du gradient, comme illustré ci-dessous avec des exemples où le vecteur gradient est représenté par la flèche en pointillés.



Etant donnée une image `im`, on applique l'algorithme suivant :

- Soient `imx` et `imy` les tableaux contenant les dérivées partielles par rapport à x et y, `imn` le tableau contenant la norme du gradient. On rappelle que ces trois tableaux ont la même taille que l'image `im` mais ne sont pas des images en tant que tels ; ainsi il n'est pas nécessaire que leurs valeurs soient des entiers dans l'intervalle $[0, 255]$. Soit `imc` l'image des contours obtenue par seuillage de la norme du gradient
- Pour chaque pixel `(u,v)` qui n'est pas sur le bord de l'image et tel que `imc(u,v) = 255` :
  - Calcul de l'angle ϕ du vecteur gradient (`imx(u, v)`, `imy(u, v)`) avec l'axe x, à l'aide de la fonction `atan` de `scilab` : `phi = atan(imy(u,v), imx(u,v))`.
  - Si $\phi \leq 0$, on remplace ϕ par $\phi + \pi$, ce qui ne modifie pas la direction et permet de se ramener à une valeur dans l'intervalle $[0, \pi]$.
  - On détermine les pixels voisins qui sont les plus proches de la direction du gradient parmi les quatre cas illustrés ci-dessus en fonction de l'angle ϕ :

    - Si $\phi \leq \pi/8$ ou $\phi \geq 7\pi/8$: cas 1.
    - Si $\pi/8 < \phi \leq 3\pi/8$: cas 2.
    - Si $3\pi/8 < \phi < 5\pi/8$: cas 3.
    - Si $5\pi/8 \leq \phi < 7\pi/8$: cas 4.

  - On compare la valeur de la norme du gradient dans le pixel central `(u,v)` à celles des deux voisins de l'étape précédente ; si une des deux valeurs est strictement supérieure, le pixel central n'est pas un maximum local et on considère qu'il ne fait plus partie du contour : `imc(u,v) = 0`.

Exercice 7 :

Identification des voisins les plus proches d'une direction donnée. Dans le fichier `contours.sci`, ajoutez une fonction `indices_voisins` avec la signature suivante :

```
function [u1,v1,u2,v2] = indices_voisins(u,v,phi)
```

Complétez-la pour qu'elle renvoie les indices `[u1,v1,u2,v2]` des pixels voisins du pixel `(u,v)` les plus proches de la direction donnée par l'angle `phi`. Cette fonction devra avoir le comportement suivant :

```
exec('contours.sci', -1);
u = 5; v = 5;
//// 1er cas ////
phi = %pi/16;
[u1,v1,u2,v2] = indices_voisins(u,v,phi);
// on attend (u1,v1) = (5,4) et (u2,v2) = (5,6)
//// 2eme cas ////
phi = %pi/4;
[u1,v1,u2,v2] = indices_voisins(u,v,phi);
// on attend (u1,v1) = (6,4) et (u2,v2) = (4,6)
//// 3eme cas ////
phi = %pi/2;
[u1,v1,u2,v2] = indices_voisins(u,v,phi);
// on attend (u1,v1) = (6,5) et (u2,v2) = (4,5)
//// 4eme cas ////
phi = 3*%pi/4;
[u1,v1,u2,v2] = indices_voisins(u,v,phi);
// on attend (u1,v1) = (6,6) et (u2,v2) = (4,4)
```

Exercice 8 :

Mise en oeuvre de l'algorithme.

- Dans le fichier `contours.sci`, ajoutez une fonction `contours_max` avec la signature suivante :
  `function im_out = contours_max(im, p)`
  Complétez-la pour qu'elle prenne en argument une image `im` et un pourcentage `p`, calcule l'image des contours par seuillage, applique l'algorithme détaillé plus haut pour éliminer les non maxima locaux, puis renvoie l'image des contours.
- Vous pouvez la tester avec l'image de l'exercice 5, vous devez obtenir comme contours deux traits verticaux d'épaisseur égale (en faisant abstraction des bords de l'image).
```
exec('contours.sci', -1);
p = 0.8;
// im est l'image de l'exercice 1
afficher_image(contours_max(im, p));
```
- Pour tester la bonne prise en compte des directions du gradient, vous pouvez réaliser le test suivant :
```
exec('contours.sci', -1);
p = 0.8;
// la fonction disque est fournie dans contours.sci
im = disque();
afficher_image(contours_max(im, p));
```
  Vous devez obtenir un cercle.

7 - Bruit, lissage, et détection des contours

Exercice 9 :

Reprenez les images de l'exercice 4 et les valeurs de `sigma` et `p` que vous aviez choisies, et testez l'effet de l'élimination des non maxima locaux en comparant avec les résultats précédents.

Exercice 10 :

Appliquer le masque flou vu dans le dernier exercice du TP3 avant d'appliquer votre méthode de détection de contours. Que constatez vous ?