Traitement des images numériques

TP 4 : Filtrage, rehaussement de contours et segmentation

Université Paris 13, Institut Galilée
Master Ingénierie et Innovations en Images et Réseaux - 1ère année

2017-2018

Consignes

- Récupérer le fichier `TP4.zip` sur le site

  `http://www.laurentoudre.fr/tin.html`

- Ouvrir MATLAB et créer un répertoire de travail. Dézipper le fichier `TP4.zip` dans ce répertoire.
- A la fin de la séance, récupérer les scripts que vous avez écrits et les envoyer par e-mail au chargé de TP ainsi qu'à vous même afin de les conserver pour la prochaine séance.

Rendu

- Quatre fichiers : `TP4Partie1.m`, `TP4Partie2.m`, `TP4Partie3.m`, `TP4Partie4.m` et `TP4Partie5.m` Chaque fichier doit contenir votre nom, votre prénom et la date.
- Compte-rendu succinct à rendre à la fin de la séance, contenant les observations, commentaires et réponses aux questions. Le compte rendu doit contenir votre nom et votre prénom.

Plan de l'étude

1 Réhaussement de contours

1.1 Rappels de cours

Le but de la détection de contours est de repérer les points d'une image numérique qui correspondent à un changement brutal de l'intensité lumineuse. Ces changements de propriétés de l'image traduisent en général des événements importants ou des formes qu'il est important de détecter. Certaines méthodes de détection de contours utilisent des filtres (linéaires ou pas), qui vont permettre de mettre en évidence les contours : on appelle cette tâche le réhaussement de contours.

Nous allons étudier ici le problème du réhaussement des contours grâce au filtrage linéaire. Ces filtres sont définis par leur masque $h(m,n)$ de taille $M_h \times N_h$. A l'inverse des filtres vus dans le TP3, les filtres réhausseurs de contours sont des passe-bande ou des passe-haut, c'est-à-dire qu'ils laissent les moyennes ou les hautes fréquences presque inchangées et qu'ils atténuent les basses fréquences.

Ces filtres vont être conçus pour mettre en évidence des changements brutaux d'amplitude dans une direction donnée. Nous allons considérer deux types de filtres :

- Les filtres du premier ordre (gradient et Sobel) sont basés sur un calcul de dérivée. Les contours (dans la direction définie par le masque) sont visibles dans l'image filtrée car ils correspondent aux pixels ayant des amplitudes élevées (en valeur absolue)
- Les filtres du second ordre (laplacien et LOG) sont basés sur un calcul de dérivée seconde. Cette fois-ci, les contours correspondent dans l'image filtrée à des transitions entre des valeurs négatives et positives.

Certains filtres vont produire des amplitudes élevées pour les changements foncé vers clair, d'autres pour les changements clair vers foncé, et d'autres pour les deux types de changements. Attention, même si l'image originelle prend des valeurs entre 0 et 1, l'image filtrée peut comporter des coefficients négatifs ou supérieurs à 1. Il faudra donc souvent retraiter l'image filtrée avant de l'afficher (par exemple en en prenant la valeur absolue, ou en utilisant la commande `mat2gray` déjà utilisée dans le TP précédent).

1.2 Etude sous MATLAB

1. Créer sous MATLAB un script vide nommé `TP4Partie1.m`
2. Créer une image `X` de taille 256×256 représentant un carré gris clair sur un fond gris foncé. Le carré se situera vers le milieu de l'image.
3. **Filtres gradients.**

$$h1 = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad h2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

   (a) Appliquer le filtre de masque `h1` et stocker l'image filtrée dans la matrice `Y1`. Afficher sur la même figure `X`, `mat2gray(abs(Y1))` et `mat2gray(Y1)`. Dans quelle direction ce filtre réhausse-t-il les contours ? Est-il sensible aux transitions clair/foncé, foncé/clair ou aux deux ?

   (b) Appliquer le filtre de masque `h2` et stocker l'image filtrée dans la matrice `Y1`. Afficher sur la même figure `X`, `mat2gray(abs(Y2))` et `mat2gray(Y2)`. Dans quelle direction ce filtre réhausse-t-il les contours ? Est-il sensible aux transitions clair/foncé, foncé/clair ou aux deux ?

4. **Filtres de Sobel.**

$$h3 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad h4 = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \qquad h5 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad h6 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$$

   (a) Appliquer le filtre de masque `h3` et stocker l'image filtrée dans la matrice `Y3`. Afficher sur la même figure `X`, `mat2gray(abs(Y3))` et `mat2gray(Y3)`. Dans quelle direction ce filtre réhausse-t-il les contours ? Est-il sensible aux transitions clair/foncé, foncé/clair ou aux deux ? Comparer les résultats avec ceux du filtre `h1`.

   (b) Appliquer le filtre de masque `h4` et stocker l'image filtrée dans la matrice `Y4`. Afficher sur la même figure `X`, `mat2gray(abs(Y4))` et `mat2gray(Y4)`. Dans quelle direction ce filtre réhausse-t-il les contours ? Est-il sensible aux transitions clair/foncé, foncé/clair ou aux deux ?

   (c) Appliquer le filtre de masque `h5` et stocker l'image filtrée dans la matrice `Y5`. Afficher sur la même figure `X`, `mat2gray(abs(Y5))` et `mat2gray(Y5)`. Dans quelle direction ce filtre réhausse-t-il les contours ? Est-il sensible aux transitions clair/foncé, foncé/clair ou aux deux ? Comparer les résultats avec ceux du filtre `h2`.

(d) Appliquer le filtre de masque `h6` et stocker l'image filtrée dans la matrice `Y6`. Afficher sur la même figure `X`, `mat2gray(abs(Y6))` et `mat2gray(Y6)`. Dans quelle direction ce filtre réhausse-t-il les contours ? Est-il sensible aux transitions clair/foncé, foncé/clair ou aux deux ?

5. **Filtre laplacien.**

$$h7 = \frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Appliquer le filtre de masque `h7` et stocker l'image filtrée dans la matrice `Y7`. Afficher sur la même figure `X`, `mat2gray(abs(Y7))` et `mat2gray(Y7)`. Dans quelle direction ce filtre réhausse-t-il les contours ?

6. **Filtre LOG.** Le filtre de Marr et Hildreth, aussi appelé LOG pour Laplacien du[Of] Gaussien est très utilisé pour détecter les contours car il permet de résoudre certains problèmes rencontrés avec le filtre laplacien. En effet, le filtre laplacien réalise une double dérivation, qui est extremement sensible au bruit. L'idée est donc d'appliquer le laplacien non pas directement sur l'image, mais sur une image déjà filtrée par un filtre Gaussien (donc potentiellement débruitée). Le filtre LOG dépend donc d'un paramètre σ qui est l'écart-type du filtre Gaussien appliqué en pré-traitement. On choisit σ de façon à réduire l'effet nuisible dû au bruit.

Pour construire le masque d'un filtre LOG, on utilise comme d'habitude la fonction `fspecial` de MATLAB :

```
h = fspecial('log',[5 5],1); % Filtre LOG de taille 5 x 5 et d'ecart type 1
```

(a) Appliquer un filtre LOG de taille 15×15 et d'écart-type $\sigma = 1$ (dont on stockera le masque dans `h8`) et stocker l'image filtrée dans la matrice `Y8`. Afficher sur la même figure `X`, `mat2gray(abs(Y8))` et `mat2gray(Y8)`. Dans quelle direction ce filtre réhausse-t-il les contours ?

(b) Faire varier la valeur de σ et commenter.

(c) Comparer les résultats du filtre laplacien et du filtre LOG.

7. Relancer tout votre script, en définissant cette fois-ci `X` comme l'image `cameraman.tif` renormalisée. Commenter les résultats obtenus.

8. Appliquer pour chacun des filtres définis ci-dessus, la commande `AffichageFiltrage(X,h)` déjà utilisée au TP3, pour afficher les spectres de l'image originelle, de l'image filtrée ainsi que la réponse en fréquence du filtre. Commenter les réponses fréquentielles des différents filtres. Attention, l'image filtrée présentée ici n'est pas retraitée (pas de valeur absolue ni de `mat2gray`), ce qui explique que tous les coefficients négatifs soient mis à 0 (noir), et que tous les coefficients supérieurs à 1 soient mis à 1 (blanc).

La fonction `AffichageFiltrage(X,h)` fournie réalise les étapes suivantes :

- Prend en entrée une image renormalisée `X`, et un masque de convolution `h`
- Calcule l'image filtrée `Y`
- Affiche sur la même image, de gauche à droite et de haut en bas :
  - L'image originelle *(en haut à gauche)*
  - Le spectre de l'image originelle en échelle linéaire *(en haut au milieu gauche)*
  - Le spectre de l'image filtrée en échelle linéaire *(en haut au milieu droit)*
  - La réponse en fréquence du filtre en échelle linéaire *(en haut à droite)*
  - L'image filtrée *(en bas à gauche)*
  - Le spectre de l'image originelle en échelle logarithmique *(en bas au milieu gauche)*
  - Le spectre de l'image filtrée en échelle logarithmique *(en bas au milieu droit)*
  - La réponse en fréquence du filtre en échelle logarithmique *(en bas à droite)*

2 Segmentation d'image

La segmentation d'image est une opération de traitement d'images qui a pour but de rassembler des pixels entre eux suivant des critères pré-définis. Les pixels sont ainsi regroupés en régions, qui constituent un pavage ou une partition de l'image. Il peut s'agir par exemple de séparer les objets du fond. Il existe de nombreux types de méthodes de segmentation d'image : certaines sont basées sur les contours, d'autres sur un seuillage des pixels en fonction de leur intensité, d'autres découpent l'image en régions connexes , etc....

Nous allons ici considérer une image composée de pièces de monnaie sur un fond relativement uniforme, et tenter de segmenter cette image avec différentes techniques.

2.1 Segmentation par seuillage de valeurs

Nous allons dans un premier temps réaliser une segmentation grâce à un seuillage des valeurs des pixels. Notre but ici est que les pièces apparaissent en blanc et que le fond devienne noir. Nous allons donc créer une image binaire booléenne, composée uniquement de 0 et de 1. Nous allons donc définir deux seuils T_{min} et T_{max} : tous les pixels dont la valeur sera comprise entre T_{min} et T_{max} seront mis à 1 (blanc), tandis que tous les autres seront mis à 0 (noir). La question restante est donc : comment choisir ces seuils ?

1. Créer sous MATLAB un script vide nommé `TP4Partie2.m`
2. Ouvrir l'image `coins.png`, la stocker dans une matrice `X` et la renormaliser. Afficher l'image.
3. Tracer l'histogramme de l'image (avec un pas de 0.001) et identifier les différents pics de l'histogramme. Quelles sont les zones correspondant au fond ? aux pièces ?

Pour calculer et tracer un histogramme d'une image renormalisée sous MATLAB, on peut utiliser :

```
bins = 0:0.001:1;   % Liste des niveaux de gris que l'on veut considerer
[h]=hist(X(:),bins);    % Cree l'histogramme pour les niveaux definis dans le vecteur bins
bar(bins,h);            % Trace l'histogramme sous forme de barres
```

4. En observant l'histogramme, déterminer les seuils T_{min} et T_{max} à utiliser.
5. Créer une image binaire booléenne `X_bin` de même taille que `X`, valant 1 pour les pixels de `X` compris entre T_{min} et T_{max} et 0 sinon. La zone correspondant au fond sera donc noire, et les zones correspondant aux pièces seront blanches. Afficher `X` et `X_bin` sur la même figure.

Pour créer une matrice booléenne sous MATLAB, il suffit d'écrire une condition portant sur une matrice : tous les pixels pour lesquels cette condition est vraie seront mis à 1, et tous les autres à 0.

```
X_bin = (X>0.1);
% X_bin a la meme taille que X : tous les pixels de X superieurs a 0.1
% sont mis a 1 dans X_bin, les autres sont mis a 0
X_bin= (X>0.1 & X<0.3);
% X_bin a la meme taille que X : tous les pixels de X compris entre 0.1 et 0.3
% sont mis a 1 dans X_bin, les autres sont mis a 0
```

6. Tester plusieurs valeurs de T_{min} et T_{max} jusqu'à avoir une segmentation de bonne qualité.
7. Si il reste des points isolés qui ne sont pas de la bonne couleur, nous pouvons essayer d'appliquer un filtrage médian sur l'image `X_bin` grâce à la commande `medfilt2` (comme nous avions fait pour du bruit poivre et sel). Réaliser cette opération : vous devriez obtenir une segmentation parfaite.

2.2 Segmentation par détection de contours

Nous allons maintenant essayer de réaliser une autre segmentation qui va se baser non pas sur les valeurs des pixels, mais sur une détection de contours. Nous n'allons donc pas travailler sur l'image originelle, mais sur une image filtrée ayant des contours rehaussés. Nous allons définir un seuil T : tous les pixels de l'image réhaussée dont la valeur est supérieure à T seront mis à 1, tandis que tous les autres seronts mis à 0. Les contours des pièces seront donc en blanc et le reste en noir.

1. Créer sous MATLAB un script vide nommé `TP4Partie3.m`
2. Ouvrir l'image `coins.png`, la stocker dans une matrice `X` et la renormaliser.
3. Calculer les images `Y1` et `Y2` correspondant au filtrage de `X` respectivement par les masques `h1` et `h2`. Calculer `Y=sqrt(abs(Y1).^2+abs(Y2).^2)` (pseudo-gradient). Afficher `Y` en utilisant `mat2gray`.
4. Tracer l'histogramme de `Y` (avec un pas de 0.001) afin de déterminer le seuil T à utiliser.
5. Créer une image `Y_bin` de même taille que `Y` valant 1 pour les pixels de `Y` supérieurs à T et 0 sinon. Les contours des pièces seront donc blancs et tout le reste de l'image sera noir. Afficher `X` et `Y_bin` sur la même figure.
6. Tester plusieurs valeurs de T jusqu'à voir apparaître tous les contours des pièces.

2.3 Fusion des deux méthodes

Nous avons étudié deux méthodes de segmentation : la première méthode nous a permis de faire une image noire avec les pièces de couleur blanche, alors que la deuxième nous a permis de trouver les contours des pièces. En revanche la deuxième méthode a donné de moins bons résultats : alors comment obtenir des contours de pièces de bonne qualité ? Une des façon de résoudre le problème est de fusionner les résultats. Au lieu de détecter les contours sur l'image originelle, nous allons détecter les contours sur l'image déjà segmentée par la première méthode.

1. Créer sous MATLAB un script vide nommé `TP4Partie4.m`
2. Ouvrir l'image `coins.png`, la stocker dans une matrice `X` et la renormaliser.
3. Reprendre les commandes MATLAB pour générer l'image segmentée finale `X_bin` du script `TP4Partie2.m`
4. Reprendre les commandes MATLAB pour générer l'image segmentée finale `Y_bin` du script `TP4Partie3.m`, mais en prenant en entrée l'image `X_bin` convertie en `double` (et non l'image `Y`).
5. Afficher sur une même figure `X` et `Y_bin` : le résultat doit être parfait

2.4 Décomposition QuadTree pour la segmentation

La décomposition QuadTree permet de représenter une image sous forme de blocs cohérents (souvent carrés), qui peuvent ensuite être fusionnés pour former une segmentation (approche*split & merge*). L'idée est la suivante : on part en général d'une image originelle de taille carrée dont la taille est une puissance de 2 (on rajoute si besoin des pixels supplémentaires si ce n'est pas le cas). On divise l'image en 4 carrés et on regarde si chacun des carrés formés est homogène (selon un critère à déterminer). Si le carré considéré est homogène, on passe au carré suivant. Si le carré considéré n'est pas homogène, on le redivise en 4 carrés plus petits et on étudie l'homogénéité de chacun des 4 petits carrés ainsi formés. Le processus est réitéré jusqu'à ce que tous les carrés soient homogènes, ou que les carrés atteignent une taille suffisamment petite (par exemple 2×2).

Cet algorithme de décomposition prend donc en entrée deux paramètres : la taille minimale d'un carré (critère d'arrêt), et un critère d'homogénéité. On peut par exemple décider qu'un carré est homogène si l'écart type des pixels le composant est inférieur à un seuil. On peut également s'intéresser à l'écart entre le pixel ayant la valeur la plus élevé et le pixel ayant la valeur la plus faible, et décider que le carré est homogène si cet écart ne dépasse pas un seuil.

Une fois cette décomposition réalisée (phase *split*), la segmentation consiste à fusionner (phase *merge*) les carrés qui se ressemblent afin de former une segmentation.

La fonction `splitmerge.m` fournie réalise ces deux étapes et montre d'une part la décomposition en carrés qui a été faite, et d'autre part la segmentation effectuée.

1. Créer sous MATLAB un script vide nommé `TP4Partie5.m`
2. Ouvrir l'image `coins.png`, la stocker dans une matrice `X` et la renormaliser.
3. Tester la fonction `splitmerge.m` sur l'image `X`, en choisissant une taille minimale de 2 pixels, et le critère d'homogénéité par défaut.

> Pour lancer la fonction `splitmerge.m` on utilisera les commandes suivantes :
>
> ```
> taille_min=2;
> splitmerge(X,taille_min,@predicate);
> ```
>
> La fonction `predicate.m` contient le critère définissant la condition pour qu'un carré soit subdivisé. Un exemple est proposé par défaut, mais vous pouvez essayer de faire varier ce critère.

4. Faire varier le critère d'homogénéité dans la fonction `predicate.m` et commenter les résultats.