

Recherche de contours – segmentation

6 octobre 2025 – B. COLOMBEL

Sommaire

1	1 Filtres de Sobel : Exemple de recherche de contours
2	2 Segmentation
2	2.1 Segmentation par seuillage de valeurs
3	2.2 Segmentation par détection de contours
3	2.3 Fusion des deux méthodes
3	3 Seuillage automatique — Méthode d’Otsu
3	3.1 Motivations
4	3.2 Méthode d’Otsu

Dans ce TP, nous utiliserons la bibliothèque `scikit-image` disponible sur les postes de l'IUT dans la distribution `anaconda`.

```
from skimage import data, io, exposure, filters, util
```

⚠ Pour simplifier, nous n'utiliserons que des images **renormalisées** c'est-à-dire que l'on a multiplié les valeurs de chaque pixel pour qu'elles soient comprises entre 0 et 1.

Il suffit pour cela, si l'image est quantifiée sur b bits, de diviser l'image par 2^{b-1} (qui est la valeur maximale possible pour un pixel).

Dans notre cas, travaillant sur 8 bits, on pourra utiliser la commande :

```
coins = io.imread('coins.pgm') // image en uint8
print(coins)
```

```
coins = util.img_as_float(coins) // image normalisée
print(coins)
```

1 Filtres de Sobel : Exemple de recherche de contours

Les filtres de Sobel se trouvent dans le module `filters` de `scikit-image`.

Le filtre détecteur de ligne horizontales est `sobel_h()` et celui des lignes verticales est `sobel_v()`

Exercice 1. Filtres de Sobel.

1. Charger l'image `lena.pgm`, la stocker dans la variable `lena` et la normaliser.
2. Lui appliquer les filtres de convolutions `sobel_h()` et `sobel_v()`. Afficher les deux images `lena_h` et `lena_v` obtenues.
3. Écrire une fonction `grad(I)` en **Python** qui calcule le module du gradient obtenu à partir des convolutions $I_x = I * \text{Sobel_h}$ et $I_y = I * \text{Sobel_v}$.
⚠ Pour visualiser le résultat, les coefficients du résultat doivent être des flottants compris entre 0 et 1 ; pour cela, on peut utiliser la fonction `img_as_float` du module `util`.
4. Afficher l'image obtenue.

Exercice 2.

1. Afficher l'histogramme de l'image obtenue dans l'exercice précédent.
2. Proposer une valeur de seuillage pour mettre en valeur les contours obtenus.
3. Afficher l'image obtenue.

2 Segmentation

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 255. 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 255 (blanc), tandis que tous les autres seront mis à 0 (noir). La question restante est donc : comment choisir ces seuils ?

Exercice 3.

1. Ouvrir l'image `coins.pgm`, la stocker dans une matrice `X`. Afficher l'image.
2. Tracer l'histogramme de l'image et identifier les différents pics de l'histogramme.
Quelles sont les zones correspondant au fond? aux pièces?
3. En observant l'histogramme, déterminer les seuils T_{min} et T_{max} à utiliser.
4. Créer une fonction `Xbin = bin1(X, Tmin, Tmax)` qui retourne une image binaire booléenne `Xbin` de même taille que `X`, valant 255 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 `Xbin` sur la même figure.
5. Tester plusieurs valeurs de T_{min} et T_{max} jusqu'à avoir une segmentation de bonne qualité.
6. S'il reste des points isolés qui ne sont pas de la bonne couleur, on peut appliquer un filtrage médian sur l'image `Xbin`.
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 rehaussée dont la valeur est supérieure à T seront mis à 255, tandis que tous les autres seront mis à 0. Les contours des pièces seront donc en blanc et le reste en noir.

Exercice 4.

1. Ouvrir l'image `coins.pgm`, la stocker dans une matrice Y . Afficher l'image.
2. Appliquer la fonction `grad` de l'exercice 1 pour déterminer le gradient Y_1 de Y .
3. Tracer l'histogramme de Y_1 afin de déterminer le seuil T à utiliser.
4. Créer fonction `Ybin = bin2(Y, T)` qui retourne une image Y_{bin} de même taille que Y valant 255 pour les pixels de Y_1 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.
5. 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 façons 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.

Exercice 5.

1. Ouvrir l'image `coins.pgm`, la stocker dans une matrice Z .
2. Générer l'image Z_1 obtenue par la méthode de la partie 2.1.
3. Générer l'image Z_{bin} obtenue par la méthode de la partie 2.2 en prenant en entrée l'image Z_1 convertie en double (et pas Z).
4. Afficher Z et Z_{bin} sur une même figure : le résultat doit être parfait.

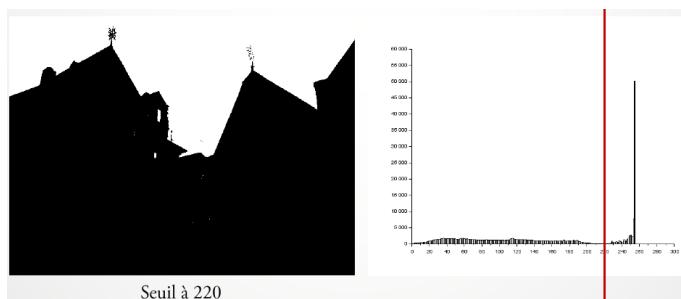
3 Seuillage automatique — Méthode d'Otsu

3.1 Motivations

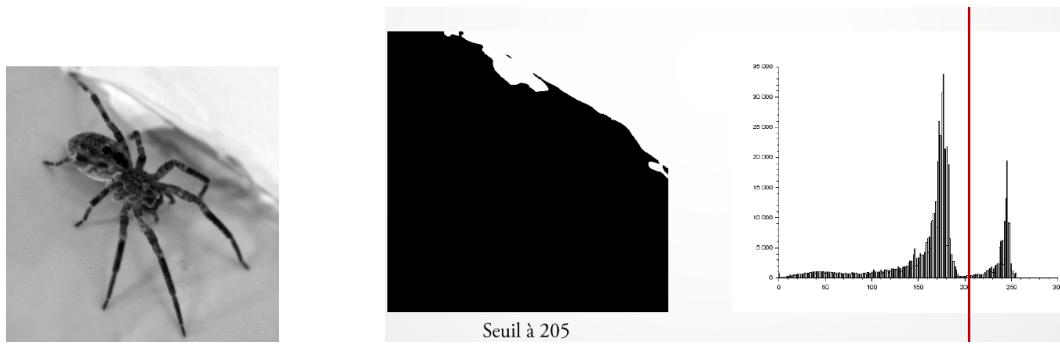
Une segmentation finit souvent par un seuillage.

Plus précisément, réaliser un seuil (`threshold` en anglais) d'une image consiste à placer tous les pixels au dessus d'un certain niveau de gris T à la valeur blanc (255) et tous les autres à la valeur 0. Le procédé paraît simple mais le choix du seuil est parfois complexe comme le montre les exemples ci-dessous.

En examinant l'histogramme d'une image, on peut parfois voir un seuil « évident » apparaître :



Mais « évident » ne veut pas toujours dire « intéressant » :



Une méthode permet de trouver un seuil « sympathique » automatiquement ; elle a été proposé par Obuyuki OTSU en 1979.

3.2 Méthode d'Otsu

La binarisation au moyen du seuillage combiné prend en paramètre un niveau de seuillage t .

Il existe différentes stratégies afin de déterminer un niveau de seuil automatiquement et ainsi avoir une fonction de seuillage sans paramètre.

La méthode d'OTSU fait partie des méthodes les plus connues pour cela. Elle est basée sur une approche classification : on considère que seuiller l'image à un niveau t revient à classifier les pixels de l'image en 2 classes blanc et noir. On peut alors mesurer la qualité d'un niveau de seuillage par la qualité de la classification qu'il génère. Seuiller l'image de manière automatique revient alors à trouver le niveau de seuil qui génère la meilleure classification des pixels.

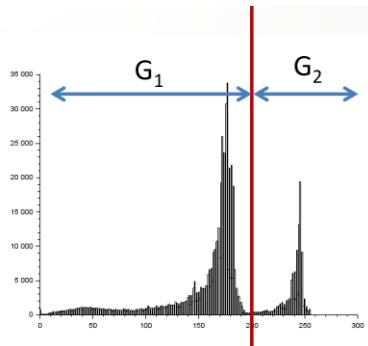
Plus précisément. Lorsque l'on choisit un seuil T , on divise l'histogramme H de l'image en deux groupes G_1 et G_2 .

On peut voir le problème de deux façons différentes :

1. chercher deux groupes où les pixels « se ressemblent » au sein d'un même groupe, c'est-à-dire **minimiser l'inertie intra-classe**;
2. chercher deux groupes les plus dissemblables possibles (dont les moyennes sont les plus éloignées), c'est-à-dire **maximiser l'inertie inter-classe**.

Nous avons vu dans la partie *classification*, que si l'on note

- μ_1 la moyenne et V_1 la variance de G_1 ;
- μ_2 la moyenne et V_2 la variance de G_2 ;
- N le nombre de pixels ;



alors

$$I_{\text{intra}} = \frac{|G_1|}{N} V_1 + \frac{|G_2|}{N} V_2 \quad \text{et} \quad I_{\text{inter}} = \frac{|G_1||G_2|}{N^2} (\mu_1 - \mu_2)^2$$

De plus, on a

$$V = I_{\text{intra}} + I_{\text{inter}} \quad (\text{valeur fixe})$$

Ainsi, le problème revient soit à minimiser l'inertie intra-classe ou à maximiser la variance inter-classe.

Il est un peu plus simple de calculer l'inertie inter-classe que l'inertie intra-classe car la première n'est basée que sur les moyennes.

Exercice 6. On implémentera cette méthode en testant tous les seuils possibles et en retenant celui qui maximise l'inertie inter-classe. Testez votre code avec `PetiteBete.png`.