L3 SI

Traitement d'image

TP N° 2 : Détection de contours

1. Introduction

N.B. : Il est conseillé de lire intégralement le sujet avant de débuter le TP. Pour les besoins de cette séance, il est nécessaire d'importer les modules suivants :

```
>>import numpy as np
>>import cv2 as cv
>>from matplotlib import pyplot as plt
```

Ce TP a pour objet l'évaluation des performances de deux détecteurs de contour décrits dans le cours : l'algorithme de Canny et la méthode basée sur le passage par zéro du laplacien de l'image. L'image test utilisée sera le photophore du tp précédent.

La détection de contour par la méthode de Canny sera réalisée à l'aide d'une fonction d'OpenCV appelée à partir de python par la commande :

```
>>edge = cv.Canny(im, seuil_bas, seuil_haut)
```

La variable **`im`** correspond à l'image source sur laquelle le contour est détecté, les variables **`seuil_bas`** et **`seuil_haut`** sont respectivement les seuils bas et haut de l'algorithme de Canny (c.f. cours). La valeur de **`seuil_bas`** est généralement choisie comme étant égale à : **0,7 x `seuil_haut.`**

On rappelle que la méthode de détection de contour par la méthode du passage par zéro du laplacien est basé sur quatre étapes principales :

1. Calcul du laplacien de l'image **`imlap`** par convolution avec un masque laplacien donné par :

0	1	0
1	-4	1
0	1	0

Pour construire ce masque vous pouvez procéder de la façon suivante :

Créer dans un premier temps une image de zéros de taille 3x3 :

```
>> flt1 = np.zeros((3,3))
```

Modifier ensuite les 5 pixels dont la valeur est différente de zéro de façon individuelle. Exemple :

```
>> flt1[1][1] = -4
```

permet d'affecter la valeur -4 au pixel central du masque.

L'opération de convolution sera ensuite réalisée à l'aide de la fonction **`cv.filter2D()`** (c.f. TP 1).

2. Création d'une image de polarité booléenne **`impol`** qui vérifie **`impol = True`** si **`implap`** > 0.

3. Création d'une image **`imzero`** des passages par zéro :

`imzero(i,j) =` 1 si **`(i,j)`** correspond à une transition (0 1) ou (1 0) de **`impol`** (0 sinon).

4. Le résultat est ensuite affiné en éliminant les points de passage par zéro associés à un gradient faible.

2. Manipulation

2.1. Réalisation du détecteur de contour

Dans cette première partie il est demandé de réaliser une fonction python permettant de calculer l'image des contours par la méthode du passage par zéro du laplacien. On rappelle qu'une fonction python débute par le mot réservé def suivi du nom de la fonction et de ses paramètres entre parenthèses.

Exemple : **`def contour_laplacien(im_src, seuil):`**

La fonction sera écrite dans un fichier texte qui sera chargé par la commande **`import`**.

Le reste du programme est constitué des instructions permettant de réaliser le traitement. Le résultat final sera sauvegardé dans la variable **`result`** qui sera retournée par l'instruction **`return`** à la fin du programme.

Quelques indications.

Afin de réaliser les étapes 2., 3. et 4. de l'algorithme, il est possible d'utiliser avec python des opérations booléennes qui seront appliquées à chaque pixel.

Exemple :

```
>> pos = im  > 20
```

permet de calculer une image **`pos`** pour laquelle les pixels sont des valeurs booléennes correspondant à la règle suivante :

`pos(i,j) = True` si **`im(i,j) > 20, pos(i,j) = False`** sinon.

Pour réaliser l'étape 3 il est nécessaire de comparer l'image **`impol`** avec deux versions décalées d'elle-même, une de un pixel horizontalement et une de un pixel verticalement. Il faut pour cela extraire trois sous images de **`impol`** : **`impol0`** qui correspond à **`impol`** sans la dernière ligne et le dernière colonne, **`impol1`** qui correspond à **`impol`** sans la première ligne et le dernière colonne et **`impol2`** qui correspond à **`impol`** sans la dernière ligne et le première colonne. Les transitions horizontales et verticales (0,1) seront détectées en comparant respectivement **`impol0`** à **`impol1`** et **`impol0`** à **`impol2.`**

Pour extraire une sous image il suffit de préciser les limites inférieures et supérieurs séparées par le caractère **`:`** (deux points) en x et y. Exemple, l'instruction :

```
>> impol_b = impol[0:20, 50:70]
```

permet de créer l'image **`impol_b`** comme une copie des pixels de **`impol`** compris entre le point supérieur gauche (0,50) et le point inférieur droit (20,70).

2.2. Comparaison des détecteurs de contour

Dans cette partie nous allons comparer les deux détecteurs de contour par rapport à leur sensibilité au bruit. Le critère utilisé sera le taux de pixels mal classés.

Pour cela il est nécessaire de calculer au préalable une image de référence en appliquant les deux détecteurs sur l'image du photophore sans bruit :

```
>> refcanny = cv.Canny(im, seuil_haut, seuil_bas)
>> reflapla = contour_laplacien(im, seuil)
```

Ces images de référence seront ensuite comparées aux résultats obtenus (respectivement pour chacun des détecteurs) avec différents bruits. Le taux d'erreur est déterminé en calculant le nombre de pixels mal détectés c'est à dire en sommant les fausses détections et les absences de détection.

Exemple : pour un écart type du bruit **et** = **5,** le taux d'erreur pour l'algorithme de Canny est calculé par :

```
>> difCanny05 = refcanny ¡= canny05
>> tauxCanny05 = difcanny05.sum()
```

N.B. : **`canny05`** est ici l'image des contours calculée pour un bruit d'écart type 5.

Calculer les taux d'erreur pour les deux algorithmes pour des bruits d'écarts types respectifs de 05, 10, 15, 20. En déduire l'algorithme le plus sensible au bruit.