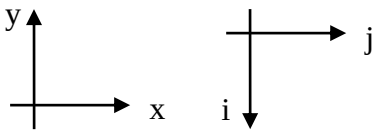
# TP4 : Détection de contours

**Préliminaire important :**
Pour faciliter la lisibilité du code les conventions suivantes sont adoptées.




## 1/ Introduction

Ce TP met en oeuvre différentes techniques de détection de contours d'une image à niveau de gris. Dans l'ensemble du TP on recherchera à calculer des frontières par un maximum local d'une approximation de la norme du gradient de l'image $|\nabla I(x,y)|$. Le but sera donc de pouvoir détecter les contours d'un objet tout en restant robuste face à une source de bruit.

Pour calculer le gradient, une différence finie de dérivation simple (Roberts) dans la direction *x* et *y* sera utilisée. Afin d'améliorer les résultats, la méthode de filtrage de Sobel sera alors utilisée.

Enfin, la méthode de détection des maxima locaux de la norme du gradient sera déjà implémentée, ainsi que le double seuillage à hystéresis. Un algorithme de chaînage spatial permettant d'obtenir les frontières significatives des objets sera à implémenter.

## 2/ Dérivation par simples différences finies

*2.1/ Filtre de Roberts*

Le filtre de convolution le plus simple que l'on puisse utiliser pour évaluer la dérivée est donné par la différence finie suivante sur 2 pixels consécutifs :

$$\frac{\partial I(x,y)}{\partial x} \approx \frac{\Delta I[i,j]}{\Delta j} = I[i, j+1] - I[i, j]$$

$$\frac{\partial I(x,y)}{\partial y} \approx \frac{\Delta I[i,j]}{\Delta i} = I[i+1, j] - I[i, j]$$

L'obtention des 2 gradients peut être réalisée par convolution. Cela peut être directement mis sous forme de convolution à l'aide des noyaux de convolution [1 -1] et $[1\ -1]^T$ (T : transposition pour obtenir le noyau en colonne).

- Compléter la fonction ` « `roberts_differential` » afin de réaliser cette convolution.
- Calculer la norme du gradient de l'image. Former l'image du gradient en x, en y ainsi que celle de la norme.
- Observer leur aspect. Entre quelles valeurs varient ces fonctions ? Observer s'il y a des effets de bord.

*2.2/ Filtre de Sobel*

Afin de limiter l'influence du bruit, on va désormais filtrer (type passe-bas : binomial, ou moyenne) l'image dans la direction perpendiculaire au passage du filtre dérivateur.

L'opérateur de dérivation est cette fois défini comme la différence finie centrée, de noyau de convolution [1 0 -1] (pour une ligne), et l'intégrateur est défini par le noyau binomial [1 2 1] (pour une ligne).

En combinant ces 2 noyaux de convolution, former le filtre de Sobel en *x* et en *y*.

- Réaliser la convolution à l'aide de ces noyaux dans la fonction « `sobel_differential` »,
- Observer l'aspect des composantes du gradient ainsi que de sa norme. Comparer à la norme obtenue précédemment sans filtrage intégrateur

*2.3/ Ajout d'un bruit gaussien*

On ajoute désormais un bruit Gaussien centré *b(x,y)* sur l'image, comme ce qui a été fait au TP3. Pour paramétrer la puissance du bruit (sa variance), on se donne le rapport signal/bruit (exprimé en dB) à avoir sur l'image bruitée. La fonction matlab « `add_gaussian_noise` » réalise cela.

- Ajouter à l'image originale un bruit gaussien d'amplitude tel que son rapport Signal sur Bruit soit de 10dB.
- Observer la norme du gradient ainsi que l'aspect de ces 2 composantes en utilisant les noyaux de Roberts et Sobel. Conclure sur la robustesse des méthodes face au bruit.

**3/ Recherche des maxima locaux dans la direction du gradient**

L'image du gradient peut directement être segmentée afin pour ne garder que l'intensité maximale. Cependant on peut noter que la frontière des objets peut être caractérisée plus précisément par un maximum local de la norme du gradient.

Nous allons donc définir une image binaire donc la valeur 1 correspondra à la présence d'un maximum local de la norme du gradient, selon la direction du gradient.

Pour cela, il faut comparer la valeur de la norme du gradient pour une position (x,y) donnée, à la valeur de cette norme sur des positions en avant et en arrière selon la direction du gradient (direction de plus grande pente de l'image au point considéré).

L'algorithme est donc le suivant (paramètre de voisinage : *d*):

```
calculer grad_x
calculer grad_y
calculer grad_norm

normer grad_x et grad_y

pour tout (x,y) de l'image

  si (grad_norm(x,y)>grad(x+d*grad_x,y+d*grad_y) &&
```

```
        grad_norm(x,y)>grad(x-d*grad_x,y-d*grad_y))

       max_local(x,y)=1

    fin si

  fin pour
```

Cependant, les positions *x +- d×grad(x)* et *y +- d×grad(y)* ne tombent pas forcément sur des valeurs entières (grille de pixels de l'image). Il est donc nécessaire d'interpoler les valeurs du gradient pour ces positions.
Une interpolation au plus proche voisin ou une interpolation bilinéaire peut être réalisée.

La fonction « `keep_local_maxima` » réalise l'algorithme de recherche des maxima locaux. Elle fait appel à une interpolation au plus proche voisin ou à une interpolation bilinéaire.

- Executer la fonction « `keep_local_maxima` » en choisissant l'un puis l'autre interpolation. Comparer et commenter ces 2 résultats.

**4/ Seuillage et chaînage des points de contour**

Une fois les maxima locaux détectés, on utilise un seuillage à hysteresis à double seuil, puis un chaînage en réalisant la méthode de chaînage vue en cours.

Pour cela, on segmente l'image par deux seuils $s_1$ et $s_2$ distincts tels que $s_1 < s_2$. Trois cas peuvent se produire :

1. Les pixels tels que $|\nabla I(x,y)| > s_2$ sont considérés comme de vrais maxima. Ils sont notés PC
2. Ceux inférieurs à $s_1$ sont définitivement éliminés. Ils sont notés NPC
3. Et enfin, ceux compris entre s1 et s2 ne sont considérés sont des points de contours candidats. La décision finale s'ils sont ou non points de contours est reportée au moment du chaînage. Ils sont notés PCP.

Par une méthode automatique, la fonction « `hysteresis_segmentation` » réalise cette segmentation en 3 classes des points de maxima de la norme du gradient dans l'orientation du gradient:

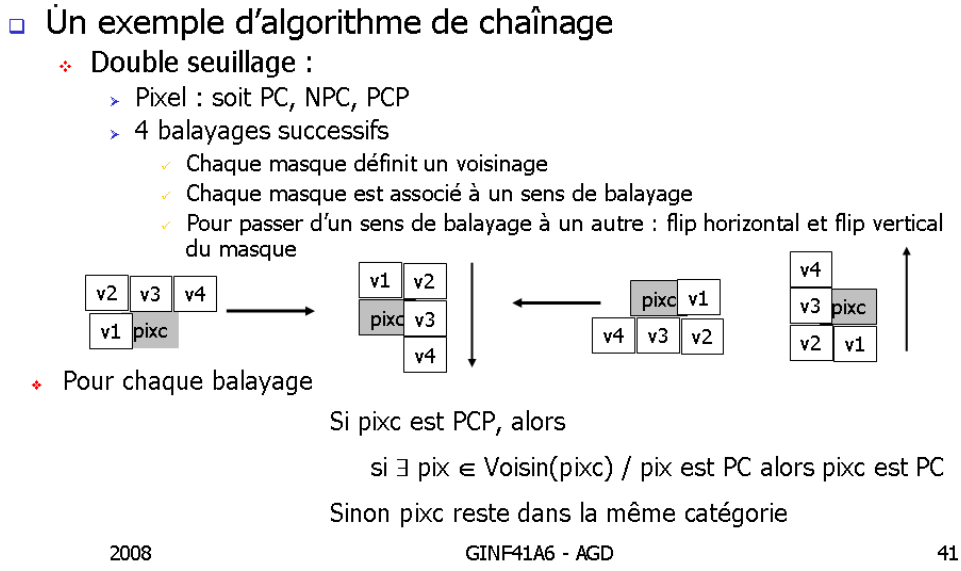
1. les pixels pour lesquels cette norme est inférieure au seuil bas,
2. ceux pour lesquels elle est supérieure au seuil haut
3. et ceux intermédiaires entre ces 2 seuils.

La fonction « `hysteresis_segmentation` » renvoie la carte binaire des pixels « PC » et des pixels « PCP ». La fonction de seuillage automatique est réalisée par « `segment_picture` ». Elle utilise la segmentation de l'histogramme des valeurs des maxima locaux de la norme du gradient par l'algorithme de « KMoyennes » (« KMeans »).

Le chaînage a pour objectif d'éviter les points de contours isolés tout en fermant correctement le bord des vrais objets. Les pixels de type « PCP » deviendront « PC » à la condition qu'il existe un chemin (au sens du voisinage V8-au 8 plus proches voisins- ici) les reliant à un pixel déjà « PC ».

Le travail à faire consiste à réaliser l'algorithme de chaînage. Pour le chaînage, il est nécessaire de réaliser les 4 passes (balayages de l'image) du chaînage vu en cours afin de déterminer quels pixels font définitivement partis ou non du contour. Voir ci-après, l'explication de l'algorithme à implémenter.

- Compléter la fonction « `link_picture` » afin de réaliser les quatre passages du chaînage.




L'algorithme est le suivant :

```
initialiser edges_pixels (I[i,j]>s1)
initialiser is_edges_pixels (s2<I[i,j]<s1)

%passe n.1 (utiliser le masque de voisinage 1)
pour tout i = [1:N_i]
      pour tout j = [1:N_j]
            si is_edges_pixels[i,j]=1
                  pour tous les voisins [i_v1,j_v1] du masque 1
                     si edges_pixels[i_v1,j_v1]=1 alors edges_pixels[i,j]=1
                  fin pour
            fin si
      fin pour
fin pour

%passe n.2 (utiliser le masque de voisinage 2)
pour tout j = [N_j:1]
      pour tout i = [1:N_i]
.
.
.
%passe n.3
pour tout i = [N_i:1]
      pour tout j = [N_j:1]
.
.
.
%passe n.4
pour tout j = [1:N_j]
```

```
      pour tout i = [N_i:1]
.
.
.
```

**5/ Exécution de la chaîne complète de détection de contours**

- Observer les contours obtenus par les différentes méthodes (Roberts et Sobel) en faisant varier le niveau de bruit.
- Faire plusieurs simulations en changeant la valeur du paramètre *d*. Interpréter les résultats obtenus.