

## PARTIE I – QUELQUES FILTRES AVEC OPENCV

### I.1. FILTRES DE CONVOLUTION - FILTRE MEDIAN – FILTRE BILATERAL

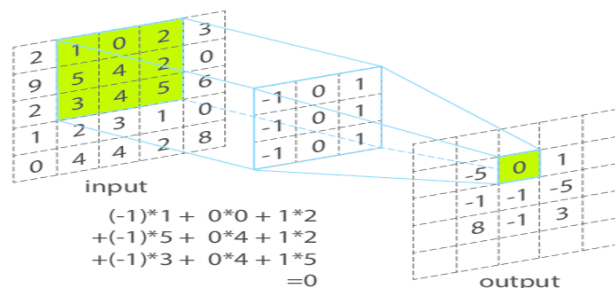
Rappels issus du site de B. Perret<sup>1</sup>, des compléments sont aussi disponibles dans le cours TIVO de Master 1 sur Arche accessible par tous.

Formellement, le **produit de convolution** est une opération entre deux images en niveau de gris  $f$  et  $g$ , notée  $f*g$  défini par :

$$\forall (x,y) \in \mathbb{Z}^2, (f*g)(x,y) = \sum_i \sum_j f(i,j)g(x-i,y-j)$$

Dans cette expression on peut reconnaître :

- $g(x-i,y-j)$  : qui comporte une opération de translation par le vecteur  $(x,y)$  et une symétrie centrale (inversion des coordonnées  $i$  et  $j$ ) : il s'agit du déplacement de l'image  $g$  à la position  $(x,y)$  ( $g$  joue donc le rôle de fenêtre glissante),
- la multiplication *point à point* de  $f$  par la translatée de  $g$  : c'est l'opération de pondération,
- la sommation du tout : c'est l'opération de moyennage.



Le produit de convolution est une application bilinéaire, associative et commutative. C'est-à-dire que pour toutes images  $f, g, h$  et pour tout scalaire  $\lambda$ , on a :

- $f*(g+\lambda h) = (f*g) + \lambda(f*h)$

<sup>1</sup> <https://perso.esiee.fr/~perretb/I5FM/TAI/convolution/index.html> -

- $(f*g)*h=f*(g*h)$
- $f*g=g*f$

Le choix du **noyau de convolution** va permettre d'obtenir différents effets.

L'idée du **filtre médian** reprend le principe de la fenêtre glissante mais ne s'intéresse pas à une combinaison linéaire des valeurs des pixels dans la fenêtre (contrairement au produit de convolution). Au lieu de cela, on va trier les pixels situés sous la fenêtre par ordre de niveau de gris croissant et prendre le pixel se trouvant au milieu : on parle d'élément **médian**. Par exemple, si on considère la liste de valeur (1,3,6,9,223), cette liste est triée et comporte 5 éléments ; le médian de cette liste est le 3ème élément, c'est-à-dire le 6. Par rapport à la moyenne, le médian a l'avantage d'être robuste aux valeurs extrêmes qui ne sont pas forcément représentatives de la distribution des valeurs dans la liste. Dans le cas précédent, la moyenne vaut 48.4, elle est fortement influencée par la valeur 223 qui n'est pas vraiment représentative du reste des éléments de la liste. Alors que le médian à 6 semble effectivement mieux représenter les valeurs de la liste. Exemple ci-dessous de calcul avec un filtre médian de taille 3x3 :

24	32	128	240	255
12	42	111	154	222
4	23	123	176	243
15	155	145	134	172
27	12	98	75	143

→

		134		

$$23 \leq 42 \leq 111 \leq 123 \leq 134 \leq 145 \leq 154 \leq 155 \leq 176$$

L'opération de convolution est efficace pour débruiter une image et lisser les zones texturées. Malheureusement, elle a également une forte tendance à flouter les contours. Il existe une variation de l'opération de convolution appelée **filtre bilatéral** qui résout ce problème en introduisant une seconde pondération afin de ne pas donner trop de poids à un pixel dont la valeur est éloignée de la valeur du pixel courant. Cette méthode est décrite sur de nombreux sites Web, par exemple :

[https://people.csail.mit.edu/sparis/bf\\_course/slides/03\\_definition\\_bf.pdf](https://people.csail.mit.edu/sparis/bf_course/slides/03_definition_bf.pdf)

## I.2. EXERCICES

On considère les images du fichier *imagesFiltres.zip*.

1. Créer un fichier **filtrage\_Q1\_TP.cpp** afin de pouvoir tester sur chacune des images les filtres moyenneur (**blur**), gaussien (**GaussianBlur**) et médian (**medianBlur**). Consulter la documentation *opencv*<sup>2</sup> et un exemple d'utilisation dans le tutoriel<sup>3</sup>

**CR** - Remplir le tableau ci-après en y incluant les images filtrées.

<sup>2</sup> [https://docs.opencv.org/3.4/d4/d86/group\\_imgproc\\_filter.html](https://docs.opencv.org/3.4/d4/d86/group_imgproc_filter.html)

<sup>3</sup>















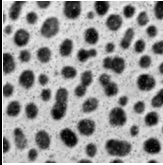
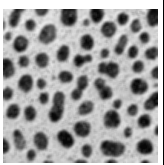
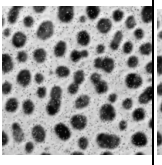
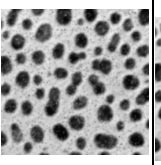
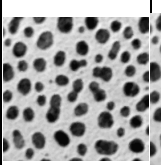
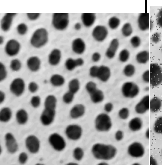
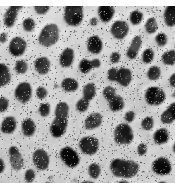
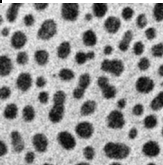
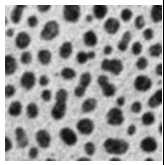
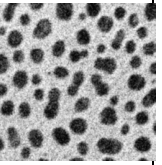
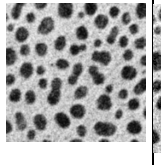
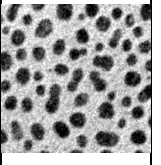
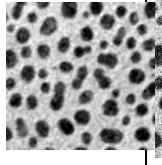
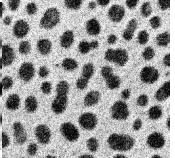
[https://docs.opencv.org/3.4.2/dc/dd3/tutorial\\_gaussian\\_median\\_blur\\_bilateral\\_filter.html](https://docs.opencv.org/3.4.2/dc/dd3/tutorial_gaussian_median_blur_bilateral_filter.html)

### **CR -** Que peut-on déduire de ce tableau au sujet du comportement de ces filtres ?

Le filtre moyenneur prends un ensemble du pixels et calcule le valeur moyenne dans tout les valeurs de la fenetre de convolution. Le resultat est une image avec moins bruit, mais les contours ne sont pas preservé. On peut voire dans les images que les contours sont en fait "blurred": le valeur moyenne sur le contour est gris, parce-que il y a le meme nombre de pixel blanc et noir.

Le filtre Gaussien est aussi un type de filtre moyenneur, mais les valeurs sont ponderé en fonction de la distance du pixel. Le resultat est une image avec moins bruit, dont les contours sont preservé plus que avec le filtre moyenneur. Pour exemple dans l'image "carre" le contour est preservé presque intact.

Le filtre médian calcule pour chaque pixel le valeur median de lui et ses voisins. De ce trois filtres il est le mieux pour éliminer de bruit "à pois", comme dans l'image "NoisyMultBlobs", et aussi pour préserver les contours. Il introduit des nouveaux erreurs dans quelque type d'image: dans l'image "carre" il change les coins (ils sont plus rond, parce que une part est éliminé) et dans l'image "casseroles" l'arrete diagonal est du tout supprimé.

Images	Filtre moyenneur 3x3	Filtre moyenneur 5x5	Filtre Gaussien 3x3	Filtre Gaussien 5x5	Filtre médian 3x3	Filtre médian 5x5	Filtre bilatéral 5X5
carre							
casserole							
NoisyMultBlobs							
NoisyAddBlobs							

- Compléter le fichier créé en 1. : ajouter un filtre bilatéral (*bilateralFilter*), tester plusieurs paramètres sur les fichiers *snack-orig.png* et *onion-ori.png* (consulter la documentation opencv<sup>4</sup> pour le choix des paramètres).

**CR** –Préciser l’effet de ce filtre en illustrant avec les images filtrées.

Le filtre bilatéral est un type de filtre gaussien qui calcule aussi la similarité dedans les pixels. L'effet est une suppression du textures. Dans l'image ici on peut voire que les détails sur les onions ne sont plus visible, mais les contours et les ombres sont préservé. C'est pourquoi le couleur d'un objet est d'habitude different que le couleur des objets proches et aussi different de son part ombreuse. Dans l'image en fait le contour de l'onion plus proche au pain est le plus brouillé.

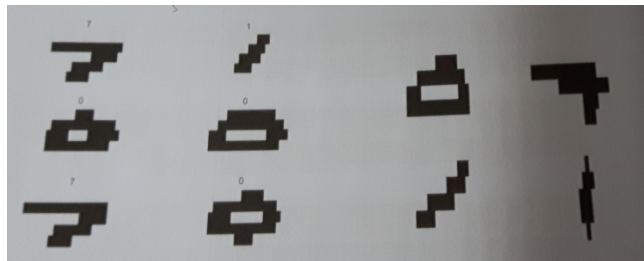


<sup>4</sup>[https://docs.opencv.org/3.4/d4/d86/group\\_imgproc\\_filter.html\\_ga9d7064d478c95d60003cf839430737ed](https://docs.opencv.org/3.4/d4/d86/group_imgproc_filter.html_ga9d7064d478c95d60003cf839430737ed)

## PARTIE II – COMPARAISON DE CONTOURS – DISTANCE D'EDITION - CODE DE FREEMAN

1. Programmer l'algorithme vu en cours d'extraction du contour de Freeman d'un objet dans une image binaire. Le tester sur les fichiers proposés sur Arche à la séance 1.
2. Faire une recherche sur les algorithmes de *Boyer et Moore (77)* et de *Wagner et Fischer(74)*. **CR - Donnez-en les principes. Quelles en sont les limites selon vous dans la comparaison de contours codés avec le code de Freeman ?**
3. **Etude de l'algorithme de Wagner et Fischer.** Un code en Matlab de cet algorithme est disponible à l'adresse :  
<https://fr.mathworks.com/matlabcentral/fileexchange/17585-calculation-of-distance-between-strings>.

On considère les images de la figure ci-contre dont vous trouverez les matrices ou les fichiers pgm dans le fichier *imagestests.zip*.



- a. Déterminer les codes de Freeman des contours de chaque image (utiliser pour cela le programme créé en 1.). Remplir ensuite le tableau ci-dessous en indiquant **la distance d'édition**, résultat de l'algorithme de Wagner et Fischer, dans les cases vides entre les codes des contours concernés. Afin de diminuer l'impact des différentes longueurs des codes des contours, vous pourrez diviser chaque distance obtenue par la somme ou par le maximum des longueurs des deux codes comparés. Prendre un poids de 1 pour chaque opération substitution/insertion/suppression.

**CR - Mettre dans le compte-rendu 2 tableaux de résultats : sans et avec pondération.**  
**Commenter les résultats obtenus.**

Sans pondération:

	xt0_1	xt7_1	xt1_1	xt1_2
xa7_1	23	13	25	36
xa1_1	24	31	12	10
xa0_1	7	22	15	26
xa0_2	11	17	20	30
xa7_2	29	14	31	43
xa0_3	9	22	13	24

- b. Proposer une variante du calcul dans l'algorithme en mettant un poids différent selon les codes de Freeman au niveau des substitutions/insertions/suppression. Commenter les résultats obtenus. Quelle solution vous semblerait mieux adaptée ?

2)

#### BOOYER-MOORE:

L'algorithme calcule les instances d'une substring dans un text. La substring est placé au debut du text. Les caractères sont comparé à partir de la fin de la substring. Si tous les caractères sont pareil, une instance est trouvé. Si non, la substring avance à d'apres de deux regles: la regle de mauvais caractère et la regle de bon suffixe. La première déclare quoi si la substring et le text sont pareil avant de un caractère 't', la substring avance jusqu'a le caractère est pareil et du nuveaux les caractères sont comparé à partir de la fin. La regle de bon suffixe dit quoi, si la substring et le text sont pareil avant de un caractère 't', la substring avance jusqu'a une autre part, egale à le bon suffix déjà trouvé, est trouvé.

Limit pour le contour de Freeman:

le codage de Freeman utilise un petit nombre de caractères, donc dans deux codages il y a une grande probabilité de avoir beaucoup de suffixes pareil (comparé au recherche dans un text). Ca veut dire quoi la complexité de l'algorithme est proche au cas pire  $O(nm)$  (le cas de text et substring avec un caractère). Plus les caractères sont pareil, plus l'abilité de l'algorithme de avancer la substring se réduit.

#### WAGNER-FISCHER:

L'algorithme calcules la distance entre deux mots avec la distance de Levenshtein. La distance de Levenshtein utilise les operations de supprimer, ajouter ou substitution des caractères. L'algorithme produces une matrice avec le prix des changement et les operations nécessaires, avec des etapes itératives. Pour exemple, pour les mots "AB" et "CD", il calcule la distance de A à C, de AB à C, et de AB à CD.

Le problème pour le contour de Freeman est que la distance de Levenshtein ne donne pas des informations géométrique et aussi les operations de supprimer, ajouter et substitution sont des critères pas dépendant de la forme originale.

3)

Avec pondération:

	xt0_1	xt7_1	xt1_1	xt1_2
xa7_1	0.30	0.16	0.37	0.72
xa1_1	0.48	0.56	0.29	0.42
xa0_1	0.10	0.31	0.25	0.63
xa0_2	0.15	0.22	0.32	0.67
xa7_2	0.35	0.16	0.41	0.75
xa0_3	0.13	0.31	0.22	0.59

Les deux tableaux sont assaiz similaires concernant la reconnaissance de l'élément plus simil. Dans la colonne de xt7\_1 la relation entre les valeurs sans et avec pondération est presque la meme. Dans la colonne de xt1\_1 le resultat sans pondération est mieux: la figure plus proche est xa1\_1, alors que avec pondération la forme plus proche est xa0\_3 (sans pondération c'est le deuxième mieux resultat: 12 xa1\_1 et 13 xa0\_3). Les colonnes de xt0\_1 et xt1\_2, comme xt7\_1, ont des resultats tres similaires dans les deux tableaux.

Les valeurs pondère sont plus significatif pour faire des comparaisons, parce que ils sont toujours  $< 1$  alors que les valeurs sans pondération n'ont pas un limite supérieur.

b)

En hausse le poid pour la substitution le resultat hausse beacoup pour les formes plus différent et moin pour les formes correct (ou le valeur est le meme).

En hausse le poid pour la insertion/suppression le resultats sont trop dépendant de la longage et ne sont pas fiable. Pour exemple, xa7\_1 - xt1\_1 a presque le meme valeur, mais xa1\_1 - xt1\_1 (le resultat correct) a 60 à la place de 12.