

TD-3 – Introduction au traitement d'images II

Informatique Graphique et Vision

Les exercices suivants seront réalisés en TD avec le langage Python (3.X), et les bibliothèques NumPy, OpenCV (4.X) et Matplotlib.

A. Rappel

Installation de la bibliothèque OpenCV

```
pip install opencv-python
```

Lancer un notebook Jupyter

- Ouvrez le terminal « Anaconda Prompt »
- Saisissez le code ci-dessous :

```
jupyter notebook
```

Inclusion des bibliothèques nécessaires à ce TD dans votre script

```
01. import cv2
02. import numpy as np
03. from matplotlib import pyplot as plt
```

B. Fonctions OpenCV

Vous trouverez ci-dessous des exemples d'utilisation des principales fonctions OpenCV dont vous aurez besoin afin de réaliser ce TD.

Lecture d'une image en couleur à partir d'un fichier

```
01. fichier = "image.jpg"
02. img = cv2.imread(fichier, cv2.IMREAD_COLOR)
```

Affichage d'une image

```
01. cv2.imshow("Titre", img) # la variable img contient une image BGRT
02. cv2.waitKey(0) # a ne pas oublier
```

Morphologie mathématique - opérations d'érosion

```
01. elementStructurant = np.ones((taille_element, taille_element), np.uint8)
02. resultat_erosion = cv2.erode(img, elementStructurant)
```

Morphologie mathématique - opérations de dilatation

```
01. elementStructurant = np.ones((taille_element, taille_element), np.uint8)
02. resultat_erosion = cv2.dilate(img, elementStructurant)
```

Définition d'un noyau

- i. On définit toutes les valeurs

```
Noyau = numpy.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]])
```

- ii. On utilise une fonction NumPy

```
Noyau = numpy.ones((5, 5), np.float32) / 25 #création d'un noyau de 5x5
```

Filtrage avec un masque de convolution

```
01. #image : votre image
02. Noyau = numpy.ones((5, 5), np.float32) / 25 #création d'un filtre moyenneur de taille 5x5
03. img = cv2.filter2D(src=image, ddepth=-1, kernel=Noyau)
```

Filtre médian

```
resultat = medianBlur(imgSource, tailleNoyau)
```

Filtre Gaussien

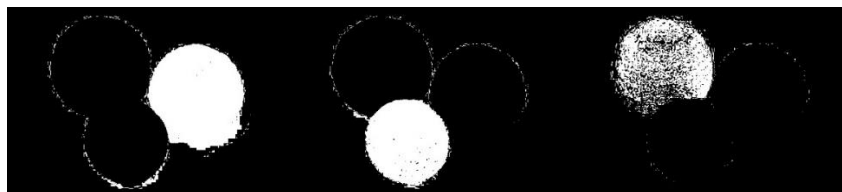
```
cv2.GaussianBlur(imgSource, (tailleNoyauHorizontale, tailleNoyauVerticale), 0)
```

Détecteur de Canny

```
cv2.Canny(imgSource, seuilInferieur, seuilSuperieur, tailleSobel)
```

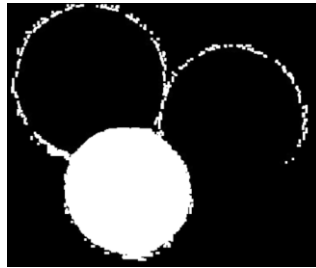
C. Exercices

Exercice 1 – Suppression d'artefacts dans l'image



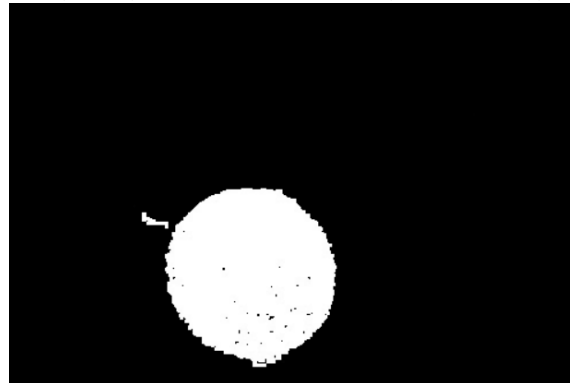
1. Écrivez des fonctions qui appliquent des opérations d'érosion et de dilatation à une image. Montrez le résultat de vos fonctions en utilisant les images ci-dessus ([balle1.jpg](#), [balle2.jpg](#), [balle3.jpg](#)). Analysez l'influence du paramètre « taille de l'élément structurant » sur le résultat (e.g., 5, 7 et 9). Que pouvez-vous conclure à propos de ces opérations et de l'influence de ce paramètre ?

Êtes-vous capable de supprimer les artefacts de l'image sans détériorer l'objet à segmenter ?



- Écrivez deux nouvelles fonctions qui réalisent les opérations d'ouverture (érosion puis dilatation) et de fermeture (dilatation puis érosion) sur une image (binaire), respectivement. Appliquez ces fonctions aux images utilisées dans l'exercice précédent.

Quelle est l'opération et les tailles de noyau qui vous permettent de supprimer les artefacts de l'image et au même temps de préserver le mieux les pixels de l'objet cible ?



Exercice 2 – Filtrage par convolution

- Écrivez des fonctions qui appliquent les filtres définis ci-dessous en utilisant la fonction « filter2D ». Montrez le fonctionnement de vos fonctions avec l'image « [ville.jpg](#) ».

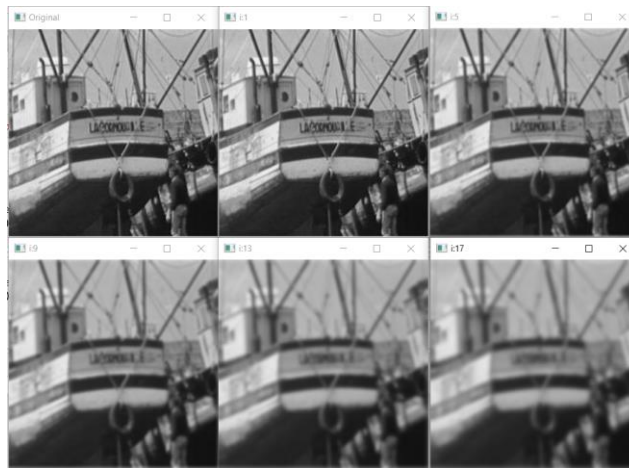
Filtre 1	Filtre 2	Filtre 3
0 0 0	1 1 1	Filtre 2 / 9
0 1 0	1 1 1	
0 0 0	1 1 1	



Pouvez-vous identifier les masques qui correspondent aux filtres identité et moyenneur ?

Exercice 3 - Filtre Gaussien

- Ecrivez une fonction qui applique le filtre Gaussien à une image. La taille du noyau ainsi que l'écart-type à utiliser pour générer le filtre Gaussien seront des arguments de la fonction.
- Utilisez l'image « [Bateau.png](#) » afin de réaliser les actions ci-dessous.
 - Reproduisez l'affichage ci-dessous en variant la taille du noyau du filtre.



2.2 Reproduisez l’affichage ci-dessous en variant l’écart-type utilisé pour générer le filtre gaussien.



Exercice 4 – Bruit sel et poivre

Si vous appliquez le filtre gaussien à l’image « [saltpeppernoise1.png](#) », vous observerez que le résultat obtenu n’est pas satisfaisant. Le filtre médian est plus adapté pour supprimer de bruits de type « sel et poivre ».



1. Écrivez une fonction qui applique le filtre médian à une image. L'image et la taille du noyau seront des arguments de la fonction. Analysez différentes tailles de noyau afin de retrouver un résultat qui s'approche de celui donné ci-dessous.



Exercice 5 – Détection de contour avec le filtre de Sobel

L'objectif de cet exercice est de retrouver les contours de l'image. Nous allons maintenant développer une fonction qu'applique le filtre détecteur de contours de Sobel à une image.

- ❖ Convertissez l'image en niveau de gris.
- ❖ Appliquez un filtre Gaussien de taille 5x5 à l'image.
- ❖ Utilisez la fonction « [Sobel](#) » afin de calculer le gradient horizontal de l'image.
- ❖ Utilisez la fonction « [convertScaleAbs](#) » afin de récupérer la valeur absolue de pixels.
- ❖ Affichez l'image résultat.
- ❖ Répétez la procédure précédente pour calculer et afficher le gradient vertical.
- ❖ Combinez les deux images en utilisant la fonction « [addWeighted](#) » afin de produire une seule image contenant les gradients vertical et horizontal.

1. Utilisez la fonction développée pour extraire les contours de l'image « [ville.jpg](#) » :

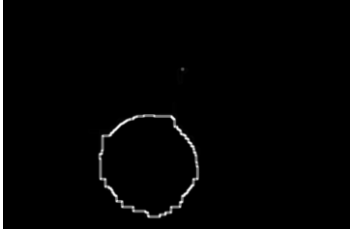
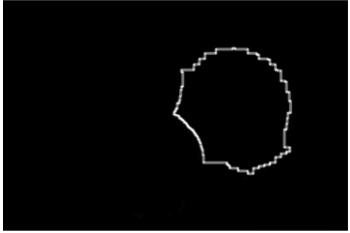
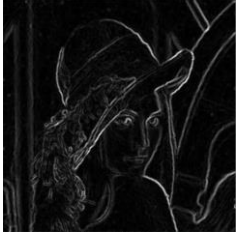



Défis : coder la fonction Sobel en utilisant les filtres vus en cours

Exercice 6 – Extraction de contours

L'objectif de cet exercice est d'extraire les contours des images ci-dessous en combinant les connaissances en filtrage obtenues pendant les exercices précédents. Le tableau ci-dessous présente les images originales (première ligne), ainsi que le résultat attendu (deuxième ligne).

Images originales			

Contours			
			
Balle1.jpg	Balle2.jpg	saltpeppernoise1.png	Bateau.png

Exercice 7 – Détections de contours revisitée – la méthode de Canny

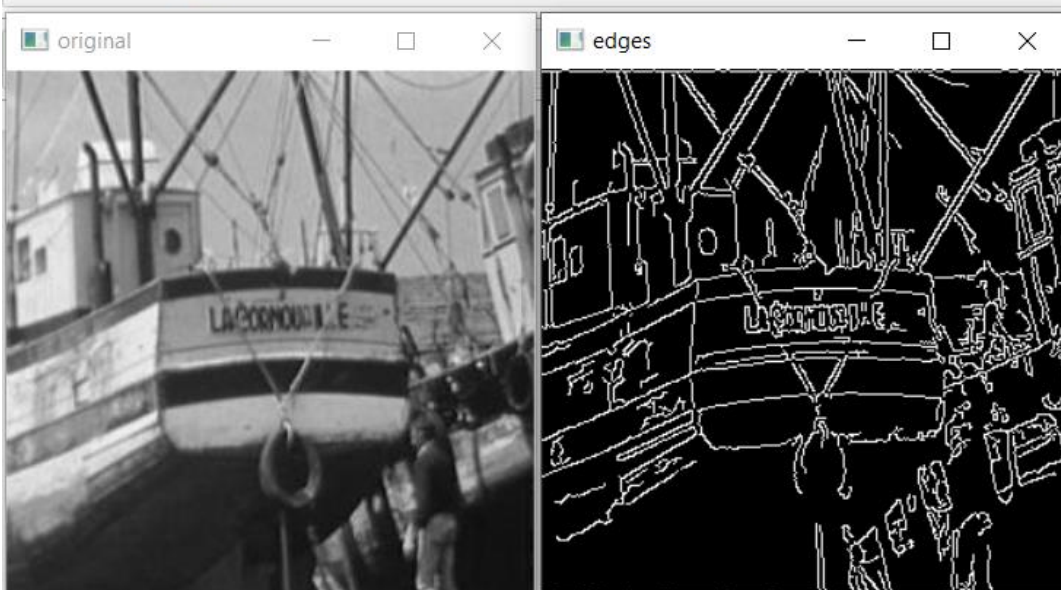
Le détecteur de Canny, proposé par John Canny en 1986, est utilisée dans le cadre du traitement d'images et de la vision par ordinateur afin de détecter les contours d'une image. Il a été conçu afin de répondre à trois critères : bonne détection, bonne localisation, ainsi que s'assurer que pour un contour donné, une seule détection est proposée [Maître, 2022].

Pendant cet exercice, vous allez analyser cette méthode en utilisant la fonction « Canny » proposée par la bibliothèque OpenCV. La méthode de Canny utilise deux seuils afin de détecter les contours dans une image. Le premier seuil, le seuil inférieur, nous sert à supprimer les pixels qui ne sont pas pertinents, tant que le deuxième seuil nous aide à identifier les pixels qui appartiennent à des contours. Les pixels restants, avec une intensité entre les deux seuils, sont analysés de la façon suivante. S'ils sont connectés à de pixels acceptés, ils sont préservés, sinon ils sont supprimés.

1. Utilisez la fonction « *Canny* » afin d'obtenir les contours des images de l'exercice précédent. Vous devez tester différentes valeurs pour les seuils inférieur et supérieur jusqu'à obtenir des résultats que vous jugez satisfaisants.

Obtenez-vous exactement les mêmes contours ?

```
img = cv2.imread('D://L2//Lyon2//Aulas//2022-2023//L3-Vision//images/Bateau.png',0)
edges = cv2.Canny(img,100,200)
cv2.imshow("original", img)
cv2.imshow("edges", edges)
cv2.waitKey(0)
```



Lecture proposée : La détection des contours dans les images par H. Maître

Références :

- ❖ OpenCV, Canny Edge Detection. Consulté le 06 décembre 2022, de https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
- ❖ H. Maître, « La détection des contours dans les images ». Consulté le 30 novembre 2022, de https://perso.telecom-paristech.fr/bloch/TDI/poly_contours.pdf
- ❖ Documentation OpenCV – Filtre de Canny. Consulté le 06 décembre 2022, de https://docs.opencv.org/4.x/dd/d1a/group_imgproc_feature.html#ga04723e007ed888ddf11d9ba04e2232de