

# INTERFAÇAGE NUMÉRIQUE

Travaux Pratiques

Semestre 6

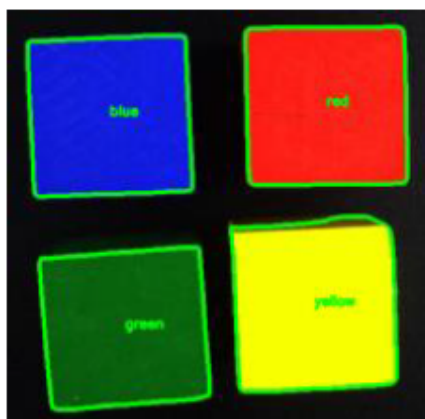
---

## Vision Industrielle

---

TP 2 / Bases de traitement d'images sous OpenCV

---



*Ce sujet est disponible au format électronique sur le site du LEnSE - <https://lense.institutoptique.fr/> dans la rubrique Année / Première Année / Interfaçage Numérique S6 / Bloc 2 Vision Industrielle.*



Lors de cette séance, vous devrez écrire vos propres scripts en **Python** (avec l'IDE PyCharm par exemple) permettant de réaliser des opérations de base de manipulation d'images, à l'aide notamment de la célèbre bibliothèque **OpenCV**.

## Ressources

Un tutoriel sur les bases d'OpenCV est disponible à l'adresse suivante :

<https://iogs-lense-training.github.io/image-processing/>

Des codes en Python, proposant des exemples à tester, sont disponibles sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc 2 Vision Industrielle / Répertoire vers codes à tester*.

*Un fichier archivé, nommé \_STEP\_BY\_STEP.ZIP, regroupe l'ensemble des codes à tester au cours de cette séance, ainsi que les images à traiter.*

Un **kit d'images** est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc 2 Vision Industrielle / Kit d'images*.

## Accumulation de preuves / Méthode de travail

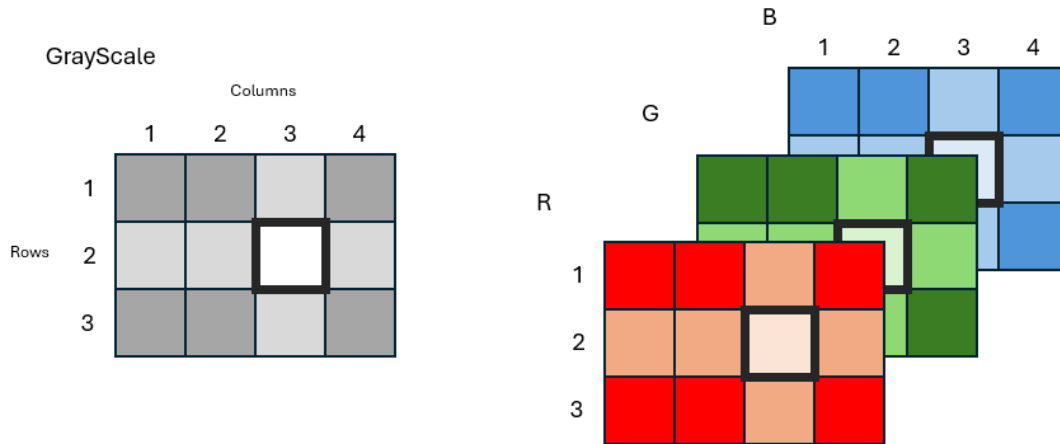
Il est conseillé pour ce TP de **créer un nouveau projet PyCharm** sur votre session (*attention à l'endroit où vous stockerez ce projet - U : sur les sessions Windows de l'IOGS*).

Les différents algorithmes proposés dans ce TP, ainsi que ceux que vous serez amenés à modifier ou créer, pourront vous resservir dans d'autres projets. Nous vous conseillons donc fortement de les **sauvegarder** précieusement et de les **commenter** autant que possible afin de retrouver rapidement les principes mis en jeu derrière les fonctionnalités de OpenCV.

Il serait également pertinent de votre part de rédiger un **journal de bord** sur ce TP en incluant les résultats (images, histogrammes...) et vos analyses des fonctionnalités et de leur intérêt en traitement d'images.

## Rappel sur les images numériques

Une image RVB contient 3 canaux (Rouge, Vert, Bleu ou *RGB* en anglais), tandis qu'une image en niveaux de gris n'en a qu'un. Une image en niveau de gris sera **3 fois plus rapide** à analyser qu'une image en couleur RVB mais toute notion de couleur sera alors perdue.



La couleur des objets peut s'avérer inutile lorsqu'on cherche, par exemple, à détecter des formes particulières ou des contours dans une image.

De nombreux algorithmes d'analyse d'image ou de vision par ordinateur travaillent plus efficacement sur des images en niveaux de gris, permettant notamment d'uniformiser l'entrée des algorithmes et de réduire les informations redondantes liées à la couleur.

## A - Ouvrir une image [20 min]

- M Ouvrir le fichier 01\_OPEN\_IMAGE.PY du répertoire des codes à tester.
- M Exécuter ce code.
- Q Que fait ce programme ? Quelle est la taille de chacune des images ? Quel est le type de données d'un élément ?
- Q Que vaut le premier pixel de chacune des images ? A quoi correspondent les données fournies ?

## B - Calculer l'histogramme d'une image et l'afficher [20 min]

- M Ouvrir le fichier 02\_HISTOGRAM\_IMAGE.PY du répertoire des codes à tester. Exécuter ce code.
- Q Que fait ce programme ? A quoi correspond la valeur affichée pour la ligne `histogram[100]` ?
- Q A quoi correspond la valeur affichée par l'exécution de la ligne `print(np.sum(histogram))` ?

Il peut être intéressant de **créer une fonction qui affiche automatiquement l'histogramme** d'une image à partir de ses données. Elle sera très utile dans la suite du TP pour voir l'impact des effets appliqués sur les images.

## C - Améliorer numériquement la qualité d'une image [20 min]

Pour modifier le contraste et la luminosité d'une image, il faut appliquer une transformation linéaire à **chaque pixel** de l'image pouvant être exprimé mathématiquement comme suit :

$$P_{new} = \alpha \cdot P_{old} + \beta$$

où  $\alpha$  est le facteur de contraste. Une valeur supérieure à 1 augmente le contraste, tandis qu'une valeur entre 0 et 1 le réduit.

$\beta$  est l'offset de luminosité. Une valeur positive rend l'image plus lumineuse, tandis qu'une valeur négative l'assombrit.

On utilise la fonction `cv2.convertScaleAbs()` pour modifier le contraste et la luminosité de l'image.

→ **M** Ouvrir le fichier `03_ENHANCE_CONTRAST_BRIGHTNESS.PY` du répertoire des codes à tester. Exécuter ce code.

→ **M** Modifier le contraste de l'image et comparer les histogrammes de l'image originale et de la version modifiée pour différente valeur de  $\alpha$ .

→ **M** Calculer la moyenne de tous les pixels de l'image originale et comparer à la moyenne des pixels de l'image modifiée.

→ **M** Modifier la luminosité de l'image et comparer les histogrammes de l'image originale et de la version modifiée pour différente valeur de  $\beta$ .

→ **M** Calculer la moyenne de tous les pixels de l'image originale et comparer à la moyenne des pixels de l'image modifiée.

→ **Q** Que pouvez-vous conclure sur les effets du contraste et de la luminosité ?

## D - Binariser l'image [20 min]

La binarisation d'une image consiste à transformer une image en niveaux de gris (ou parfois en couleur) en une image ne contenant que deux valeurs possibles : généralement 0 (noir) et 1 (blanc).

L'objectif de la binarisation est de réduire drastiquement les informations à traiter tout en essayant de séparer clairement le premier plan (objets) de l'arrière-plan.

→ **M** Ouvrir le fichier `04_THRESHOLD.PY` du répertoire des codes à tester. Exécuter ce code. Modifier la valeur du seuil et comparer les images résultantes.

→ **Q** Que fait ce code ? Est-ce que le choix du seuil a un impact sur la valeur résultante pour la méthode d'Otsu ?

→ **M** Tester ces méthodes sur une autre image.

*Vous trouverez un peu plus de détails sur les méthode de binarisation à l'adresse suivante :*

[https://docs.opencv.org/4.x/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html)

## E - Opérations de pré-traitement [40 min]

Les opérations de pré-traitement dans le traitement d'images sont essentielles pour améliorer la qualité des images avant d'appliquer des algorithmes plus complexes, comme la segmentation, la détection d'objets ou la classification. Ces étapes de pré-traitement visent à réduire le bruit ou améliorer la structure de l'image.

Parmi les opérations de pré-traitement classiques, on peut citer :

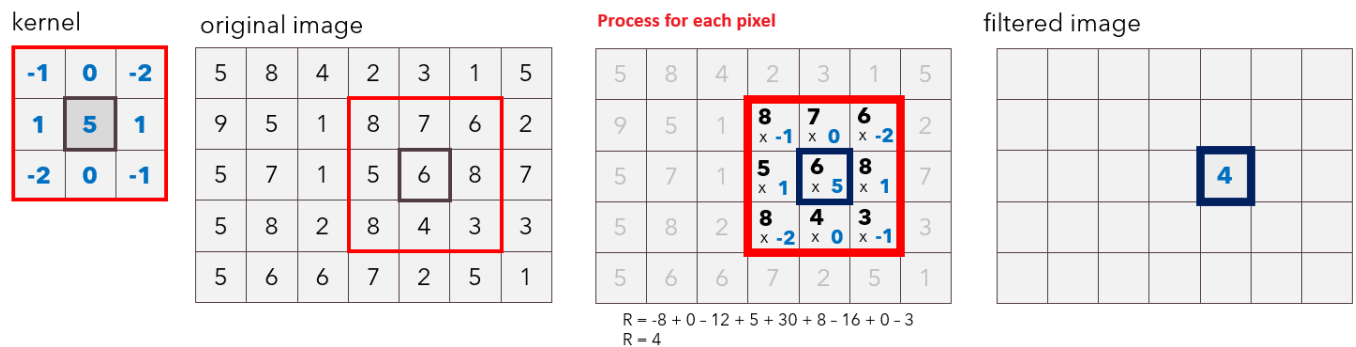
- **Correction des couleurs** : Balance des blancs, Correction gamma, Amélioration de contraste...
- **Réduction de bruit** : Filtrage linéaire pour atténuer les bruits sans trop affecter les détails importants de l'image, Filtrage non linéaire pour éliminer les bruits impulsionnels, Filtrage anisotrope...
- **Opérations morphologiques** : érosion pour éliminer du bruit, dilatation pour combler des lacunes dans les objets, ouverture et fermeture pour enlever les petites anomalies ou remplir les petits trous dans une image
- **Filtrage fréquentiel** pour éliminer ou atténuer des fréquences particulières (comme des motifs de bruit répétitifs)

### Éléments structurants d'une opération morphologique (noyau)

Notions : *Éléments structurants*

Les **transformations dites morphologiques** se basent sur l'application d'un **élément structurant** (ou noyau) que l'on va superposer sur chaque pixel de l'image.

Un **élément structurant** (ou noyau) est une petite matrice (généralement de taille et de forme prédéfinies, comme un carré, un disque, une ligne, etc.) qui sert de sonde pour inspecter et modifier les pixels d'une image.



Les éléments structurants jouent un rôle clé en traitement d'image, notamment dans les opérations de morphologie mathématique. Ces opérations sont principalement utilisées pour analyser et traiter des images binaires ou en niveaux de gris en modifiant leurs formes ou en extrayant des structures spécifiques.

- M Ouvrir le fichier 05\_KERNEL.PY du répertoire des codes à tester. Exécuter ce code.
- Q Quel est le type de des objets noyaux résultants ?
- M Générer un noyau en forme de croix de taille 3 par 3 pixels et afficher ce noyau.
- M Générer un second noyau en forme de cercle (ellipse) de taille 5 par 5 pixels et afficher ce noyau.

## Opérations d'érosion et de dilatation

Notions : *Erosion* - *Dilation*

- M Ouvrir le fichier 06\_EROSION\_DILATION.PY du répertoire des codes à tester. Exécuter ce code.
- Q Quelle est la taille de l'image originale? Quelle est la taille (approximative) des défauts dans cette image?
- Q Quelles sont les opérations effectuées sur l'image *a\_letter\_noise.jpg*? Quels sont les noyaux utilisés pour ces opérations?
- M Modifier la taille des noyaux et tester à nouveau ces opérations.
- Q Que pouvez-vous conclure sur l'utilité des opérations d'érosion et de dilatation sur une image? Sur l'impact de la forme et de la taille du noyau?

*Il est également intéressant de retenir la façon d'afficher plusieurs images sur un même graphique afin de pouvoir les comparer rapidement.*

## Opérations d'ouverture et de fermeture

Notions : *Opening* - *Closing*

- M Ouvrir le fichier 07\_OPENING\_CLOSING.PY du répertoire des codes à tester. Exécuter ce code.
- Q Quelles sont les opérations effectuées sur l'image *a\_letter\_noise.jpg*? Quels sont les noyaux utilisés pour ces opérations?
- M Modifier la taille des noyaux et tester à nouveau ces opérations.
- Q Que pouvez-vous conclure sur l'utilité des opérations d'ouverture et de fermeture sur une image? Sur l'impact de la forme et de la taille du noyau?

## Opération de gradient

- M Ouvrir le fichier 08\_GRADIENT.PY du répertoire des codes à tester. Exécuter ce code.
- Q Quelle est l'opération réalisée sur l'image *grad\_image\_cross\_3*? Quelle est la suite d'opérations réalisée sur l'image *grad\_sub\_image\_cross\_3*?
- Q Que pouvez-vous conclure sur le lien entre ces deux méthodes?
- M Proposer une méthode "fiable" pour comparer facilement ces deux images. Mettre en oeuvre cette méthode.
- M Modifier la taille des noyaux et tester à nouveau ces opérations.
- Q Que pouvez-vous conclure sur l'utilité de l'opération de gradient sur une image? Sur l'impact de la forme et de la taille du noyau?

## F - Calculer la FFT d'une image [30 min]

- M Ouvrir le fichier 09\_FFT.PY du répertoire des codes à tester. Exécuter ce code.
  - Q Quelle est la taille de l'image initiale ? Quel est le type et quelle est la taille de l'objet *fft\_image* ? De quel type sont ses éléments ?
  - Q Expliquer l'intérêt de l'affichage du logarithme de la valeur absolue de la FFT.
- 
- Q Que permet de faire la fonction *circular\_mask* sur la FFT ? Commenter le résultat obtenu sur l'image recréée à partir de cette FFT modifiée.
  - Q Quel est l'impact de la taille du masque ? A quoi sont dues les "vaguelettes" que l'on aperçoit sur l'image résultante ?
  - Q Que se passe-t-il si on applique un masque rectangulaire ?
- 
- M Générer une trame sinusoïdale en deux dimensions (200 x 300 pixels) à l'aide de la fonction *sine\_trame* de la bibliothèque *images\_manipulation*. Calculer sa FFT et afficher les résultats.
  - Q Commenter le résultat obtenu pour différentes périodes spatiales et différents angles.

## G - Appliquer un filtre moyenneur sur une image [30 min]

L'utilisation de filtres permet de prendre en compte les pixels voisins en exploitant les relations locales dans une image, ce qui est crucial pour des tâches comme la suppression de bruit, la détection de contours, l'extraction de caractéristiques, et l'amélioration de la qualité visuelle. Travailler sur des pixels isolés limite l'analyse à des informations ponctuelles, tandis que considérer les voisins permet une compréhension plus riche et contextuelle de l'image.

Notions : *Blur with OpenCV*

- M Ouvrir le fichier 10\_BLUR\_MEAN.PY du répertoire des codes à tester. Exécuter ce code.
- Q Commenter les effets des deux méthodes utilisées pour lisser l'image.
- Q Quelle méthode est utilisée ici pour montrer de manière objective les modifications apportées à l'image ?

Le fichier 10A\_BLUR\_MEAN\_SLICE.PY propose une autre méthode pour comparer les effets du lissage.

- M Ouvrir le fichier 10A\_BLUR\_MEAN\_SLICE.PY du répertoire des codes à tester. Exécuter ce code.
- Q Quelle est cette méthode ?
- M Calculer et afficher la FFT de chacune des images précédentes (originale et modifiées).
- Q Quelle est la fonction réalisée par les deux filtres précédents ?

On cherche à présent à voir l'impact de la taille du noyau sur l'image finale.

- M Ouvrir le fichier 10B\_BLUR\_MEAN\_KERNEL.PY du répertoire des codes à tester. Exécuter ce code.
- Q Que pouvez-vous conclure sur l'impact de la taille du noyau ? Vous pourrez également comparer les FFT...



## H - Détecter une ligne [20 min]

- M Ouvrir le fichier 11\_LINE\_DETECTION.PY du répertoire des codes à tester. Exécuter ce code.
- Q Commenter les résultats. Quel type de masque est utilisé ? Quelle est sa taille ?
- M Modifier le noyau pour détecter des lignes d'au moins 40 pixels.

## I - Appliquer un filtre passe-haut sur une image [40 min]

Le **filtre moyennneur** (vu précédemment) permet de conserver les **éléments à basse fréquence spatiale** dans l'image. C'est une méthode intéressante pour supprimer des bruits ponctuels (des éléments isolés et donc "rapides"). Il est également possible en choisissant un autre élément structurant de réaliser l'opération complémentaire qui supprime le fond continu et ne conserve que les transitions de fréquence spatiale élevée (bords d'un objet par exemple).

Il est possible d'utiliser la fonction `cv2.filter2D()` pour appliquer un noyau particulier sur une image.

### Opérateur de Roberts

L'**opérateur de Roberts** est l'un des premiers filtres de **détection de contours**. Il repose sur la convolution avec deux petits noyaux 2x2, conçus pour approximer les dérivées en diagonale de l'image.

Les noyaux de convolution sont les suivants :

$$K_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}, \quad K_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}.$$

Il est alors possible de calculer l'amplitude du gradient par l'opération suivante :

$$\text{Amplitude} = \sqrt{G_x^2 + G_y^2}$$

où  $G_x$  est le résultat de la convolution de l'image par le noyau  $K_x$  et  $G_y$  le résultat de la convolution de l'image par le noyau  $K_y$ .

Une forte amplitude indique un contour ou un bord marqué. Une faible amplitude indique une région où l'intensité est relativement constante.

- M Ouvrir le fichier 12\_ROBERTS.PY du répertoire des codes à tester. Exécuter ce code.
- Q Commenter les résultats. Que pouvez-vous conclure sur l'effet de ce filtre ?
- M Appliquer cet effet sur l'image *noise\_vi\_small.png*.
- Q La détection de contours est-elle optimale ?

---

→ M Appliquer un filtre gaussien à l'aide d'un noyau de taille 11 x 11 et d'écart-type de 2. Appliquer l'effet précédent à l'image résultante.

- Q La détection de contours est-elle améliorée ?

## Opérateur de Sobel

L'**opérateur de Sobel** permet de réaliser une opération similaire à celui de Roberts, mais en étant moins sensible aux bruits dans l'image, puisqu'il se base sur un noyau plus large et ainsi lisse l'image dans la direction perpendiculaire au gradient mesuré.

Les noyaux de convolution sont les suivants :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

De la même façon que précédemment, on peut calculer l'amplitude du gradient par l'opération suivante :

$$\text{Amplitude} = \sqrt{G_x^2 + G_y^2}$$

où  $G_x$  est le résultat de la convolution de l'image par le noyau  $K_x$  et  $G_y$  le résultat de la convolution de l'image par le noyau  $K_y$ .

- M Ouvrir le fichier 12\_SOBEL.PY du répertoire des codes à tester. Exécuter ce code.
- M Visualiser l'image originale, le résultat du filtrage selon X et le résultat du filtrage selon Y.
- Q Commenter les résultats. Que pouvez-vous conclure sur l'effet de ce filtre ?
- M Appliquer cet effet sur l'image *noise\_vi\_small.png*.
- Q La détection de contours est-elle optimale ?

---

→ M Appliquer un filtre gaussien à l'aide d'un noyau de taille 11 x 11 et d'écart-type de 2. Appliquer l'effet précédent à l'image résultante.

- Q La détection de contours est-elle améliorée ?

## Méthodes de Harris et de Canny

L'**algorithme de Canny** est une technique de détection qui a pour objectif d'identifier les bords significatifs d'une image tout en minimisant le bruit.

Cet algorithme comprend les étapes suivantes : **lissage** (réduction du bruit avec un filtre gaussien), **calcul du gradient** (obtention de la magnitude et de la direction des contours), **suppression non maximale** (affinage des contours pour ne garder que les crêtes locales) et **seuils double** (application de deux seuils pour conserver les contours forts et connecter les contours faibles pertinents).

L'**opérateur de Harris** permet de détecter des coins ou points d'intérêt dans une image, utiles pour la reconnaissance d'objets, le suivi de mouvement ou la reconstruction 3D.

Il est basé sur la mesure de la variation de l'intensité d'image dans des zones de l'image (par convolution avec un noyau). Les coins correspondent à des régions où la variation de l'intensité est forte dans plusieurs directions. Enfin un calcul d'une matrice de structure et d'une fonction de réponse R pour identifier les coins significatifs est réalisé.

- M Ouvrir le fichier 13\_HARRIS\_CANNY.PY du répertoire des codes à tester. Exécuter ce code.
- Q Commenter les résultats.
- M Appliquer cet effet sur d'autres images.

## **INTERFAÇAGE NUMÉRIQUE**

### **Travaux Pratiques**

Semestre 6

---

## **Ressources**

---

Bloc Vision Industrielle

**Liste des ressources**

— ???