Institut d'Optique Graduate School
Interfaçage Numérique

Interfaçage Numérique

Travaux Pratiques

Semestre 6

Images et OpenCV

Pré-traitements, masques, filtres et segmentation

---

2 séances

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 Images et OpenCV.

Images et OpenCV

À l'issue des séances de TP concernant le **bloc de traitement d'images avec OpenCV**, les étudiant·es seront capables d'utiliser OpenCV pour manipuler des images et appliquer des traitements simples : erosion/dilatation, filtres de lissage, masques...

Ressources

Un tutoriel sur les bases d'OpenCV est disponible à l'adresse suivante :
https://iogs-lense-training.github.io/image-processing/

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

Des **fichiers de fonctions** sont disponibles sur le site du LEnsE dans la rubrique*Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Répertoire vers codes à tester*.

Quelques exemples et explications sur les différents pré-traitements d'images est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Image Processing with OpenCV*.

Déroulement du bloc

Etape 0 - 30 min Ouvrir une image et faire son histogramme

Etape 1 - 30 min Couleur vers niveau de gris

Etape 2 - 30 min Seuillage et binarisation d'une image

Etape 3 - 60 min Erosion, Dilatation et Gradient

Etape 4 - 90 min Lissage du bruit

Etape 5 - 90 min Isoler des éléments verticaux (ou horizontaux) dans une image grâce à des opérateurs morphologiques spécifiques

Etape 6 - 90 min Détecter des contours et des points d'intérêt

Etape 7 - 90 min Segmenter une image par la méthode de Watershed

Primitives

En traitement d'image, les **primitives** sont les éléments fondamentaux ou les structures de base qui composent une image, sur lesquels des algorithmes peuvent opérer pour effectuer des analyses ou des traitements.

Les primitives servent de **points de départ** pour la reconnaissance d'objets, l'analyse de scène, la segmentation d'image ou la reconstruction 3D. Par exemple, pour détecter un visage dans une image, l'algorithme peut commencer par identifier des primitives simples comme les yeux (points ou régions sombres), puis les relier pour former une structure cohérente.

On peut distinguer 3 catégories de primitives.

Primitives de bas niveau

Ce sont les entités les plus simples extraites directement des pixels de l'image.
Par exemple :

— Points : des pixels isolés ou des points d'intérêt
— Lignes ou segments : des ensembles de pixels alignés détectés par des algorithmes de détection de bord
— Contours : les frontières des objets définis par des changements d'intensité ou de couleur
— Régions : des groupes de pixels connectés ayant des propriétés similaires.

Primitives de niveau intermédiaire

Ces primitives sont obtenues en combinant ou en analysant les primitives de bas niveau.
Par exemple :

— Formes géométriques : rectangles, cercles, polygones
— Lignes ou segments : des ensembles de pixels alignés détectés par des algorithmes de détection de bord
— Objets simples : identification d'objets à partir de leurs contours ou formes.

Primitives de haut niveau

Ces primitives sont plus abstraites et dépendent de la compréhension sémantique de l'image.
Par exemple :

— Objets complexes : reconnaissance d'éléments comme des personnes ou des animaux
— Relations spatiales : liens entre différents objets (par exemple, un objet en avant d'un autre).

Ouvrir une image sous OpenCV et afficher son histogramme

Temps conseillé : 30 min

Notions : *Ouvrir une image* - *Afficher une image* - *Calculer l'histogramme*

➜ **M** Créer un nouveau projet sous PyCharm et impoter la bibliothèque OpenCV2.

➜ **M** Ouvrir et afficher l'image *robot.jpg* du kit d'images fourni, en niveau de gris.

➜ **Q** Quelle est la taille de l'image ? Quel est le type d'un élément ?

➜ **M** Calculer l'histogramme de l'image et l'afficher.

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.*

Couleur vers niveau de gris

Temps conseillé : 30 min

➜ **M** Ouvrir et afficher l'image *robot.jpg* du kit d'images fourni, au format RGB.

➜ **Q** Quelle est la taille de l'image ? Quel est le type de données d'un élément ?

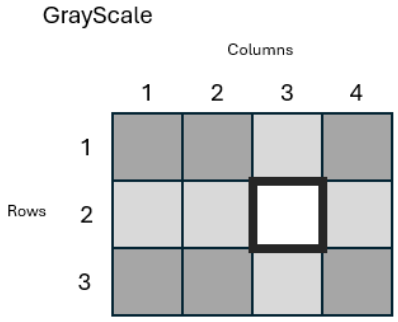
➜ **M** Créer une copie de la matrice image (fonction *copy()* de Numpy).

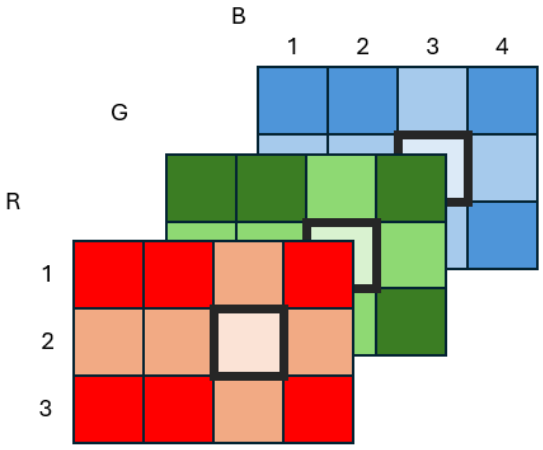
➜ **M** Forcer à 0 tous les pixels du canal bleu de la copie de l'image et afficher la nouvelle image.

➜ **Q** Est-il possible d'afficher un histogramme de l'image ?

RVB vs Niveau de gris

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.

Plusieurs méthodes de conversion

Plusieurs méthodes existent pour passer d'une image RVB à une image en niveau de gris :

— Calculer la **moyenne des valeurs** des trois canaux de couleur (Rouge, Vert, Bleu) pour chaque pixel.
— Utiliser des **poids spécifiques pour les canaux R, V et B**, basés sur leur contribution relative à la perception humaine.
— Convertir l'image dans un **autre espace de couleur**, comme YUV, HSL ou HSV, et extraire la composante de luminosité.

Moyenne des canaux R,V,B

Cette méthode est la plus simple. Chaque pixel de l'image en gris est la moyenne des pixels des canaux rouge, vert et bleu de l'image en couleur :

$$Pixel_{Gray} = \frac{Pixel_R + Pixel_V + Pixel_B}{3}$$

➜ **M** Créer une image en nuance de gris utilisant la méthode de la moyenne des trois canaux.

➜ **M** Afficher l'image résultante.

Pondération en fonction de la perception humaine

Cette méthode est une moyenne pondérée des valeurs des pixels R, V, B de l'image couleur :

$$Pixel_{Gray} = 0.299 \cdot Pixel_R + 0.587 \cdot Pixel_V + 0.114 \cdot Pixel_B$$

Les coefficients de cette méthode proviennent de la sensibilité relative de l'œil humain aux différentes couleurs et ont été standardisés à l'origine pour la télévision analogique (NTSC). Ils sont aujourd'hui utilisés comme une approximation fidèle de la perception visuelle de la luminosité.

La méthode de conversion fournie par la bibliothèque OpenCV se base sur cette pondération.

➜ **M** Convertir l'image RVB en niveau de gris par l'instruction suivante :

```
1 image_gray = cv2.cvtColor(image_rgb, cv2.COLOR_BGR2GRAY)
```

➜ **M** Comparer les images obtenues par la moyenne classique et cette moyenne pondérée.

Utilisation d'un espace colorimétrique différent

L'espace colorimétrique RVB est très utilisé dans le domaine du numérique (affichage, acquisition d'images) pour sa facilité de mise en oeuvre.

Cependant, ce n'est **pas** le plus **adapté vis-à-vis de la perception humaine** où la luminance et la couleur sont séparées.

Des espaces comme YUV, YIQ, ou YCbCr séparent la composante de luminance (Y) des composantes de chrominance (U et V).

➜ **M** Convertir l'image RVB dans l'espace YUV par l'instruction suivante :

```
1  image_yuv = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2YUV)
```

➜ **M** Comparer alors l'image en niveau de gris obtenue par la méthode de moyennage pondérée et le canal Y de cette conversion.

➜ **Q** Que pouvez-vous conclure sur la méthode de calcul utiliser pour la luminance (Y) ?

Il existe d'autres espaces colorimétriques dans le domaine numérique. Voici un résumé non exhaustif :

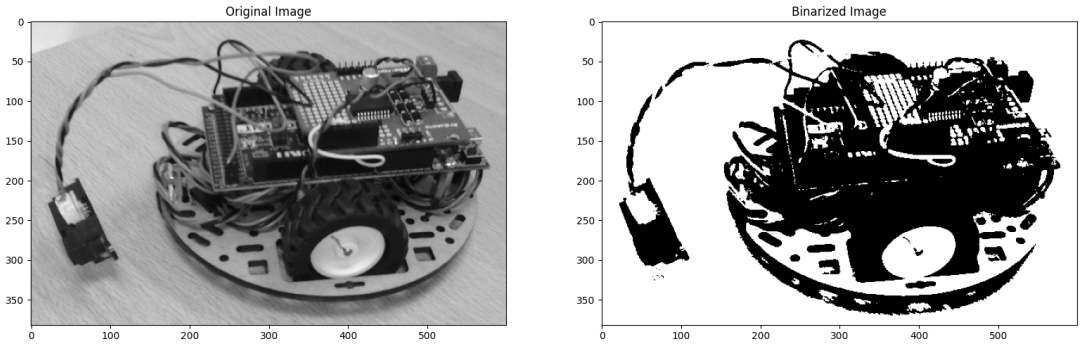
Espace colorimétrique	**Avantages**
RGB	Simple, utilisé pour les écrans et le rendu des couleurs.
HSV / HSL	Intuitif pour manipuler la couleur (teinte, saturation).
YUV / YCbCr	Sépare luminance et chrominance.
CIE-Lab	Uniformité perceptuelle, idéal pour mesurer les différences de couleur.
CMY(K)	Optimisé pour l'impression.
XYZ	Modèle basé sur la perception humaine.

Seuillage et binarisation

Temps conseillé : 30 min

Notions : *Binarisation d'une image* - *Mesurer un temps d'exécution*

Le seuillage (ou binarisation) d'une image est une technique fondamentale en traitement d'image qui consiste à convertir une image en niveaux de gris en une image binaire, composée uniquement de deux niveaux (0 ou 1, ou encore noir et blanc).



Méthode classique

➜ **M** Calculer l'image binarisée de l'image initiale - robot.jpg - (en niveau de gris) avec un seuil de 120, par la méthode CV2.THRESH_BINARY.

➜ **M** Afficher l'image résultante.

➜ **M** Tester également pour différente valeur de seuil. Tester également la méthode CV2.THRESH_BINARY_IN

➜ **Q** Que pouvez-vous conclure sur l'utilisation de cette méthode.

Méthode d'Otsu

La **méthode d'Otsu** permet d'effectuer un seuillage global automatique d'une image. Cette méthode est idéale pour les images où les niveaux de gris des objets et de l'arrière-plan forment **deux classes** bien distinctes. Dans ce cas, le seuil optimal pour séparer les niveaux de gris en deux classes (par exemple, objets d'intérêt et arrière-plan) est trouvé automatiquement : le seuil doit minimiser la variance intra-classe (dispersion des niveaux de gris dans chaque classe) et maximiser la variance inter-classe (différence entre les classes).

Cependant, son efficacité peut être limitée dans des cas de bruit ou de complexité multimodale.

Il est possible d'utiliser cette méthode à l'aide de la fonction *threshold()* d'OpenCV de la façon suivante :

```
otsu_val, binary_image = cv2.threshold(image_gray, 0, 255,
                    cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

➜ **M** Tester cette méthode sur l'image **robot.jpg**. Comparer alors l'image résultante avec la méthode classique de seuillage.

➜ **Q** A quoi correspond la valeur *otsu_val* ?

Vous pourrez aussi comparer les temps d'exécution des deux méthodes...

Erosion, dilatation et gradient

Temps conseillé : 60 min

On se propose ici d'analyser l'impact de différents procédés de pré-traitements (érosion, dilatation et gradient) sur une image.

Les pré-traitements à étudier sont à réaliser sur l'image *a_letter_noise.jpg* du kit d'images fourni. Vous pourrez utiliser la fonction *zoom_array()* fournie dans le fichier *images_manipulation.py* afin d'augmenter la taille des images à analyser.

Pour faciliter l'analyse des images, on propose le code suivant permettant d'afficher 3 images en parallèle sur un même graphique :

```
fig, ax = plt.subplots(nrows=1, ncols=3)
ax[0].imshow(image_data_1, cmap='gray')
ax[0].set_title('Title Image 1')
ax[1].imshow(image_data_2, cmap='gray')
ax[1].set_title('Title Image 2')
ax[2].imshow(image_data_3, cmap='gray')
ax[2].set_title('Title Image 3')
```

Opérations de pré-traitement

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)

Eléments structurants d'une convolution (noyau)

Notions : *Structuring Elements (kernels)*

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.

➜ **M** Générer un noyau en forme de croix de taille 3 par 3 pixels et afficher ce noyau.

➜ **Q** Quel est le type de l'objet noyau résultant ?

➜ **M** Générer un second noyau en forme de carré de taille 3 par 3 pixels et afficher ce noyau.

Opérations d'érosion et de dilatation

Notions : *Erosion* - *Dilation*

➜ **M** Appliquer une opération d'érosion sur l'image *a_letter_noise.jpg* avec, indépendamment, les deux noyaux précédemment générés.

➜ **M** Afficher les deux images ainsi que l'image originale sur un même graphique pour les comparer.

➜ **M** De la même manière, utiliser une opération de dilatation sur cette même image à l'aide des deux noyaux précédemment générés. Afficher également un comparatif des images résultantes.

➜ **Q** Que pouvez-vous conclure sur l'utilité des opérations d'érosion et de dilatation sur une image ?

Opérations d'ouverture et de fermeture

Notions : *Opening* - *Closing*

➜ **M** Appliquer une opération d'ouverture (*opening*) sur l'image *a_letter_noise.jpg* avec, indépendamment, les deux noyaux précédemment générés.

➜ **M** Afficher les deux images ainsi que l'image originale sur un même graphique pour les comparer.

➜ **M** De la même manière, utiliser une opération de fermeture sur cette même image à l'aide des deux noyaux précédemment générés. Afficher également un comparatif des images résultantes.

➜ **Q** Que pouvez-vous conclure sur l'utilité des opérations d'ouverture et de fermeture sur une image ?

Opération de gradient

Une autre opération, appelée **gradient**, calcule la différence entre une dilatation et une érosion sur une même image.

Il est possible de la mettre en pratique à l'aide de l'instruction suivante :

```
1  gradient_image = cv2.morphologyEx(image, cv2.MORPH_GRADIENT, kernel)
```

➜ **M** Appliquer une opération de gradient sur l'image *robot.jpg* avec, indépendamment, les deux noyaux précédemment générés.

➜ **M** Afficher les deux images ainsi que l'image originale sur un même graphique pour les comparer.

➜ **Q** Que pouvez-vous conclure sur l'utilité de l'opération de gradient sur une image ?

Lissage du bruit

Temps conseillé : 90 min

Générer du bruit sur des images

Notions : *Histogram of an image*

On se propose d'étudier la fonction *generate_gaussian_noise_image()* fournie dans le fichier *images_manipulation*

➜ **M** Tester l'exemple fourni dans le fichier *noise_test1.py*.

➜ **Q** Comment vérifier la distribution du bruit généré par cette fonction ?

On se propose d'étudier la fonction *generate_uniform_noise_image()* fournie dans le fichier *images_manipulation*

➜ **M** Tester l'exemple fourni dans le fichier *noise_test2.py*.

➜ **Q** La distribution du bruit généré par cette fonction est-elle uniforme ?

➜ **M** A l'aide de la fonction *generate_gaussian_noise_image_percent()*, générer un bruit gaussien de moyenne 30 et d'écart-type 20 sur 10% de l'image *robot.jpg* ouverte précédemment en nuance de gris. Visualiser le résultat.

Comparer différents filtres de lissage

On se propose à présent d'analyser l'impact de différents filtres de lissage (flou gaussien, filtre médian et filtre moyenneur) sur une image.

Pour ces trois types de filtres, répéter les étapes suivantes :

➜ **M** Appliquer une opération de lissage avec le filtre souhaité sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels.

➜ **M** Stocker dans une matrice la différence entre l'image originale et l'image lissée.

➜ **M** Afficher l'image originale, l'image lissée et la différence de deux images sur un même graphique pour les comparer.

➜ **M** Ajouter du bruit gaussien sur l'image et appliquer à nouveau le filtre gaussien. Afficher l'image originale, l'image lissée et la différence de deux images sur un même graphique pour les comparer.

➜ **Q** Que pouvez-vous conclure sur l'utilité d'un tel filtre ?

Vous pourrez également regarder l'impact de la taille du noyau sur l'image lissée finale.

Filtre de type gaussien

➜ **M** Appliquer une opération de lissage de type GAUSSIAN BLUR sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels (*cv2.GaussianBlur*).

Filtre de type médian

➜ **M** Appliquer une opération de lissage de type MEDIAN BLUR sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels (*cv2.medianBlur*).

Filtre de type moyenneur (mean ou box)

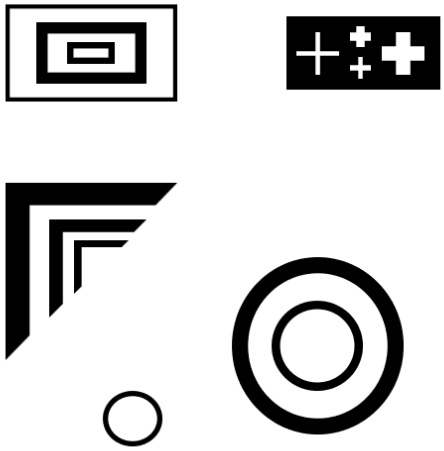
➜ **M** Appliquer une opération de lissage de type AVERAGING BLUR sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels (*cv2.blur*).

Isoler des éléments d'une image grâce à des opérations linéaires

Temps conseillé : 90 min

Les opérateurs d'érosion et de dilatation permettent d'extraire des informations particulières dans l'image à partir du moment où les éléments structurants (noyaux de convolution) sont judicieusement choisis.

On va chercher ici à détecter les lignes horizontales et verticales de l'image *forms_opening_closing.png* :



➜ **M** Tester l'exemple fourni dans le fichier *line_detection.py*.

➜ **M** Afficher les images aux différentes étapes du traitement.

➜ **Q** Analyser les différentes phases du traitement. Quelle est la forme du noyau utilisé ? Quel est l'impact de sa taille sur les éléments détectés ?

➜ **M** A partir de l'exemple précédent, écrire un script qui permet de détecter les lignes verticales de cette image et afficher le résultat.

➜ **M** Tester ces deux exemples sur d'autres images.

Détecter des contours et des points d'intérêt

Temps conseillé : 90 min

Le traitement numérique des images, de manière automatisée, vise souvent à **extraire des informations** d'une image en se basant sur de la détection des caractéristiques très spécifiques (ou *features detection*) :

— détection de coins (Harris...)
— détection de bords (Canny, Sobel, Prewitt...)
— détection de régions homogènes (Laplacien, Difference-of-Gaussian...)
— détection de lignes (Hough...)

Il existe également d'autres procédés plus complexes, notamment invariants à l'échelle et à la rotation, pour détecter des objets dans une image (SIFT, SURF, BRIEF...) ou basés sur l'apprentissage profond (R-CNN...).

Détection de coins par la méthode de Harris

Un **coin** est un point dans une image où l'**intensité change fortement dans plusieurs directions** (par opposition aux bords, où l'intensité change principalement dans une seule direction).

La **méthode de Harris** détecte ces coins en analysant les variations locales d'intensité des pixels. Elle repose sur une matrice appelée matrice de structure (ou matrice de Harris), qui résume la distribution locale des gradients dans une région autour d'un pixel. Cette matrice est définie comme suit :

$$M = \begin{pmatrix} I_x^2 & I_x \cdot I_y \\ I_x \cdot I_y & I_y^2 \end{pmatrix}$$

où : I_x et I_y sont les dérivées partielles de l'intensité de l'image dans les directions x et y. Ces dérivées sont calculées à l'aide d'un filtre Sobel ou similaire.

Pour chaque pixel, Harris calcule un score basé sur les valeurs propres (λ_1, λ_2) de M, en utilisant la formule suivante :

$$R = det(M) - k \cdot (trace(M))^2$$

où : $det(M) = \lambda_1 \cdot \lambda_2$ est le déterminant de M, $trace(M) = \lambda_1 + \lambda_2$ est la trace de M et k est une constante (généralement entre 0.04 et 0.06).

Le score R permet de distinguer les caractéristiques suivantes :

— $R > 0$: Coin (les deux directions présentent une variation significative) ;
— $R \approx 0$: Région plate (pas de variation significative) ;
— $R < 0$: Bord (variation dans une seule direction).

➜ **M** Ouvrir l'image *robot.jpg* du kit d'images fourni, en niveau de gris.

➜ **M** Appliquer le code suivant sur l'image ainsi ouverte et afficher l'image résultante :

```
image_harris = cv2.cornerHarris(image_gray, 2, 3, 0.04)
# result is dilated for marking the corners, not important
image_harris = cv2.dilate(image_harris, None)
# Threshold for an optimal value
image_gray[image_harris > 0.01 * image_harris.max()] = 0
```

➜ **M** Afficher également le résultat de la méthode *cornerHarris()*.

➜ **Q** Que pouvez-vous conclure sur l'intérêt de la méthode de Harris ? Vous pourrez faire varier certains paramètres de la méthode *cornerHarris()*...

Détection de contour par la méthode de Canny

La méthode de Canny est un algorithme classique utilisé pour **détecter les bords** dans une image. On cherche ici à calculer des gradients d'intensité G_x et G_y pour estimer les changements d'intensité dans les directions horizontale et verticale (souvent à l'aide de filtres Sobel).

On peut alors définir l'amplitude du gradient G et sa direction θ par les formules suivantes :

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = arctan\frac{G_y}{G_x}$$

On souhaite ensuite ne conserver que les pixels correspondant aux crêtes (maxima locaux) des gradients dans la direction du gradient. Pour chaque pixel, on compare l'amplitude du gradient avec celles des pixels adjacents dans la direction θ. Si ce n'est pas un maximum, le pixel est supprimé.

Puis deux seuils (T_{haut} et T_{bas}) sont alors appliqués pour classifier les pixels :

— $G > T_{haut}$: pixels forts

— $T_{bas} < G < T_{haut}$: pixels faibles

— $T_{bas} > G$: pixels supprimés

Enfin, on détecte les pixels faibles qui sont connectés à des pixels forts. Ces ensembles sont conservés comme bords. Cela permet de relier les fragments de bords discontinus tout en éliminant les bords isolés ou bruités.

➜ **M** Ouvrir l'image *robot.jpg* du kit d'images fourni, en niveau de gris.

➜ **M** Appliquer le code suivant sur l'image ainsi ouverte et afficher l'image résultante :

```
1  edges = cv2.Canny(image_gray, min_value, max_value)
```

Pour en savoir plus sur l'algorithme de détection de contours de Harris :
https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

Segmenter une image par la méthode de Watershed

Temps conseillé : 90 min

La méthode de **Watershed** (littéralement "ligne de partage des eaux") est une technique de **segmentation d'image** utilisée pour diviser une image en différentes régions ou objets. Elle est basée sur l'idée de traiter l'image comme une topographie où les niveaux de gris représentent des altitudes. L'objectif est de séparer les régions connectées tout en identifiant les limites précises entre elles.

Cette méthode nécessite au préalable quelques étapes de pré-traitement, basées sur les méthodes vues au cours de ce TP.

L'algorithme complet, que vous allez mettre en oeuvre à présent, est le suivant :

1. Ouverture de l'image en niveau de gris et seuillage (par la méthode d'Otsu)
2. Réduction du bruit (ouverture morphologique)
3. Identification des zones inconnues (ni fond, ni objets)
4. Identification des objets
5. Application de l'algorithme de Watershed

Pour le test et la compréhension de l'algorithme, il est intéressant d'afficher les images obtenues à chaque étape et de comprendre les différences avec l'étape précédente.

Ouverture et seuillage

➜ **M** Ouvrir l'image *bricks2.jpg* du kit d'images fourni, en niveau de gris. Puis appliquer un seuillage selon la méthode d'Otsu.

Réduction du bruit

➜ **M** Créer un élément structurant elliptique (*cv2.MORPH_ELLIPSE*) de taille 3 par 3.

➜ **M** Utiliser ce noyau pour réaliser une opération morphologique d'ouverture sur l'image binaire précédemment obtenue.

Identification des zones inconnues

Afin de pouvoir distinguer les objets sur l'image et ainsi les extraire du fond, nous allons chercher à séparer le fond (*background*) des objets (*foreground*).

➜ **M** Tester le code suivant sur l'image obtenue lors de l'étape précédente :

```
# sure background area
sure_bg = cv2.dilate(opening, kernel, iterations=1)

# Finding sure foreground area
k_dist = 0.4
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L1, 5)
ret, sure_fg = cv2.threshold(dist_transform,
          k_dist * dist_transform.max(), 255, 0)

# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)
```

➜ **Q** A quoi correspondent les images : *sure_bg*, *sure_fg*, *unknown* et *dist_transform* ?

➜ **Q** Que se passe-t-il en modifiant le coefficient *k_dist* ?

Identification des objets

Les pixels connectés dans les zones identifiées précédemment (*sure_fg*) peuvent alors être considérés comme faisant partie d'un même objet. Il s'agit à présent de les identifier comme étant des objets différents.

➜ **M** Tester le code suivant sur l'image obtenue lors de l'étape précédente :

```
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)
# Add one to all labels so that sure background is not 0, but 1
markers = markers + 1
# Now, mark the region of unknown with zero
markers[unknown == 255] = 0
```

➜ **M** Afficher l'image *markers*.

➜ **Q** A quoi correspond-elle ? La détection des objets est-elle optimale ? Pourquoi ?

Application de l'algorithme de Watershed

➜ **M** Tester le code suivant sur l'image obtenue lors de l'étape précédente (*markers*) et sur l'image RGB originale (*image_rgb*) :

```
image_rgb2 = image_rgb.copy()
markers2 = cv2.watershed(image_rgb, markers)
image_rgb[markers2 == -1] = [255, 0, 0]
```

➜ **Q** A quoi sert la première ligne de ce code ?

➜ **M** Comparer l'image originale, l'image marquée par la méthode de Watershed et l'image finale.

➜ **Q** Concluer sur l'intérêt d'un tel procédé et sur les paramètres qui peuvent modifier le résultat final.

Interfaçage Numérique

Travaux Pratiques

Semestre 6

Ressources

Bloc Images et OpenCV

Liste des ressources

— Image Processing / Key concepts

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

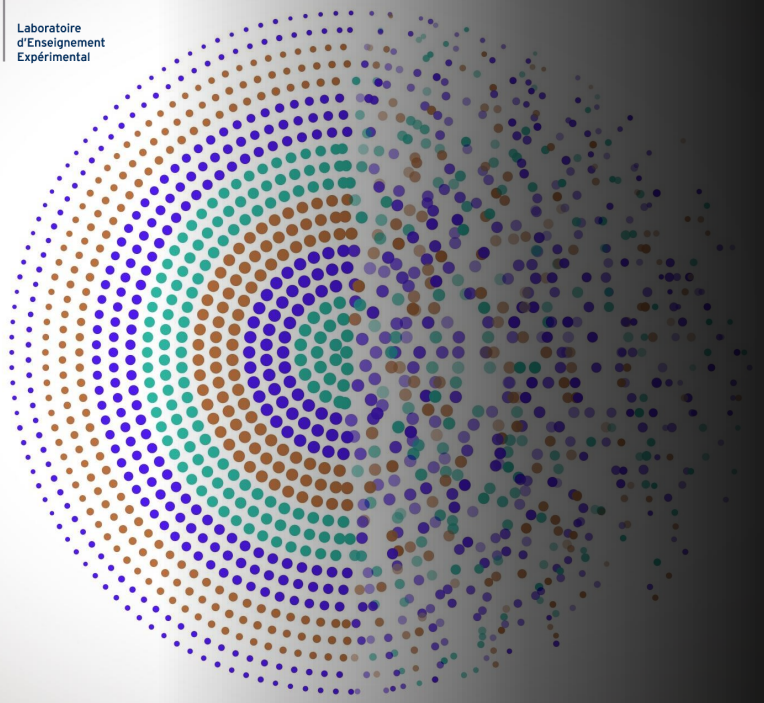
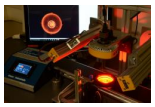


Image processing with OpenCV

Institut d'Optique – Engineers Training

Semester 6 – Digital Interface

Julien VILLEMEJANE



Goal of processing an image

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

Image from the camera

- **Noise**
- Bad contrast
- Inhomogeneous Lighting
- …

Desired image with objects with **well-defined contours**

- Homogeneous zones
- Transition zones

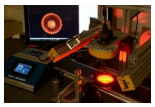


Image processing

Steps for processing an image

Acquisition	Light, Camera…	
Pre Processing	Noise Reduction / Filtering Contrast Enhancement Normalization	Ensuring clarity and reducing unwanted information Making features of interest stand out Standardizing the image scale or intensity
Segmentation	Thresholding Edge Detection Region of Interest Selection	Isolating objects of ROI / Separating product from background Identifying boundaries and contours Focusing only on relevant portions of the image
Feature Extraction	Geometric Features Texture Analysis Color Analysis	Extracting measurements (size, shape, position…) Recognizing patterns, symbols, points of interest
Classification	Object detection Template Matching Classification	Identifying and labelling objects Checking correct shape, size or orientation Categorizing objects to specific groups (defective or non-def.)
Decision	Tolerances, pass/fail, real-time feedback…	

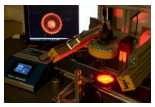


Image processing with OpenCV

Python 3 and OpenCV

Installing OpenCV for Python 3

```
pip install opencv-python
```

Testing OpenCV importation in a script

```
import cv2
Cv2.__version__
```

http://opencv.org

LEnsE.TECH

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

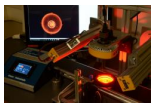
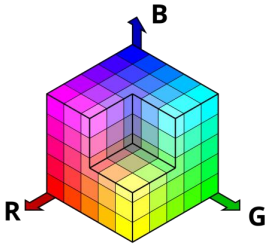


Image processing with OpenCV

Digital Images / Color Spaces

RGB

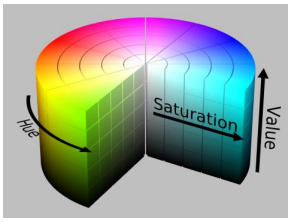
Used primarily in **electronic displays** like computer screens, cameras, and scanners. The combination of these three primary colors at various intensities can produce any color.



HSV

Used in **image editing**. It separates image's color from its brightness.

Hue : type of color
Saturation : intensity of the color
Value : Brightness of the color



CMYK
LAB
YUV

Images Source : Wikipedia

Color Space

Model for **representing colors** in a consistent and reproducible way

Each color space uses a different method for organizing and describing color, depending on the purpose or application

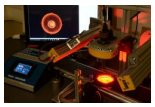


Image processing with OpenCV

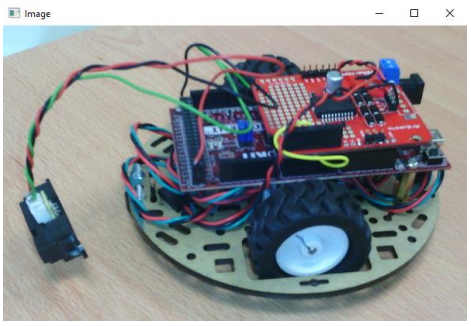
OpenCV / Open and display an Image

Acquisition

```
import cv2

image_rgb = cv2.imread('path/to/image.png')
image_gray = cv2.imread('path/to/image.png', cv2.IMREAD_GRAYSCALE)
```

```
image = cv2.imread("../../_data/robot.pgm")
print(type(image))
          <class 'numpy.ndarray'>
print(image.shape)
          (382, 600, 3)
```




```
cv2.imshow('Image ', image_rgb)
cv2.waitKey(0)
```

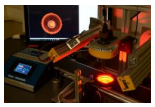
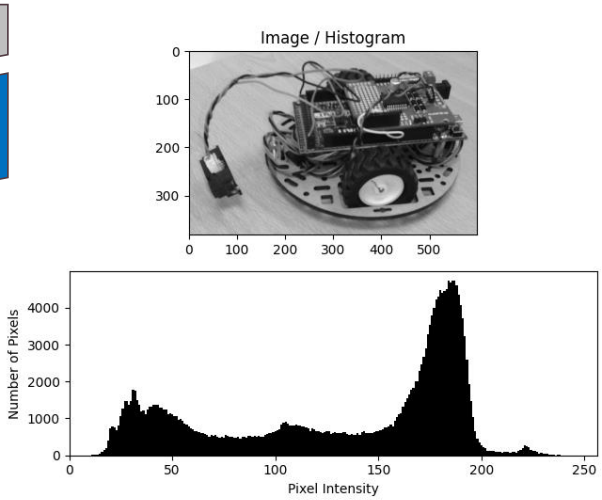


Image processing with OpenCV

OpenCV / Histogram of an image

Acquisition

Pre Processing



Histogram

Graphical representation that shows the **distribution of pixel intensity values** in an image

cv2.**calcHist**([image], [chan], Mask, [bins_nb], [min, max])

```
histogram = cv2.calcHist([image], [0], None, [256], [0, 256])
```

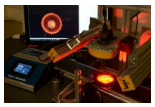
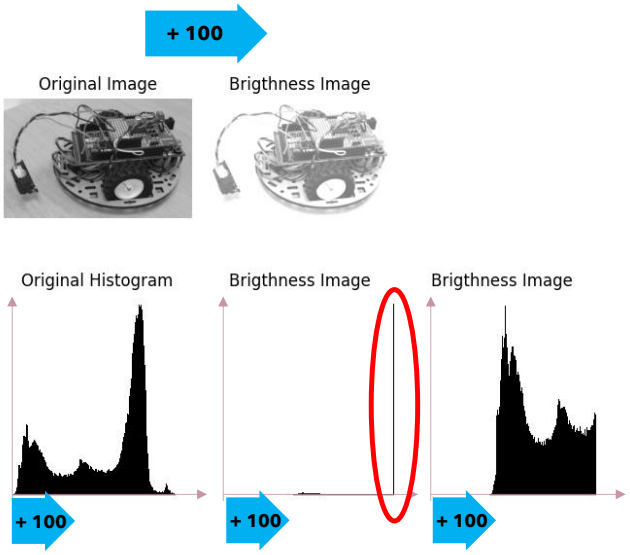


Image processing with OpenCV

OpenCV / Contrast and Brightness

Acquisition

Pre Processing



new_img = cv2.**convertScaleAbs**(image, beta=100)

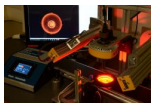
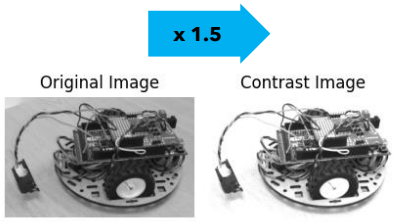


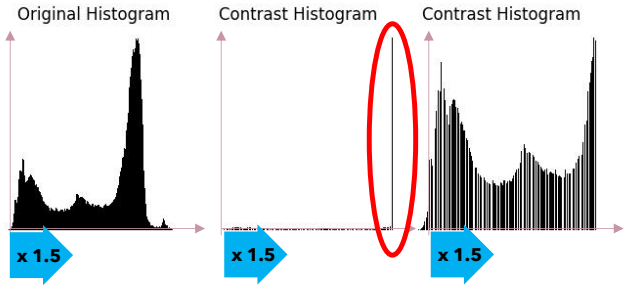
Image processing with OpenCV

OpenCV / Contrast and Brightness

Acquisition

Pre Processing





new_img = cv2.**convertScaleAbs**(image, alpha=1.5)

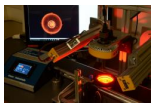


Image processing with OpenCV

OpenCV / Convolution Kernel *(or Structuring Element)*

Acquisition

Pre Processing

kernel = cv2.***getStructuringElement***(cv2.MORPH_xx, (M,N))

Cross Kernel

0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

cv2.**MORPH_CROSS**

Rect Kernel

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

cv2.**MORPH_RECT**

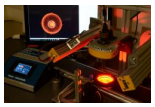
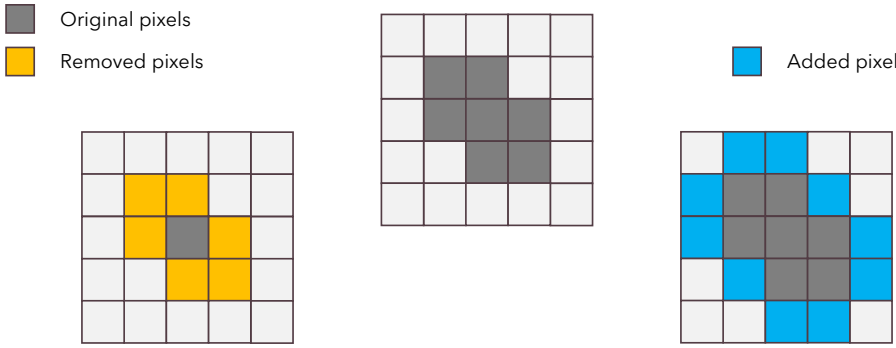


Image processing with OpenCV

OpenCV / Erosion and Dilation

Acquisition

Pre Processing



kernel

0	1	0
1	1	1
0	1	0

Erosion

Shrinking the foreground by **removing pixels** to the boundaries of objects

https://www.youtube.com/watch?v=fmyE7DialYQ

Dilation

Enlarging the foreground by **adding pixels** to the boundaries of objects

https://www.youtube.com/watch?v=xO3ED27rMHs

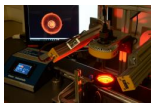
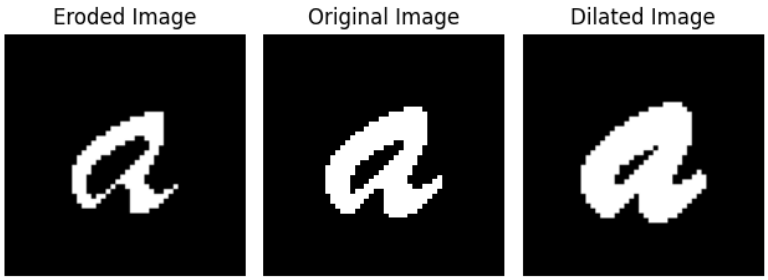


Image processing with OpenCV

OpenCV / Erosion and Dilation

Acquisition

Pre Processing



kernel

0	1	0
1	1	1
0	1	0

Erosion

Shrinking the foreground by **removing pixels** to the boundaries of objects

Dilation

Enlarging the foreground by **adding pixels** to the boundaries of objects

```
eroded_image = cv2.erode(image, kernel, iterations=1)
dilated_image = cv2.dilate(image, kernel, iterations=1)
```

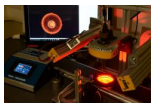
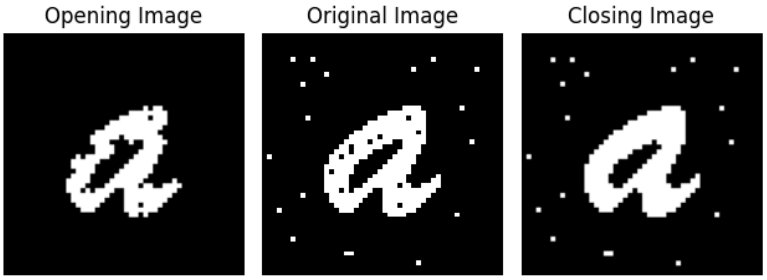


Image processing with OpenCV

OpenCV / Opening and Closing morphological transforms

Acquisition

Pre Processing



kernel

0	1	0
1	1	1
0	1	0

Opening

Erosion then **Dilation**

Removing small objects, in the background

Closing

Dilation then **Erosion**

Filling in small holes in the foreground

```
opening_image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
closing_image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
```

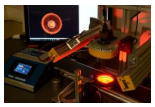
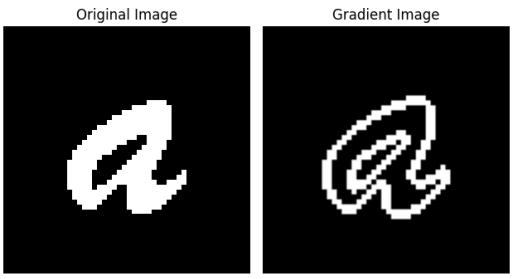


Image processing with OpenCV

OpenCV / Gradient morphological transforms

Acquisition

Pre Processing



kernel

0	1	0
1	1	1
0	1	0

Gradient

Difference between a **dilation** and an **erosion**

Unknown pixels classification : background or foreground ?

```
gradient_image = cv2.morphologyEx(image, cv2.MORPH_GRADIENT, kernel)
```

cv2.imshow('Image Window', image)

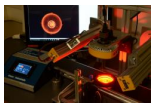


Image processing with OpenCV

OpenCV / Blur and mean

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

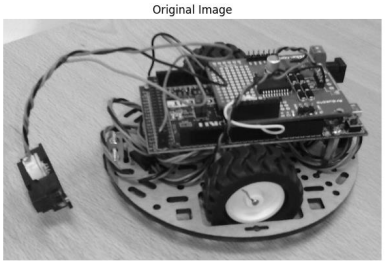
Acquisition

Pre Processing

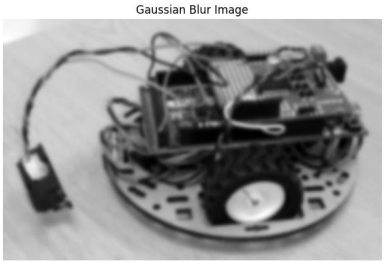
```
kernel_size = (N,M)

blurred_image_gauss = cv2.GaussianBlur(image, kernel_size, 0)
blurred_image_box = cv2.blur(image, kernel_size)
```

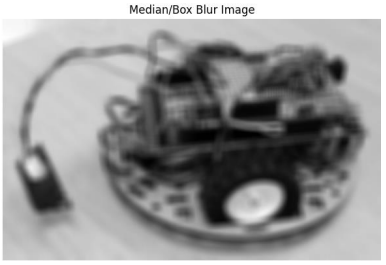
Original Image



Gaussian Blur Image



Median/Box Blur Image



1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Gaussian Kernel
(x 1/273)

Mean Kernel (x 1/(N*M))

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Removing irrelevant details

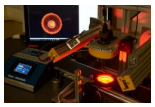


Image processing

Fourier Transform and filtering

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

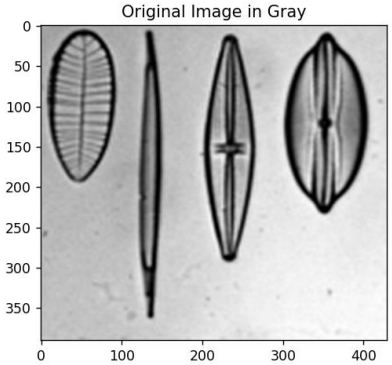
Acquisition

Pre Processing

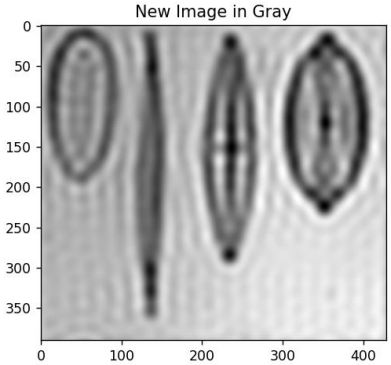
Segmentation

Feature Extraction

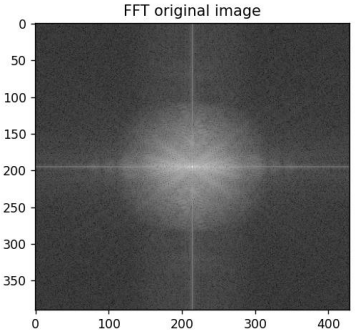
Original Image in Gray



New Image in Gray



FFT original image



Masked FFT