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 : *Open an image* - *Display an image* - *Calculate the histogram*

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

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

➜ **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

Voir l'impact de la loi de transformation de RGB vers Gray.

Seuillage et binarisation

Temps conseillé : 30 min

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 :

```
1 fig, ax = plt.subplots(nrows=1, ncols=3)
2 ax[0].imshow(image_data_1, cmap='gray')
3 ax[0].set_title('Title Image 1')
4 ax[1].imshow(image_data_2, cmap='gray')
5 ax[1].set_title('Title Image 2')
6 ax[2].imshow(image_data_3, cmap='gray')
7 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)

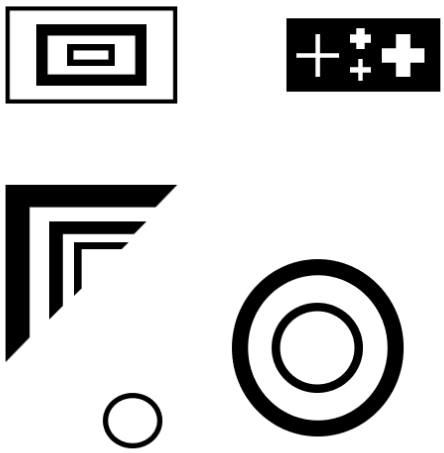
➜ **M** Appliquer une opération de lissage de type AVERAGING BLUR sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels (*cv2.blur*).

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

Canny et Harrys

Temps conseillé : 90 min

Segmenter une image par la méthode de Watershed

Watershed

Temps conseillé : 90 min

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

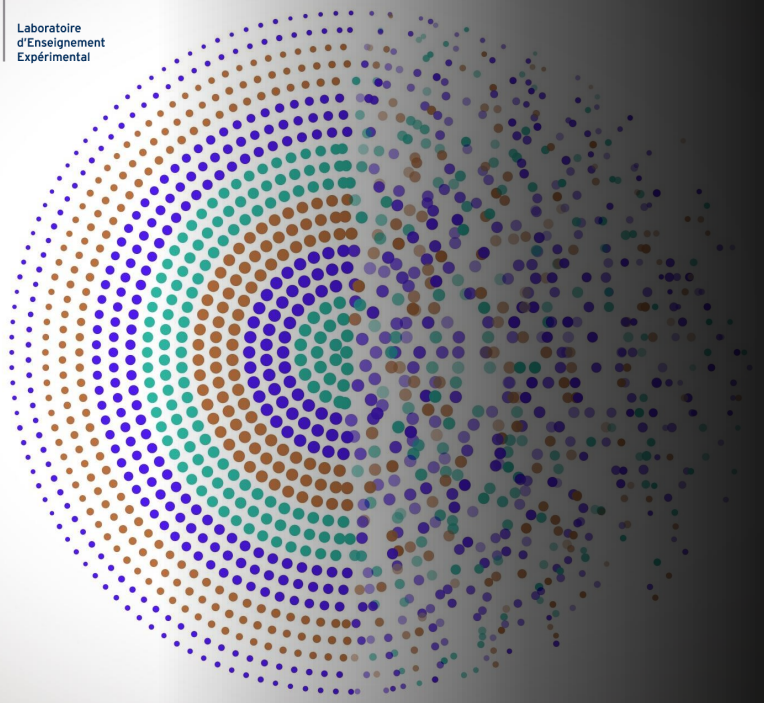
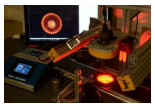


Image processing with OpenCV

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

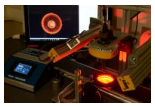


Image processing

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…	

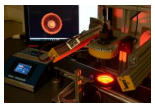


Image processing with OpenCV

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/

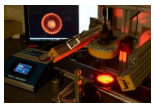
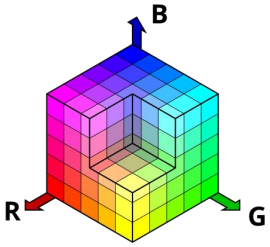


Image processing with OpenCV

Digital Images / Color Spaces

RGB

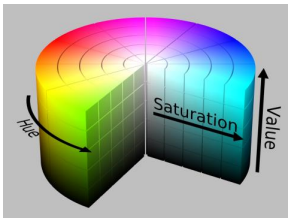
Used primarily in **electronic displays** like computer screens, cameras, and scanners. The combination of these three primary colors at various intensities can produce any color.



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

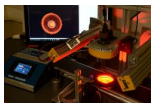


Image processing with OpenCV

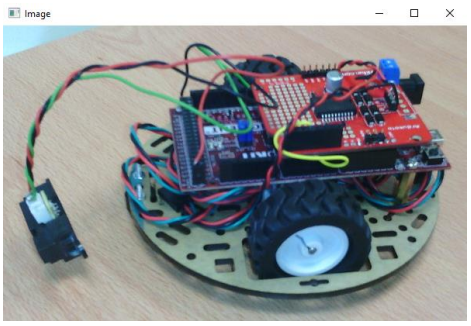
OpenCV / Open and display an Image

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)
```

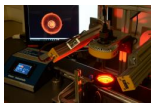
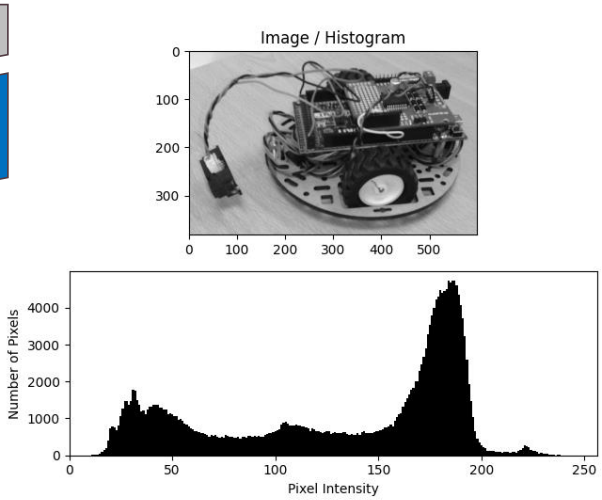


Image processing with OpenCV

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])
```

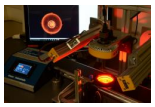
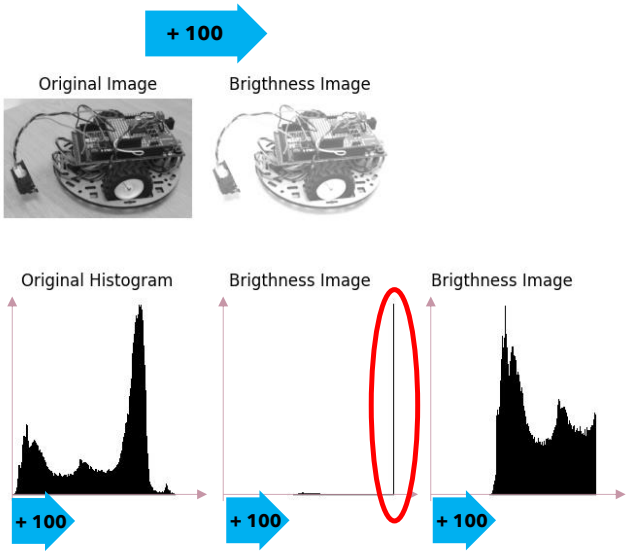


Image processing with OpenCV

OpenCV / Contrast and Brightness

Acquisition

Pre Processing



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

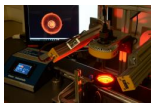
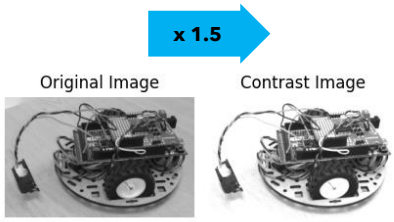


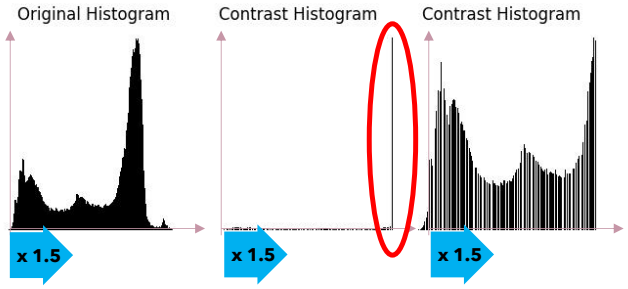
Image processing with OpenCV

OpenCV / Contrast and Brightness

Acquisition

Pre Processing





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

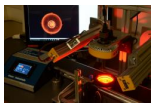


Image processing with OpenCV

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**

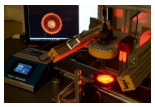
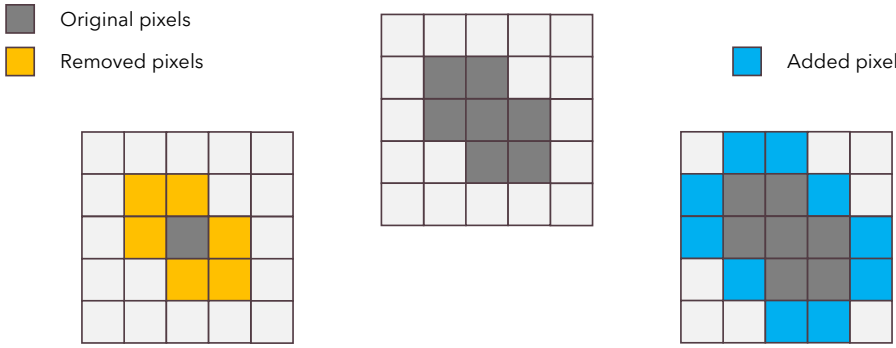


Image processing with OpenCV

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

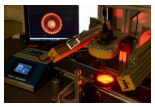
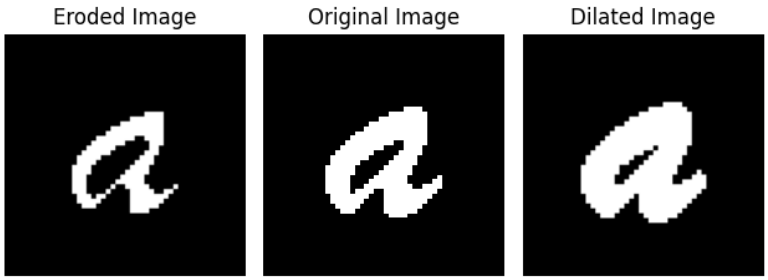


Image processing with OpenCV

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)
```

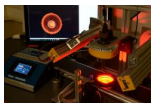
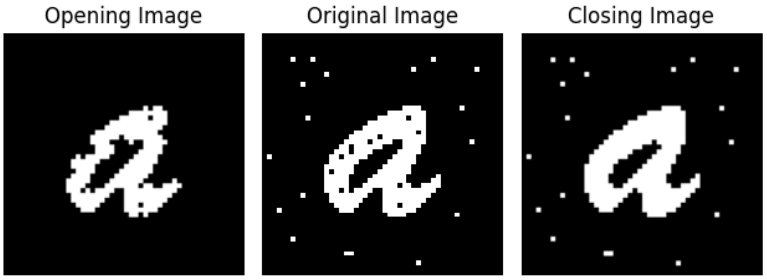


Image processing with OpenCV

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)
```

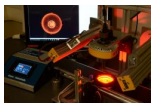
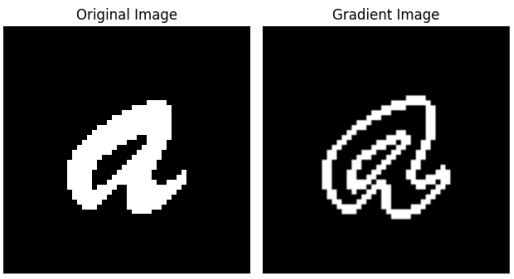


Image processing with OpenCV

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)

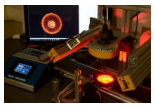


Image processing with OpenCV

OpenCV / Blur and mean

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

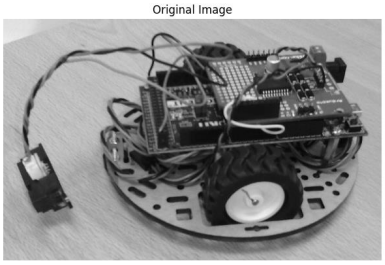
Acquisition

Pre Processing

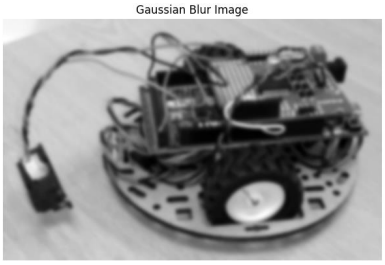
```
kernel_size = (N,M)

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

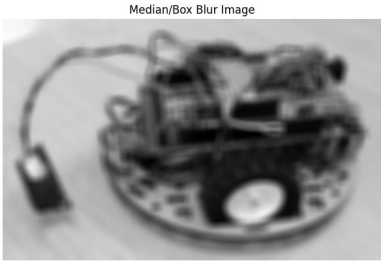
Original Image



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

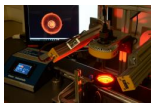


Image processing

Fourier Transform and filtering

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

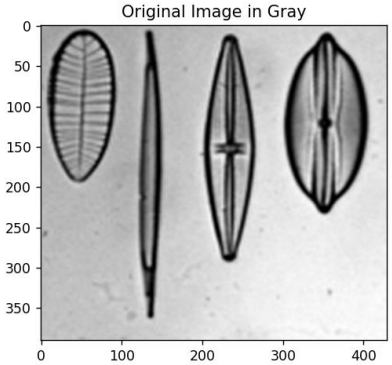
Acquisition

Pre Processing

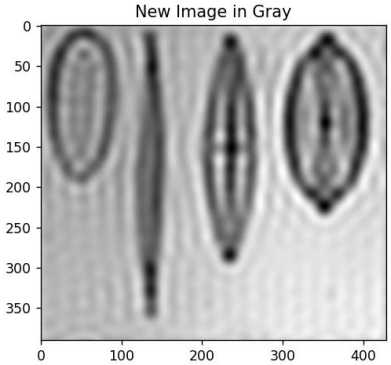
Segmentation

Feature Extraction

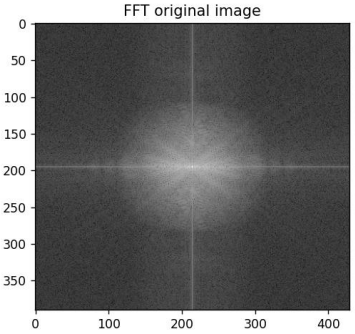
Original Image in Gray



New Image in Gray



FFT original image



Masked FFT