

INTERFAÇAGE NUMÉRIQUE

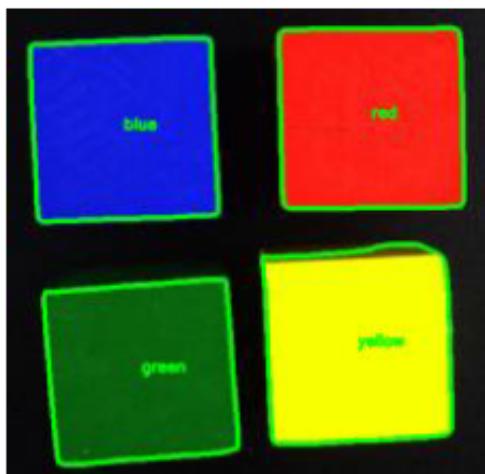
Travaux Pratiques

Semestre 6

Vision Industrielle

De la physique de la chaîne d'acquisition
à l'exploitation logicielle

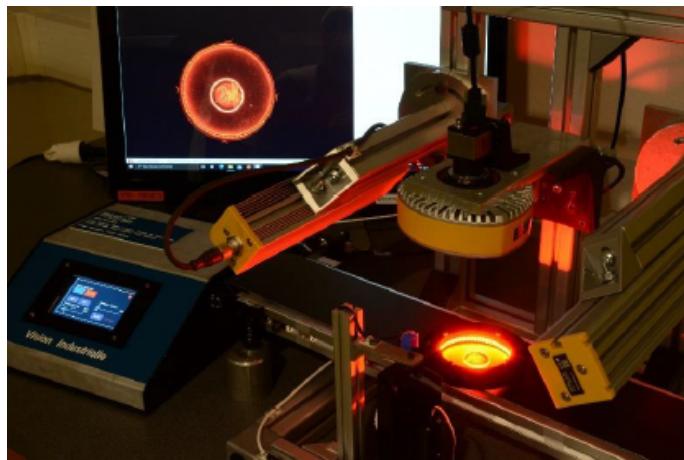
4 séances



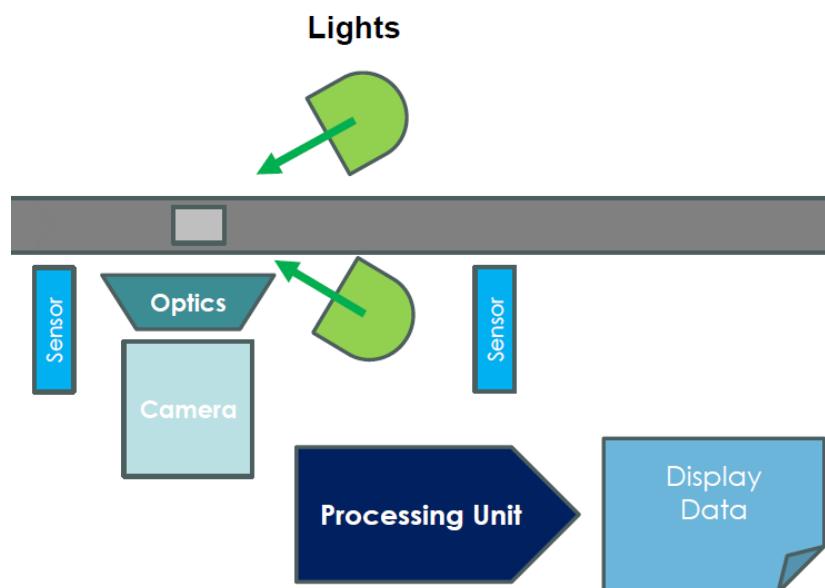
Ce sujet est disponible au format électronique sur le site du LEnSE - <https://lense.institutoptique.fr/> dans la rubrique Année / Première Année / Interfaçage Numérique S6 / Bloc 2 Vision Industrielle.

Vision Industrielle

La vision industrielle est une technologie qui permet à des machines d'**analyser automatiquement** des scènes pour **contrôler**, **guider** ou **inspecter** des objets sur des processus de production. Elle repose sur l'utilisation de **caméras**, d'**optique**, d'**éclairages** spécifiques (ou contraints), de **capteurs** et d'algorithmes de **traitement d'image**.



Elle a pour but de **prendre des décisions automatiques** (ou aider l'être humain dans sa prise de décision) vis-à-vis d'un (ou plusieurs) objet(s) dans une scène spécifique : détecter des défauts ou des irrégularités, compter ou trier..., en rejetant ou validant automatiquement des produits, tout en assurant une constance de la qualité et de la répétabilité des opérations.



Ce sujet a été co-conçu par une équipe d'étudiant·es - Joséphine BECHU, Justine GABRIEL et Paul CHENEAU - lors d'un projet DEPhI de 2A - 2025-2026 - et l'équipe pédagogique d'Interfaçage Numérique.

Acquis d'Apprentissages Visés (AAV)

À la fin de cette série de 4 séances, les étudiant·es seront capable de :

- Décomposer et paramétrer une chaîne de vision industrielle complète (du capteur au traitement de l'image),
- Comprendre les compromis physiques et numériques de chaque maillon,
- Réaliser un prototype d'inspection ou de mesure simple (tri d'objets),
- Justifier leurs choix de configuration (résolution, focale, éclairage...)

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 2 Caméra, Images et Interfaces / 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 2 Caméra, Images et Interfaces / 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 2 Caméra, Images et Interfaces / Images et OpenCV / Image Processing with OpenCV*.

Déroulement

Les sujets **TP1** et **TP2** se font en binôme et sont interchangeables (4 binômes commenceront par la TP1 lors de la première séance et les 4 autres binômes commenceront par le TP2).

Les sujets **TP3** et **TP4** se font par groupe de 4 étudiant·es.

TP1 - Banc de vision industrielle

Le **TP1** se fera sur un banc de vision industrielle simple incluant une caméra, un objectif (focale : xx mm), un éclairage Effilux Ring-RGB et un ordinateur.

- [20'] Prendre en main l'interface - caméra (temps d'intégration, histogramme) + éclairage RGB
- [20'] Tester les outils de base proposés dans l'interface (coupe, lissage, seuillage)
- [20'] Vérifier l'uniformité de l'éclairage sur objet uniforme - coupe dans l'image
- [30'] Valider la linéarité de la caméra sur objet uniforme - pour différents temps d'intégration (en blanc et R/G/B)
- [60'] Contrôler le champ de vision du système et sa résolution spatiale
 - placer une règle pour mesurer le FOV (Field of View)
 - placer une mire pour mesurer la plus petite taille résoluble
 - Modifier la profondeur binaire (8, 10, 12 bits) et voir l'impact sur la résolution spatiale / sur la qualité de l'image
 - Revenir aux caractéristiques de l'objectif optique
- [30'] Contrôler le contraste du système en plaçant des mires en fonction du temps d'intégration, de l'éclairage
- [60'] Analyser l'impact des propriétés des objets et des sources sur la valeur mesurée par la caméra
 - Étudier les réflectances des cubes et le spectre des sources (fournis)
 - Comparer les niveaux de gris obtenus sous différents éclairages des différents objets mis à disposition (cubes, formes...)

TP2 - Manipulations de base sous OpenCV

Codes fournis (à modifier?)

- [40'] Ouvrir une image - différence Gray/RGB
- [20'] Calculer l'histogramme d'une image
- [20'] Améliorer numériquement une image / contraste-luminosité
- [20'] Appliquer un seuillage sur une image
- [30'] Appliquer une érosion et une dilatation sur une image
- [20'] Appliquer une ouverture et une fermeture sur une image
- [20'] Appliquer un gradient sur une image
- [30'] Calculer la FFT d'une image
- [20'] Appliquer un filtre moyenneur sur une image
- [20'] Appliquer un filtre passe-haut - Sobel

TP3 - Manipulations avancées sous OpenCV / Limites du banc de vision

Sous OpenCV :

- [40'] Déetecter des formes
- [40'] DéTECTER DES COULEURS
- [40'] Recolorisation (matrice de changement de base)

Segmentation de l'image ?

Sur le banc de VI :

- [40'] Calibration de la chaîne sur les cubes de couleurs (couleur)
- [40'] Calibration de la chaîne sur des formes colorées
- [40'] Calibration de la chaîne pour des mesures de distance

TP4 - Détection d'objets - Contrôle qualité

Le but est à partir d'une série d'images (1 en RGB ou 3 en niveau de gris sous éclairage particulier) de détecter le nombre d'objets d'une forme et d'une couleur particulière et de valider si les pièces requises pour un assemblage sont bien présentes (simuler le packaging d'un produit fini - 6 pièces de forme et de couleurs distinctes par exemple).

La validation du packaging final devra se faire sans l'intervention de l'être humain mais un affichage propre des couleurs devra être fait pour le contrôle par le/la manipulateur·trice.

Bonus : mesure de la conformité de la taille des pièces ?

Objectifs du bloc

L'objectif principal de ce bloc est de découvrir les **différents paramètres impactant la qualité d'une prise d'image par une caméra** dans un environnement industriel ainsi que la **manipulation d'images numériques**, à l'aide de la bibliothèque **OpenCV**.

On s'intéressera à l'impact du **temps d'intégration**, de la **Résolution** de la caméra et de l'**éclairage** (couleur et intensité) de la scène et des objets sur l'image résultante.

Cette étude sera complétée par un TP plus détaillé sur les bruits associés à l'utilisation des caméras CMOS en 2ème année du cycle ingénieur à Palaiseau.

Des cours et des projets autour du traitement d'images sont également proposés dans les prochaines années de formation, quelque soit le site. Ces deux séances sont une introduction à ces modules plus avancés.

*Le bloc **Images et OpenCV** (facultatif) de ce module permet d'aller également plus loin dans le pré-traitement des images et leur manipulation.*

Séance 1 / Modélisation primaire d'une chaîne d'acquisition

Ce bloc de travaux pratiques utilise un **banc de vision industrielle** avec une lampe de type Effi-Ring RGB, une caméra Basler et une interface développée en **Python** (*PyQt6*) et qui utilise des fonctionnalités de la bibliothèque **OpenCV**.

Les documentations de la caméra et de l'éclairage sont disponibles aux adresses suivantes :

- Basler a2A 1920 - 160ucBAS : <https://docs.baslerweb.com/a2a1920-160ucbas#specifications>
- Effi-Ring : <https://www.effilux.com/fr/produits/annulaire/effi-ring>

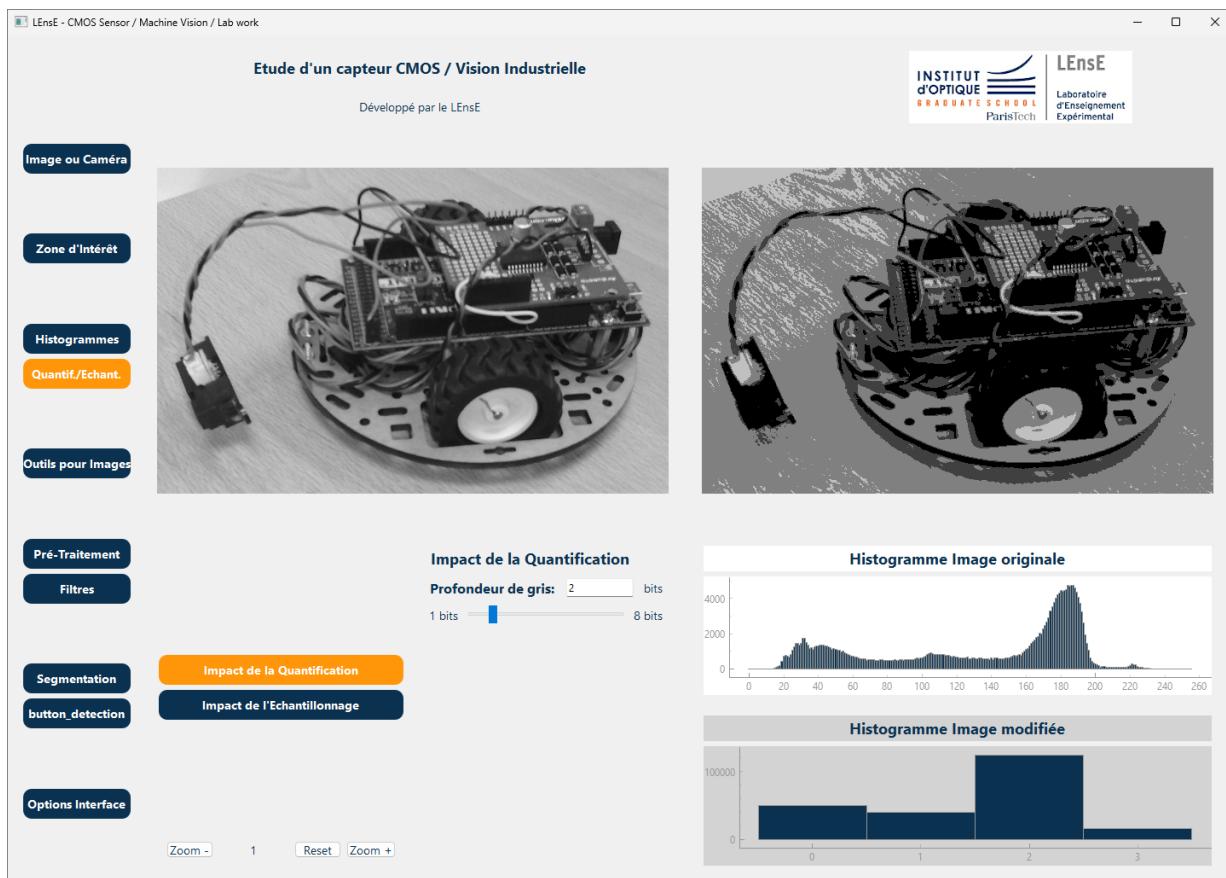
Etape 0 / Prendre en main l'interface de pilotage

Temps conseillé : 30 min

→ M Lancer l'application CMOS_Machine_Vision depuis le bureau des ordinateurs.

La dernière version officielle est sur le dépôt GitHub suivant :

<https://github.com/IOGS-LEnsE-ressources/machine-vision-gui> (version Basler)



→ M Ouvrir la première caméra disponible dans l'onglet IMAGE OU CAMÉRA / SÉLECTIONNER UNE CAMÉRA / OUVRIR LA PREMIÈRE CAMÉRA.

Eclairage et zone d'intérêt

→ M Allumer l'éclairage annulaire du banc (trois couleurs).

→ Q Quelle couleur d'éclairage obtient-on ?

→ M Placer un objet (un cube de couleur par exemple) dans le champ de la caméra.

→ M Ajuster la zone d'intérêt (ou *Area of Interest - AOI*) à l'aide de l'onglet ZONE D'INTÉRÊT pour ne sélectionner qu'une partie de l'image autour de l'objet.

→ Q Que pouvez-vous dire des deux histogrammes affichés ? Quelles sont les valeurs minimale et maximale prises par les pixels de la caméra ? Quelle est alors la résolution de la caméra utilisée ?

→ Q A quoi servent les deux bagues présentes sur l'objectif ?

Pour la suite du TP, on s'assurera de prendre une zone d'intérêt à peu près centrée dans l'image et d'une taille d'environ 500 par 500 pixels.

Etape 1 / Décrire le rôle des principales caractéristiques d'une caméra CMOS

Temps conseillé : 30 min

Temps d'intégration

→ M Dans l'onglet IMAGE OU CAMÉRA, sélectionner un *Black Level* de 0. Modifier le temps d'intégration de la caméra.

→ Q Que constatez-vous sur l'histogramme de l'image ? Sur la moyenne et l'écart-type de la répartition de la luminosité des pixels ?

→ M Placer un cache devant la caméra (ou Prévoir une caméra annexe avec cache - pour éviter d'enlever l'objectif et mettre un cache...) pour vous placer dans l'obscurité.

→ Q La répartition de la luminosité perçue par chaque pixel est-elle uniforme ? Que pouvez-vous en conclure ?

Echantillonnage et quantification

→ M Replacer l'objectif sur la caméra. Placer des objets dans le champ de la caméra. Sélectionner une zone d'intérêt d'environ 500 pixels par 500 pixels autour de l'objet à visualiser. Ajuster le temps d'intégration pour obtenir une image non saturée avec un éclairage blanc.

→ M Dans la section QUANTIF./ECHANT., modifier la profondeur de quantification des informations.

→ Q Que peut-on conclure sur l'effet de la quantification sur l'image ? Vous pourrez vous appuyer sur l'histogramme pour analyser le résultat.

→ M De la même façon, dans la section QUANTIF./ECHANT., modifier le nombre de pixels de sous-échantillonnage.

On parle ici d'un phénomène de **binning**. La résolution de l'image est "dégradée" numériquement dans ce cas et les nouveaux pixels affichés sont la moyenne de N x N pixels de l'image initiale.

Dans le cas présent, ce phénomène peut simuler le changement de résolution de la caméra sur l'acquisition d'une image numérique.

→ Q Que peut-on conclure sur l'effet de la résolution de la caméra sur l'image ? Vous pourrez vous appuyer sur l'histogramme pour analyser le résultat.

Etape 2 / Analyser l'impact des outils de base de la manipulation d'images

Temps conseillé : 60 min

Dans cette section, nous allons nous intéresser à quelques fonctionnalités permettant de **manipuler des images** pour les rendre utilisables : suppression du bruit, seuillage, amélioration du contraste...

Contraste et Luminosité

→ M Placer un objet dans le champ de la caméra. Sélectionner une zone d'intérêt d'environ 500 pixels par 500 pixels autour de l'objet à visualiser. Ajuster le temps d'intégration pour obtenir un histogramme dont le pixel maximum a une valeur de l'ordre des 2/3 de la valeur maximale de la caméra.

→ M Dans la section PRÉ-TRAITEMENT, puis sous-section CONTRASTE / LUMINOSITÉ, modifier les valeurs de contraste et de luminosité de l'image.

→ Q Quelles sont les opérations mathématiques réalisées sur les pixels par ces deux fonctionnalités ? Vous pourrez vous appuyer sur les histogrammes des images brutes et modifiées pour analyser vos résultats.

→ M Dans la sous-section AMÉLIORATION DU CONTRASTE, tester l'effet des deux curseurs.

→ Q Proposer une interprétation de l'opération effectuée sur chacun des pixels.

Seuillage

→ M Dans la section PRÉ-TRAITEMENT, puis sous-section SEUILLAGE, sélectionner le seuillage *Normal* et modifier la valeur du seuil.

→ Q Que pouvez-vous conclure sur l'intérêt du seuillage ? Vous pourrez essayer avec des objets de taille, de forme et de couleurs différentes.

→ M Tester également le mode *Inversé* et *Double*.

→ Q Que pouvez-vous conclure sur ces deux modes ?

Filtrage

→ M Dans la section FILTRES, puis sous-section FILTRE DE LISSAGE, sélectionner le filtre *Blur Moyen* et un noyau de taille 15.

→ Q Que se passe-t-il sur l'image ? Vous pourrez également vous appuyer sur la différence entre l'image de base et l'image modifiée, en cliquant sur l'option *Image - Effet*, pour analyser les effets sur l'image.

→ Q Quel est l'effet de la taille du noyau sur le filtrage ?

→ Q Qu'en est-il avec le filtre de type *Médian* ?

Les aspects théoriques liés au filtrage de données (signaux et images) sont abordés dans les modules MATHS ET SIGNAL (semestre 5) et TRAITEMENT DU SIGNAL (semestre 6).

La mise en oeuvre de ces filtres sur des images sera abordée en TD de ce module et également dans des modules de traitement d'images dans vos prochaines années de formation.

Etape 3 / Analyser l'impact du choix de la couleur de l'éclairage

Temps conseillé : 60 min

L'acquisition d'image réalisée par l'interface est monochrome. Il est toutefois possible de détecter des couleurs dans les objets à analyser en jouant sur le choix de l'éclairage et sa couleur.

Acquisition à des longueurs d'onde différentes

Vous avez à votre disposition plusieurs objets de couleurs différentes.



→ M Placer les cubes de 4 couleurs différentes dans le champ de la caméra. Placer également la surface plane noire dans le champ.

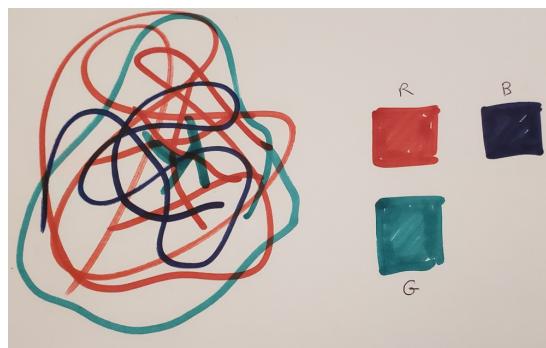
→ M Eclairer successivement la scène en rouge, puis en bleu, puis en vert. Visualiser les images acquises ainsi que les histogrammes associées.

Il est préférable de faire ces analyses dans une environnement sombre, voir dépourvu de tout éclairage parasite.

→ Q A partir des histogrammes, proposer une méthode de détection des couleurs des objets.

→ M Tester votre méthode. Faire valider par un·e enseignant·e.

→ M Placer à présent la feuille qui présente 3 carrés de couleurs différentes et un dessin.



→ Q Proposer une méthode de détection de ces couleurs, afin de n'extraire qu'un des carrés.

→ M Tester votre méthode. Faire valider par un·e enseignant·e.

→ Q Quelle lettre se cache dans votre dessin ?

Vous pouvez également demander l'oeuvre de Thierry A. et rechercher la lettre mystère...

Etape 4 / Modélisation simple d'une chaîne d'acquisition

Ouverture d'une image avec OpenCV

Pour ouvrir les images, nous allons utiliser la bibliothèque **OpenCV** (plus de détails dans la séance 2 de ce TP). Nous utiliserons la version développée pour le langage Python.

Pour installer cette extension sous Python, il faut exécuter la commande suivante dans un terminal (ou Invite de commande sous Windows) :

```
1 pip install opencv-python
```

Si l'on souhaite ouvrir l'image *test.png* stockée dans le même que votre script en Python, à l'aide de la bibliothèque OpenCV et l'afficher, il faut utiliser les instructions suivantes :

```
1 import cv2
2 from matplotlib import pyplot as plt
3
4 grayscale_image = cv2.imread('test.png', cv2.IMREAD_GRAYSCALE)
5 plt.imshow(grayscale_image, cmap='gray')
6 plt.show()
```

L'objet *grayscale_image* est une matrice en deux dimensions, issue de la bibliothèque *Numpy*. Il est alors possible de faire des calculs sur cette matrice : moyenne, somme, addition et autres opérations mathématiques avec d'autres matrices...).

Il sera également possible d'appliquer d'autres fonctionnalités de pré-traitement ou de filtrage. Ces notions seront abordées dans la séance de TP suivante.

Traitements avec OpenCV

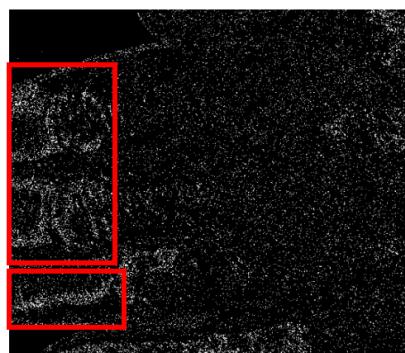
→ M Réaliser les premières étapes du protocole afin de prendre des images exploitables du fond blanc choisi dans les 3 conditions d'éclairage proposées (R, G et B) ainsi que les images d'une main dans les mêmes conditions d'éclairage et d'acquisition pour la caméra.

→ M Réaliser un script Python permettant d'ouvrir et d'afficher les 6 images.

→ M Ajouter à ce script la possibilité de calculer l'image résultante du traitement proposé pour détecter les veines.

Il pourra être intéressant de calculer le logarithme de l'image résultante et d'afficher le résultat. Afin d'isoler plus facilement les pixels intéressants, vous pourrez utiliser l'interface de la caméra en chargeant l'image du logarithme de *I* et utiliser le seuillage double pour isoler les objets intéressants...

→ Q Est-ce concluant ? Vous pourrez répéter ces étapes afin d'obtenir une meilleure qualité d'image (en jouant sur le temps d'intégration notamment).



Séance 2 / Manipulation d'images (OpenCV) et filtrage

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/>

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*.

Etape 1 / Ouvrir une image sous OpenCV

Temps conseillé : 20 min

Notions : *Open an image - Display an image*

- M Créer un nouveau projet sous PyCharm.
- M Impoter la bibliothèque **OpenCV2** (*cv2*).

*Pour la suite, il est nécessaire de placer tous les fichiers dans le même dossier : votre script Python, les différentes images et les bibliothèques de fonction (fichier *images_manipulation.py*)*

- M Ouvrir 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 Ouvrir 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 de données d'un élément ?

Etape 2 / Calculer l'histogramme d'une image et l'afficher

Temps conseillé : 20 min

Notions : *Calculate the histogram*

- M Calculer l'histogramme de l'image précédente 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.

Etape 3 / Améliorer numériquement la qualité d'une image

Temps conseillé : 20 min

Notions : *Enhance Contrast/Brightness*

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

β est l'offset de luminosité. Une valeur positive rend l'image plus lumineuse, tandis qu'une valeur négative l'assombrît.

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

- M Ouvrir l'image *robot.jpg* du kit d'images fourni, en niveau de gris.
- M Modifier le contraste de l'image et comparer les histogrammes de l'image originale et de la version modifiée pour différente valeur de α .
- M Modifier la luminosité de l'image et comparer les histogrammes de l'image originale et de la version modifiée pour différente valeur de β .

Etape 4 / Appliquer un filtre moyenneur sur une image

Temps conseillé : 90 min

La transformation précédente ne prend pas en compte les pixels voisins. Il est pourtant intéressant dans de nombreuses situations de capturer des relations spatiales entre les pixels.

Les filtres prenant en compte les pixels voisins exploitent 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.

Eléments structurants ou noyau

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

kernel	original image	Process for each pixel	filtered image
$\begin{bmatrix} -1 & 0 & -2 \\ 1 & 5 & 1 \\ -2 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 5 & 8 & 4 & 2 & 3 & 1 & 5 \\ 9 & 5 & 1 & 8 & 7 & 6 & 2 \\ 5 & 7 & 1 & 5 & 6 & 8 & 7 \\ 5 & 8 & 2 & 8 & 4 & 3 & 3 \\ 5 & 6 & 6 & 7 & 2 & 5 & 1 \end{bmatrix}$	$\begin{bmatrix} 5 & 8 & 4 & 2 & 3 & 1 & 5 \\ 9 & 5 & 1 & 8 & 7 & 6 & 2 \\ 5 & 7 & 1 & 5 & 6 & 8 & 7 \\ 5 & 8 & 2 & 8 & 4 & 3 & 3 \\ 5 & 6 & 6 & 7 & 2 & 5 & 1 \end{bmatrix}$ x -1 x 0 x -2 x 1 x 5 x 1 x -2 x 0 x -1 $R = -8 + 0 - 12 + 5 + 30 + 8 - 16 + 0 - 3$ $R = 4$	$\begin{bmatrix} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{bmatrix}$ $\boxed{4}$

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.

Filtre moyenneur gaussien

Notions : *Blur with OpenCV*

- M Créer un nouveau projet sous PyCharm.
- M Impoter la bibliothèque **OpenCV2** (*cv2*).
- M Ouvrir l'image *bricks2.jpg* du kit d'images fourni, en niveau de gris.
- M Appliquer un filtre gaussien (fonction *cv2.GaussianBlur()*) sur l'image précédente, de taille 5 x 5 pixels.
- M Afficher les deux images pour les comparer.
- Q Comment peut-on montrer de manière objective les modifications apportées à l'image ?
- M Mettre en oeuvre la méthode proposée.

On se propose d'utiliser la fonction *compare_blur_fft()* fournie dans le fichier *images_manipulation.py* pour comparer l'effet.

Ce fichier est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Répertoire vers codes à tester*.

- M Tester cette fonction sur l'image précédente.
 - Q Etudier cette fonction. Quelles méthodes sont utilisées pour comparer les images ?
 - Q Quelle est la fonction réalisée par le filtre précédent ?
- On cherche à présent à voir l'impact du noyau sur l'image finale.
- M Tester la fonction précédente avec des noyaux de taille différente et comparer les résultats à la fois sur l'image obtenue mais également sur la transformée de Fourier.
 - Q Que pouvez-vous conclure sur l'impact de la taille du noyau ?

Bruit sur une image

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

- 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.py`.

→ 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_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.

Il peut-être nécessaire de normaliser les images bruitées afin que les données entières de chaque pixel soient comprises entre 0 et 255, afin que les images puissent être affichées correctement.

→ M Tester la fonction `compare_blur_fft()` fournie dans le fichier `images_manipulation.py` et comparer les résultats à la fois sur l'image obtenue mais également sur la transformée de Fourier.

Etape 5 / Appliquer un filtre passe-haut sur une image

Temps conseillé : 60 min

Le **filtre moyenneur** précédent 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 Ecrire un script qui permet d'appliquer cette opération sur l'image `bricks2.jpg` du kit d'images fourni, en niveau de gris.

→ M Visualiser l'image originale, le résultat du filtrage selon X et le résultat du filtrage selon Y.

→ Q Que pouvez-vous conclure sur l'effet de ce filtre ?

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 Ecrire un script qui permet d'appliquer cette opération sur l'image *bricks2.jpg* du kit d'images fourni, en niveau de gris.

→ M Visualiser l'image originale, le résultat du filtrage selon X et le résultat du filtrage selon Y.

→ Q Que pouvez-vous conclure sur l'effet de ce filtre ?

Etape 6 / Appliquer un filtre par l'intermédiaire de la transformée de Fourier

Temps conseillé : 30 min

On s'intéresse ici à la **transformée de Fourier discrète en deux dimensions** permettant de représenter l'image dans le domaine fréquentiel, plutôt que dans le domaine spatial.

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

→ M Calculer la transformée de Fourier discrète de cette image à l'aide de la fonction *fft2()* de la bibliothèque **Numpy / FFT**. Afficher l'image et sa transformée de Fourier. Pensez à utiliser la fonction *fftshift...*

On se propose d'utiliser la fonction *circular_mask()* fournie dans le fichier *images_manipulation.py* pour appliquer un masque sur la transformée de Fourier de l'image.

Ce fichier est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Répertoire vers codes à tester*.

→ M Appliquer un masque circulaire sur la transformée de Fourier de l'image, à l'aide de la fonction *circular_mask()*. Afficher le résultat.

→ M Calculer l'image résultante à l'aide de la fonction *ifft2()* de la bibliothèque **Numpy / FFT**. Afficher l'image résultante et la comparer à l'image originale.

→ M Ecrire et tester une fonction permettant de réaliser un masquage rectangulaire sur une image, dont on pourra préciser la largeur et la longueur, ainsi que le barycentre du rectangle (point d'intersection des diagonales).

→ Q Que se passe-t-il lorsque vous appliquez un masque rectangulaire (dont la largeur et la longueur sont différentes) sur une image ? Qu'en est-il d'un masque circulaire ?

INTERFAÇAGE NUMÉRIQUE

Travaux Pratiques

Semestre 6

Ressources

Bloc Caméra

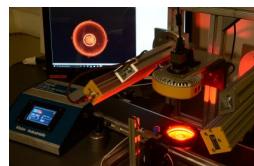
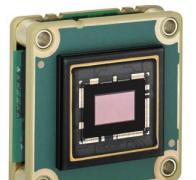
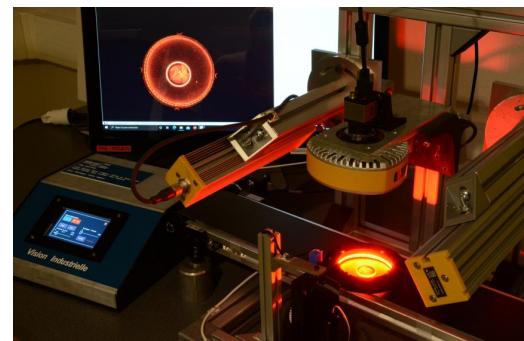
Liste des ressources

- Camera and sensor / Key concepts

SC 19 – Machine Vision

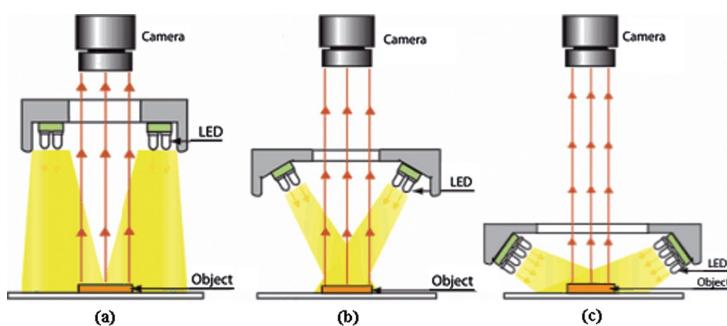
Cameras and Interfaces

Julien VILLEMEJANE



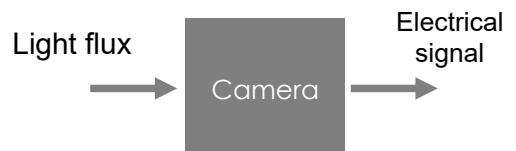
SC19 – Cameras and Interfaces

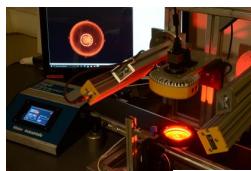
Camera in a machine vision chain



Camera

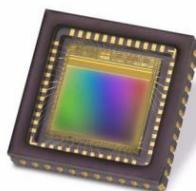
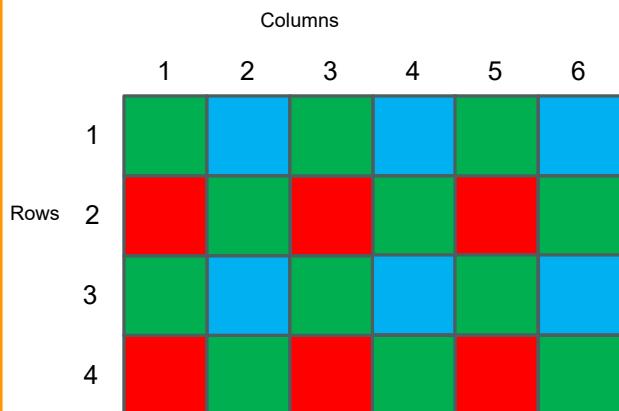
Device that transforms a **light flux** into a **measurable electrical signal**



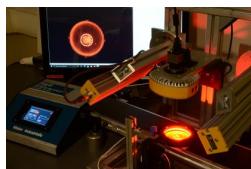
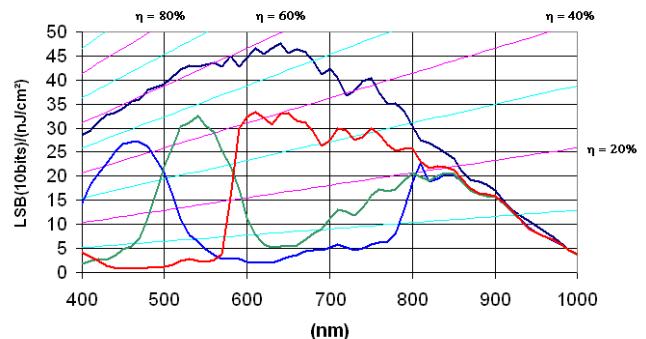


SC19 – Cameras and Interfaces

Camera / Bayer filter for color sensors

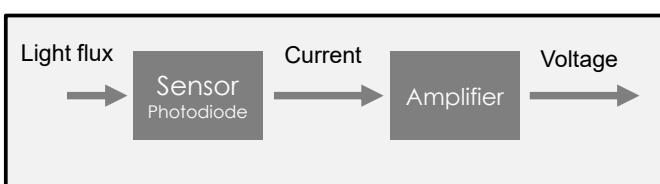
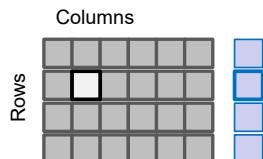


e2v sensor EV76C560ACT



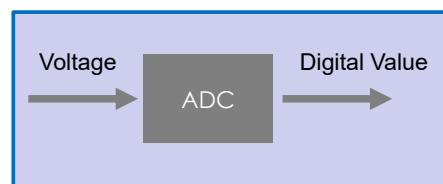
SC19 – Cameras and Interfaces

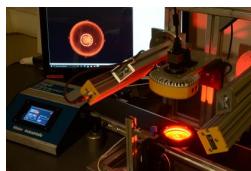
Camera / From analog to digital signal



Digital Camera

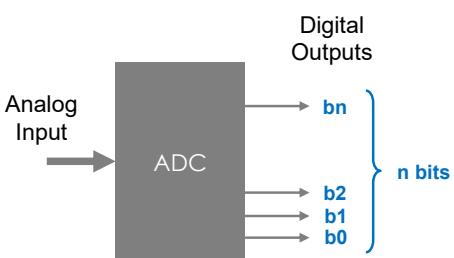
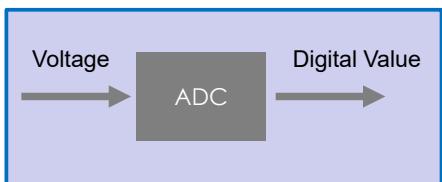
Device that transforms an array of **light flux sensors** into **digital data** called pixels





SC19 – Cameras and Interfaces

How an Analog to Digital Converter works ?

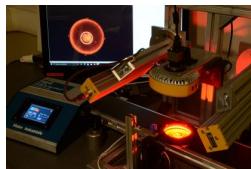
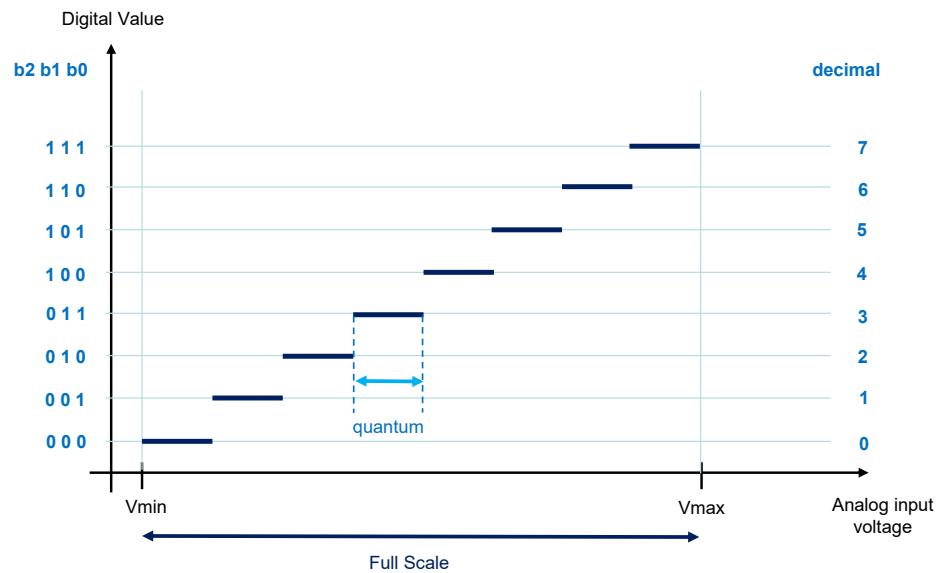


Each bit can have one of two values: **0** or **1**.

The **number of different values** that can be represented by **n bits** is 2^n .

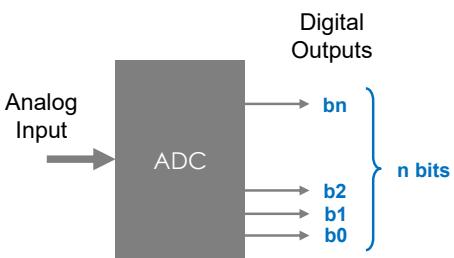
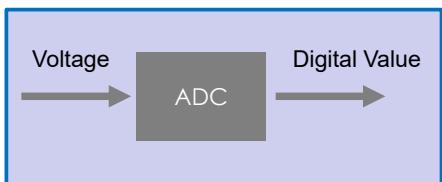
Example for $n = 3$ bits

Quantization



SC19 – Cameras and Interfaces

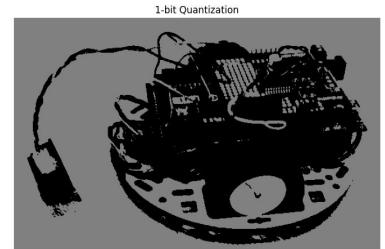
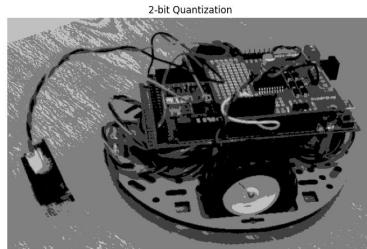
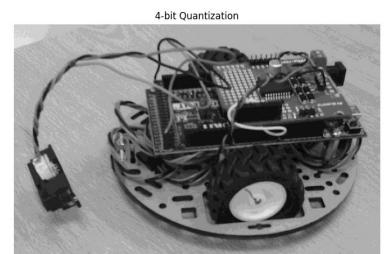
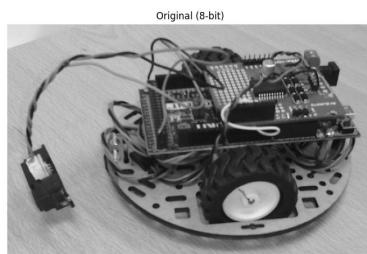
Sampling and quantization of an image

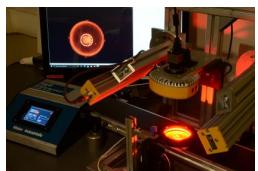


Each bit can have one of two values: **0** or **1**.

The **number of different values** that can be represented by **n bits** is 2^n .

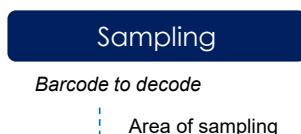
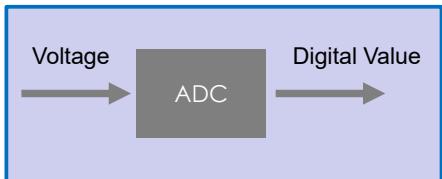
Quantization





SC19 – Cameras and Interfaces

Sampling and quantization of an image



<https://barcode-coder.com/fr/specification-ean-13-102.html>

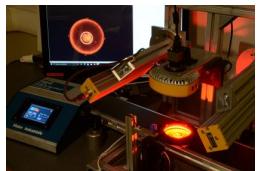
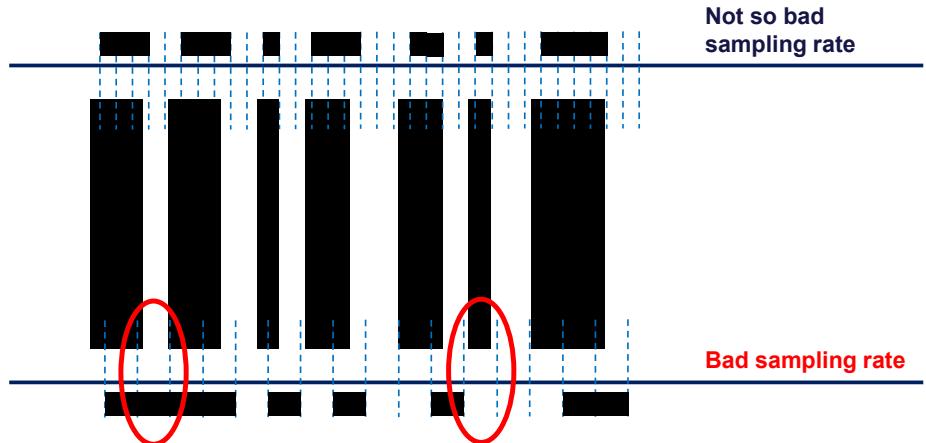


Sampling theorem

Nyquist–Shannon sampling theorem

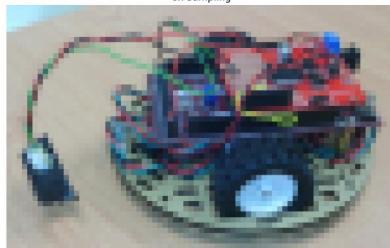
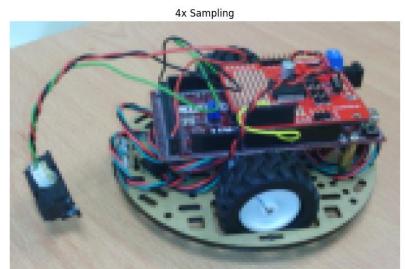
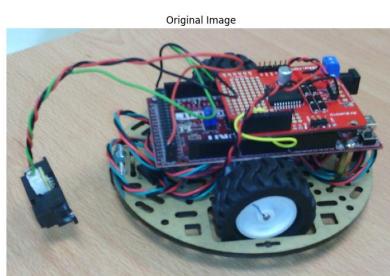
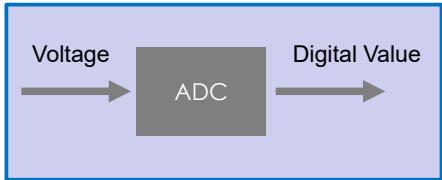
The sampling frequency must be equal to or **greater than twice** the frequency associated with the finest detail in the image (edges).

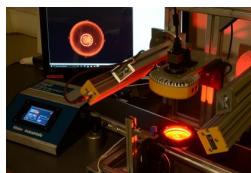
With a grid spacing of d , a periodic component with a period higher than $2.d$ can be reconstructed.



SC19 – Cameras and Interfaces

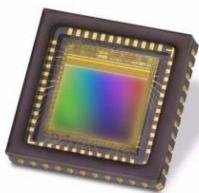
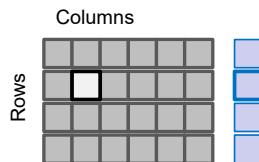
Sampling and quantization of an image



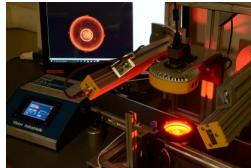
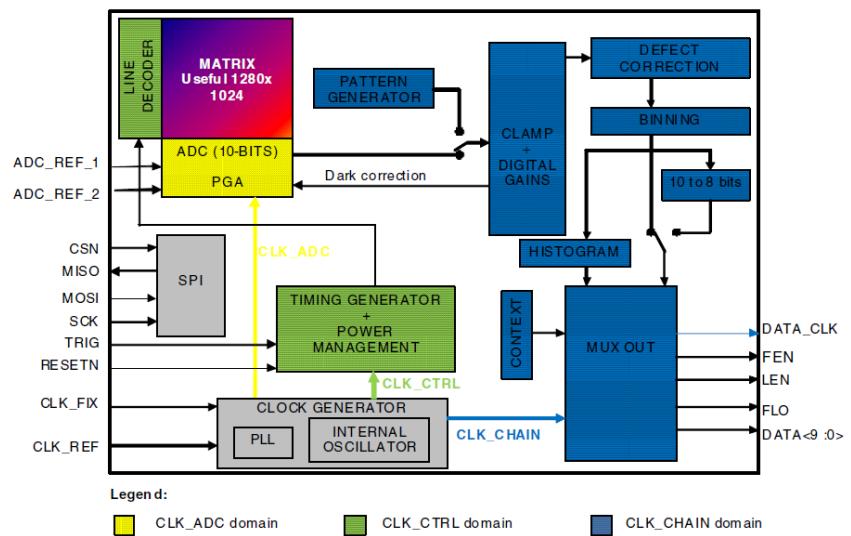


SC19 – Cameras and Interfaces

Inside a real sensor



e2v sensor EV76C560ACT

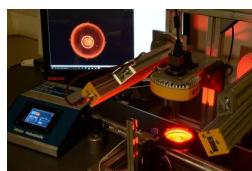


SC19 – Cameras and Interfaces

Interface for data transfer

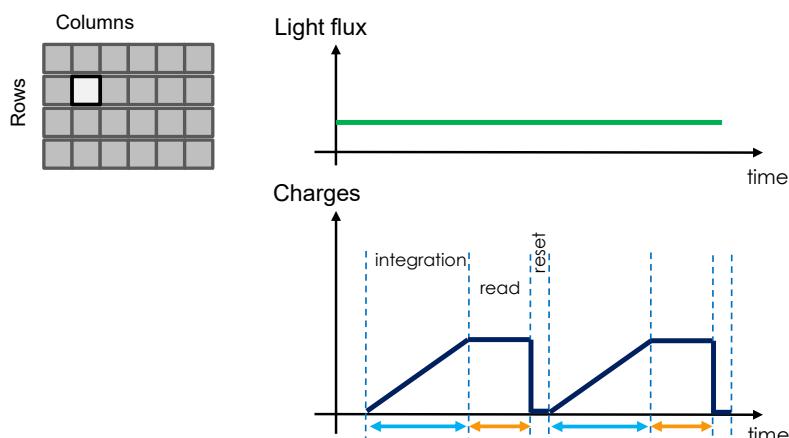
The data from a camera is transferred via **an interface**.
There are several types of standard interfaces.

	USB 3.0	10 GigE	CameraLink	Coaexpress
Bandwidth	5 to 20 Gbit/s	1.2 Gbits/s	Base : 2 Gbits/s Full : 5.4 Gbits/s (2 cables)	12.5 Gbits/s per cable
Cable length	3 m	100 m	7 to 15 m	20 to 40 m
Power	4.5 to 25 W	30 W *	Optional	13 W / cable
Frame Grabber	Not Required	Not Required	Required	Required
GeniCam	Required	Required	Optional	Required



SC19 – Cameras and Interfaces

Exposure Time

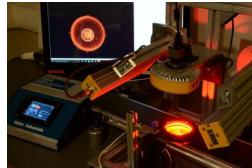
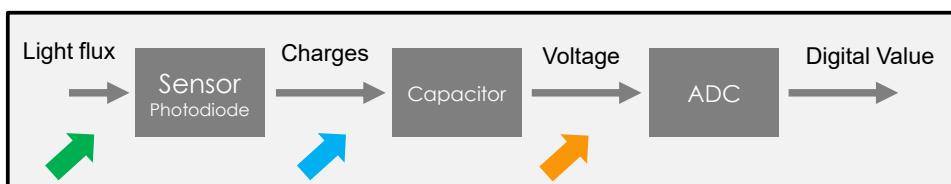


Exposure Time

Duration for which the **camera's sensor is exposed to light**, when capturing an image.

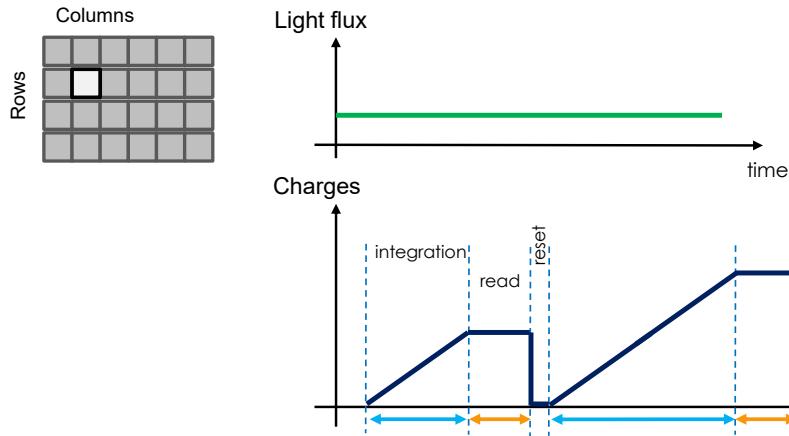
This parameter determines the amount of light collected.

i.e. the amount of collected charges coming from the sensor stored in a capacitor



SC19 – Cameras and Interfaces

Exposure Time

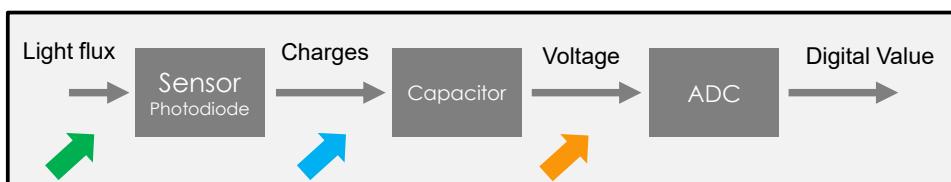


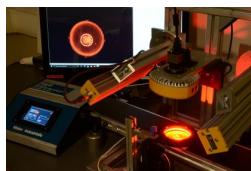
Exposure Time

Duration for which the **camera's sensor is exposed to light**, when capturing an image.

This parameter determines the amount of light collected.

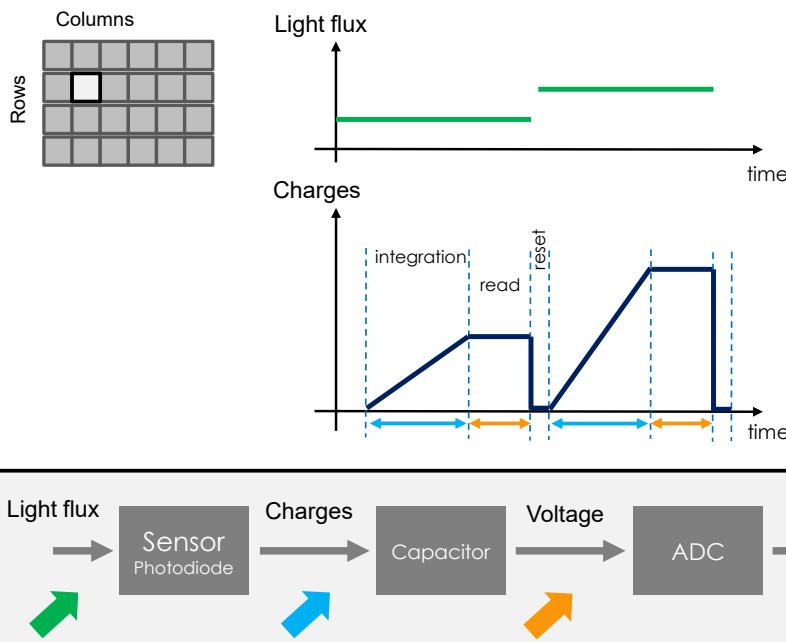
i.e. the amount of collected charges coming from the sensor stored in a capacitor





SC19 – Cameras and Interfaces

Exposure Time

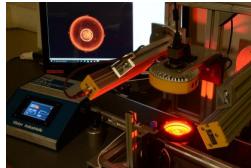


Exposure Time

Duration for which the **camera's sensor is exposed to light**, when capturing an image.

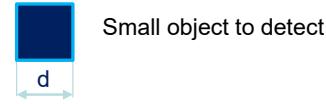
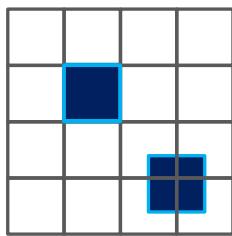
This parameter determines the amount of light collected.

i.e. the amount of collected charges coming from the sensor stored in a capacitor



SC19 – Cameras and Interfaces

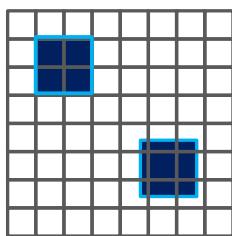
Spatial Resolution



$$P = d$$

Spatial resolution / P

Distance observed by a single pixel in a given direction



Security factor S

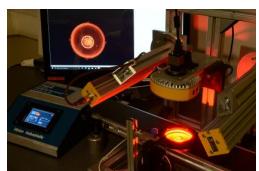
$$P = \frac{d}{S}$$

This security factor is due to the Nyquist-Shanon theorem.

And $S \geq 2$



To verify if the spatial resolution is good enough, **calibration target** can be used. (Foucault)

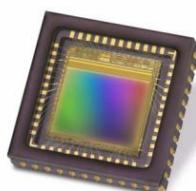


SC19 – Cameras and Interfaces

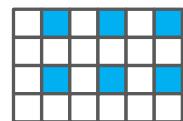
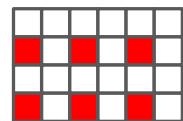
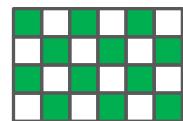
Camera / Bayer filter for color sensors

Columns

	1	2	3	4	5	6
1	Green	Blue	Green	Blue	Green	Blue
2	Red	Green	Red	Green	Red	Green
3	Green	Blue	Green	Blue	Green	Blue
4	Red	Green	Red	Green	Red	Green



e2v sensor EV76C560ACT



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.

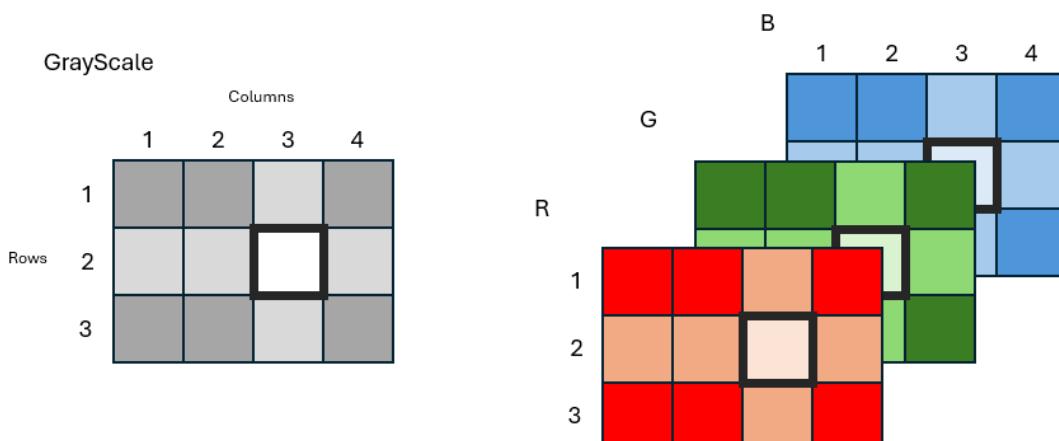
Couleur vers niveau de gris

Temps conseillé : 30 min

- M Ouvrir et afficher l'image *couleurs_4.png* 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 ?
- Q Est-il possible d'afficher un histogramme de l'image ?
- 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.

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 :

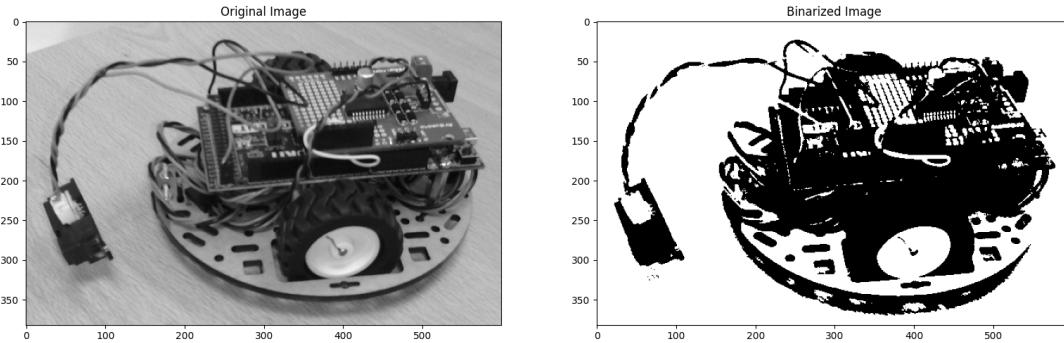
Espace colorimétrique	Avantages
RGB	Simple, utilisé pour les écrans et le rendu des couleurs.
HSV / HSL	Intuitif pour manipuler la couleur (teinte, saturation).
YUV / YCbCr	Sépare luminance et chrominance.
CIE-Lab	Uniformité perceptuelle, idéal pour mesurer les différences de couleur.
CMY(K)	Optimisé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

→ 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 :

```
1 otsu_val , binary_image = cv2.threshold(image_gray , 0 , 255 ,  
2 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 :

```
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 impulsifs, 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

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

→ 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.py`.

→ 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_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.

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. 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)

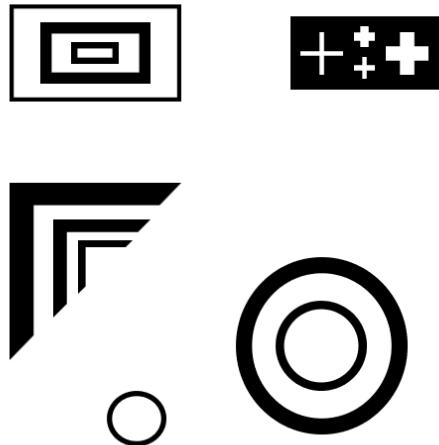
→ M Appliquer une opération de lissage de type AVERAGING BLUR sur l'image `robot.jpg` avec un noyau de 15 x 15 pixels (`cv2.blur`).

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.

Déetecter 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 :

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

où : $\det(M) = \lambda_1 \cdot \lambda_2$ est le déterminant de M , $\text{trace}(M) = \lambda_1 + \lambda_2$ 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 :

```
1 image_harris = cv2.cornerHarris(image_gray, 2, 3, 0.04)
2 # result is dilated for marking the corners, not important
3 # image_harris = cv2.dilate(image_harris, None)
4 # Threshold for an optimal value
5 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 **déetecter 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 :

```
1 # sure background area
2 sure_bg = cv2.dilate(opening, kernel, iterations=1)
3
4 # Finding sure foreground area
5 k_dist = 0.4
6 dist_transform = cv2.distanceTransform(opening, cv2.DIST_L1, 5)
7 ret, sure_fg = cv2.threshold(dist_transform,
8     k_dist * dist_transform.max(), 255, 0)
9
10 # Finding unknown region
11 sure_fg = np.uint8(sure_fg)
12 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 :

```
1 # Marker labelling
2 ret, markers = cv2.connectedComponents(sure_fg)
3 # Add one to all labels so that sure background is not 0, but 1
4 markers = markers + 1
5 # Now, mark the region of unknown with zero
6 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*) :

```
1 image_rgb2 = image_rgb.copy()
2 markers2 = cv2.watershed(image_rgb, markers)
3 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](#)

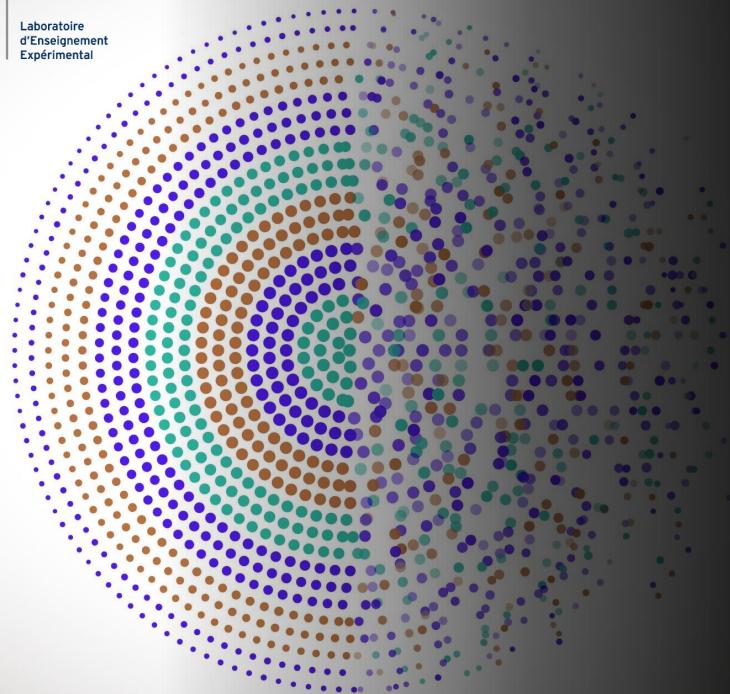


Image processing with OpenCV

Institut d'Optique – Engineers Training
Semester 6 – Digital Interface

Julien VILLEMEJANE

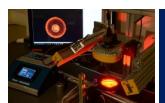


Image processing

Goal of processing an image



Image from the camera

- **Noise**
- Bad contrast
- Inhomogeneous Lighting
- ...

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

- Homogeneous zones
- Transition zones

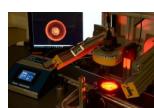


Image processing

Steps for processing an image

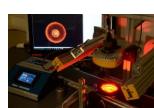
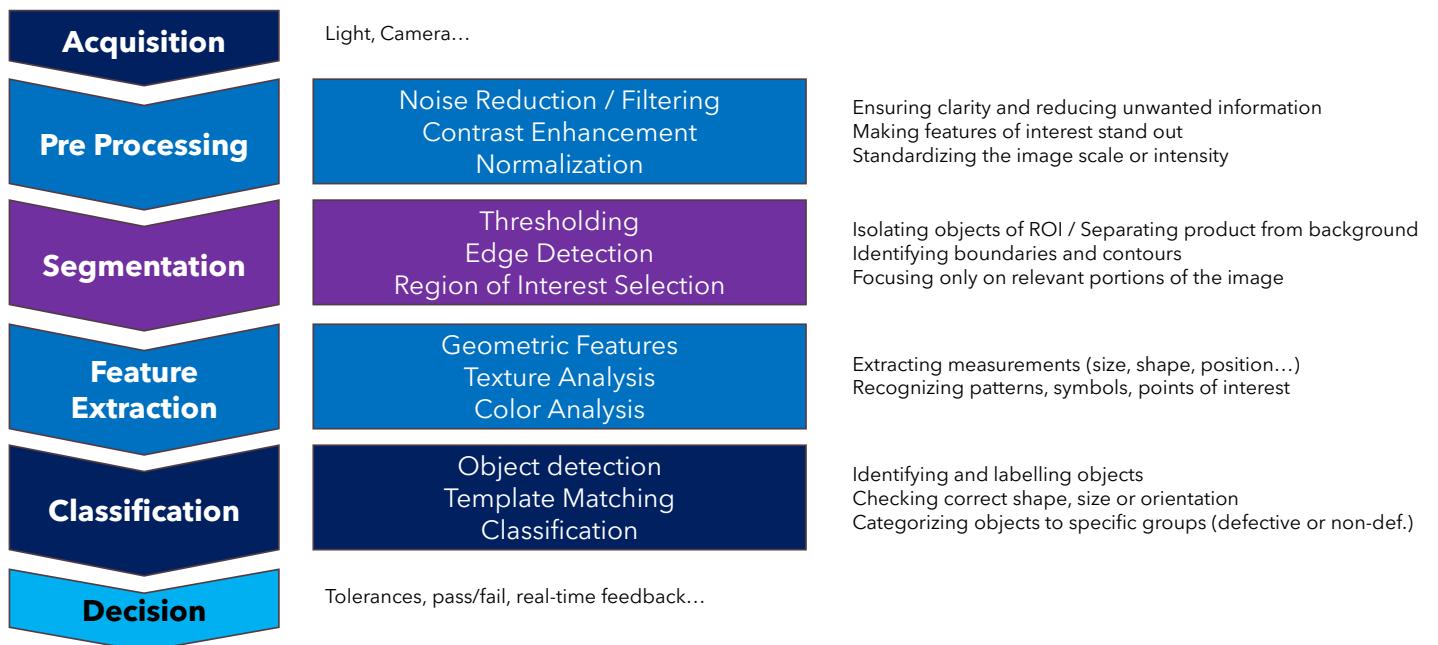


Image processing with OpenCV

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>

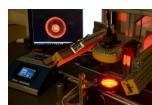
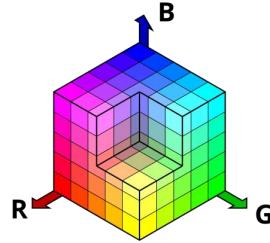


Image processing with OpenCV

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.



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

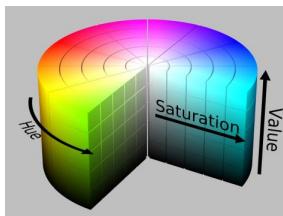
HSV

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

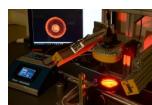


Image processing with OpenCV

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)
```

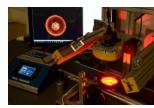
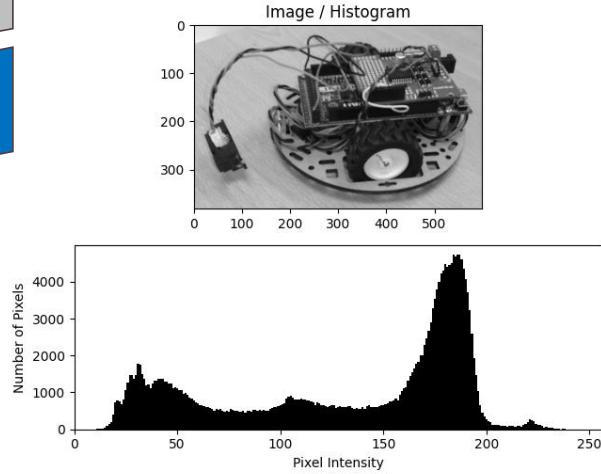


Image processing with OpenCV

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])
```

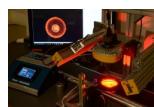
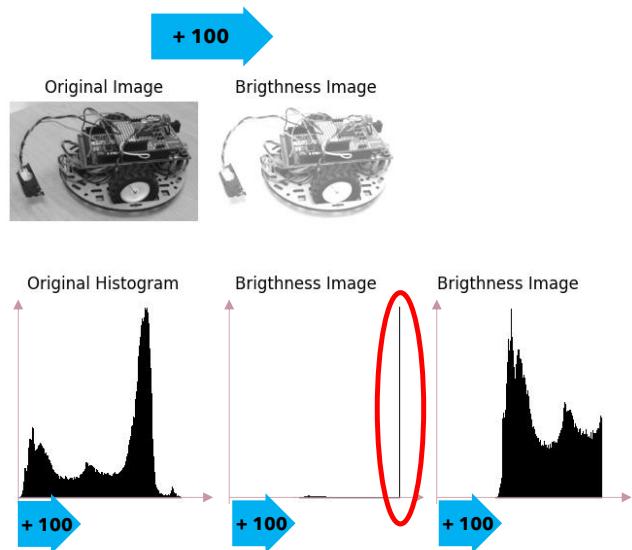


Image processing with OpenCV

OpenCV / Contrast and Brightness

Acquisition

Pre Processing



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

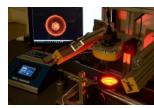
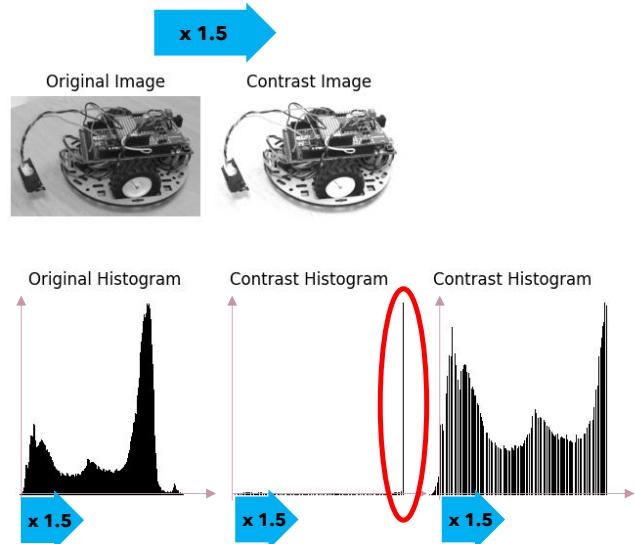
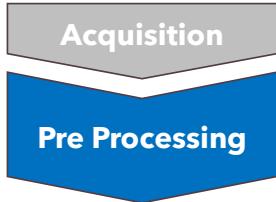


Image processing with OpenCV

OpenCV / Contrast and Brightness



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

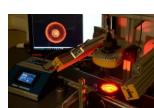
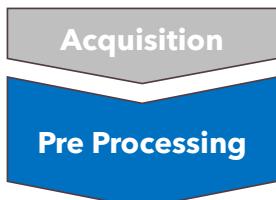


Image processing with OpenCV

OpenCV / Convolution Kernel (or Structuring Element)



```
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

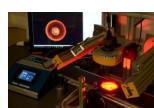
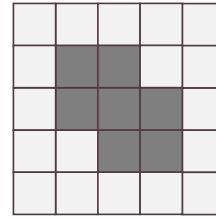
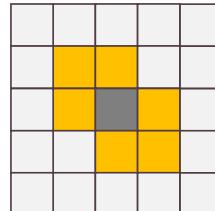


Image processing with OpenCV

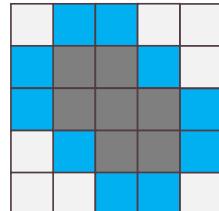
OpenCV / Erosion and Dilation



Original pixels
Removed pixels



Added pixels



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

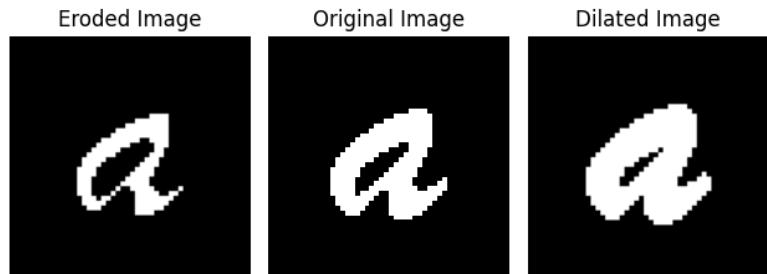
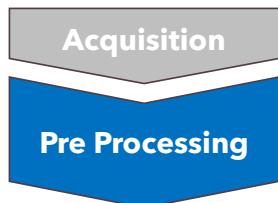
<https://www.youtube.com/watch?v=fmyE7DialYQ>

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



Image processing with OpenCV

OpenCV / Erosion and Dilation



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)
```

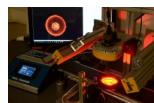
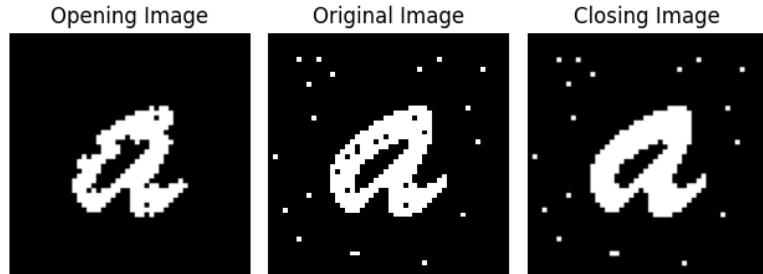
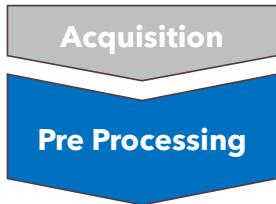


Image processing with OpenCV

OpenCV / Opening and Closing morphological transforms



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)
```

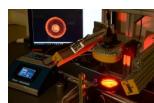
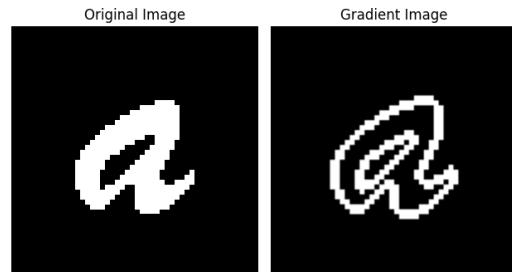
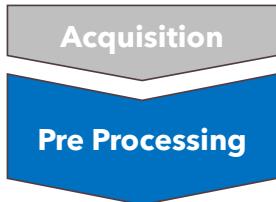


Image processing with OpenCV

OpenCV / Gradient morphological transforms



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)
```

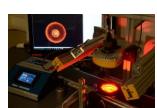


Image processing with OpenCV

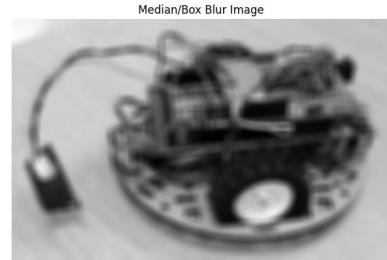
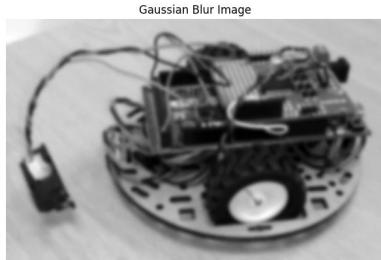
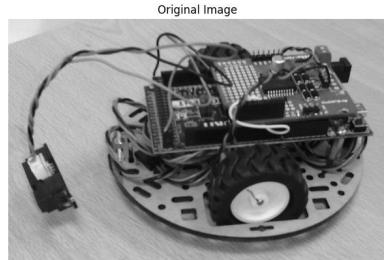
OpenCV / Blur and mean

Acquisition

Pre Processing

kernel_size = (N,M)

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



Removing irrelevant details

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)

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

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

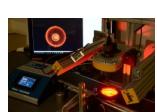


Image processing

Fourier Transform and filtering

Acquisition

Pre Processing

Segmentation

Feature Extraction

