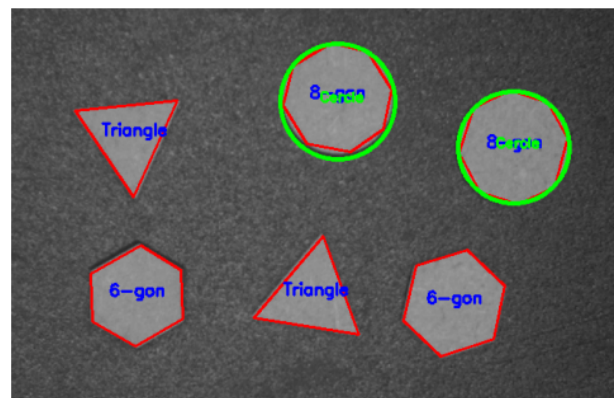
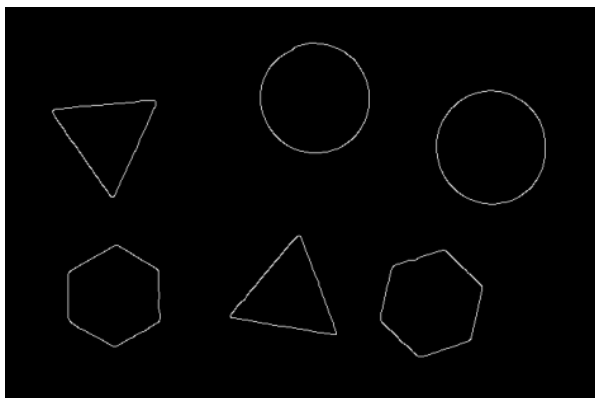


Travaux Pratiques

Semestre 6

Vision Industrielle

Objectifs des 4 séances
TP 3-4 / Détection d'objets



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.

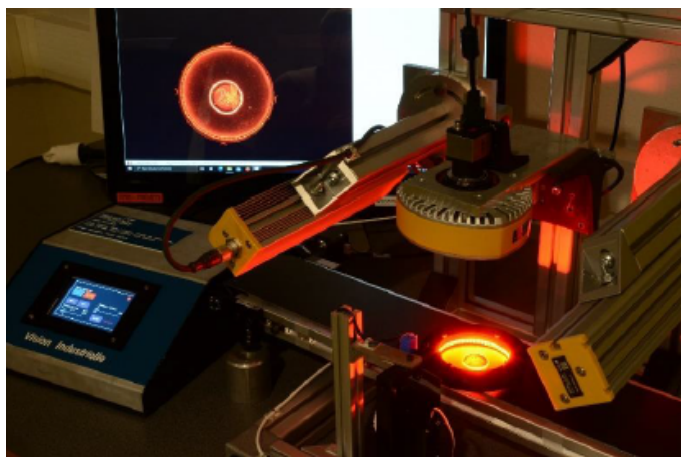


© 2025 by LEnsE-IOGS

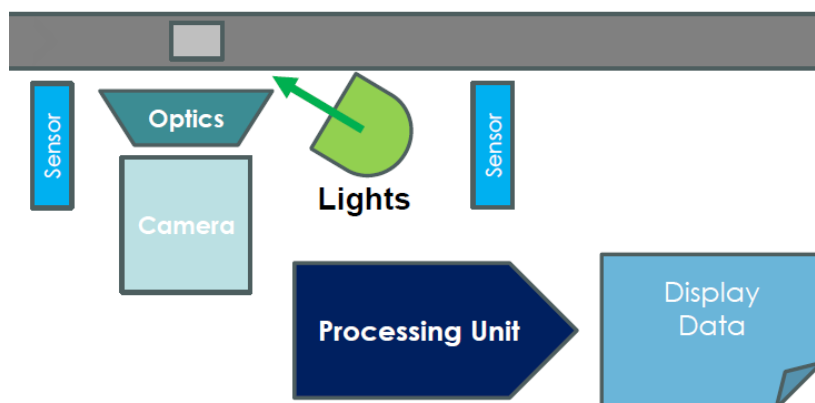
*Ce sujet a été co-conçu par une équipe d'étudiant-es - Joséphine BECHU, Justine GABRIEL et Paul CHENEAU
- lors d'un projet DEPhI de 2A - 2025-2026 - et l'équipe pédagogique d'Interfaçage Numérique.*

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.



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

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

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

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 : 5 / 8 / 10 mm), un éclairage Effilux Ring-RGB et un ordinateur.

Voir le sujet de la séance TP1. Le 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*.

TP2 - Manipulations de base sous OpenCV

Le **TP2** se fera sur un ordinateur possédant une installation de Python (3.11 ou 3.13) et des bibliothèques standard, dont OpenCV.

Voir le sujet de la séance TP2. Le 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*.

TP3-4 - Détection d'objets

Le détail des séances 3 et 4 est donné dans la suite de ce document.

Accumulation de preuves / Méthode de travail

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

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

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

Objectif des séances 3 et 4

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

Sous OpenCV :

- Détecter des formes [40 min]
- Détecter des couleurs [40 min]
- Changer d'espace de couleur [40 min]

Segmentation de l'image ?

Sur le banc de VI :

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

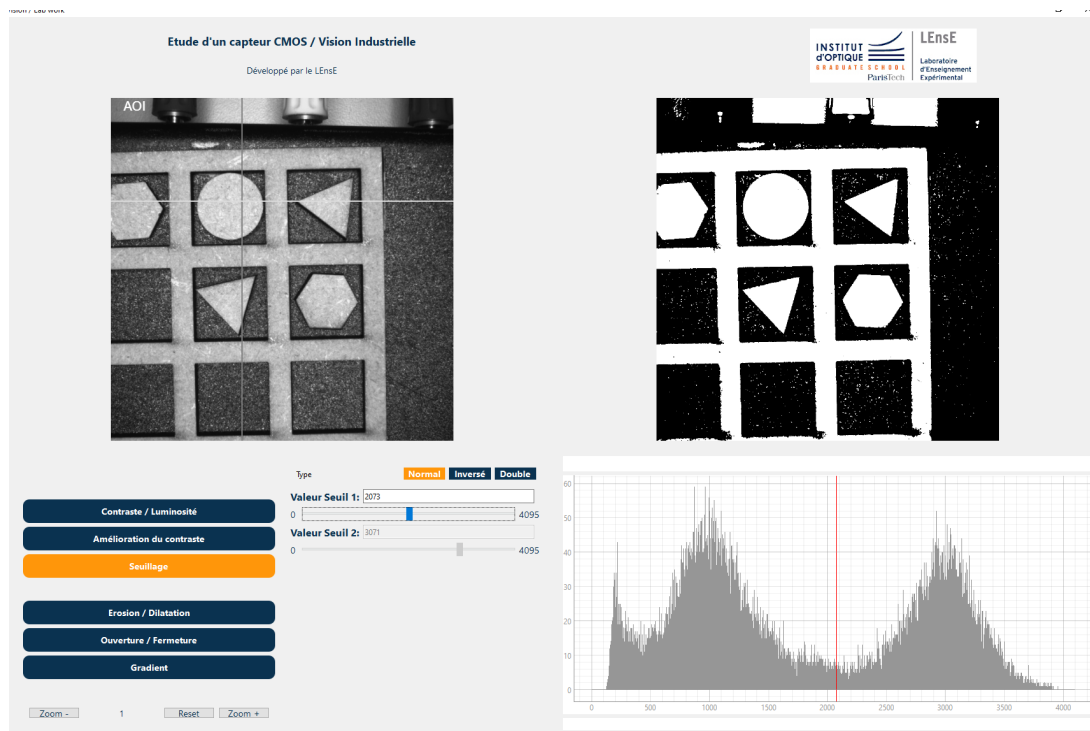
TP4 - Détection d'objets - Contrôle qualité

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 ?

Premiers essais sur une grille de jeu TicTacToe (morpion) :



OpenCV - Détection de formes et de couleurs

Une partie des algorithmes de traitement sont donnés. Il est fortement conseillé de **réaliser une analyse poussée** de chacune des étapes de ces algorithmes afin de comprendre l'intérêt de l'utilisation de ces techniques de base.

Vous pouvez par exemple **décrire ces algorithmes** par l'intermédiaire d'un graphique (algorithme) et **afficher les résultats intermédiaires** (et en faire une capture d'écran) tout en **commentant les résultats**.

CV-A - Détecter des formes [40 min]

Détection de contours sur une image "propre"

- M Ouvrir le fichier 20_DETECTION_FORME.PY du répertoire des codes à tester.
- M Exécuter ce code.
- Q Que fait ce programme ? A quoi correspondent les images affichées ? A quoi correspondent les données affichées ?

→ M Modifier le paramètre MIN_AREA (100, 1000, 10000).

→ Q A quoi sert ce paramètre ? Quelle est son unité ?

→ M Modifier le paramètre APPROX_FACTOR (0.01, 0.05, 0.1).

→ Q A quoi sert ce paramètre ?

Détection de cercles

- M Ouvrir le fichier 21_DETECTION_FORME_CERCLES.PY du répertoire des codes à tester.
- M Exécuter ce code.
- Q Quelle amélioration apporte ce programme ?

Détection de contours sur une image "réelle"

L'image précédente était "propre" (sans bruits dûs au capteur par exemple). Nous allons nous intéresser à présent à une image provenant d'une chaîne d'acquisition.

→ M A partir du code précédent, ouvrir l'image FORMES_BLANC_30MS.PNG du répertoire du kit d'images.

→ Q L'algorithme est-il efficace ?

→ M Ouvrir le fichier 22_DETECTION_FORME_PLUS.PY du répertoire des codes à tester.

→ M Exécuter ce code.

→ Q Quelle étape est-elle ajoutée par rapport à la version précédente du code ?

→ M Ajouter un traitement morphologique de type *opening* (ouverture) sur l'image binarisée (noyau carré de 3x3, puis 5x5).

- M Ajouter l'affichage de l'image ainsi traitée.
- Q Que permet cette étape dans l'algorithme de détection ?
- Q L'algorithme de détection de cercles fonctionne-t-il avec cette étape supplémentaire ?

Détection de cercles sur une image "réelle"

L'**algorithme de Hough** qui permet la recherche de cercles dans une image se base sur des images en nuance de gris (et non binarisée). Il s'appuie sur des gradients d'intensité et des bords continus et doux pour détecter les transitions (contours des cercles).

Il n'est donc pas possible d'utiliser une opération morphologique d'ouverture sur l'image.

- M Ouvrir le fichier `23_DETECTION_FORME_BLUR_CERCLES.PY` du répertoire des codes à tester.
- M Exécuter ce code. Vérifier que l'algorithme n'est pas efficace.

→ M Ajouter un filtrage de type *blur* gaussien (`cv2.GaussianBlur`) sur l'image grise (noyau carré de 7×7 et $\sigma = 1.9$).

- M Ajouter l'affichage de l'image ainsi traitée.
- Q Que permet cette étape dans l'algorithme de détection ?
- M Modifier la taille du noyau (3×3 , 5×5 , 15×15) et de la valeur de σ (1.9, 3, 5, 10).
- Q Quel est l'impact du choix de ces deux paramètres ?

CV-B - Détecter des couleurs [40 min]

Jouer avec les canaux de couleurs

- M Ouvrir et afficher l'image `couleurs_4.png` du kit d'images fourni, au format RGB.
- 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 et afficher la nouvelle image.

Transformer en nuances de gris

CV-C - Changement d'espace de couleur [40 min]

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.

Banc VI - Calibration

VI-A - Calibrer la chaîne pour les cubes de couleur [40 min]

VI-B - Calibrer la chaîne pour les formes colorés [40 min]

VI-C - Calibrer la chaîne pour des mesures de distance [40 min]