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

Institut d'Optique Graduate School

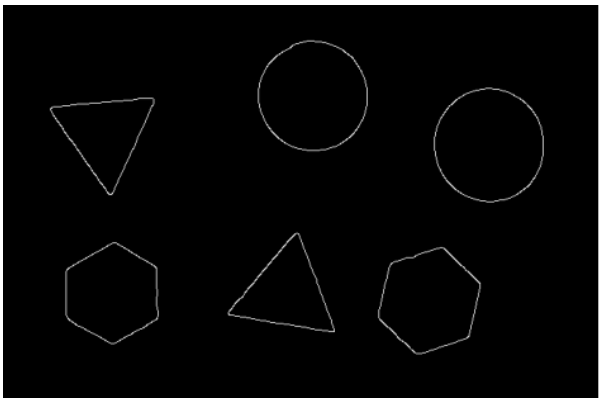
INTERFAÇAGE NUMÉRIQUE

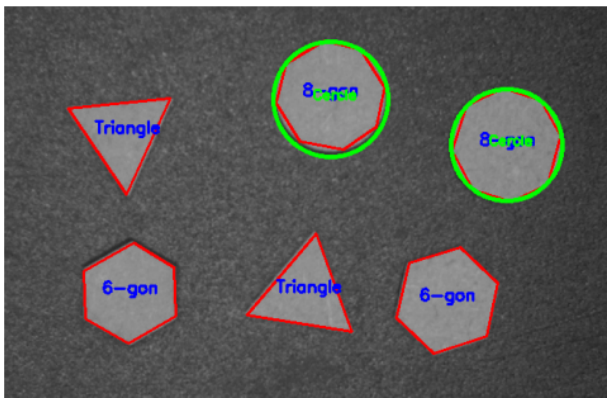
Travaux Pratiques

Semestre 6

Vision Industrielle

2 séances
Détection d'objets : formes et couleurs





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.

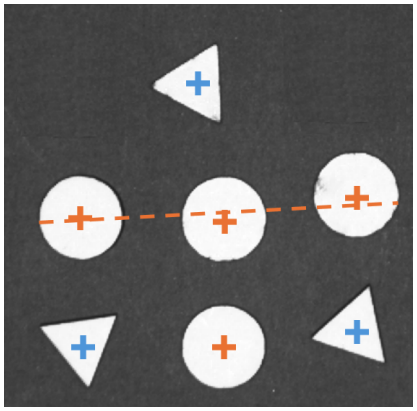
CC BY NC SA

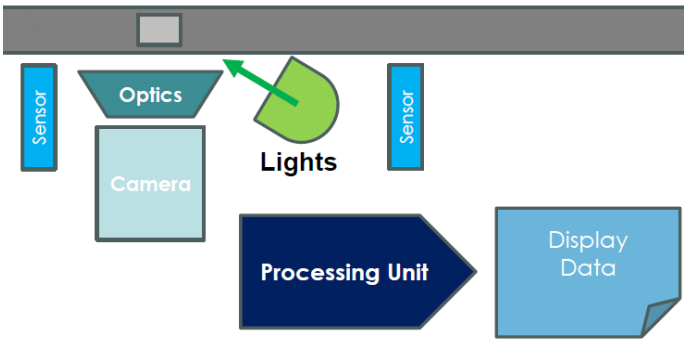
© 2027 by LEnsE-IOGS

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 des 2 séances autour de la **Vision Industrielle**, les étudiant·es seront capable de :

— Réaliser un prototype d'inspection ou de mesure simple (tri d'objets)

Au semestre 5...

A travers un TP, vous avez pris en main un **banc de Vision industrielle**, incluant : une caméra, un objectif, un éclairage et une interface de pilotage. Vous avez pu **observer l'impact** de l'ensemble de ces éléments **sur l'image finale** : temps d'intégration, uniformité de l'éclairage, contraste entre les objets...

Dans un second TP, vous avez pu **appréhender les premières briques de traitement d'image** (via la bibliothèque OpenCV) : données d'une image numérique, histogramme, flou dans une image, opérations d'ouverture/fermeture... Ces étapes sont essentielles pour **préparer l'image à la segmentation et la classification** des objets qu'elle contient.

Dans ces deux nouvelles séances...

Lors de ce semestre, en deux séances, vous allez **mettre en place deux systèmes indépendants d'inspection de pièces**, en "temps réel" sous caméra, pour se rapprocher des systèmes industriels :

— le premier permettant de **détecter la couleur d'un objet** placé sous la caméra (cubes)
— le second permettant de **détecter l'alignement de formes sur une grille**, en se basant sur la règle du jeu du Morpion (ou *Tic Tac Toe* en anglais).

Ressources

Un **fichier archivé**, nommé _STEP_BY_STEP_S6.ZIP, regroupant l'ensemble des codes à tester au cours de la séance de découverte d'OpenCV, ainsi que les images à traiter, est à téléchargé sur le site du LEnsE :

Année / Première Année / Interfaçage Numérique S6 / Bloc 2 – Vision Industrielle / Ressources

— Téléchargez ce fichier dans un répertoire du disque dur **U :** de l'ordinateur.
— Faites un clic droit sur le fichier *zip* puis **Extraire tout**. Décompressez l'archive.

Accumulation de preuves / Méthode de travail

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.

Validation de la compétence C4

Valider une solution technologique

Il est également crucial pour les chaînes de production, par exemple, d'assurer une **cadence de fabrication et de validation importante**. Les algorithmes de détection doivent donc pouvoir s'exécuter rapidement et la vitesse de défilement des pièces sous le système imageant peut devenir limitant.

Il est enfin nécessaire de valider ces systèmes de prise de décision en mesurant leur **taux d'erreur** vis-à-vis des grandeurs à tester. Ce taux d'erreur doit être proche de 0, une tolérance peut être acceptée et peut être différente selon le type de défauts à identifier, le type de matériaux utilisés...

Dans le cadre de ces TP, il sera important de **définir des protocoles de test pour ces deux aspects** (temps d'exécution des algorithmes et taux d'erreur) et de donner quelques mesures quantitatives.

Objectif des deux séances

L'utilisation d'un banc de vision industrielle est indispensable pour les deux séances.

TP1 - Manipulations avancées / Formes et couleurs

Dans cette première séance, vous allez devoir mettre en place un système de détection d'objets (formes simples) et de couleurs, afin d'identifier la couleur du (ou des) cube(s) de couleur que vous mettrez sous la caméra.

Des **scripts** vont vous permettre, **pas à pas**, de :

- récupérer le flux de la caméra depuis OpenCV (calibration de la caméra, boucle d'acquisition des images...)
- détecter des formes (à partir de la quantité de contours - segments - d'un objet)
- détecter la couleur de ces objets (utilisation d'espaces colorimétriques appropriés)

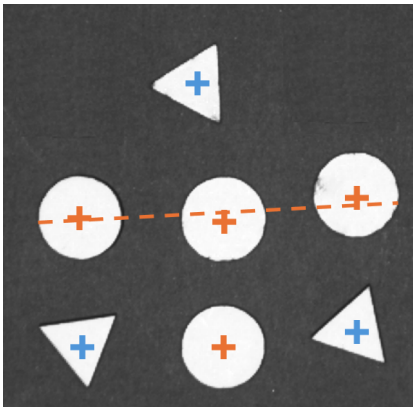
Vous devrez, à partir de ces scripts de base, **réaliser** et **faire valider une application simple** mais **robuste** de détection de la couleur d'objets placés sous la caméra.

TP2 - Détection d'objets - Contrôle qualité

L'objectif final de ce second TP est de **détecter le nombre d'objets d'une forme particulière et leur alignement sur une grille**, en temps réel sous caméra.

On se basera sur la règle du **jeu du Morpion** (ou *Tic Tac Toe* en anglais) qui oppose deux joueurs qui doivent aligner 3 marques (de même forme ou couleur) sur une grille de 3 x 3 cases.

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



Cette mission "ludique" est cependant très proche des **tâches d'inspection en industrie**, afin par exemple 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).

On pourra ajouter également des mesures de surface des pièces, de colorimétrie pour valider la couleur des dépôts faits sur les pièces... pour vérifier leur **conformité à un cahier des charges**.

Travaux Pratiques

Semestre 6

TP1 - Manipulations avancées sous OpenCV
Formes et couleurs

Méthode de travail (conseillée)

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 (algorigramme) et **afficher les résultats intermédiaires** (et en faire une capture d'écran) tout en **commentant les résultats**.

Faites des copies des scripts fournis, afin de garder une trace des évolutions que vous avez apportés.

Mesure de temps d'exécution

Il est possible d'avoir une **estimation du temps d'exécution** d'un code en Python par la méthode proposée ci-dessous :

```
1 import time
2
3 start = time.perf_counter()
4 # Code to evaluate
5 end = time.perf_counter()
6
7 print(f"Total Time : {(end - start)*1000:.2f} ms")
```

A - Flux de la caméra (30 min)

➜ **M** Allumer l'**éclairage annulaire** du banc (trois couleurs). Placer un **fond uniforme** sous l'éclairage (feuille blanche, paillasse, fond sombre...) et quelques objets (cubes de couleur par exemple).

➜ **M** Ouvrir le fichier 101_FLUX_CAMERA_LENSEPY_SIMPLE.PY du répertoire des codes à tester.

➜ **M** Exécuter ce code.

*Si la caméra est bien connectée en USB à l'ordinateur, vous devriez voir s'afficher son flux. Ce script se base sur la bibliothèque **lensepy_app** développée par le LEnsE.*

➜ **Q** Détaillez les parties du script fourni.

➜ **Q** De quelle taille est l'image ?

➜ **M** Ouvrir le fichier 102_FLUX_CAMERA_LENSEPY_HISTO.PY du répertoire des codes à tester. Exécuter ce code. Modifier le temps d'exposition pour obtenir une image exploitable.

➜ **Q** Sur quel(s) critère(s) vous basez-vous pour valider l'exploitabilité de votre prise d'image ?

➜ **M** Ouvrir le fichier 103_FLUX_CAMERA_LENSEPY_PROCESS.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Détaillez les parties du script fourni.

B - Détection de formes (60 min)

➜ **M** Ouvrir le fichier 111_DETECTION_FORME.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Détaillez les parties du script fourni.

➜ **Q** Ce code fournit-il un résultat satisfaisant ? A quoi servent notamment les paramètres MIN_AREA et APPROX_FACTOR ?

➜ **Q** Et dans le cas de la seconde image proposée (FORMES_BLANC_30MS.PNG), le résultat est-il satisfaisant ? Quel est la différence majeure entre ces deux images ?

➜ **M** Ouvrir le fichier 112_DETECTION_FORME_PLUS.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Quelle étape a été ajoutée par rapport à la version précédente ? Le résultat est-il satisfaisant ?

➜ **M** Modifier la valeur de MIN_AREA pour optimiser la détection de contours valides (suppression du bruit induit par le fond).

➜ **Q** L'ajout d'une étape de moyennage (*blur*) est-il pertinent ?

➜ **Q** L'association d'un flou (*blur*) et d'une binarisation permet-elle d'éliminer les erreurs de détection de contours ? (sans utilisation de MIN_AREA).

➜ **M** Ouvrir le fichier 113_DETECTION_FORME_CERCLES.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Détaillez les étapes de ce code et le résultat obtenu.

➜ **Q** Dans le cas de la seconde image proposée (FORMES_BLANC_30MS.PNG), le résultat est-il satisfaisant ?

➜ **M** Ouvrir le fichier 113B_DETECTION_FORME_CERCLES_PLUS.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Quelle étape a été ajoutée par rapport à la version précédente ? Le résultat est-il satisfaisant ?

➜ **M** Modifier la taille du noyau et la valeur de σ (entre 1 et 5). Cela améliore-t-il le résultat ? Quelles sont les valeurs limites ?

C - Détection de formes et de couleurs sous caméra (60 min)

➜ **M** Ouvrir le fichier 121_FLUX_CAMERA_CROP.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Que permet-il de faire ?

➜ **M** Ajouter un effet de filtre gaussien uniquement sur la partie découpée de l'image (noyau de 15 pixels de côté et σ de 3).

➜ **M** Ouvrir le fichier 122_FLUX_CAMERA_MASK.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Que permet-il de faire ?

➜ **M** Adapter la taille de la découpe à la taille d'un cube de couleur (en laissant quelques dizaines de pixels autour de l'objet).

➜ **Q** A quoi correspond l'image appelée `Mask` ?

➜ **Q** La détection des objets est-elle meilleure sur fond clair ou sur fond sombre ?

➜ **M** Ouvrir le fichier 123_FLUX_CAMERA_MASK_SQUARE.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Que fait-il de plus que le précédent ?

➜ **M** A partir des scripts précédents (également ceux étudiés en section A) et des différentes observations effectuées dans la section B, mettre en place un script qui, à partir du flux de la caméra :

1. Affiche le flux de la caméra

2. Détecte un objet de forme carrée dans l'image (cube placé sous la caméra)
3. Affiche les valeurs RVB moyennes de l'objet

Penser à régler correctement le temps d'intégration.

➜ **Q** Est-il facile de déterminer la couleur d'un objet par cette méthode ?

D - Détection de couleurs efficace (30 min)

Utiliser l'espace colorimétrique HSV

L'espace colorimétrique **HSV** (*Hue, Saturation, Value*) représente les couleurs de manière intuitive en les séparant selon :

- la **teinte** (Hue), qui correspond au type de couleur et est généralement exprimée comme un angle sur un cercle (rouge, vert, bleu, etc.),
- la **saturation** (Saturation), qui mesure la pureté ou l'intensité de la couleur (d'une couleur vive à un gris),
- la **valeur** (Value), qui représente la luminosité globale.

Contrairement à l'espace RVB, HSV est conçu pour se rapprocher de la perception humaine, ce qui le rend particulièrement adapté aux tâches de vision par ordinateur, notamment la détection et la segmentation de couleurs, car il permet de dissocier plus facilement l'information de couleur des variations d'éclairage.

➜ **M** Ouvrir le fichier 131_COULEUR_HSV.PY du répertoire des codes à tester.

➜ **M** Exécuter ce code.

➜ **Q** Que représentent les deux lignes de figure affichées par ce programme ?

➜ **Q** Comment pouvez-vous interpréter les deux histogrammes ?

➜ **M** Ajouter un filtrage de type *blur* gaussien (*cv2.GaussianBlur*) sur l'image RVB FORMES_BRUIT.PNG (noyau carré de 7x7 et $\sigma = 1.9$).

➜ **Q** Quel est l'effet sur l'histogramme associé à cette image ? Quel serait l'inconvénient de cette technique, par rapport à la détection de couleur via l'espace HSV ?

Segmenter une image en fonction de la couleur

➜ **M** Ouvrir le fichier 132_HSV_RGB.PY du répertoire des codes à tester.

➜ **M** Exécuter le code.

➜ **Q** Expliquer le principe utilisé dans chacune des segmentations, en RVB et HSV. On s'intéressera en particulier aux bornes utilisées.

➜ **Q** Quelle méthode est la plus efficace pour détecter la couleur rouge ?

➜ **M** Ajouter la détection de la zone bleue, de l'image initialement ouverte, par la méthode qui vous parait la plus pertinente. Afficher une nouvelle image avec le résultat final.

Segmenter une image en fonction de la couleur

➜ **M** A partir du script que vous avez mis en place dans la partie C (détection d'une couleur RVB d'un objet), réaliser un script qui permet de **déterminer les valeurs HSV** des différents cubes à votre disposition.

➜ **M** Mesurer les valeurs HSV de chacun des cubes de couleur à votre disposition.

➜ **Q** Quelles valeurs sont dépendantes de la quantité de photons captées par la caméra ?

Vous pourrez également tester sur plusieurs cubes d'une même couleur pour tester la répétabilité de vos mesures et vérifier la robustesse du protocole que vous allez devoir mettre en place par la suite.

Cette étape servira de calibration pour la détection de la couleur des cubes que vous mettrez en place dans la section suivante.

E - Application de détection de couleurs (60 min)

➜ **M** A partir de l'ensemble des éléments précédents, vous devez mettre en place une solution technologique (script) permettant :

1. de **détecter la position d'un cube** placé sous la caméra
2. d'**afficher la couleur du cube** (parmi les 4 possibles)

On ne placera dans un premier temps qu'un seul cube de couleur sous la caméra.

➜ **Q** Faire valider votre système ainsi que ces performances.

➜ **M** Etendre votre solution à plusieurs cubes placés sous la caméra.

Faire valider votre application par un·e encadrant·e.

F - Niveaux de gris (Bonus)

Il existe plusieurs méthodes pour transformer une image en RVB vers une image en nuance de gris.

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 :

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

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.

➜ **M** Ouvrir le fichier 141_COLORSPACE.PY du répertoire des codes à tester.

➜ **Q** Que fait ce programme ?

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

Utiliser l'espace colorimétrique YUV

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** Ouvrir le fichier 142_COLORSPACE _YUV.PY du répertoire des codes à tester.

➜ **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.

Travaux Pratiques

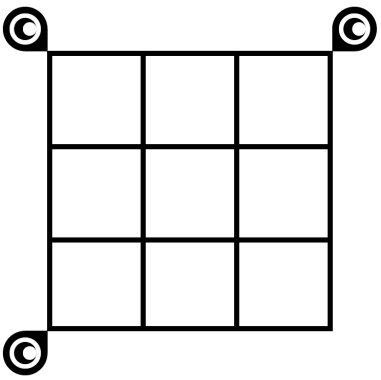
Semestre 6

TP2 - Détection d'objets / Contrôle qualité

Validation d'une grille de TikTakToe

Dans ce second TP, vous allez devoir mettre en place un **système d'inspection** de grilles du jeu **TikTakToe**. Ce système s'appuie sur l'utilisation d'un **banc de vision industrielle** et d'un algorithme de **traitement de l'image**, que vous allez devoir concevoir, développer et valider.

On s'intéressera dans un premier temps au **repérage de la grille dans l'espace**. Des repères spécifiques ont été ajoutés sur cette grille.



Dans un second temps, vous mettrez en place la **détection de formes à l'intérieur des 9 cases** de la grille afin d'en déduire si l'un des deux joueurs à gagner la partie.

G - Détection de repères

Des **repères** sont souvent ajoutés sur des objets à détecter : QR Code, marqueurs ArUco... Ceux-ci servent principalement à permettre de **détecter**, **localiser**, **orienter** et **transformer** correctement un objet dans une scène.

Trois repères sont souvent nécessaires pour :

- déterminer les limites de l'objet,
- connaître son angle de rotation,
- retrouver la géométrie en cas de prises d'image en biais (perspective).

Premier essai sur images

On se propose d'étudier le fichier 201_CORRELATION.PY, associé aux images MIRE_REP.PNG et MIRE_TIK.PNG (du répertoire _DATA_MARKER).

➜ **M** Ouvrir les deux images.

➜ **Q** Que représentent-t-elles ?

➜ **M** Exécuter le script et commenter le résultat.

➜ **M** Ouvrir le script 202_CORRELATION_ROTATION.PY, exécuter-le et commenter le résultat.

➜ **Q** Qu'est-ce qui a changé entre les deux exécutions ? Tester d'autres angles de rotation.

➜ **M** Tracer l'évolution du maximum de corrélation pour des angles de rotation allant de -30° à 30° par pas de 2°.

Faire valider votre résultat par un·e encadrant·e.

Premier essai sous flux caméra

➜ **M** Placer une mire avec des repères carrés sous la caméra.

➜ **M** Ouvrir le script 203_CORRELATION_CAMERA.PY, exécuter-le et commenter le résultat.

➜ **Q** Pourquoi la corrélation n'est pas efficace dans ce cas ?

➜ **M** Changer le facteur de grandissement de l'image de référence de 1 :

- à **1.83** pour une focale de **16 mm**,
- à **0.91** pour une focale de **8 mm**.

➜ **Q** Que se passe-t-il lorsqu'on fait tourner la mire sous la caméra ?

OpenCV et la détection d'objets

La bibliothèque *OpenCV* propose différentes fonctions permettant de repérer des objets dans une scène. En voici une liste non exhaustive.

Détection de contours	cv2.findContours()	Détecter les formes
Détection de coins	cv2.goodFeaturesToTrack()	Trouver des points caractéristiques
Détection de marqueurs	cv2.aruco	Identifier des repères connus
Transformation affine	cv2.getAffineTransform()	Rotation, translation, cisaillement
Transformation perspective	cv2.getPerspectiveTransform()	Corriger une perspective

H - Application finale

Le script 221_CAMERA_FLUX_MARKER.PY donne une base de travail pour la réalisation de l'application finale.

➜ **M** Ouvrir ce script, exécuter-le et commenter le résultat, en plaçant la grille aux repères carrés sous le flux de la caméra.

➜ **M** Faire un algorigramme des fonctions *detect_zone()*, *display_grid()* et *main()*.

➜ **Q** A quoi correspondent les variables dans les zones *PARAMETERS* et *GRID DEFINITION* ?

A partir du script précécent, on souhaite réaliser une application qui :

- **repère les formes** dans chaque zone de la grille (et les affiche)
- valide une grille de Tic Tac Toe

Faire valider votre application finale par un·e encadrant·e.

Vous pouvez également faire valider des opérations partielles. Pensez également aux preuves pour la compétence C4 (protocoles de tests, résultats des tests, analyse partielle...).

I - Détection de repères multi-échelles (Bonus)

On se propose à présent d'étudier le fichier 211_CORRELATION_MULTI.PY, associé aux images (du répertoire _DATA_MARKER) :

- MIRE_REP_V2_1.PNG
- MIRE_TIK_V2.PNG

➜ **M** Ouvrir les deux images.

➜ **Q** Que représente-t-elle ? Qu'est-ce que cela va impliquer sur la recherche des motifs dans l'image ?

➜ **M** Ouvrir le script 211_CORRELATION_MULTI.PY et exécuter-le.

➜ **Q** Que se passe-t-il si on change l'échelle de l'image du repère :

- dans les limites prévues dans la fonction `find_all_matches_multiscale()` ?
- en dehors de ses limites ?

➜ **M** Ouvrir le script 211B_CORRELATION_MULTI.PY et exécuter-le.

➜ **Q** Expliquer ce code et son résultat.

➜ **M** A partir 203_CORRELATION_CAMERA.PY et 211_CORRELATION_MULTI.PY, faire une application qui permet de détecter les trois coins de la grille de TikTakToe.

Vous prendrez la grille avec les repères qui vous paraissent les plus pertinents.

Faire valider votre application par un·e encadrant·e.