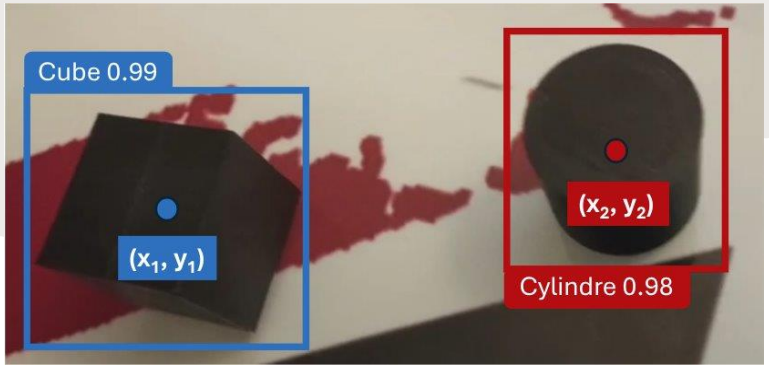
Informatique / iut de BORDEAUX

SAE S5.A.01-7 Audit application

Mélodie Daniel, Clément Gaspard, Grégoire Passault & Arnaud Pêcher

2025 - 2026

Sujet de la SAE S5.A.01-7



- Audit de performance applicative en apprentissage supervisé comparant **YOLO** et un **CNN custom** pour la détection d'objets dans une image, leur localisation 3D dans le repère caméra et l'estimation de leur position dans un repère monde à définir.

- **Compétences à acquérir**
  - Prise en main de capteurs : caméra RGB & caméra de profondeur (Depth camera en anglais).
  - Création d'une base de données (Dataset en anglais).
  - Mise en œuvre d'algorithmes d'apprentissage supervisé : YOLO, CNN custom.
  - Exploitation de bibliothèques IA et vision par ordinateur : Python, PyTorch, OpenCV, Label Studio.
  - Paramétrage, optimisation et évaluation de modèles : réglage des hyperparamètres, analyse des performances d'un modèle de Deep Learning.
  - Gestion et organisation d'un projet.
- **Préambule de la SAE S6.A.01** : "Évolution d'une application".

Application à réaliser

Spécifications fonctionnelles

- Matériel fourni : les objets à détecter et à localiser, une caméra (RealSense D435i ou D435if) et un ordinateur équipé d'une carte GPU (salle 210).
- Annotation/Labellisation de la base de données : Label Studio (https://labelstud.io/guide/index.htm).
- Méthode de référence : YOLOv11
  - https://docs.ultralytics.com/fr/models/yolo11/
  - https://github.com/yt7589/yolov11
  - https://blog.roboflow.com/what-is-yolo11/
- Méthode à développer : CNN custom avec PyTorch
  - https://melodiedaniel.github.io/deep_learning/
  - https://docs.pytorch.org/examples/index.html
- Interface graphique : exemples d'outils
  - Streamlit : https://streamlit.io/
  - Gradio : https://www.gradio.app/guides/quickstart

Informatique / iut de BORDEAUX

Modalités

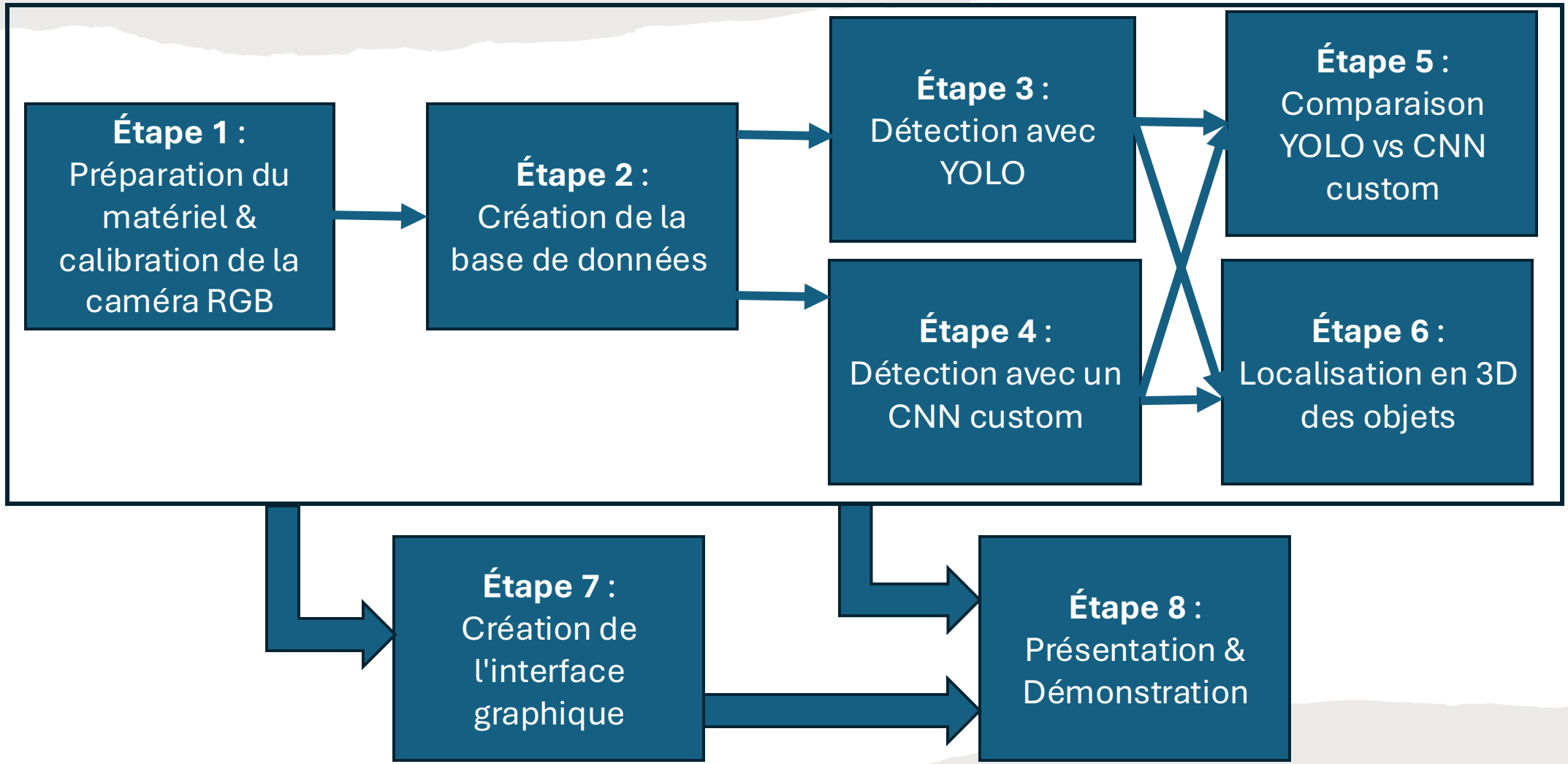
Organisation

- Par équipes de 5-6 personnes. Les groupes sont disponibles sur Moodle (https://moodle.u-bordeaux.fr/user/index.php?id=25180).
- Désignation d'un responsable du matériel par équipe.
- Planning : sur 4 jours du 02/02/26 au 06/02/26 (~36h dont ~2-3h d'évaluation orale).
- **Conseil** : créer par groupe un git repository pour le projet.

Évaluation

- 4 Points : Évaluation individuelle (auto-évaluation).
- 16 Points : Présentation et démonstration du projet (pensez à prendre des vidéos au cas où votre projet n'est pas fonctionnel le jour J).

Déroulement de la SAE



Étape 1 : Préparation du matériel & calibration intrinsèque de la caméra RGB

Prérequis : un membre de chaque groupe récupère les objets à détecter et à localiser (un cube orange et un cylindre bleu) et une caméra RealSense D435i ou D435if (https://github.com/realsenseai/librealsense) auprès de votre enseignant(e).

Rappel : des PC sous Ubuntu équipés d'une carte GPU sont disponibles dans la salle 210.

```
==========================================
Résumé: 4 caméra(s) disponible(s)
==========================================

Caméras disponibles: [0, 35, 37, 38]

Quelle caméra tester? (0/35/37/38):
```

Documents fournis sur Moodle :

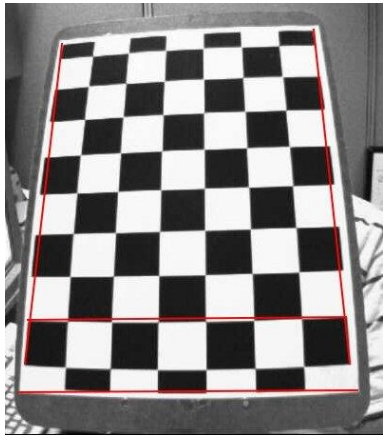
- Le fichier *scan_cameras.py* permet de trouver les id des caméras branchées et de voir l'image renvoyée par chaque caméra.
- Le fichier *test_camera.py* permet de faire une vidéo .mp4 de ce que voit la caméra.

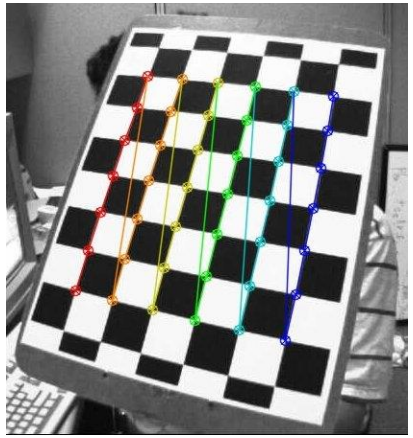
Conseil : Créer un environnement virtuel Python dans lequel vous installerez tous les packages nécessaires.

Étape 1 : Préparation du matériel & calibration intrinsèque de la caméra RGB

Calibration intrinsèque

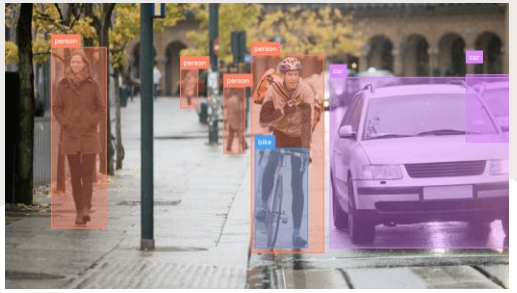
- **Objectif** : caractériser les propriétés internes de la caméra pour pouvoir convertir des coordonnées 3D → pixels.
- **Paramètres principaux de la matrice (repère) de la caméra** :
  - **Focale (f_x, f_y)** : mesure l'agrandissement (exprimée en pixels).
  - **Centre optique (c_x, c_y)** : position du point principal sur l'image.
  - **Facteur de pixel** : rapport entre la taille physique du pixel et le capteur.
  - **Distorsions de lentilles :**
    - distorsion radiale (k_1, k_2, k_3).
    - distorsion tangentielle (p_1, p_2).





- **Méthode courante** :
  - Prise de vues d'un motif connu (damier/mire ou ChessBoard en anglais).
  - Optimisation via OpenCV pour estimer les paramètres à l'aide de la méthode ***cv.calibrateCamera()***.
- **Attention** : il faudra mesurer et vérifier la précision de la calibration intrinsèque.
- **Documentation** : https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.

Étape 2 : Création de la base de données



```
dataset_yolo/
├── images/              # Toutes vos images annotées
│   ├── frame_00001.jpg
│   ├── frame_00002.jpg
│   └── ...
├── labels/              # Fichiers .txt (même nom que l'image)
│   ├── frame_00001.txt
│   ├── frame_00002.txt
│   └── ...
├── classes.txt          # Liste des classes : une par ligne
└── notes.json           # Métadonnées
```

- **Prérequis** : une vidéo .mp4 dont certaines images (Frames en anglais) contiennent un des deux objets, les deux objets ou aucun des objets.
- **Conseil** : gardez la vitesse de capture par défaut à 30 FPS mais réduisez la résolution de l'image à 640 x 480 pixels.
- **Création de la base de données** :
  - Vous devez ensuite extraire les images de la vidéo (attention au FPS: vous devrez probablement le réduire pour ne pas avoir plusieurs fois la même image).
  - Vous devrez ensuite étiqueter les images avec Label Studio à l'aide de boîtes englobantes (Bounding Boxes en anglais).
  - Attention au format d'export de votre base de données (il en faut un spécifique pour YOLO).
- **Documentation** : en plus de la documentation précédente sur Label Studio, une partie de ce processus est décrit dans le chapitre 6 du cours https://melodiedaniel.github.io/deep_learning/.

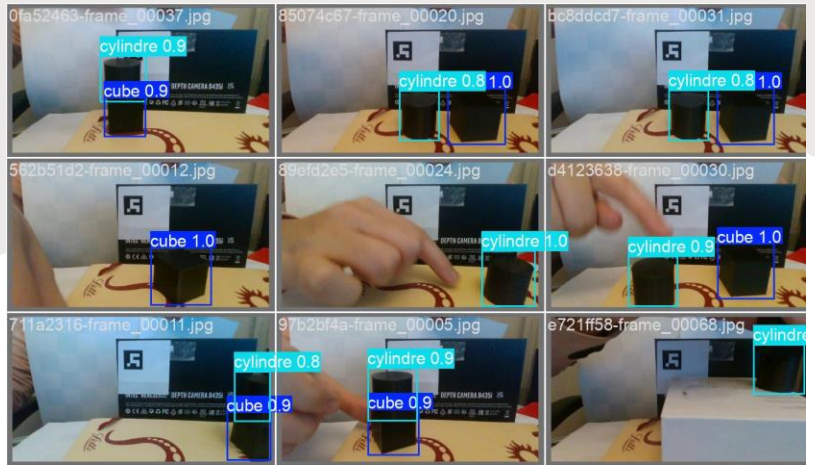
Étape 2 : Création de la base de données

- **Prérequis** : Le Dataset exporté depuis Label Studio au bon format.

- **Création de la base de données au niveau software :**
  - Décomposer le Dataset en 3 Datasets : un Dataset pour l'entraînement, un pour l'évaluation et un pour le test.
  - Dataset entraînement (Train en anglais) : sert à entraîner le modèle/CNN.
  - Dataset évaluation/Eval : sert à évaluer le CNN pendant l'entraînement pour éviter l'Overfitting (=> surapprentissage des données du train et incapacité à généraliser à d'autres données).
  - Dataset Test : sert à évaluer le CNN après l'entraînement pour tester la généralisabilité.
  - Pour cela il existe des outils de PyTorch comme par exemple la classe ***DataLoader*** ou la méthode ***random_split()***.

- **Documentation PyTorch** :
  - https://docs.pytorch.org/tutorials/beginner/basics/data_tutorial.html
  - https://docs.pytorch.org/docs/stable/data.html
  - https://melodiedaniel.github.io/deep_learning/

Étape 3 : Détection avec YOLO



- **Prérequis** : les 3 Datasets : Train, Eval et Test.
- **Création du modèle avec YOLOv11** : YOLOv11 : vous devez ensuite choisir quel modèle utiliser. La différence entre eux est l'architecture du CNN c'est-à-dire le nombre de paramètres (les poids et biais ou Weights and Biases en anglais) et le FPS maximal de traitement de la vidéo.
- **Liste de questions à laquelle il vous faudra répondre pour savoir quel modèle garder** :
  - Quel est le nombre minimal de paramètres dont on a besoin pour détecter les objets ?
  - Pendant combien d'épisodes/époques (Episode/Epochs en anglais) ?
  - Comment détermine-t-on que le modèle a bien appris ?
  - Quel impact du nombre de paramètres sur la performance et sur le temps d'entraînement ?
  - Y a-t-il des limites de détection pour certains objets (taille, couleur) selon le modèle choisi ?
  - Faut-il faire de l'augmentation de données ?
- **Conseil** : Faites plusieurs entraînements et comparer les résultats avant de faire votre choix. Attention vous devrez présenter une analyse pour motiver votre choix durant la présentation de votre travail.
- **Documentation** : En plus de la documentation YOLOv11 citée précédemment, le chapitre 6 du cours (https://melodiedaniel.github.io/deep_learning/) décrit succinctement comment entraîner et quelles sont les architectures disponibles avec YOLOv11.

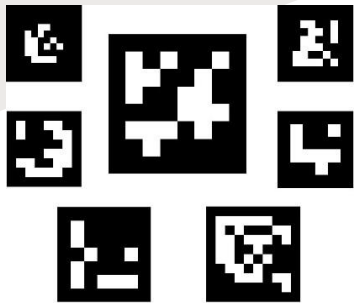
Étape 4 : Détection avec un CNN custom

- **Prérequis** : les 3 Datasets : Train, Eval et Test.
- **Création du CNN custom :**
  - L'idée est de créer un CNN custom pour détecter les mêmes objets qu'avec YOLO et ensuite de comparer les résultats (bien-sûr les résultats avec YOLO serviront de référence).
  - Attention dans le cas du CNN custom il faudra gérer les éléments suivants :
    - L'existence ou non des objets, c'est-à-dire, parvenir à dire s'il n'y a aucun objet, s'il y en a un et dans ce cas identifier si c'est un cylindre ou un cube ou dire s'il y a les deux objets.
    - Faire la détection, c'est-à-dire, générer les coordonnées des boîtes englobantes le cas échéant.
    - Faire deux fonctions de perte (ou Loss Fonction en anglais) différentes chacune traitant d'un des deux cas précédents.
    - Faire de l'Early Stopping pour éviter l'Overfitting.
- **Documentation PyTorch** :
  - https://www.youtube.com/watch?v=l0w-neGG2R8
  - https://pyimagesearch.com/2021/11/01/training-an-object-detector-from-scratch-in-pytorch/
  - https://melodiedaniel.github.io/deep_learning/

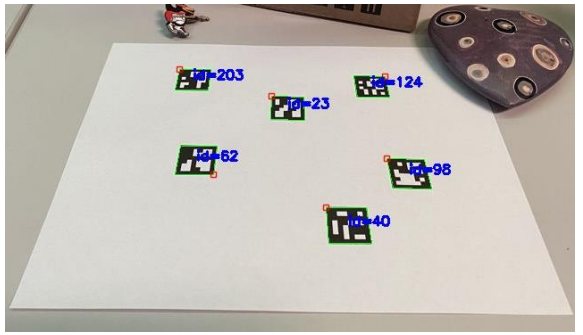
Étape 5 : Comparaison YOLO vs CNN custom

- **Prérequis** : les paramètres des deux modèles YOLO et CNN custom après entraînement.
- **Comparaison des deux architectures choisies pour YOLOv11 et pour le CNN custom** :
  - Le nombre de paramètres.
  - Les fonctions d'activations et de perte choisies.
  - La rapidité d'exécution.
  - Le temps d'entraînement.
  - Les courbes d'apprentissage, etc.
- **Comparaison des résultats en fonction des métriques :**
  - Le taux de succès (ou Success Rate en anglais) => calculer le taux de vrais-positifs, vrais-négatifs, faux-positifs et faux-négatifs à l'aide d'une matrice de confusion (ou Confusion Matrix en anglais).
  - La précision de la détection => calculer l'Intersection over Union (IoU), etc.
- **Documentation** :
  - https://docs.pytorch.org/ignite/generated/ignite.metrics.confusion_matrix.ConfusionMatrix.html
  - https://wandb.ai/site/
  - https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
  - https://github.com/optuna/optuna?tab=readme-ov-file

Étape 6 : Localisation en 3D des objets

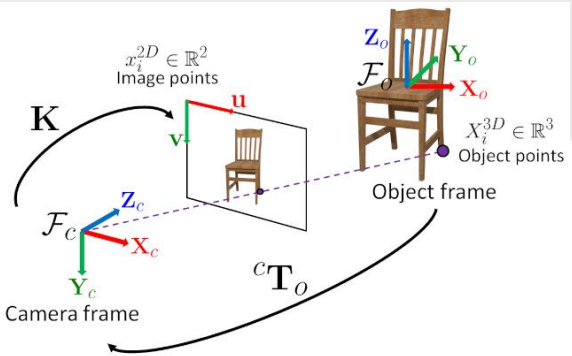


- **Prérequis** : les modèles YOLO et CNN custom entraînés et prêts à détecter des objets.
- **L'objectif** : est de définir un repère monde en 3D par rapport auquel nous souhaitons connaître la position en 3D des objets détectés avec YOLO ou le CNN custom.
- **Principe** :
  - Il faut placer un élément de référence dans l'espace et déterminer sa pose (position et orientation) par rapport au repère caméra.
  - Cette opération s'appelle la calibration extrinsèque d'une caméra.
- **Méthodes** :
  - Calculs géométriques manuels avec OpenCV
  - Détection automatique de marqueurs (méthode recommandée) :
    - Les marqueurs ArUco sont des motifs carrés détectables automatiquement par OpenCV.
    - Ils permettent une calibration extrinsèque rapide et précise.



- **Documentation OpenCV** : https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

Étape 6 : Localisation en 3D des objets



Calibration extrinsèque

- **Objectif** : Décrire la pose d'un objet (ou de la caméra) dans l'espace 3D avec la fonction solvePnP() d'OpenCV.
- **Deux éléments clés pour le changement de repère** :
  - **R** : matrice de rotation 3×3 qui décrit l'orientation de l'objet par rapport au repère caméra.
  - **t** : vecteur de translation 3×1 qui décrit la position de l'objet par rapport au repère caméra.
- **Méthode courante** :
  - Observer un motif connu dans l'espace (un damier, un marqueur, un QR code, etc.).
  - Résoudre le problème PnP pour obtenir (**R**, **t**). Le problème PnP consiste à déterminer la pose d'un objet à partir de points 3D connus dans le repère de l'objet (ex: coins d'un marqueur connaissant la taille du marqueur) et leurs correspondances 2D dans l'image (pixels détectés).
- **Formule clé : $\mathbf{X}_{cam} = \mathbf{R}\,\mathbf{X}_{objet} + \mathbf{t}$** avec
  - $\mathbf{X}_{cam}$: vecteur 3×1 qui décrit la position du point dans le repère de la caméra et
  - $\mathbf{X}_{objet}$: vecteur 3×1 qui décrit la position du point dans le repère de l'objet/marqueur.
- **Documentation** : https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html.

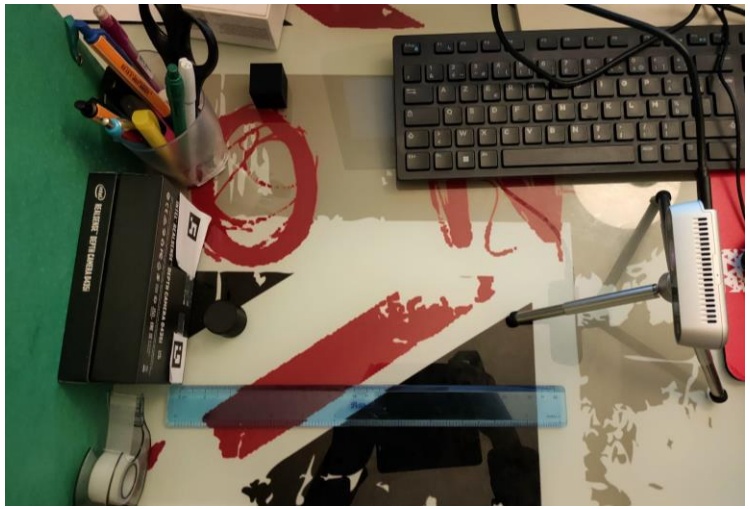
Étape 6 : Localisation en 3D des objets

- **Problème** : Les modèles YOLO/CNN custom détectent des objets en 2D (Bounding Boxes en pixels). Comment obtenir leur position 3D dans le repère caméra avant de faire la conversion vers le repère monde ?
- **Solution** : Utiliser la caméra Depth de la RealSense.
- **La RealSense D435i (ou D435if) possède deux caméras** :
  - Caméra RGB (couleur) : capture l'image classique → détection YOLO/CNN custom.
  - Caméra Depth (profondeur) : mesure la distance de chaque pixel → conversion 3D.
  - Pour la RealSense, le SDK calibre automatiquement les deux caméras (RGB et Depth) intrinsèquement et établit la correspondance spatiale entre elles. Un exemple d'utilisation est fourni dans le fichier *test_depth_realsense.py* sur Moodle.
- **Étapes de conversion** :
  - Détection 2D : YOLO/CNN custom retourne une Bounding Box → calculer le centre (u,v) en pixels.
  - Mesure de profondeur : interroger la caméra Depth pour obtenir la distance z (en mètres) au pixel (u,v). Cette distance est mesurée le long de l'axe Z du repère caméra (perpendiculaire au plan image).
  - Déprojection : convertir le pixel 2D + profondeur en coordonnées 3D dans le repère caméra.
  - Où les paramètres intrinsèques (f_x,f_y,c_x,c_y) sont fournis par le SDK.
  - API RealSense Python :
    - depth_frame.get_distance(u, v) → profondeur en mètres.
    - rs2_deproject_pixel_to_point(intrinsics, [u, v], depth) → point 3D (dans le repère caméra).

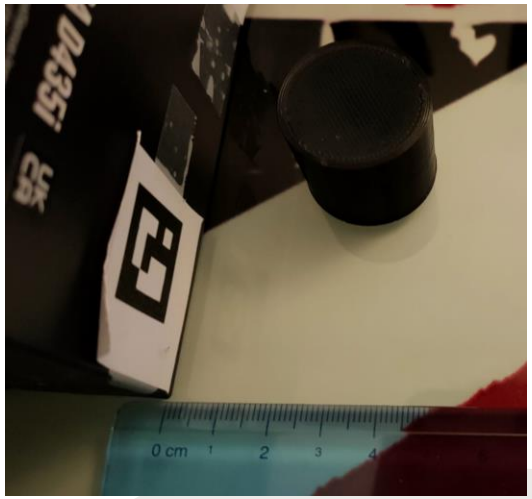
$$X_{\text{cam}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (u - c_x) \cdot z / f_x \\ (v - c_y) \cdot z / f_y \\ z \end{bmatrix}$$

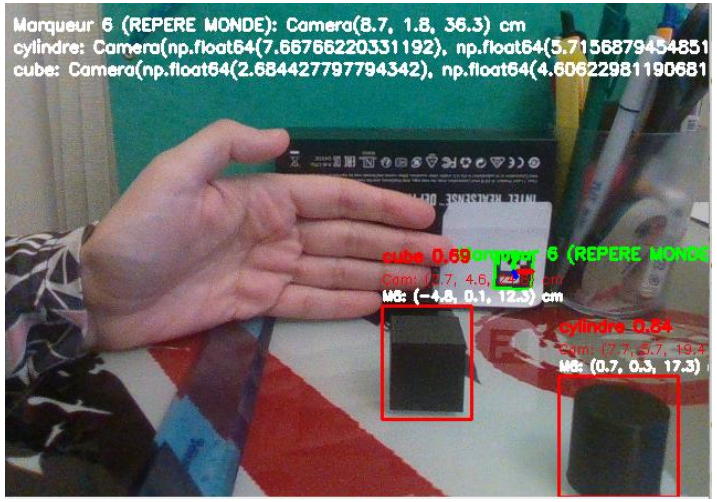
Étape 6 : Localisation en 3D des objets

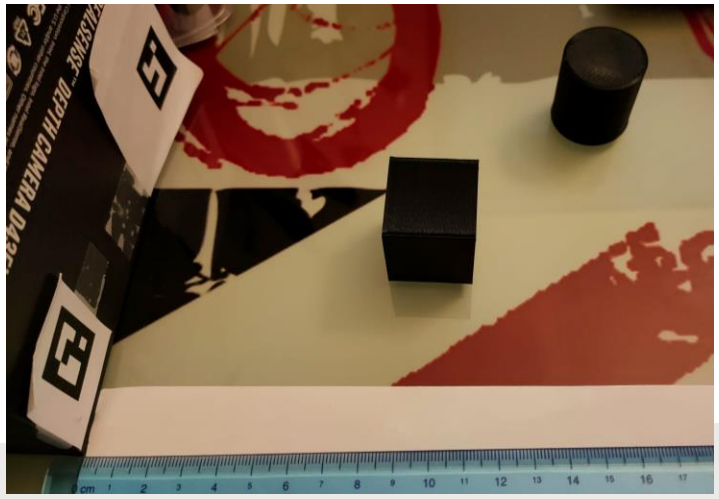
Exemples du résultat attendu



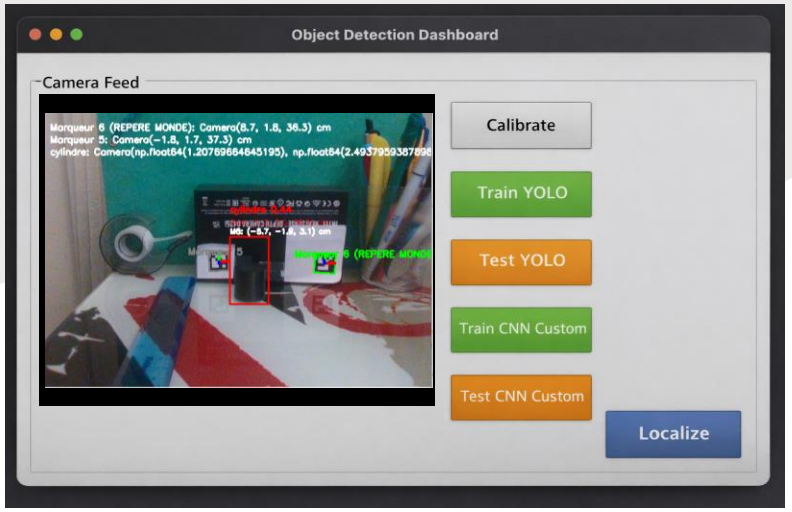
Setup initial



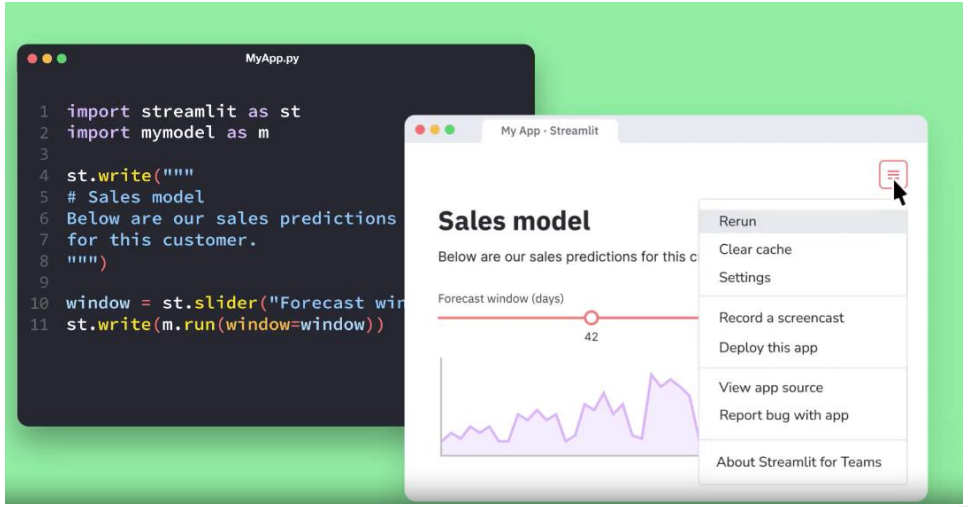


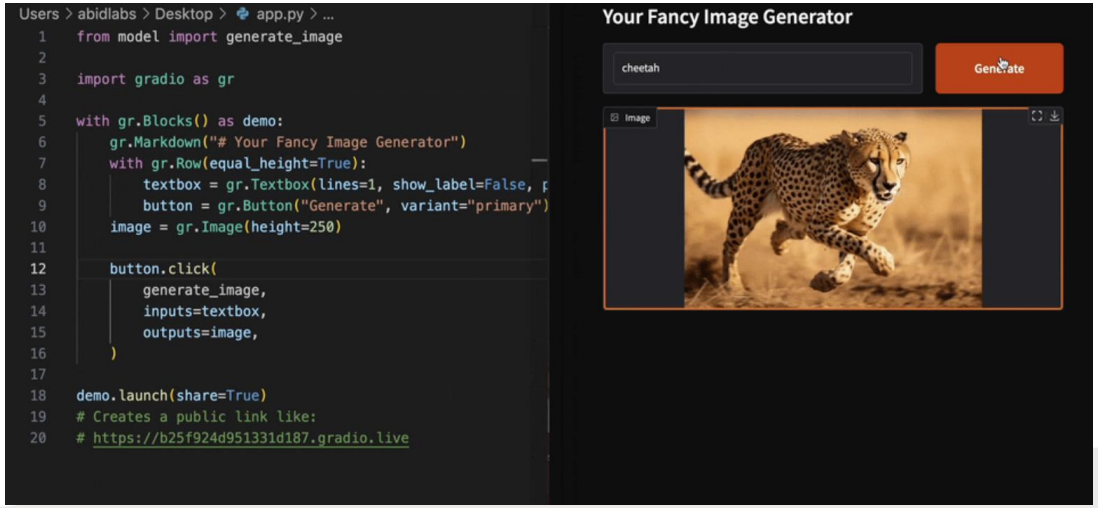


Étape 7 : Création de l'interface graphique



- **Objectif** : créer une interface graphique qui permet de piloter le projet.
- **Les outils utilisés ou le design reste à votre convenance** : (exemple d'outils)
  - Streamlit : https://streamlit.io/
  - Gradio : https://www.gradio.app/guides/quickstart





Étape 8 : Présentation & Démonstration

- **Soutenance de la SAE – 6 février 2026 à partir de 15h00** :
  - Durée : 20 à 25 minutes par groupe
    - 10 minutes de présentation (incluant 1 à 2 minutes de démonstration)
    - 10 minutes de questions par le jury (incluant des questions sur votre code)
  - **Participation** chaque membre du groupe doit parler, même brièvement (1 minute minimum) pour présenter son travail dans le cadre de la SAE. Synchronisez-vous pour ne pas perdre de temps.
  - **Attention** : prévoyez des vidéos de substitution au cas où la démonstration ne pourrait pas se faire.
- **Document d'auto-évaluation à rendre la veille, le 5 février** (1 page maximum, recto) :
  - Noter chaque membre du groupe (vous inclus) sur 4 points selon votre estimation du travail réalisé.
  - Objectif : viser une moyenne de 3/4 pour l'équipe.
  - Préciser les tâches accomplies, les défis rencontrés, etc.
  - Ce document sera pris en compte par le jury dans l'évaluation globale.
- **Évaluation technique et compréhension** (sur 16 points) :
  - 4 pts -> Préparation du matériel, calibration intrinsèque et création de la base de données.
  - 2 pts -> Détection avec YOLO.
  - 4 pts -> Détection avec CNN custom et comparaison avec YOLO.
  - 4 pts -> localisation en 3D.
  - 2 pts -> Interface graphique.

Bon Courage !