

Chapitre 3 : Implémentation et validation de l'approche de test

Introduction

Ce chapitre détaille la mise en œuvre de la méthodologie CRISP-DM pour implémenter et valider un système de test vision automatisé basé sur l'intelligence artificielle, développé pour les compteurs électriques [REDACTED]. Intégration sur le moteur de séquençement TestStand pour l'orchestration des scénarios de test, ainsi qu'une interface opérateur développée en Qt. Le système a été validé en deux phases une phase hors ligne, dédiée à l'entraînement et à l'évaluation des modèles IA, suivie d'une phase en ligne, portant sur leur intégration dans le processus de test sur le prototype.

1. Développement et validation hors ligne de la solution de test

L'objectif principal de ce projet est de remplacer la méthode actuelle de détection et d'inspection des composants de compteur, basée sur LabVIEW, Vision Builder NI et TestStand, par une solution plus flexible, automatisée et intelligente, reposant sur l'intégration de l'intelligence artificielle tel que le modèle de détection d'objets YOLOv8 et automatisé le test par le moteur de test TestStand, (Voir la Figure 36).

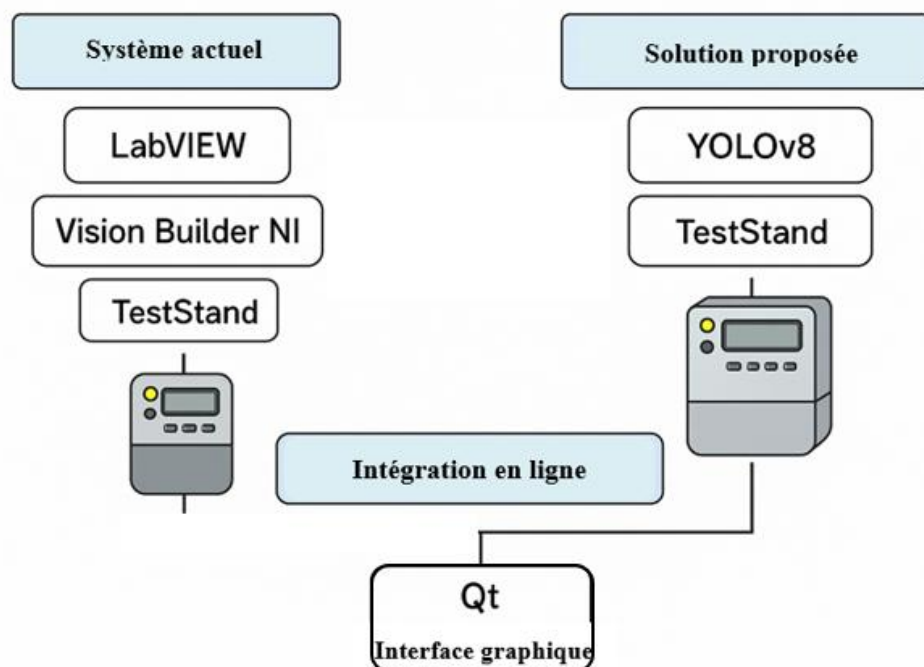


Figure 36 : Objectif de solution

Dans un premier temps, l'approche IA a été développée et validée hors ligne, c'est-à-dire en dehors du prototype industriel, afin de garantir la performance des modèles de détection dans

des conditions variées. Ensuite, une phase d'intégration en ligne a été réalisée, permettant de connecter l'approche IA directement au prototype de test pour automatiser le processus complet d'inspection.

Par ailleurs, afin d'assurer une interaction fluide avec l'opérateur et une visualisation en temps réel des résultats de test, une interface graphique a été spécifiée. Elle a été développée en Qt et assure la supervision complète du système de test visuel.

1.1 Préparation des bases de données

Afin d'entraîner efficacement des modèles de détection d'objets basés sur l'intelligence artificielle, trois bases de données distinctes ont été construites comme il est détaillé dans la figure 37, chacune correspondant à une tâche spécifique du système de vision notamment la détection des LEDs, la détection de l'écran LCD et la détection du bouton poussoir.

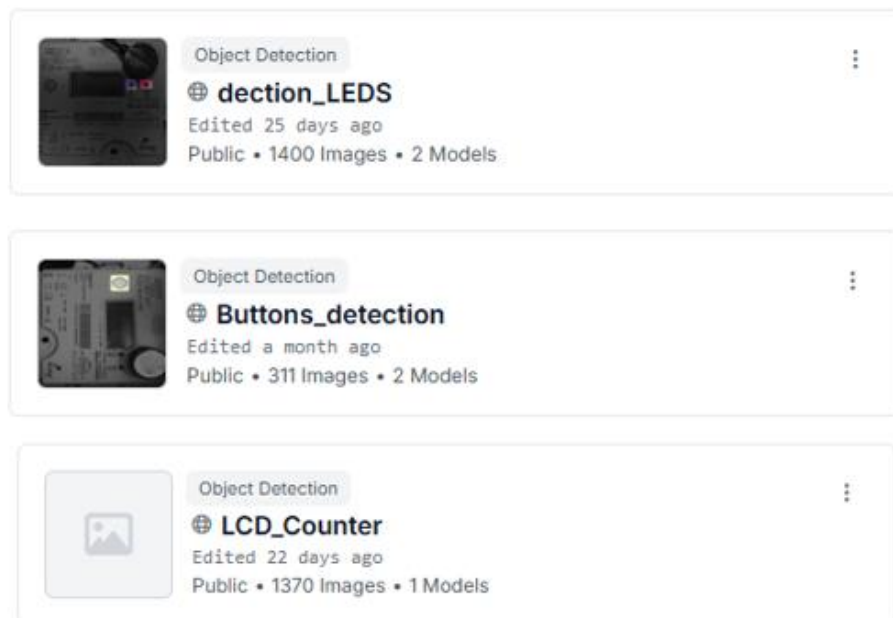


Figure 37 : Bases de données utilisées

L'ensemble des images a été capturé dans un environnement de test, directement à partir du prototype de test du compteur électrique. Comme illustré à la figure 38, les captures ont été réalisées dans des variées conditions d'éclairage, d'angles de vue, de positions du compteur, et avec divers niveaux de reflets, afin de garantir une base de données représentative des scénarios réels et robustes.



Figure 38 : Les images de la base de données

Le choix de Roboflow comme plateforme d'annotation s'est imposé pour plusieurs raisons tel que l'interface conviviale et intuitive pour l'annotation manuelle, l'intégration facile avec les pipelines de traitement de données, les fonctionnalités avancées d'augmentation et de prétraitement, la génération automatique des formats compatibles avec des modèles d'IA tels que YOLOv8. Chaque image a été annotée manuellement à l'aide de boîtes englobantes (bounding boxes) en respectant les classes spécifiques à chaque tâche.

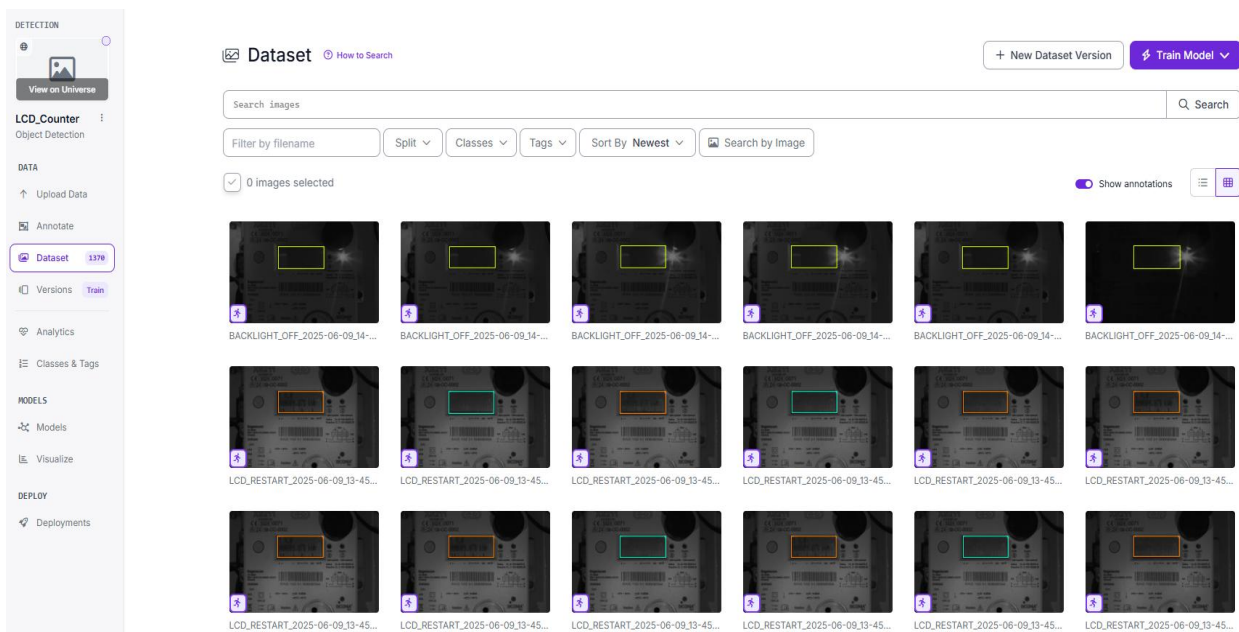


Figure 39 : Roboflow [16]

La figure 40 représente la répartition des jeux de données pour chaque base, les données ont été divisées selon la répartition standard 70 % pour l'entraînement, 20 % pour la validation et 10 % pour le test.

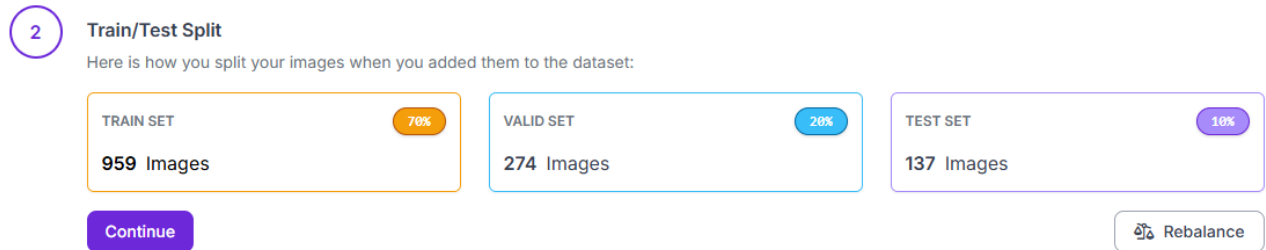


Figure 40 : Exemple de répartition des jeux de données

Pour la base détection des LEDs (Voir Figure 41) contient 4 classes correspondant aux différents états tel que led_p_on (LED active sur la phase P), led_p_off (LED éteinte sur la phase P), led_q_on (LED active sur la phase Q) et led_q_off (LED éteinte sur la phase Q).




COLOR	CLASS NAME	COUNT ↻
	P_off	885
	P_on	511
	Q_off	854
	Q_on	533

Figure 41 : Les classes de la base de données LEDs

La seconde base concerne la détection du bouton poussoir (Voir Figure 42), et comprend deux classes sont bouton_present, indiquant la présence physique du bouton sur l'image, et bouton_absent, correspondant à son absence ou invisibilité.


COLOR	CLASS NAME	COUNT ↻
	buttons	309

Figure 42 : Les classes de la base de données bouton

Enfin, la base dédiée à la détection de l'écran LCD (Voir Figure 43) est la plus variée, avec cinq classes couvrant différents états visuels de l'affichage sont lcd_backlight_on (éclairage

actif), `lcd_backlight_off` (éclairage éteint), `lcd_black` (écran noir), `lcd_clear` (affichage clair ou vide) et `lcd_restart` (affichage en cours de redémarrage).






COLOR	CLASS NAME	COUNT ↻
	LCD_BLACK	261
	LCD_BLACKLIGHT_off	293
	LCD_BLACKLIGHT_on	326
	LCD_CLEAR	268
	ICD_RESTART	214

Figure 43 : Les classes de la base de données LCD

Cette granularité permet une meilleure compréhension du comportement du compteur à travers l'évolution visuelle de son écran.

1.2 Prétraitement et augmentation des données

Avant l'augmentation, les images ont été prétraitées dans Roboflow pour améliorer leur qualité et leur homogénéité. Deux transformations principales ont été appliquées à toutes les images de la base de données, Auto-Orient cette option permet de corriger automatiquement l'orientation des images en se basant sur les métadonnées EXIF (Exchangeable Image File Format), assurant que toutes les images soient correctement orientées lors de l'annotation et de l'apprentissage. Un redimensionnement avec étirement a été appliqué à toutes les images pour les adapter au format d'entrée attendu par le modèle YOLOv8. Cela permet de standardiser la taille des images et d'assurer une compatibilité optimale avec les architectures profondes utilisées. Ces étapes de prétraitement permettent de réduire le temps d'entraînement, de garantir une entrée uniforme pour le modèle, et d'améliorer la stabilité des performances pendant l'apprentissage.

Pour améliorer la robustesse et la généralisation du modèle, des techniques d'augmentation des données ont été appliquées à chaque base dans Roboflow. Les paramètres ont été choisis en fonction des spécificités de chaque tâche.

Les paramètres d'augmentation utilisés pour entraîner le modèle à détecter les LEDs dans des conditions variées, des augmentations légères ont été appliquées. Celles-ci incluent un recadrage entre 10 % et 30 %, une rotation entre -3° et $+3^\circ$, un cisaillement de $\pm 3^\circ$ horizontal et $\pm 2^\circ$ vertical, et une modification de la luminosité entre -15 % et +15 %. De plus, un flou jusqu'à 0.5 px et un bruit affectant jusqu'à 0.02 % des pixels ont été ajoutés pour simuler un léger bruit de capteur.

Pour la détection de bouton, une augmentation modérée et équilibrée a été appliquée pour couvrir les variations réalistes de position et de couleur. Elle inclut une symétrie horizontale, Un cisaillement de $\pm 2^\circ$ dans les deux directions, un changement de teinte entre -1° et $+1^\circ$, et une saturation entre -7% et $+7\%$. La luminosité varie de -4% à $+4\%$, l'exposition entre -2% et $+2\%$, avec un flou jusqu'à 3 px et du bruit injecté jusqu'à 1.72% des pixels.

Pour la détection de LCD, des augmentations fortes ont été appliquées afin de rendre le modèle robuste aux conditions d'affichage difficiles. Cela comprend une rotation de $\pm 10^\circ$, Un cisaillement de $\pm 10^\circ$, une saturation allant de -25% à $+25\%$, une luminosité de -20% à $+20\%$, et une exposition de -10% à $+10\%$. L'image peut être tournée à 90° (horaire ou antihoraire), un recadrage jusqu'à 10% , un flou jusqu'à 3 px, un bruit de 1.05% , et une conversion en niveaux de gris appliquée sur 15% des images. Ces augmentations simulent des conditions réelles variées sans nécessité de capturer davantage d'images, tout en augmentant la taille et la diversité des données disponibles pour l'entraînement.

1.3 Entraînement et export des modèles YOLOv8

Une fois l'annotation et les augmentations terminées, chaque base a été exportée au format YOLOv8 depuis Roboflow. Chaque base de données contient les dossiers train, valid et test, avec leurs images et fichiers d'annotation correspondants. Les bases des données annotés ont été utilisés dans le pipeline d'entraînement de YOLOv8, en spécifiant le chemin d'accès aux données et les noms des classes dans le fichier data.yaml. Ce fichier permet de configurer rapidement l'entraînement sans besoin de réécrire du code.

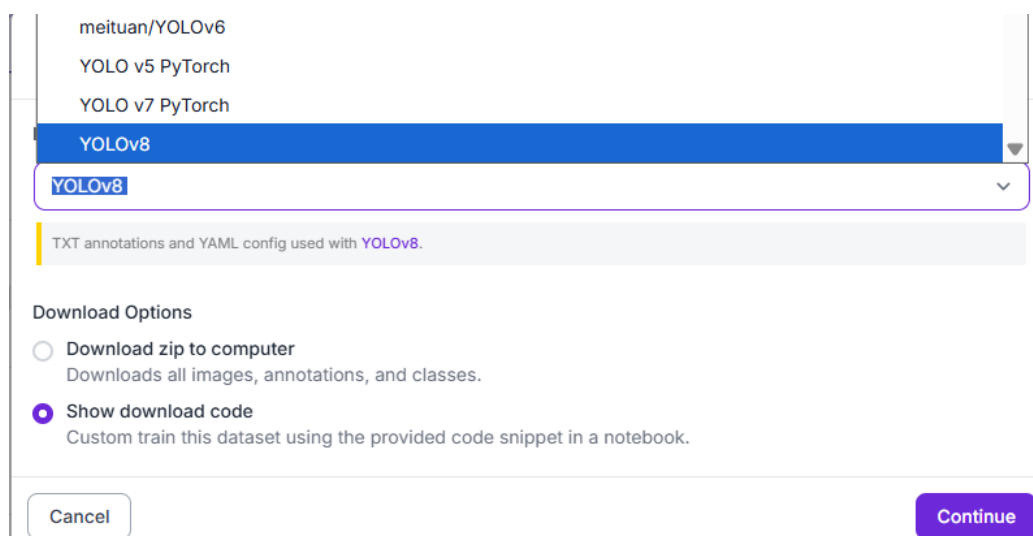


Figure 44 : Exportation de données au format YOLOv8 depuis Roboflow

Dans le cadre du développement de notre système de test visuel automatisé pour compteurs électriques, une étude comparative a été menée entre différentes variantes du modèle YOLOv8 tel que YOLOv8n (nano), YOLOv8s (petit) et YOLOv8m (moyenne). L'objectif était de sélectionner, pour chaque tâche de détection (LEDs, écran LCD, bouton), le modèle offrant le meilleur compromis entre précision, vitesse d'inférence et empreinte mémoire, en vue d'un déploiement dans un système temps réel.

Chaque modèle a été entraîné sur un jeu de données annoté, avec des hyperparamètres spécifiques, adaptés à la nature de l'objet à détecter. Voici la configuration retenue pour chaque cas dans le tableau 16 suivant.

Tableau 16 : Les hyperparamètres spécifiques de chaque base de données

Élément détecté	Optimiseur	Époques	Batch	Raison
LEDs	Adam	100	32	Objets petits et peu contrastés
LCD	AdamW	100	32	Forte variabilité (reflets, contraste)
Bouton	Adam	60	16	Élément simple, forme stable

Le choix de l'optimiseur a eu un impact significatif sur la stabilité de l'apprentissage et la capacité de généralisation des modèles. Par exemple, l'optimiseur Adam (utilisé pour la détection des LEDs et du bouton) est un optimiseur adaptatif qui combine le momentum (une technique qui permet d'accélérer la convergence en tenant compte des gradients passés) et un ajustement dynamique du taux d'apprentissage. Il convient particulièrement aux tâches simples ou peu bruitées. Sa formule est la suivante :

$$\mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \cdot \mathbf{g}_t \quad (10)$$

$$\mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot \mathbf{g}_t^2 \quad (11)$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \cdot \frac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t} + \epsilon} \quad (12)$$

AdamW est une variante améliorée de l'optimiseur Adam, qui dissocie la régularisation L2 (ajoute une pénalité à la fonction de perte pendant l'entraînement en fonction de la norme des poids du modèle pour empêcher les poids de devenir trop grands, ce qui aide à éviter le surapprentissage). Cette séparation permet une meilleure généralisation, en particulier sur des données complexes. L'utilisation d'AdamW contribue à limiter le surapprentissage, tandis que

les augmentations visuelles appliquées aux données renforcent la robustesse du modèle face aux variations d'entrée.

La formule modifiée d'AdamW est la suivante :

$$\theta_t = \theta_{t-1} - \eta \cdot \frac{m_t}{\sqrt{v_t} + \epsilon} + \lambda \cdot \theta_{t-1} \quad (13)$$

1.4 Évaluation et sélection des modèles finaux

Chaque modèle a été analysé à partir de ses courbes d'évolution des métriques. Les figures 45, 46 et 47 suivantes présentent les courbes d'entraînement et de validation du modèle YOLOv8 pour la détection de Bouton :

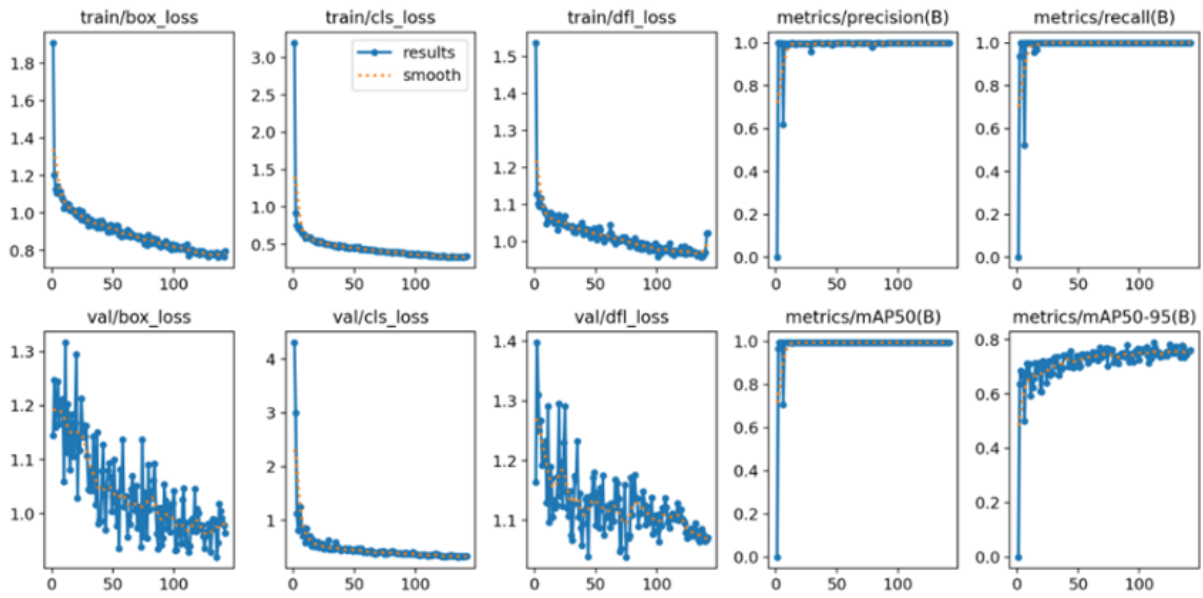


Figure 45 : YOLOv8n pour la détection de Bouton

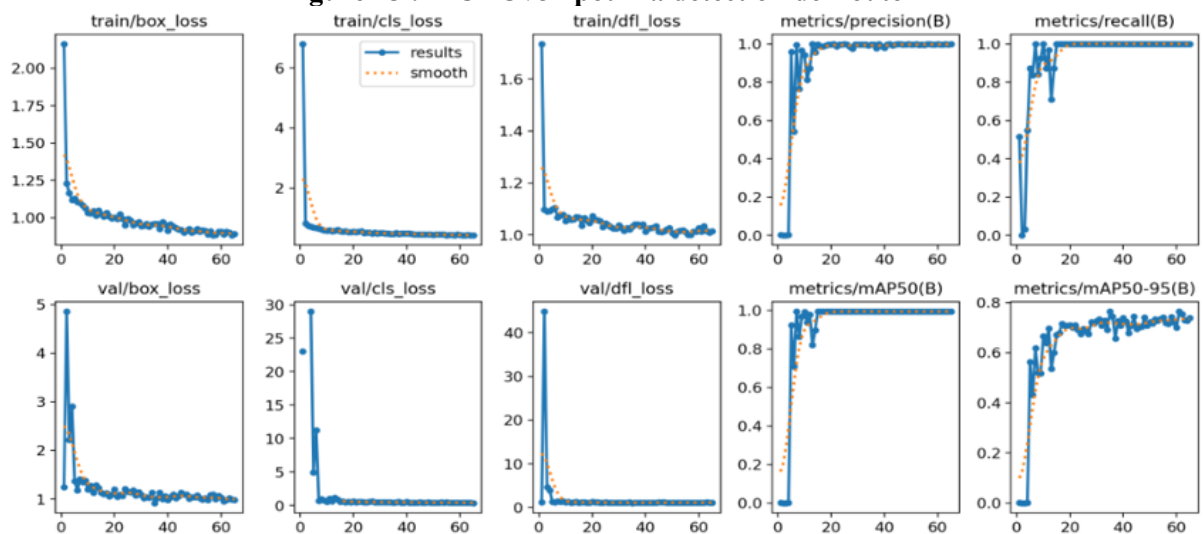


Figure 46 : YOLOv8s pour la détection de Bouton

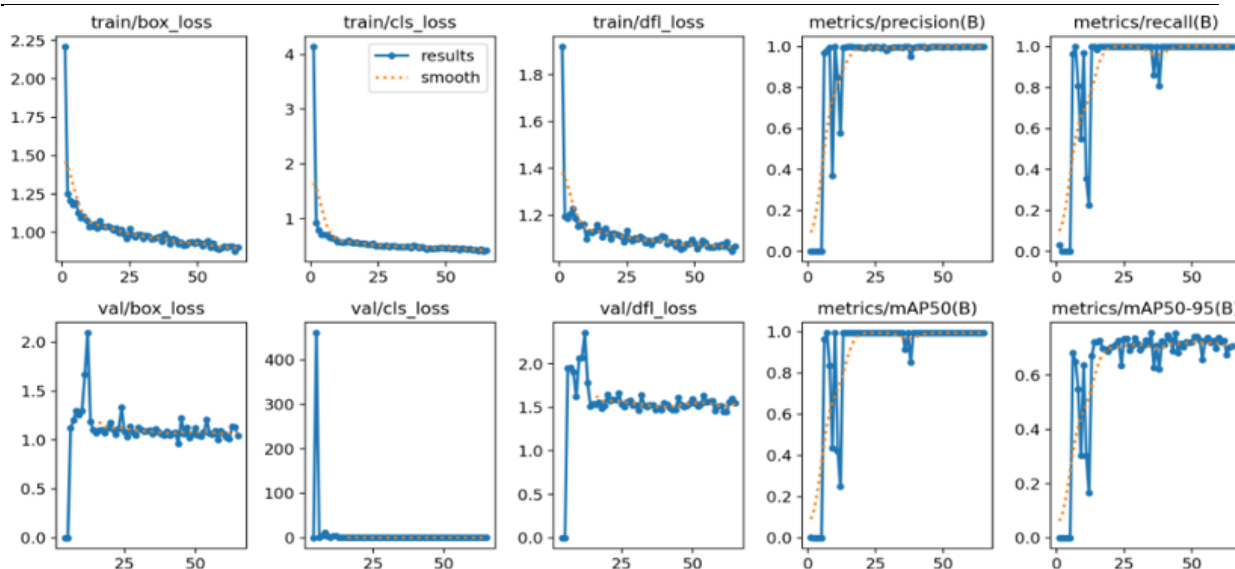


Figure 47 : YOLOv8m pour la détection de Bouton

Après avoir entraîné trois variantes du modèle YOLOv8 (n, s, m) avec les mêmes paramètres sur notre base de données, nous avons observé que le modèle YOLOv8n offre la meilleure performance globale. Il affiche une précision de 0.998, un rappel parfait, et surtout une moyenne de précision (0.5/0.95) de 0.760, supérieur aux autres versions. Ses courbes de pertes sont également plus stables, ce qui confirme sa capacité à bien généraliser sans surapprentissage. Ainsi, YOLOv8n est retenu comme le modèle optimal pour notre tâche de détection.

Les figures 48 et 49 suivantes présentent les courbes d'entraînement et de validation du modèle YOLOv8 pour la détection de l'écran LCD :

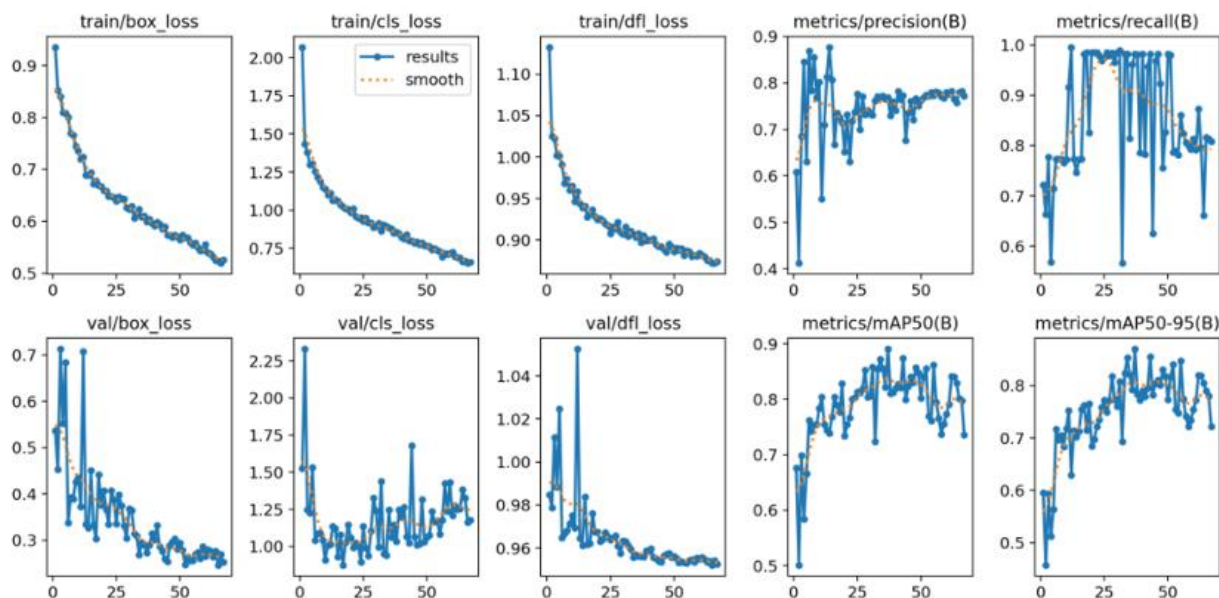


Figure 48 : YOLOv8n pour la détection de l'écran LCD

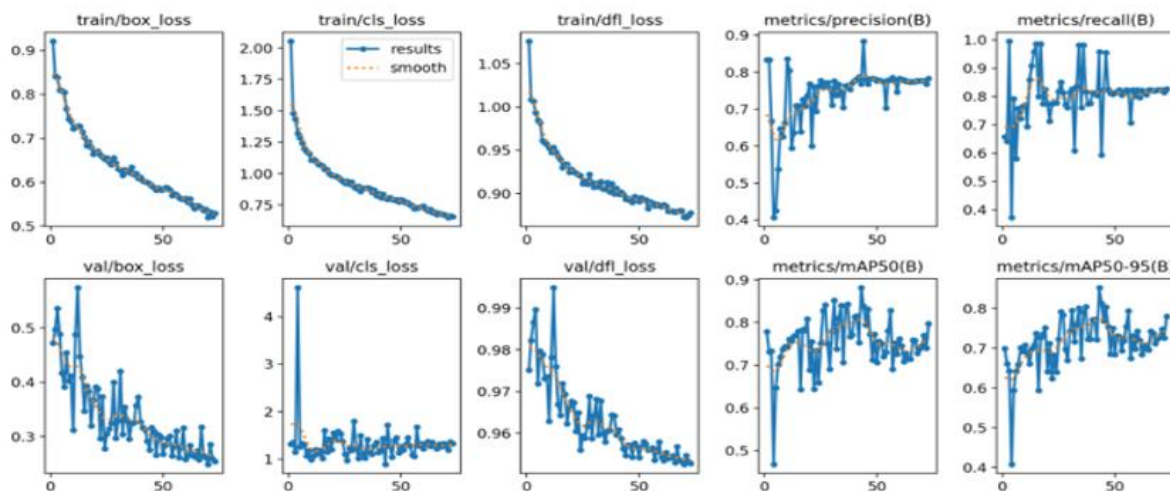


Figure 49 : YOLOv8s pour la détection de l'écran LCD

Une comparaison entre les modèles YOLOv8n et YOLOv8s a été effectuée afin de sélectionner le modèle le plus adapté à notre tâche de détection de LCD. L'analyse des courbes d'apprentissage et des métriques finales montre que YOLOv8s surpasse YOLOv8n avec un rappel de 0.960 et un score de moyenne de précision (0.5:0.95) de 0.854. Ses courbes de validation sont plus stables, traduisant une meilleure capacité de généralisation. YOLOv8n, en revanche, présente des performances moindres et des fluctuations importantes, probablement liées à une capacité insuffisante. En conséquence, le modèle YOLOv8s a été retenu comme la solution optimale pour la base de données de LCD.

Les figures 50 et 51 suivantes présentent les courbes d'entraînement et de validation du modèle YOLOv8 pour la détection des LEDs :

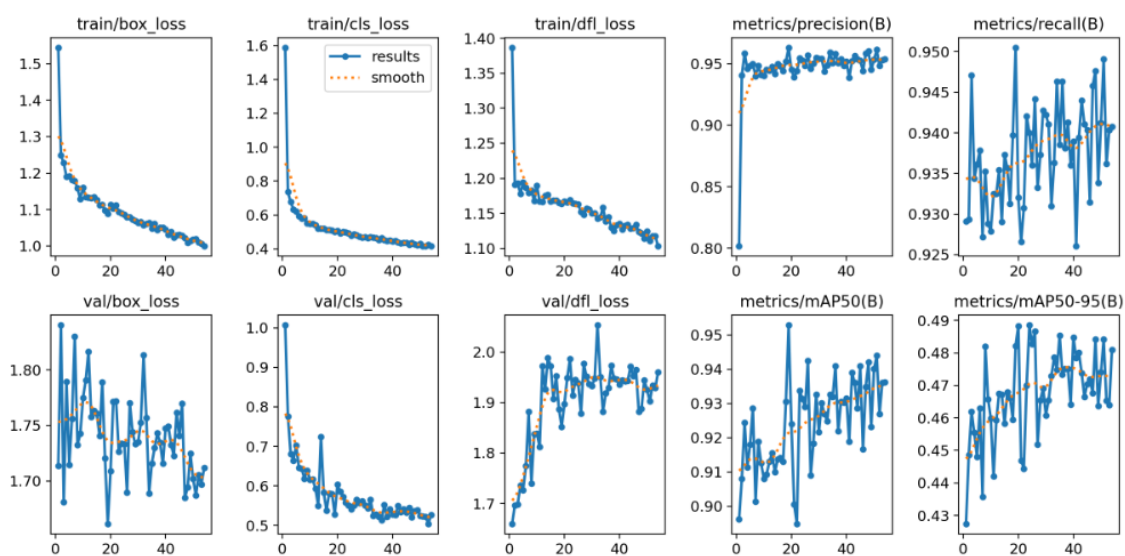


Figure 50 : YOLOv8n pour la détection des LEDs

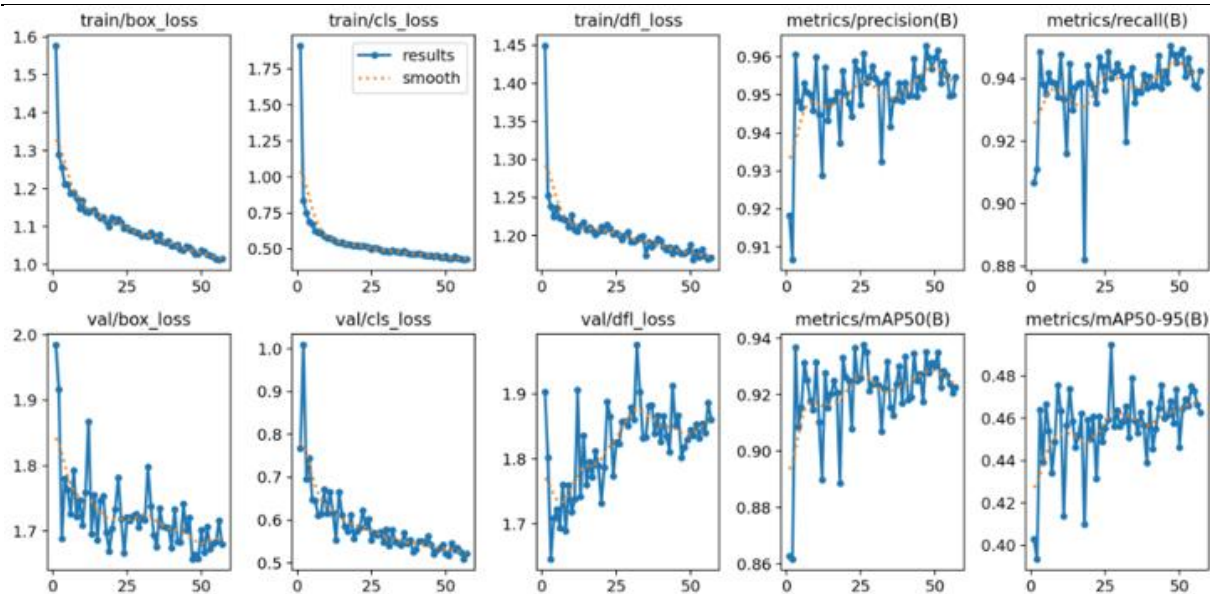


Figure 51 : YOLOv8s pour la détection des LEDs

Après avoir entraîné les variantes YOLOv8n et YOLOv8s sur la base de données LEDs, nous avons constaté que le modèle YOLOv8s présente de meilleures performances globales. Il atteint une précision de 0.955, un rappel de 0.942, une moyenne de précision de 0.923, dépassant ainsi les résultats obtenus avec YOLOv8n. De plus, les courbes de pertes et de métriques pour YOLOv8s sont globalement plus stables et plus régulières, indiquant un apprentissage plus maîtrisé et une meilleure capacité de généralisation. En comparaison, YOLOv8n présente davantage de fluctuations, notamment au niveau des pertes de validation. Par conséquent, YOLOv8s est retenu comme le modèle le plus adapté pour notre tâche de détection.

Le tableau 18 illustre toutes les valeurs des paramètres d'évaluation de la performance des modèles à chaque base de données et d'après lequel nous avons fait notre choix.

Tableau 17 : Analyse des courbes d'apprentissage

Elément	Modèles	Taille Mo	mAP 0.5	mAP 0.5/0.95	Précision	Rappel	Verdict
LEDs	YOLOv8 n	6	0.917	0.455	0.953	0.936	Très léger, bonne précision, mais légèrement moins performant
	YOLOv8 s	11	0.923	0.463	0.955	0.942	Bon compromis entre précision et stabilité, retenu

	YOLOv8m	22	-	-	-	-	Trop lourd
LCD	YOLOv8n	6	0.736	0.721	0.772	0.808	Léger mais moins précis, parfois instable.
	YOLOv8s	11	0.882	0.854	0.768	0.960	Très bonne détection, excellent rappel
	YOLOv8m	22	-	-	-	-	Trop lourd
Bouton	YOLOv8n	6	0.995	0.760	0.998	1.00	Très stable Bon compromis vitesse/précision
	YOLOv8s	11	0.995	0.738	0.998	1.00	Surdimensionné Moins stable
	YOLOv8m	22	0.995	0.706	0.998	1.00	Surapprentissage

La taille du modèle influence fortement la capacité à l'intégrer dans un système temps réel. YOLOv8m peut être il fait une meilleure performance avec les bases de données complexe tel que petits objets comme base de données de LEDs ou une base de données variantes comme les états de LDC dans la base de données LCD, mais son poids mémoire 22 Mo est un handicap pour un déploiement embarqué ou système industriel. Donc les modèles retenus pour le système final sont YOLOv8s pour la détection des LEDs et de l'écran LCD, et YOLOv8n pour la détection du bouton. Le modèle YOLOv8s offre un excellent compromis entre précision, vitesse et consommation mémoire, tandis que YOLOv8n, plus léger, est parfaitement adapté à une tâche simple comme la détection du bouton.

Ces choix garantissent une détection fiable, une intégration fluide dans l'environnement embarqué, et une exécution rapide dans le flux du banc de test automatisé.

L'entraînement du modèle d'intelligence artificielle a été conduite à l'aide de Google Colab, en exploitant un environnement GPU pour accélérer l'apprentissage.

L'environnement d'exécution sur Google Colab a été configuré avec l'accélération matérielle via GPU, permettant de tirer parti des bibliothèques comme Python 3.10, Ultralytics YOLOv8 pour la détection d'objets, Roboflow pour le téléchargement et le prétraitement des jeux de données, ainsi que PyTorch avec support CUDA pour l'entraînement optimisé sur GPU.

Grâce à cette configuration, les phases d'apprentissage ont été significativement accélérées, atteignant une vitesse moyenne de traitement de 25 à 35 millisecondes par image, en fonction du modèle utilisé et des hyperparamètres choisis.

```

Fri Jun 13 15:05:04 2025
+-----+
| NVIDIA-SMI 550.54.15                  Driver Version: 550.54.15          CUDA Version: 12.4         |
+-----+-----+-----+
| GPU   Name                               Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan   Temp   Perf              Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|====+=====+====+=====+=====+=====+=====+
|    0  Tesla T4                       Off          | 00000000:00:04:0 Off  |            0          |
| N/A   43C    P8              9W / 70W        |  0MiB / 15360MiB |      0%      Default  |
|                                           |                      | N/A           |
+-----+-----+-----+
+-----+
| Processes:                                |
| GPU   GI    CI        PID   Type   Process name                      GPU Memory |
|      ID    ID                                   name                 Usage     |
|=====+=====+=====+=====+=====+=====+
|  No running processes found              |
+-----+

```

Figure 52 : La commande nvidia-smi exécutée dans cet environnement

La Figure 52 illustre le retour de la commande `nvidia-smi` exécutée dans cet environnement. Le système dispose d'un GPU NVIDIA Tesla T4 (architecture Turing), avec les spécifications suivantes : driver NVIDIA 550.54.15, CUDA version 12.4, et une mémoire graphique totale de 15 Go (15360 MiB). Ces caractéristiques confirment que le GPU est pleinement opérationnel pour l'exécution de tâches intensives en intelligence artificielle, notamment l'entraînement de modèles profonds.

Les jeux de données annotés ont été téléchargés automatiquement via l'API Roboflow. Chaque base de données contient les répertoires train, valid, et test, et un fichier 'data.yaml' décrivant les classes. La figure 53 représente un exemple de téléchargement d'une base de données annotées par Roboflow.

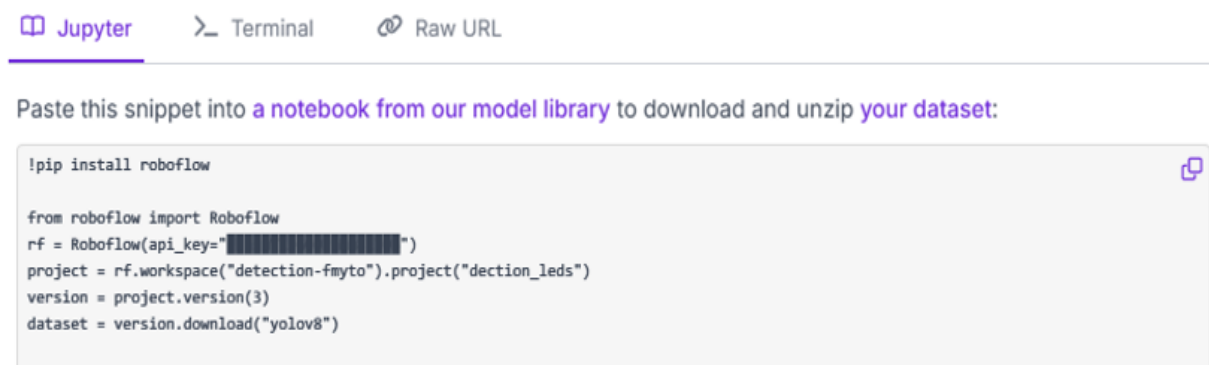


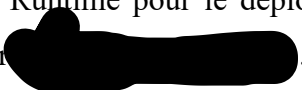
Figure 53 : Les jeux de données annotés des LEDs

Comme illustré dans le Tableau 19, chaque tâche de détection a été associée à une variante de YOLOv8, entraînée avec des hyperparamètres ajustés selon les contraintes spécifiques (complexité visuelle, fréquence d'apparition, etc.).

Tableau 18 : Les hyperparamètres

Base de données	Modèle	Epoques	Taille image	Batch	Patience	Optimiseur
LEDs	YOLOv8s	100	640	32	30	Adam
Bouton	YOLOv8n	60	640	16	30	Adam
LCD	YOLOv8s	100	640	32	30	AdamW

Une fois les modèles YOLOv8 entraînés et validés, ils ont été exportés aux formats PyTorch (format natif .pt) et ONNX (Open Neural Network Exchange) afin de permettre leur intégration dans l'environnement Python du système de vision développé. Le format PyTorch a été utilisé durant les phases de test et de validation locale sur PC. En parallèle, l'exportation au format ONNX a été privilégiée pour le déploiement industriel, en raison de sa compatibilité avec des frameworks performants tels que ONNX Runtime, OpenVINO (Intel) ou TensorRT (NVIDIA). Cette approche permet une interopérabilité maximale et garantit une exécution optimisée du modèle sur des plateformes embarquées ou des systèmes de test accélérés par GPU.

Dans le cadre de notre projet, nous avons donc retenu deux scénarios complémentaires : l'utilisation de PyTorch pour la phase de développement et de test sur machine locale, et l'intégration via ONNX Runtime pour le déploiement sur un poste industriel dédié au test automatique du compteur .

2. Validation en ligne sur le prototype

Après la validation hors ligne de l'approche basée sur l'intelligence artificielle, une phase d'intégration sur le prototype physique a été entreprise. Cette étape visait à configurer et valider l'ensemble de l'environnement matériel et logiciel en conditions réelles de test sur le prototype. La procédure d'intégration a commencé par l'ouverture de la caméra industrielle Imaging Source DFK 23G031 à l'aide du logiciel IC Capture 4.0, comme illustré dans la figure ci-dessous. L'adresse IP statique de la caméra a été configurée (192.168.0.10) via l'interface de sélection des périphériques GigE, afin d'assurer une communication stable avec le poste de test.

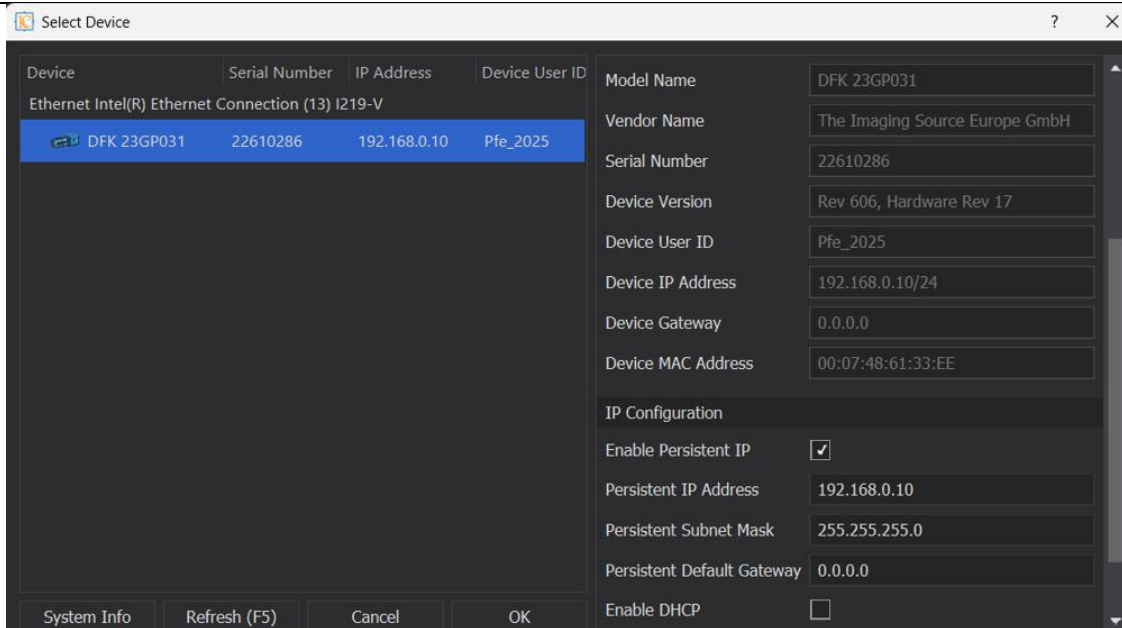


Figure 54 : Configuration de la caméra DFK 23G031 via IC Capture

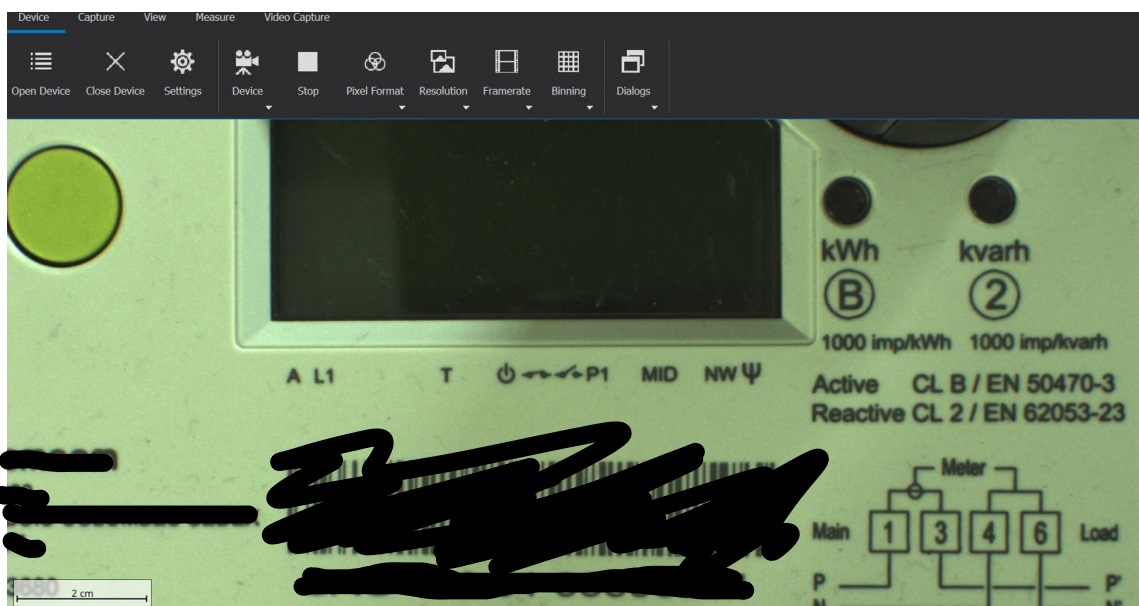


Figure 55 : Visualisation en direct de la caméra DFK 23G031 via IC Capture

Ensuite, la connexion avec le module d'entrées/sorties numériques ioLogik E1214 de Moxa a été établie. Ce module permet de piloter les différents actionneurs du prototype, tels que l'allumage, l'alimentation de prototype ou les entrées sorties de test. Et d'alimentation de compteur via l'activation et d'activation de contacteur lié directement ou Moxa. À travers l'outil ioSearch, le module a été identifié sur le réseau, puis les ports ont été testés en conditions réelles pour valider leur activation correcte (voir Figure 56).

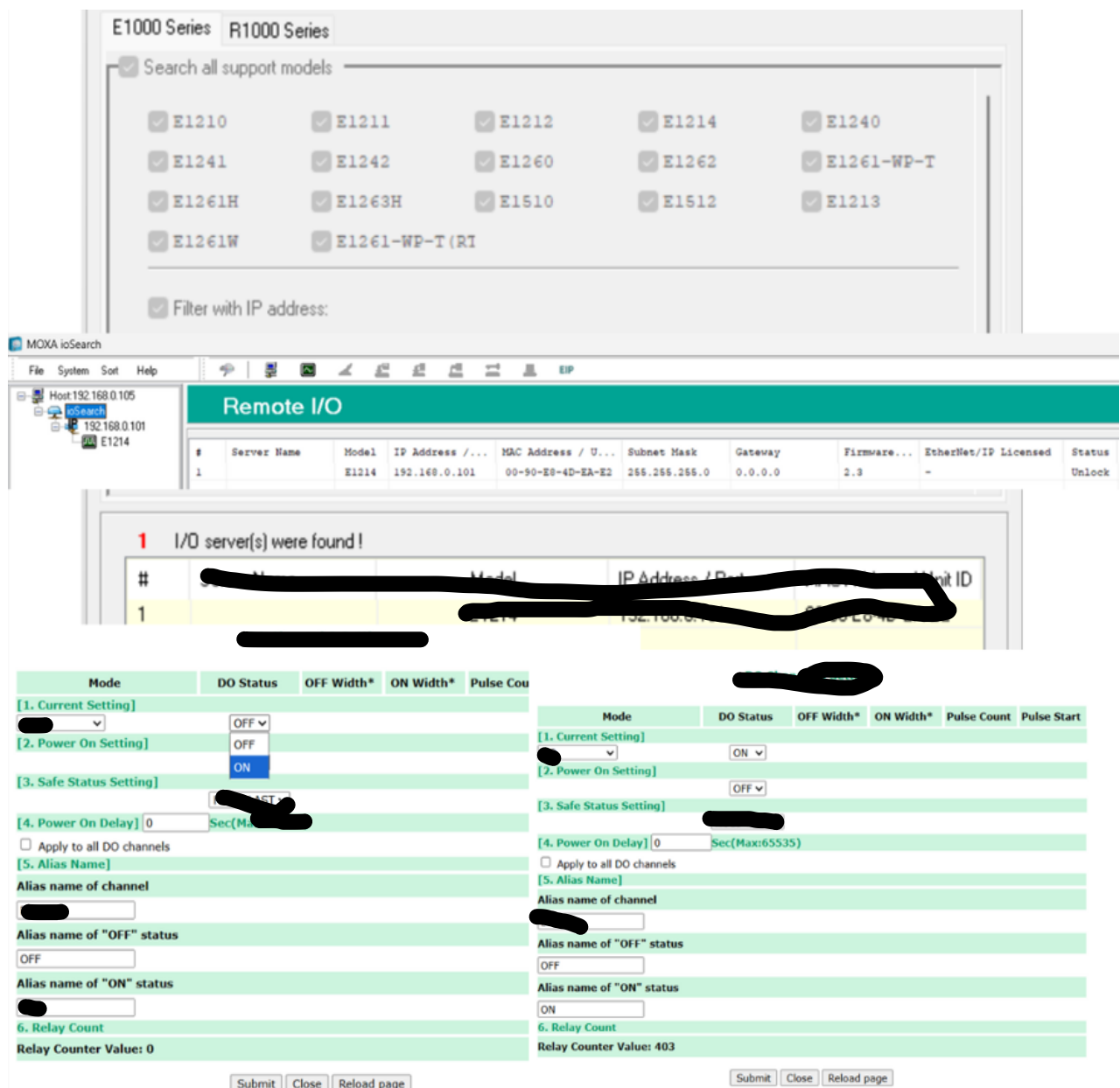


Figure 56 : Configuration et test du module ioLogik E1214

Puis un envoi des commandes de test (de type CEI 62056-21) ont été envoyées pour s'assurer que chaque élément (LEDs, bouton, écran LCD) réagit correctement.

À travers l'outil Docklight, avec une certaine configuration de port et de baud rate pour synchroniser la communication avec le compteur. (Voir Figure 56).

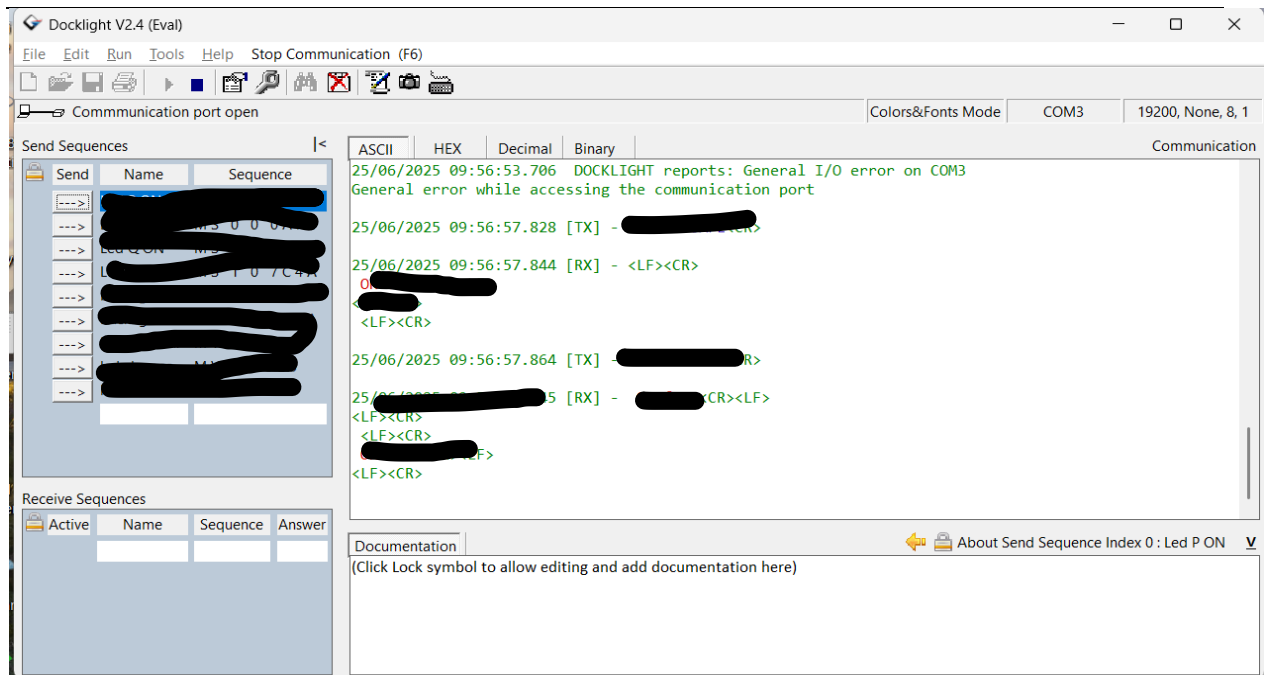


Figure 57 : Envoi de commandes CEI 62056-21 via Docklight

Une fois cette phase de vérification manuelle effectuée, les actions ont été automatisées à l'aide de scripts Python, appelés directement dans le moteur de séquencement TestStand.

Cela a permis d'intégrer l'intelligence artificielle dans le système de test de façon fluide, sans recourir à LabVIEW ni Vision Builder, contrairement au système actuellement utilisé sur la chaîne de production. Cette intégration marque ainsi une transition vers une solution plus moderne, modulaire et évolutive, dans laquelle TestStand orchestre les tests, Python exécute les inférences IA, et Qt assure l'interface opérateur en temps réel.

2.1 Validation de l'approche via le moteur de test TestStand

Pour assurer une orchestration complète et modulaire du processus de test visuel automatisé des compteurs électriques, le moteur de séquencement TestStand de National Instruments a été utilisé comme socle central du système de test.

Ce dernier permet de gérer la logique de test, d'exécuter des scripts Python intégrant les modèles YOLOv8, de prendre des décisions conditionnelles, et de générer des rapports standardisés. L'objectif principal de l'intégration était de superviser l'exécution des scripts de fonctionnement de tout process telle que l'activation des modules et caméra et intégrant les modèles IA à partir d'un environnement standardisé. Gérer les étapes de test sous forme de séquences. Automatiser les décisions en fonction des résultats d'Action.

Le Python Adapter intégré dans TestStand a été activé et configuré pour pointer vers l'environnement Python utilisé dans le projet (Python 3.9). Ce composant permet d'exécuter directement des fonctions Python définies dans des scripts externes, (Voir Figure 58).

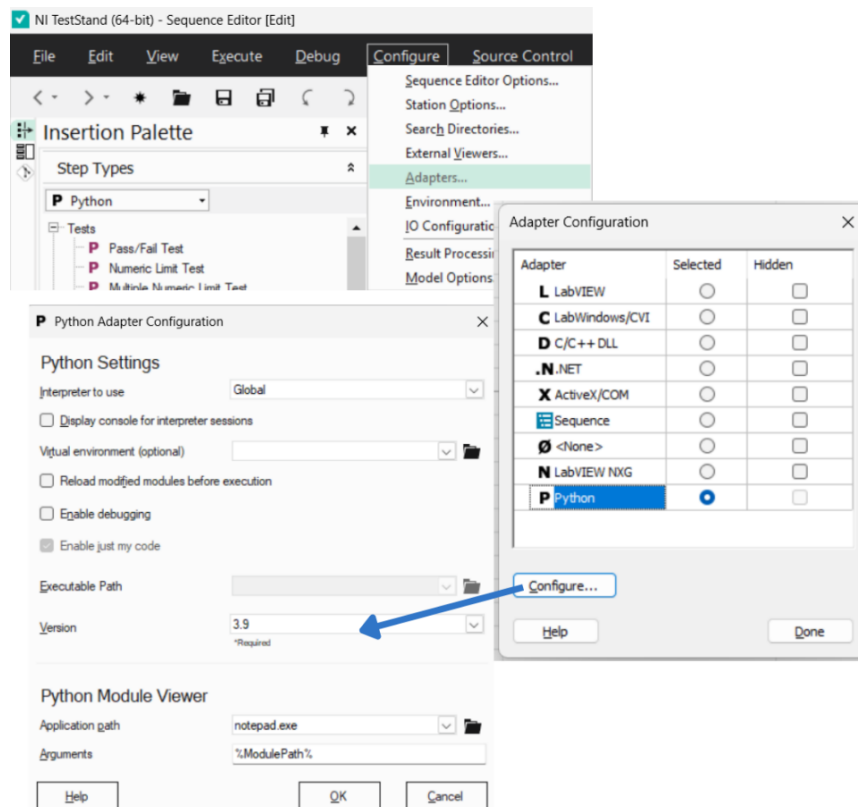


Figure 58 : Configuration du Python Adapter TestStand

Chaque tâche de détection (LED, bouton, LCD) est encapsulée dans une fonction Python dédiée, qui est appelée depuis TestStand via une étape de type "Python Action", (Voir Figure 59).

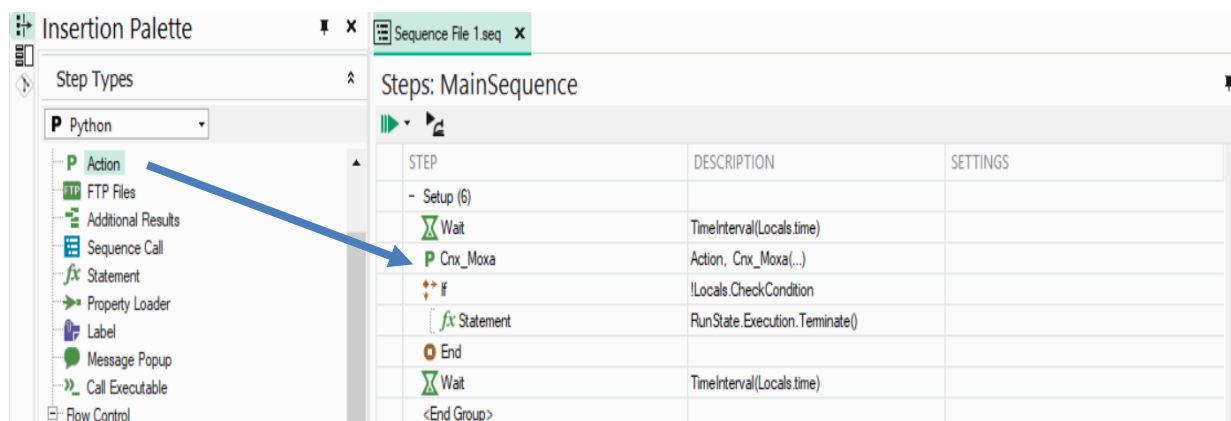


Figure 59 : Python Action TestStand

La figure 60 illustre un exemple de fonction Python utilisée. Cette fonction prend en entrée le chemin de l'image capturée, effectue une inférence avec le modèle YOLOv8 entraîné, puis retourne la liste des classes détectées.

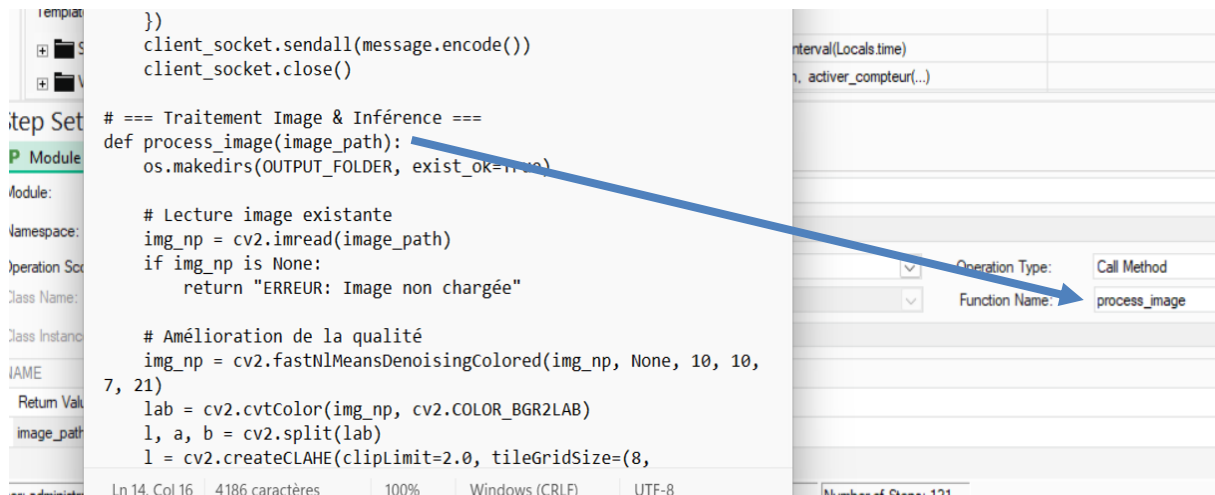


Figure 60 : Exemple de fonction Python utilisée

Une séquence typique développée dans cet environnement suit une structure modulaire, avec des étapes conditionnelles et des points de contrôle critiques permettant une automatisation fiable.

La séquence de test automatisée est structurée en plusieurs étapes successives garantissant le bon déroulement des vérifications fonctionnelles du compteur électrique.

Tout d'abord, la connexion au module d'entrées/sorties numériques ioLogik E1214 de Moxa est établie via une communication Ethernet, permettant de piloter les équipements physiques tels que les relais et capteurs. Une fois cette communication initialisée, le système vérifie la présence effective du compteur sur le banc de vision à l'aide d'un capteur de proximité connecté en entrée digitale. Si la présence est confirmée, l'alimentation du compteur est activée automatiquement par commande d'un relais de sortie via le module ioLogik.

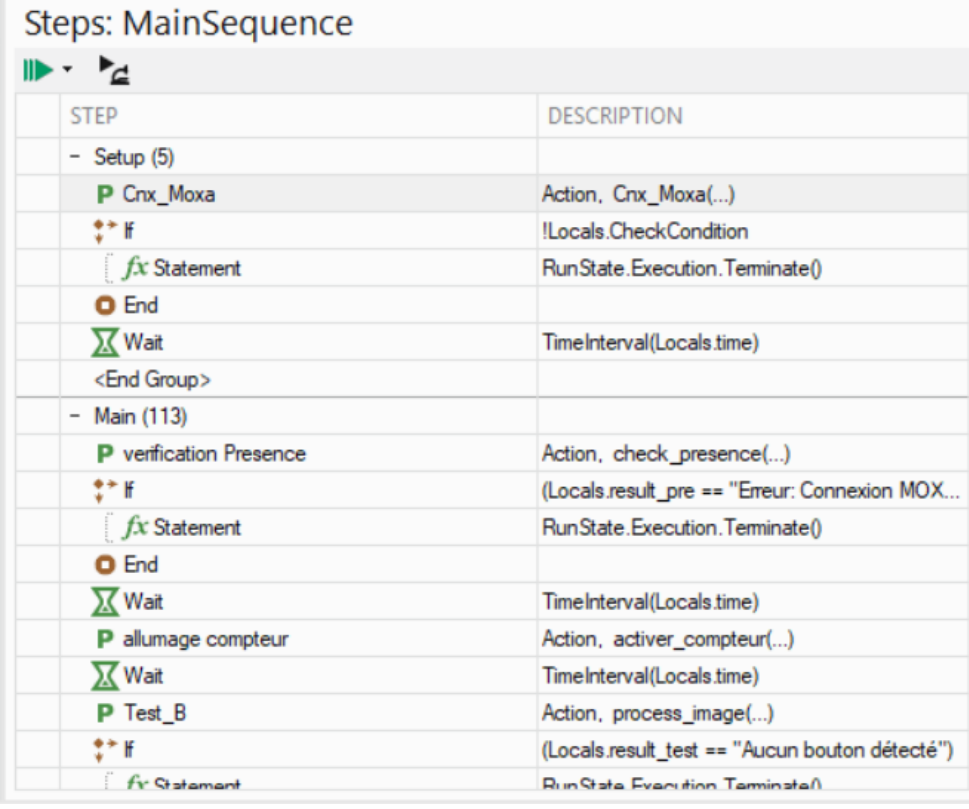
Une première inspection visuelle est alors lancée à l'aide du modèle de détection YOLOv8, afin de localiser le bouton-poussoir sur l'image capturée. En cas d'absence ou de mauvaise position du bouton, la séquence est interrompue de manière sécurisée. Dans le cas contraire, le test se poursuit avec l'envoi des commandes de test (de type CEI 62056-21) vers le compteur, en modifiant dynamiquement l'état de différents composants (LEDs, écran LCD, etc.). Ces commandes sont envoyées via un port série ou une couche logicielle dédiée.

À chaque commande transmise, une vérification de conformité est effectuée à l'aide du modèle de vision, pour s'assurer que l'effet attendu est bien visible (par exemple, l'allumage d'une

LED spécifique). Si une incohérence est détectée entre l'état visuel observé et la commande envoyée, la séquence s'interrompt immédiatement. Sinon, le processus continue jusqu'à l'exécution complète des tests.

À l'issue de la séquence, le compteur est mis hors tension en désactivant son alimentation via ioLogik, et la connexion TCP/IP avec le module est proprement fermée pour libérer les ressources.

Enfin, un verdict final (PASS ou FAIL) est généré et transmis à l'interface graphique de supervision via le protocole TCP/IP, assurant l'affichage du résultat et son archivage.



The image shows a screenshot of the TestStand 'Steps: MainSequence' window. It displays a table with two columns: 'STEP' and 'DESCRIPTION'. The steps are organized into two main groups: 'Setup (5)' and 'Main (113)'. The 'Setup' group includes steps for connecting to Moxa, checking conditions, and waiting. The 'Main' group includes steps for verifying presence, checking for errors, activating the counter, waiting, and testing the button. Each step is represented by an icon (e.g., a green 'P' for action, a yellow diamond for if-then-else, a green 'fx' for statement) and a description of the action or condition.

STEP	DESCRIPTION
- Setup (5)	
Cnx_Moxa	Action, Cnx_Moxa(...)
If	!Locals.CheckCondition
Statement	RunState.Execution.Terminate()
End	
Wait	TimeInterval(Locals.time)
<End Group>	
- Main (113)	
verification Presence	Action, check_presence(...)
If	(Locals.result_pre == "Erreur: Connexion MOX...)
Statement	RunState.Execution.Terminate()
End	
Wait	TimeInterval(Locals.time)
allumage compteur	Action, activer_compteur(...)
Wait	TimeInterval(Locals.time)
Test_B	Action, process_image(...)
If	(Locals.result_test == "Aucun bouton détecté")
Statement	RunState.Execution.Terminate()

Figure 61 : Exemple de Séquence

Pour des soucis de sécurité à chaque étape critique, des conditions d'arrêt ont été définies dans TestStand. En cas d'erreur ou de défaillance (absence de réponse, échec de détection, non-conformité visuel), la séquence est interrompue de manière anticipée afin de garantir la traçabilité des défauts et d'éviter les essais inutiles, la figure 62 décrit un exemple de condition.

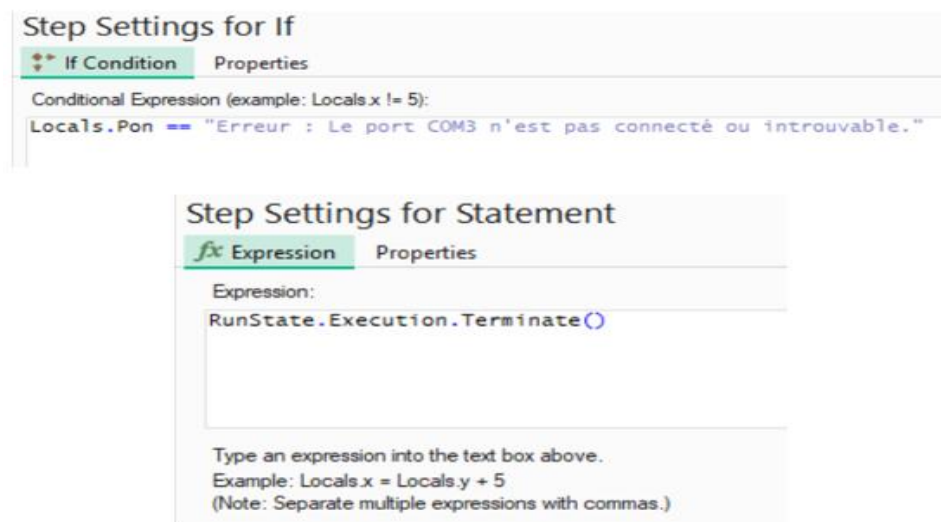


Figure 62 : Exemple de condition dans TestStand

Chaque appel Python est lié à des variables locales dans TestStand, permettant un échange fluide des données entre le moteur de test et les scripts IA, (Voir Figure 63).

Variables Sequences		
Variables		
Filter by name		
NAME	VALUE	TYPE
Locals ('MainSequence')		
A B_off	""	String
A B_on	""	String
A Camera_serial	"22610286"	String
TF CheckCondition	False	Boolean
123 conf_threshold	0.5	Number
TF ft	False	Boolean
A L_black	""	String
A L_clear	""	String
A Lcd_reset	""	String
123 Local	2	Number
→ Local_2	Nothing	Object Reference
A output_folder	"C:\Users\aya mejri\One..."	String
A output_folder2	"C:\Users\aya mejri\One..."	String
A output_folder3	"C:\Users\aya mejri\One..."	String
A path_image1	"C:\Users\aya mejri\Cam..."	String
A path_image10	"C:\Users\aya mejri\Cam..."	String

Figure 63 : Les variables locales dans TestStand

Afin de visualiser et superviser le séquençement des tests automatisés, une interface graphique (IHM) a été développée en Qt

2.2 Développement d'interface QT

Pour offrir une interface conviviale, interactive et adaptée aux environnements industriels, une interface utilisateur graphique (GUI) a été développée avec Qt, via le module PySide6 en Python.

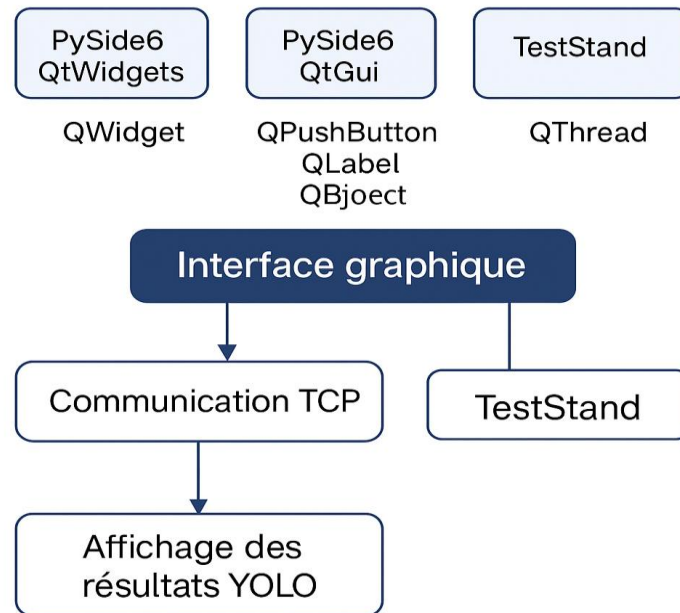


Figure 64 : Qt Creator pour le développement graphique

L'interface Qt a été conçue comme un point d'entrée ergonomique destiné aux opérateurs, leur permettant de s'authentifier facilement et d'interagir via une interface simplifiée. Elle assure la supervision complète de l'exécution des tests visuels, tout en permettant le déclenchement de commandes matérielles à destination du compteur électrique. Intégrée au cœur du système, l'IHM réceptionne et affiche dynamiquement les résultats issus du modèle YOLOv8, tels que les classes détectées, les images annotées et les verdicts associés. Enfin, elle communique de manière bidirectionnelle avec le moteur de séquençement TestStand afin de déclencher automatiquement les différentes étapes du processus de test.

L'interface est composée de plusieurs sections fonctionnelles représentées dans le tableau 20.

Tableau 19 : Les sections de l'interface Qt

Zone	Fonction Principale
Login	Authentifie l'opérateur avec identifiants sécurisés.
Contrôle de test	Démarrage et Arrêt de test, lancement des séquences TestStand
Détection de présence	Vérification du branchement du compteur

Commandes	Simulation de scénarios via envoi de commandes séries (MV)
Résultats	Affichage les verdicts (PASS/ FAIL) des différents modules.
Affichage caméra	Visualisation d'images capturées navigation entre clichés

Afin de garantir la réactivité, l'interopérabilité et la stabilité de l'interface Qt, en s'appuyant sur une sélection de bibliothèques et technologies spécifiques, chacune répondant à un besoin fonctionnel du système. Le tableau 21 suivant présente les principaux composants logiciels utilisés, ainsi que les outils ou modules associés assurant leur fonctionnement au sein de l'architecture du banc de test.

Tableau 20 : Les principaux composants logiciels utilisés

Composant	Technologie utilisée
Framework GUI	PySide6 (Qt pour Python)
Communication série	Pyserial (port COM3, 19200 bauds)
Communication réseau	socket TCP/IP (port 9000 local)
Détection visuelle IA	Modèle YOLOv8 entraîné + intégré Python
Affichage d'images	OpenCV + QPixmap
Exécution de séquences	os.startfile() vers fichier (.seq)
QThread	Exécution parallèle pour lecture/écriture réseau sans bloquer l'interface.

Le fonctionnement global de l'interface débute dès le lancement de l'application, l'opérateur est accueilli par une fenêtre de connexion sécurisée, dont le but est de restreindre l'accès au banc de test automatisé. Une fois authentifié, la fenêtre principale s'ouvre, centralisant toutes les interactions entre l'utilisateur, le compteur, le modèle d'intelligence artificielle et le moteur de test TestStand.

En cliquant sur le bouton « DÉMARRER TEST », plusieurs actions sont déclenchées automatiquement :

1. La séquence de test est exécutée via TestStand en ouvrant un fichier (.seq), grâce à la fonction Python os.startfile().
2. Un serveur TCP local est démarré sur le port 9000 à l'aide de la classe personnalisée TcpServer, pour recevoir en temps réel les résultats IA sous forme de messages JSON.
3. L'interface passe en mode supervision et attend l'image capturée par la caméra industrielle ainsi que le verdict associé.

L'analyse visuelle est assurée par un modèle YOLOv8 intégré côté serveur Python. Ce dernier traite l'image, renvoie le chemin de l'image annotée ainsi que la classe détectée, la confiance, et le statut (PASS ou FAIL). Ces informations sont reçues par le client TCP de Qt et affichées dynamiquement dans la zone image de l'interface, avec un message clair pour l'opérateur.

En parallèle, l'IHM permet également à l'utilisateur de simuler manuellement certains scénarios de test via des boutons dédiés (par exemple : LED P ON, LCD Clear, Backlight OFF). Chaque bouton déclenche une commande série CEI 62056-21 envoyée vers le compteur via le port COM3 à l'aide du module pyserial. La réponse ou l'absence de réponse du compteur permet à l'interface de mettre à jour l'état du bouton, de la LED ou du LCD avec un retour visuel immédiat.

Une fois tous les tests terminés, l'interface compile les résultats partiels et affiche un verdict global :

- ✓ Produit OK : si toutes les détections sont passées avec succès,
- ✗ Produit non conforme : si au moins une détection a échoué.

L'interface se remet ensuite en attente d'une nouvelle session de test, tout en conservant l'historique d'images et des verdicts dans une file circulaire.

Exemple de résultat visuel de l'interface Qt dans les figures suivants 65 et 66.

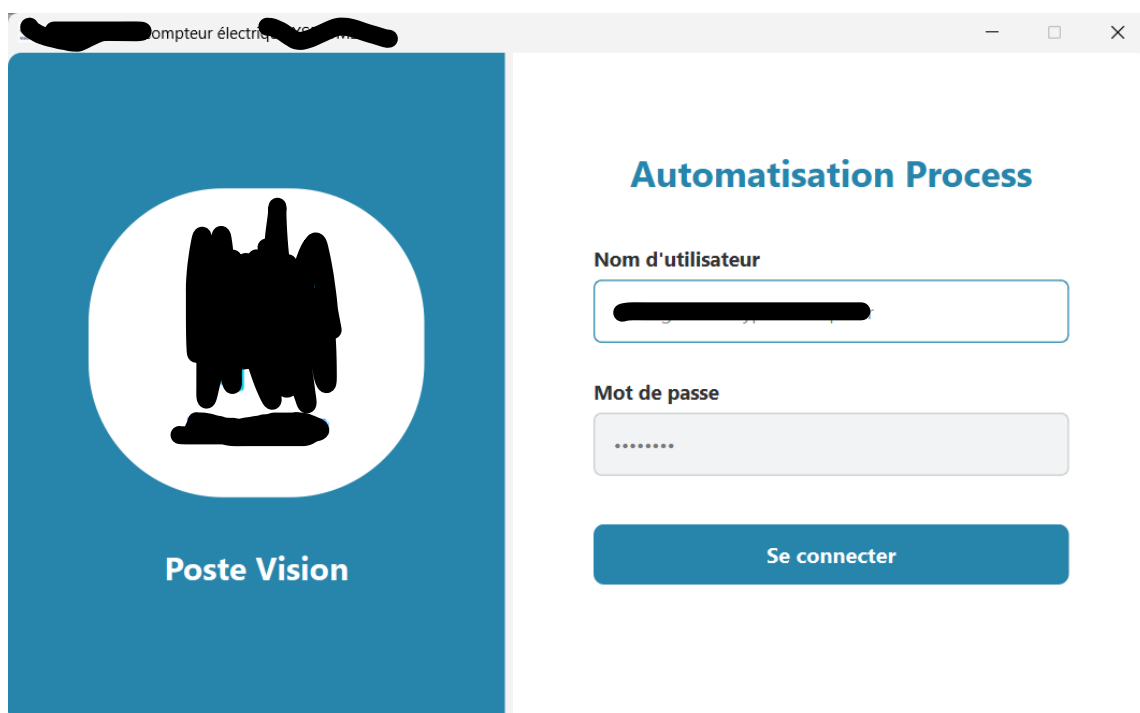


Figure 65 : Une fenêtre de connexion sécurisée

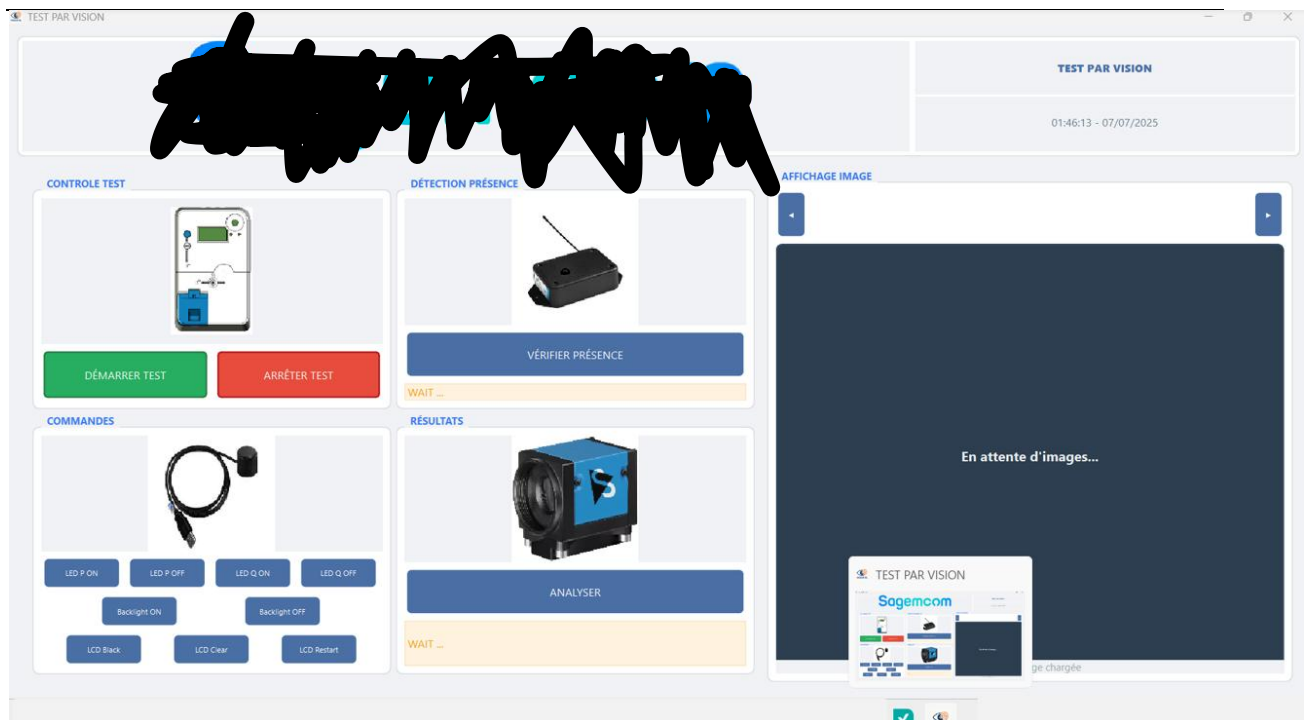


Figure 66 : La fenêtre principale de visualisation de test

2.3 Les résultats au cours de test sur le prototype

Afin d'évaluer la performance du système de test visuel automatisé développé, des expérimentations ont été menées en conditions réelles sur le compteur électrique XS211. Les objectifs principaux étaient de valider la capacité du modèle YOLOv8 à détecter avec précision les composants visuels critiques (LEDs, écran LCD, bouton), tout en maintenant un temps d'exécution compatible avec les contraintes industrielles.

Les résultats obtenus mettent en évidence une amélioration significative des indicateurs qualité par rapport au système initial Vision Builder. En outre, le temps moyen de traitement par image s'établit à 120 ms, garantissant la continuité du flux de production sans ralentissement.

La Figure 67 illustre un exemple de résultat obtenu lors d'un test sur un compteur conforme. On y observe la détection précise des LEDs en état « P_ON » et « Q_OFF », ainsi que la reconnaissance correcte de l'écran LCD en mode « BACKLIGHT ON ».

La boîte englobante (bounding box) autour des éléments atteste de la capacité du modèle à localiser et classifier chaque composant avec exactitude.

Cette précision réduit considérablement le risque de faux rejets, tout en garantissant la traçabilité complète des inspections.

Chapitre 3 : Implémentation et validation de l'approche de test

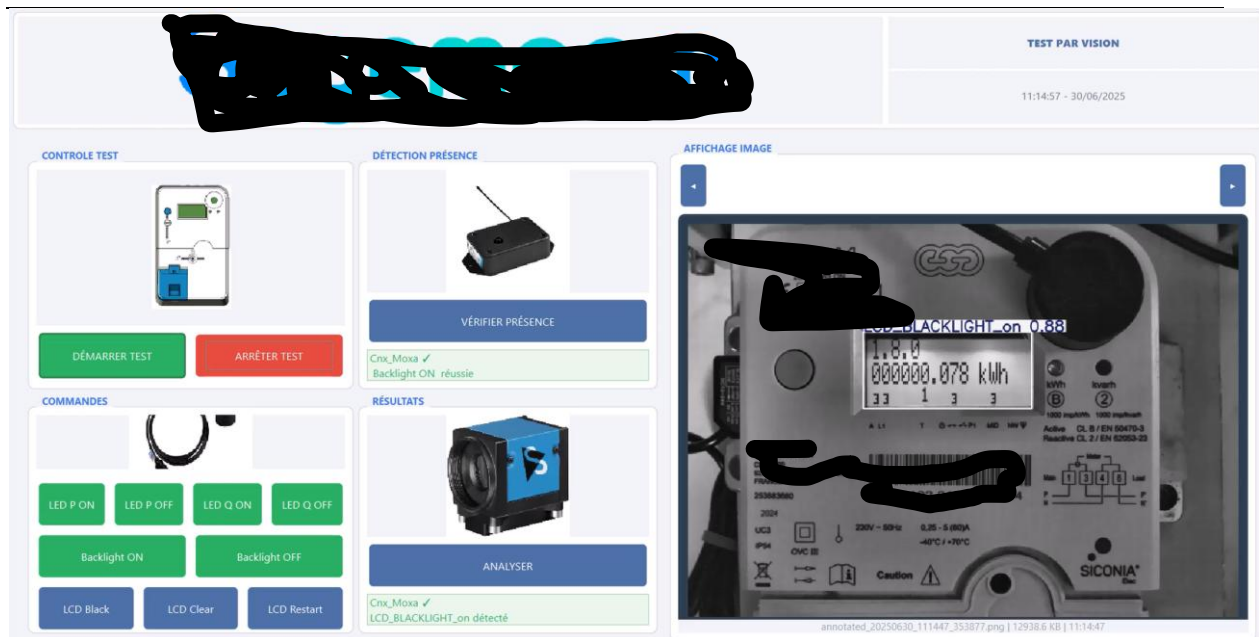


Figure 67 : Exemple de Test 1

Une fois toutes les vérifications terminées avec succès, le verdict final s'affiche dans la section Résultats : « PRODUIT OK », indiquant que le compteur est conforme et peut être validé pour expédition, (Voir Figure 68).

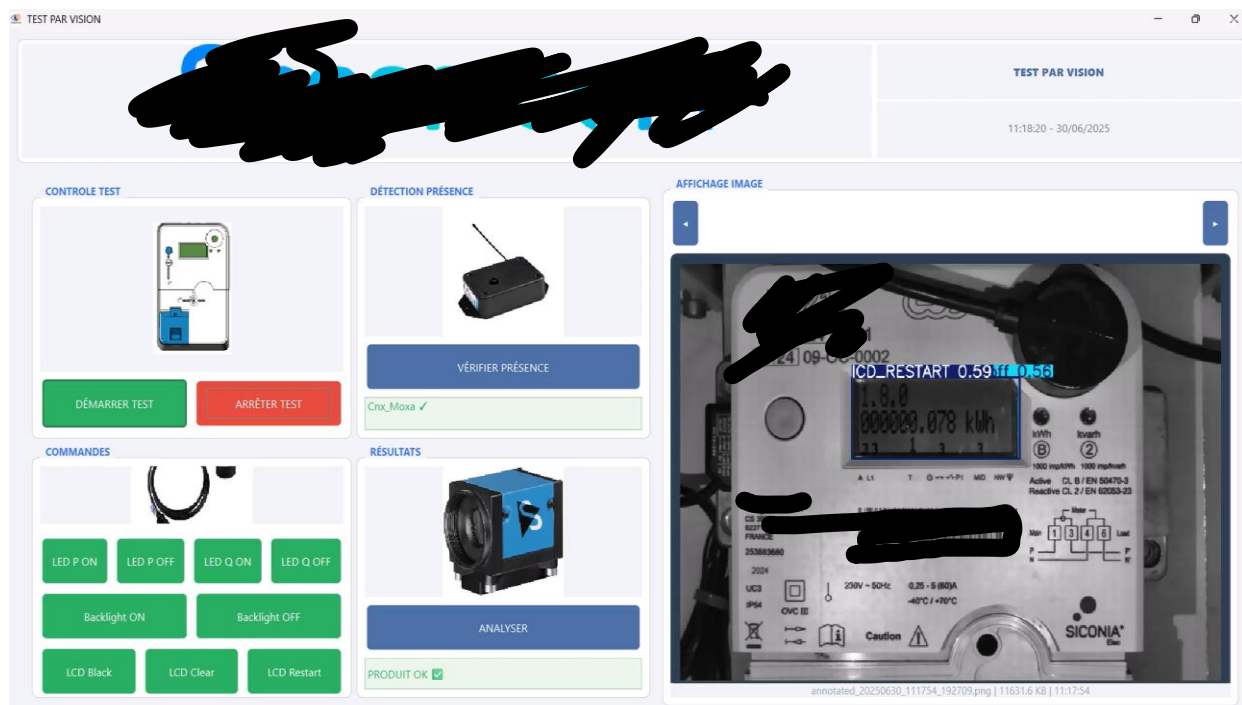


Figure 68 : Résultat de Test 1

Chapitre 3 : Implémentation et validation de l'approche de test

La Figure 69 présente un scénario où un défaut a été volontairement simulé, en retirant le bouton poussoir. Le modèle détecte l'absence d'allumage de LED Q, qui contrairement au reçu de commande par le port serie ça engendre un probleme matériel de compteur.

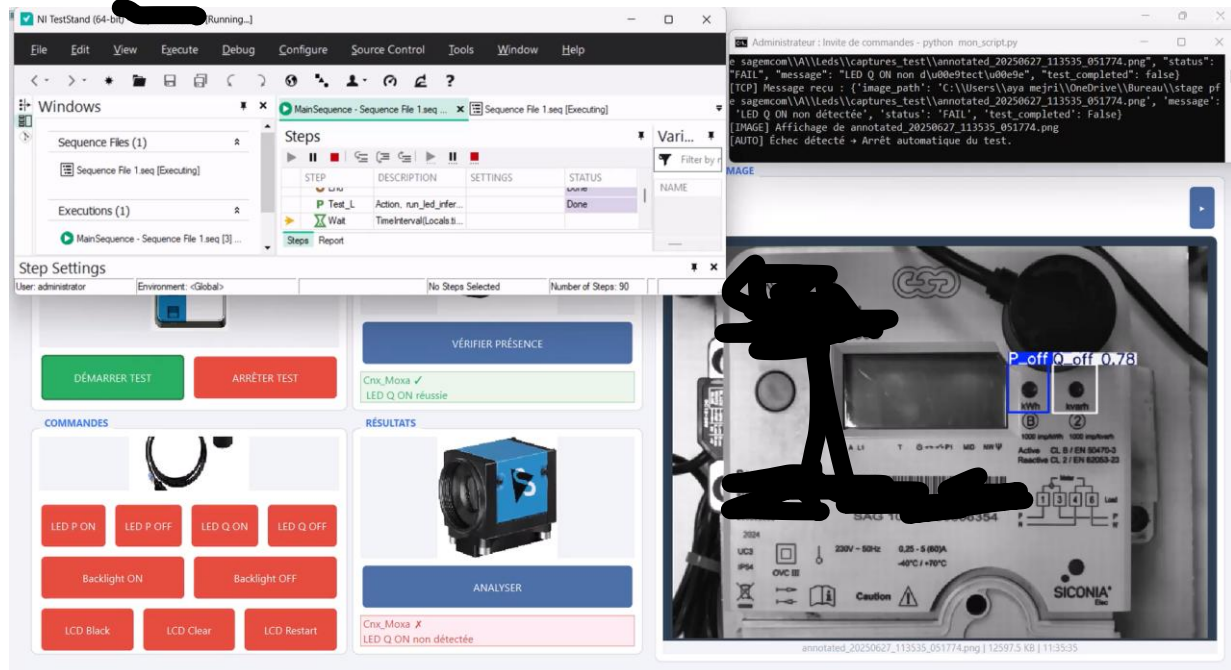


Figure 69 : Exemple de Test 2

Donc il émet immédiatement un verdict « Non conforme ». Cette fonctionnalité permet de prévenir l'expédition d'unités incomplètes ou présentant des défauts critiques, renforçant ainsi la qualité finale des produits livrés. (Voir la figure 70)

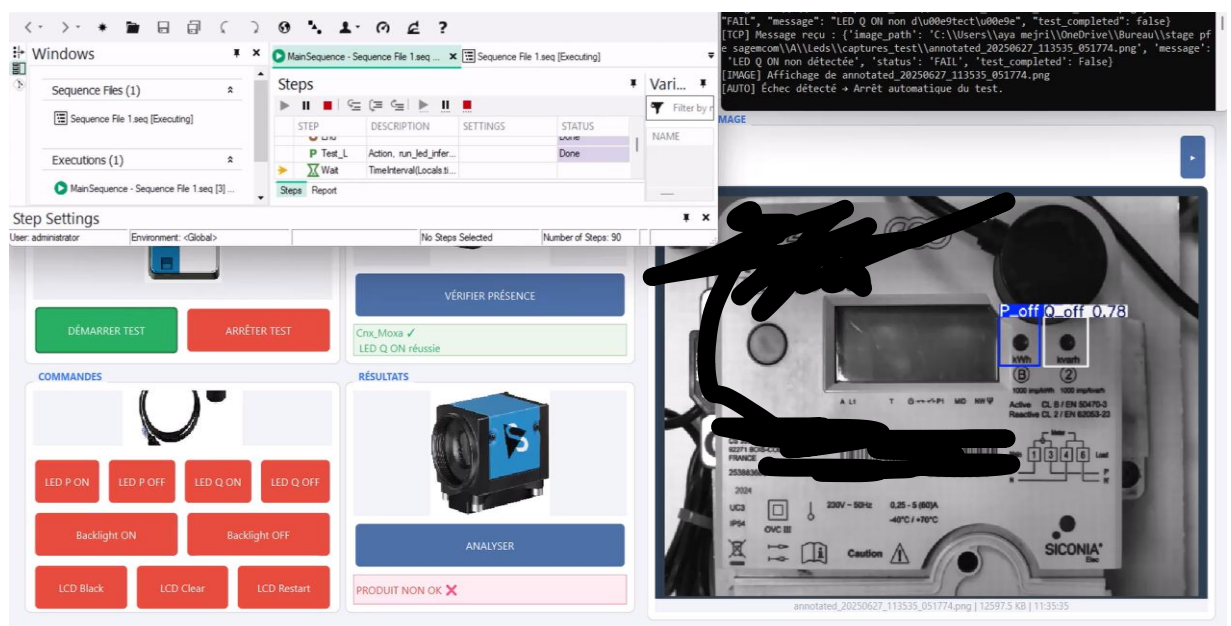


Figure 70 : Résultat de Test 2