Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

MÉMOIRE DE BACHELOR

Francisco Ribeiro

Système de Surveillance en Temps Réel pour Détection de Défauts sur Aluminium

Informatique et Systèmes de communication (ISC)

ISC-ID-25-3

Haute école spécialisée de Suisse occidentale
Faculté Ingénierie et Architecture
Haute École d'Ingénierie de Sion

Francisco Ribeiro

Système de Surveillance en Temps Réel pour Détection de Défauts sur Aluminium

Mémoire soumis pour le programme de bachelor
Informatique et Systèmes de communication (ISC) – Data engineering major
Haute École d'Ingénierie de Sion

Superviseur·e : **PROF. DR FRANCESCO CARRINO**

Expert·e : **DR EMMANUEL SENFT**

Travail soumis le : 25 juillet 2025

Abstract

This Bachelor thesis was carried out in collaboration with the company Constellium, in the field of industrial vision applied to quality control.

The main objective is to design a system capable of automatically detecting visual defects in real time on extruded aluminum profiles moving at 0.7 m/s, specifically on the lower face, which is difficult for operators to access.

The project has two main objectives:

1. The implementation of a waterproof prototype, capable of capturing images on the production line, with the aim of automatically detecting defects
2. The exploration of several unsupervised methods for visual anomaly detection.

Defects appear on average once every two months, which makes supervised approaches impractical, as we do not have high-quality images of defective parts. To test our methods, we therefore created a dataset containing images with artificial defects: these defects were segmented from photos taken with a smartphone, then integrated into normal images.

A benchmark was conducted on six unsupervised methods: Autoencoder, AnoGAN, Isolation Forest, One-Class SVM, KMeans-ResNet, and PatchCore. The dataset contained approximately 100,000 training images (without defects) and 20,000 validation images with simulated defects.

PatchCore achieved the best performance with an AUROC of 87%, demonstrating good robustness, and an average inference time of 273 ms.

The current limitations of our system include motion blur caused by the moving profiles, low lighting conditions, exposure to aluminum chips, and the model's inference time. The next steps would involve integrating a more powerful camera in a better protected capture area, and using a faster model.

Keywords: engineering, data, machine learning, Anomaly detection

Project repository:
```
https://github.com/franciscoribeiro99/TravailDeBachelor
https://github.com/franciscoribeiro99/SSTRDDA-RPI
```

Résumé

Ce travail de Bachelor a été réalisé en collaboration avec l'entreprise Constellium, dans le domaine de la vision industrielle appliquée au contrôle de qualité.

L'objectif principal est de concevoir un système capable de détecter automatiquement, en temps réel, des défauts visuels sur des profilés en aluminium extrudé se déplaçant à 0.7 m/s, plus précisément sur la face inférieure, difficilement accessible aux opérateurs.

Le projet a deux objectifs majeurs:

1. La mise en place d'un prototype étanche, capable de capturer des images sur la ligne de production, dans le but de détecter automatiquement des défauts.
2. L'exploration de plusieurs méthodes non supervisées pour la détection d'anomalies visuelles.

Les défauts appariassent en moyenne une fois tous les deux mois, ce qui rend les approches supervisées peu praticable car nous ne disposons pas d'images défectueuses de bonne qualité. Pour tester nos méthodes: nous avons donc créé un dataset contenant des images avec défauts artificiels: ces défauts ont été segmentés à partir de photos prises avec un smartphone, puis intégrés dans des images normales.

Un benchmark a été réalisé sur six méthodes non supervisées : Autoencoder, AnoGAN, Isolation Forest, One-Class SVM, KMeans-ResNet et PatchCore. Avec un dataset contenant environ 100′000 images d'entraînement (sans défauts) et 20′000 de validation avec des défauts simulés.

Patchcore a obtenu les meilleures performances avec un AUROC de 87% démontrant une bonne robustesse, avec un temps d'inférence moyen de 273 ms.

Les limitations de notre système sont le flou lié au mouvement des profilés, la faible luminosité, l'exposition aux copeaux d'aluminium et le temps d'inférence de notre modèle. Les prochaines étapes consisteraient à intégrer une caméra plus performante dans une zone de capture mieux protégée et à utiliser un modèle plus rapide.

Mots-clés : engineering, data, machine learning, Anomaly detection

Lien de dépôt du projet : `https://github.com/franciscoribeiro99/TravailDeBachelor`
`https://github.com/franciscoribeiro99/SSTRDDA-RPI`

Remerciements

Je tiens à remercier toutes les personnes ayant contribué à la réalisation de mon travail bachelor.

Je souhaite remercier, au sein de l'HES-SO, les personnes et services suivants :

- **Prof. Francesco Carrino :** pour ses conseils, disponibilité et positivisme tout au long de ce projet
- **Filière ISC :** réactivité pour la validation des commandes de matériel
- **Service technique :** Création d'un coffret électrique étanche

Je souhaite également remercier l'entreprise **Constellium** pour sa collaboration plus en particulier les personnes suivantes:

- **Raoul Rey :** pour toute la partie dessins techniques ainsi que tous les montages mécaniques.
- **William Rob Prieur :** mise à disposition de matériel pour les montages
- **Service technique :** mise à disposition de matériel électrique

Je remercie également toutes les personnes ayant relu cette thèse.

Table des matières

Chapitre 1 – Introduction

1.1 – Déclaration sur l'utilisation de l'intelligence artificielle

Dans le cadre de ce travail de bachelor, certains outils basés sur l'intelligence artificielle ont été utilisés à des fins d'assistance. ***GitHub Copilot*** *a notamment été employé pour suggérer des portions de code ; Quand* ***ChatGPT ou Claude AI*** *ont été utilisés, les passages explicitement mentionnés dans les commentaires du code source. Par ailleurs, ChatGPT a été ponctuellement sollicité pour des tâches de reformulation ou de clarification de texte.*

1.2 – Contexte et motivation

Dans les secteurs industriels de l'aéronautique, l'automobile et des véhicules ferroviaires de transport de personnes, les profilés en aluminium jouent un rôle essentiel grâce à leur compromis résistance/poids/prix. La qualité de surface et la solidité de ces profilés est essentielle pour garantir leur qualité et répondre aux attentes des clients. Des défauts visuels comme des soufflures, fissures ou inclusions peuvent compromettre la qualité du produit, ses performances, voire sa sécurité dans des applications exigeantes.

Pour maintenir des standards qualité élevés et s'aligner aux exigences, les industriels cherchent des solutions innovantes pour automatiser l'inspection visuelle et réduire le contrôle qualité manuel.

Dans notre cas spécifique, l'unité AS&I (Automotive Structures & Industry) de Constellium Valais SA à Sierre produit des profilés en aluminium mesurant environ **100 à 210 mm de hauteur**, **150 à 720 mm de largeur**, et jusqu'à **28 mètres de longueur**.

Ces profilés avancent sur des lignes d'extrusion où des défauts peuvent apparaître sur différentes faces du matériau. L'objectif de ce travail est de détecter automatiquement les défauts de surface situés sur la face inférieure du profilé, celle orientée vers le sol pendant la production.

Constellium dispose à Sierre de 2 grandes presses d'extrusion. La petite avec 7200 tonnes de poussée (presse n°29) et la grande avec 7500 tonnes (presse n°27), **sur laquelle** nous avons installé notre dispositif, extrudent entre **10'000 et 14'000 tonnes par année**. La production horaire est entre les **1.8 et 2.2 tonnes** . Cela permet de comprendre l'enjeu qui se cache derrière les arrêts d'entretiens et de révisions de ces machines pendant une durée de 3-4 heures. Nous devons être rapides, précis et efficaces pendant l'installation de notre système.

Pour avoir une idée nous allons aborder brièvement comment fonctionne le processus d'extrusion de l'aluminium.

Nous travaillons sur une presse à extrusion directe où nous avons une matrice de forme cylindrique qui va donner la forme de notre profilé en sortie.

1. Matrice (filière) pré chauffée à environ 450-500 degrés
2. Préchauffage de la billette d'aluminium qui est un bloc de l'alliage en question. Cette étape a pour but de rendre l'alliage malléable pour faciliter son extrusion.
3. Pousser le matériau dans le conteneur de la presse avec un vérin hydraulique. Ceci va remplir le récipient.
4. Maintenir la pression pour contraindre le matériau à traverser la matrice(filière) pour extruder le profilé.
5. Trempe en ligne du profilé dans le but de bloquer sa structure.
6. Tendre au-delà de la limite élastique. Permet de corriger les défauts de vrillage et de rectitude. Permet la création de plans de dislocations augmentant les caractéristiques du matériau. Permet un alignement des contraintes.
7. Débitage

Dans la Figure 1, on peut voir un aperçu de la partie 3-4 du processus dans la partie d'extrusion directe.

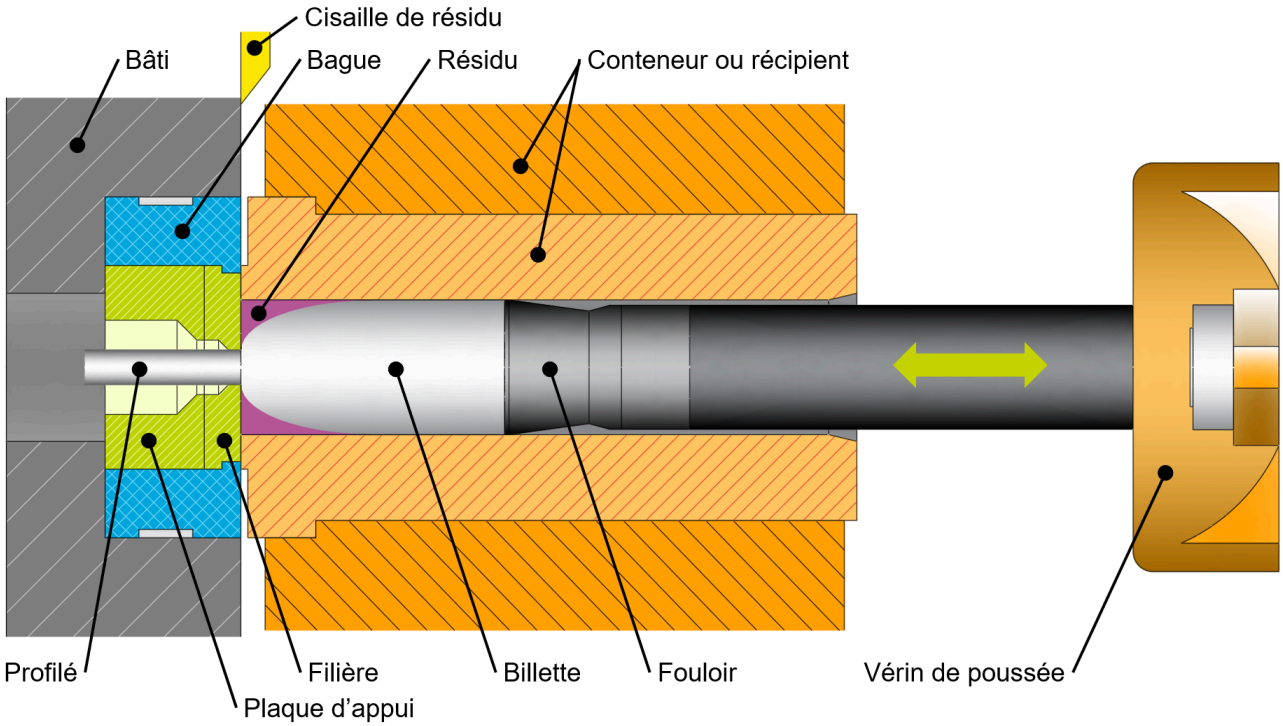


Figure 1 - Extrusion directe, fait par Raoul Rey (Constellium)

Dans la Figure 2, nous pouvons apercevoir la taille de notre système par rapport à la taille des différents éléments de cette presse.

Nous avons également plusieurs étapes que nous n'aborderons pas en détail afin de réduire le périmètre du projet. C'est une machine très complexe avec de nombreuses étapes.

Nous pouvons voir, que notre système se trouve à la fin de la ligne de production avant le débitage. Sachant que peu d'intervention humaine voir aucune existe entre la sortie de la presse et le débitage notamment parce que le métal est à température élevée.

Le choix de l'installation de notre système, à la fin de presque tout le processus, a pour but de pouvoir d'identifier le plus de défauts possibles. Les défauts telles les blister se passent à l'extrusion, mais d'autres défauts telles les bosses peuvent apparaître pendant le transfert des profilés depuis la sortie de presse jusqu'à la zone de débitage.

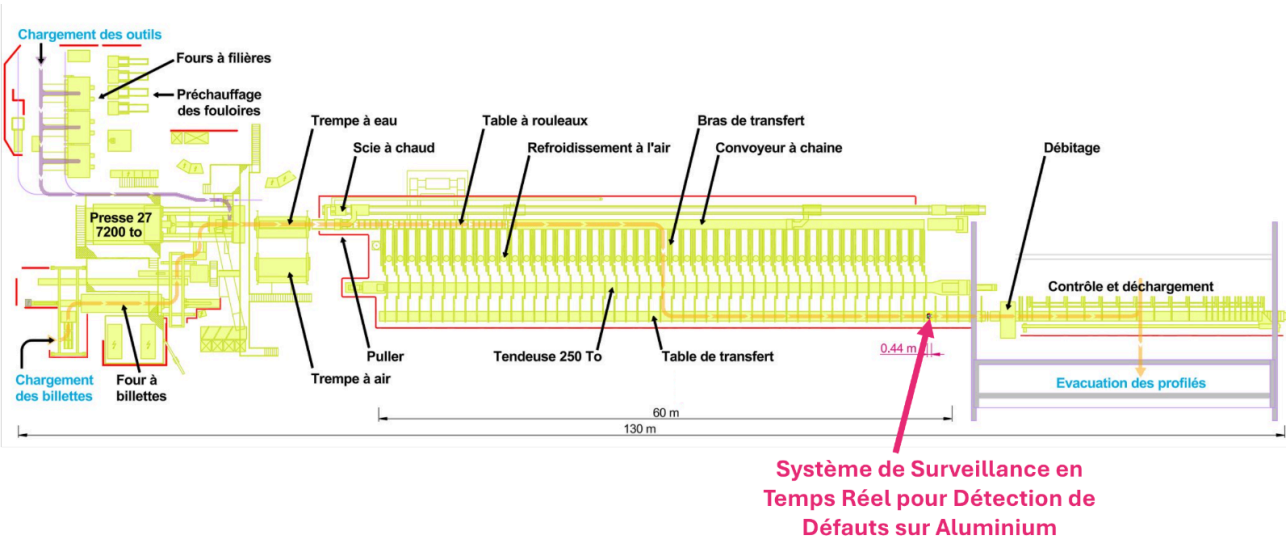


Figure 2 - Plan approximatif Constellium, fait par Raoul Rey (Constellium)

1.3 – Problématique

Actuellement, l'inspection de surface est entièrement réalisée par des opérateurs, chargés d'identifier ces défauts. Cependant, ce processus manuel présente plusieurs limitations :

- Fatigue humaine et manque de constance
- Vitesse de production élevée, donc peu de temps pour inspecter

Il existe donc un réel besoin pour une solution d'inspection automatisée en soutien à l'inspection manuelle.

Les limites du contrôle visuel manuel – accès difficile à la face inférieure des profilés, justifient le développement d'une solution automatisée. Un système idéal capturerait en continu des images des profilés sur la ligne et signalerait toute anomalie sans nécessiter un nombre important d'exemples de défauts.

Plusieurs défis techniques se présentent : d'une part, l'acquisition d'images dans une zone sombre nécessite un éclairage artificiel adapté et/ou l'utilisation de caméras conçues pour ce type d'environnement. L'aluminium est un matériau très réfléchissant. D'autre part, la détection automatique des défauts doit tenir compte de la grande variabilité des textures et de l'apparence des profilés.

1.4 – Objectif

L'objectif de ce travail de bachelor est de développer un prototype fonctionnel pour installation sur la ligne de production. Ce prototype doit être capable de capturer des images et les envoyer sur un serveur. La deuxième étape est de faire un benchmark de plusieurs méthodes non supervisé est de choisir la plus performante pour utilisation dans notre système. Nous cherchons à avoir une méthode rapide et robuste, l'objectif est d'avoir un système qui performe en temps réel. À terme, le but est de pouvoir effectuer l'inférence de cette méthode sur ce prototype.

1.5 – Structure du rapport

Ce rapport est structuré comme suit :

- **Chapitre 2** présente l'état de l'art en détection d'anomalies non supervisées.
- **Chapitre 3** détaille la méthodologie adoptée pour ce projet.
- **Chapitre 4** présente les résultats obtenus lors du benchmark des six méthodes de détection d'anomalies.
- **Chapitre 5** discute des améliorations futures possibles
- **Chapitre 6** conclut ce travail en synthétisant les contributions principales

Chapitre 2 – État de l'art

La détection de défauts consiste à identifier des images contenant des défauts par rapport à ce qui est considéré comme normal. Cette tâche est cruciale, par exemple pour repérer des produits défectueux en industrie. Une caractéristique commune des méthodes modernes est qu'elles peuvent être entraînées uniquement sur des exemples normaux, les anomalies étant souvent rares ou difficiles à énumérer. On cherche alors à apprendre la distribution des données sans défauts afin de détecter toute image qui s'en écarte significativement.

Dans le cadre de ce projet, nous allons nous concentrer exclusivement sur les approches non supervisées. Nous sommes partis dans cette idée car les défauts apparaissent qu'une fois tous les deux mois environ, rendant très difficile avoir un dataset labellisé suffisant pour utiliser des méthodes supervisées. Les approches non supervisées permettent d'apprendre la normalité à partir d'images sans défauts pour détecter automatiquement toute déviation de cette normalité.

Les sections suivantes présentent brièvement le principe de chaque méthode testée, avant d'approfondir la méthode PatchCore, qui s'est avérée la plus performante dans notre cas.

2.1 – Autoencodeur (Autoencoder) [1]

Un autoencodeur est un réseau de neurones entraîné de manière non supervisée à reconstruire ses propres données en sortie (sortie identique à l'entrée). Il est constitué d'un encodeur qui compresse l'image d'entrée en une représentation de faible dimension (latent), et d'un décodeur qui essaie de reconstruire l'image originale à partir de cette représentation. En entraînant ce modèle uniquement sur des images normales, l'autoencodeur apprend à restituer fidèlement ces images sans défauts. Lorsqu'on lui présente une image anormale qu'il n'a pas vue pendant l'entraînement, la reconstruction est généralement de moins bonne qualité. L'idée est donc que les anomalies produisent une erreur de reconstruction élevée, alors que les données normales sont bien reconstruites.

En pratique, on définit un score d'anomalie basé sur l'erreur de reconstruction (par exemple l'erreur quadratique moyenne pixel à pixel) : si ce score dépasse un certain seuil, l'image est considérée comme défectueuse. Les autoencodeurs peuvent être difficiles à entraîner efficacement pour capturer toutes les variations des données normales.

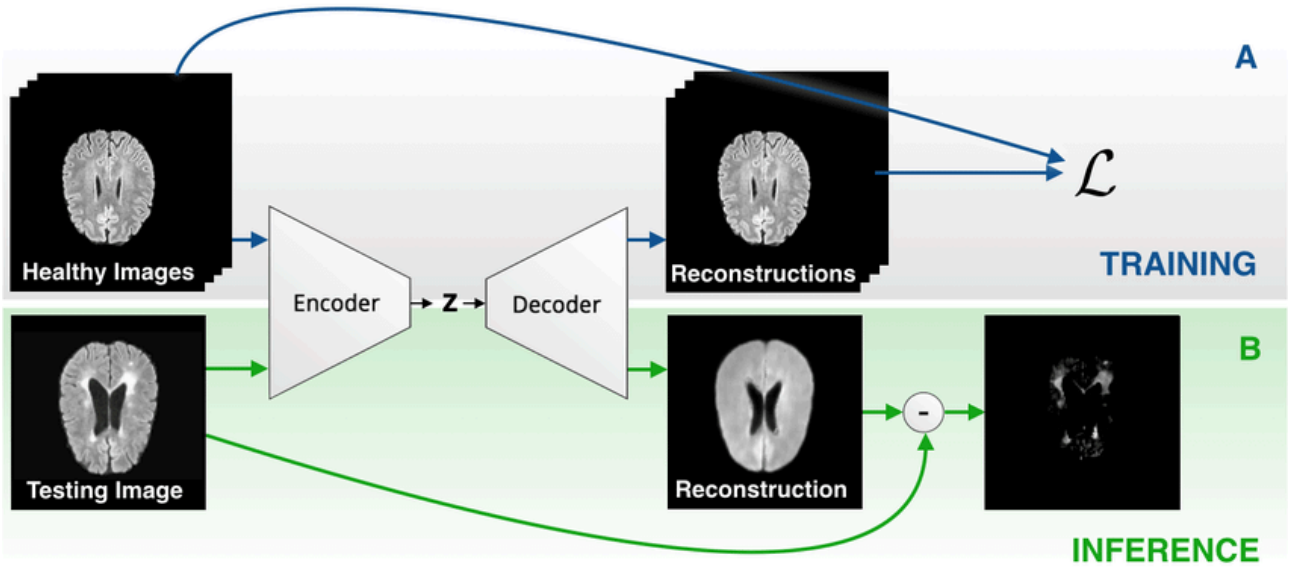


Figure 3 - The concept of Autoencoder-based Anomaly Detection/Segmentation: A) Training a model from only healthy samples and B) anomaly segmentation from erroneous reconstructions of input samples, which might carry an anomaly. [7]

2.2 – AnoGAN (Anomaly GAN) [2]

AnoGan est une méthode basée sur les réseau antagoniste génératif (GAN) [8].

Dans cette méthode, deux grandes étapes se distinguent:

1. **Pré entraînement d'un générateur :** Le générateur est un réseau neuronal convolutif qui prend en entrée un vecteur latent aléatoire. Ce dernier apprend à générer des images réalistes à partir de bruit en minimisant la distance entre les images générées et les images normales via une loss de reconstruction (MSE).
2. **Recherche du meilleur vecteur latent :** Pour détecter une anomalie, on ne donne pas une image au réseau mais on cherche directement le vecteur qui permet de générer une image qui s'approche au mieux de l'image de test. ce vecteur z est optimisé par rétropropagation à l'aide d'une fonction de coût combinant:
   - L'erreur de reconstruction pixel à pixel
   - la différence entre les features extraites de l'image générée et celles de l'image de test.

Le score d'anomalie est défini à partir de cette perte : plus l'image est difficile à reproduire, plus le score sera élevé (suspicion d'anomalie).

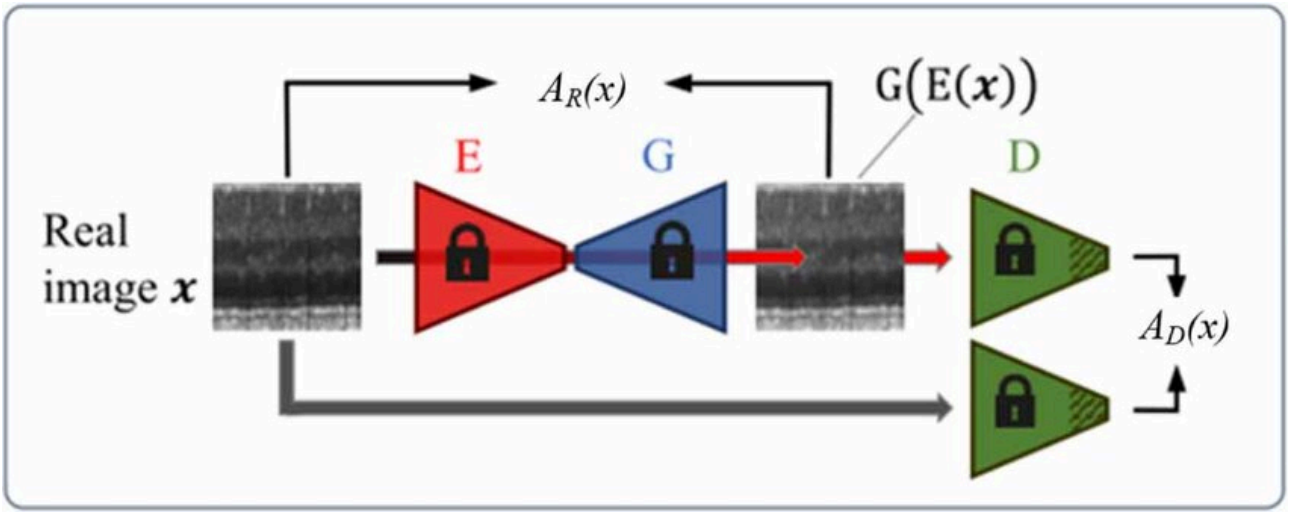


Figure 4 - The detection process of f-AnoGAN by Yikang Zhang, Jiale Li, li Junfeng, Haipeng Pan [9]

2.3 – Hybrid Resnet Kmean [3]

Cette méthode utilise un CNN avec architecture de Resnet et un modèle pré entraîne afin d'en sortir un vecteur de caractéristiques. À partir de là, on procède ainsi:

1. **Clustering avec Kmeans**

   On applique Kmeans avec un seul cluster sur tous les vecteurs des images normales (sans défauts). Ceci permet d'approcher le centre de la distribution des images normales.

2. **Évaluation des images**

   Pour les images de test, on va en sortir les vecteurs de caractéristiques et ensuite calculer la distance euclidienne avec le centroïde et les images normales. Une distance élevée indique une anomalie.

L'avantage de cette méthode est sa simplicité d'implémentation et d'interprétation.

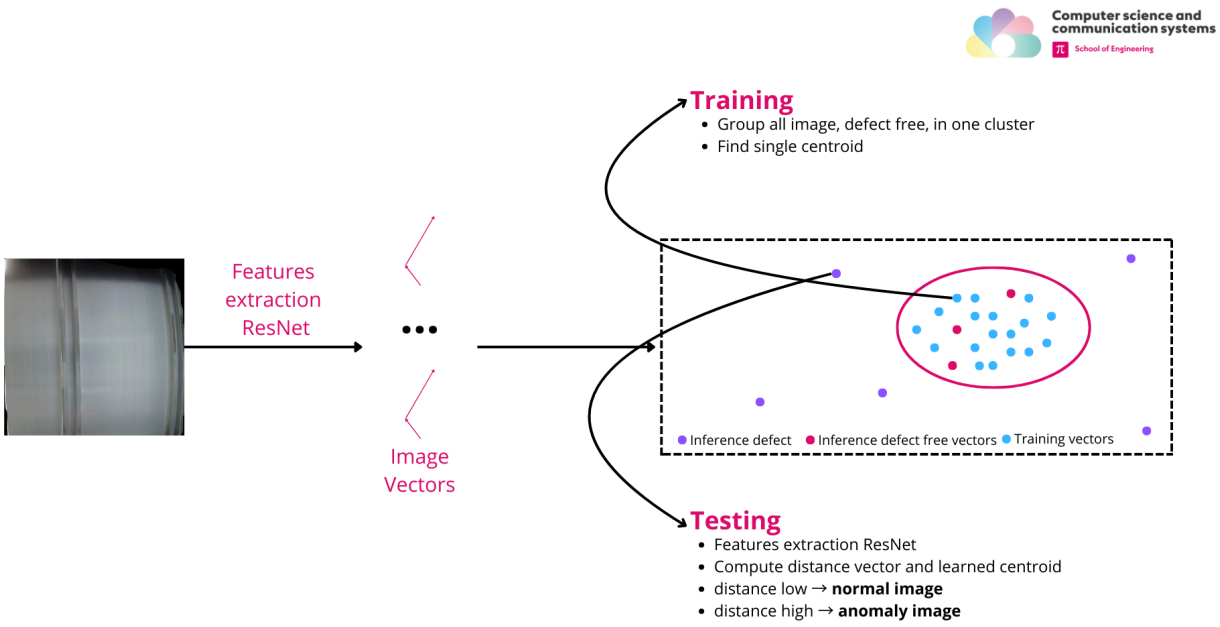


Figure 5 - Kmeans explication

2.4 – Isolation forest [4]

Isolation forest est un algorithme de détection d'anomalies basé sur la construction des arbres aléatoires.

L'idée est d'extraire des vecteurs à l'aide de ResNet avec un modèle pré entraîné. En phase « d'entraînement », plusieurs arbres binaires aléatoires sont construits: à chaque nœud, on choisit une feature et un seuil pour diviser l'espace. La profondeur moyenne reflète la normalité. À l'inférence, on considère que :

- **Chemin court**: Correspond à un point isolé donc **anomalie**.
- **Chemin Long**: correspond à des points semblables vus en entraînement.

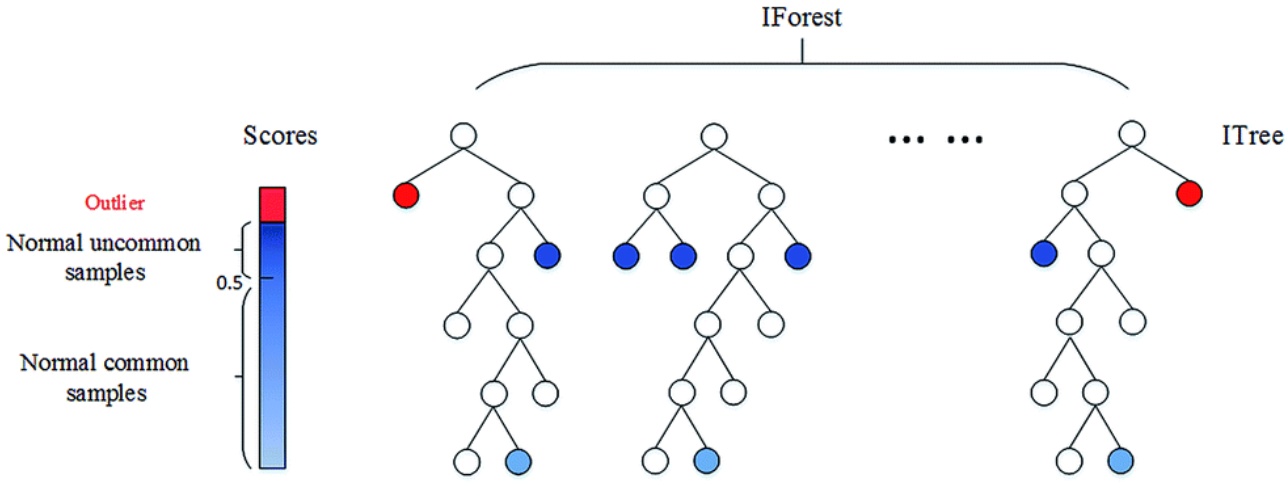


Figure 6 - Isolation Forest anomaly detection [10]

2.5 – One class SVM [5]

One class SVM est une méthode qui cherche à rassembler toutes les données normales à l'intérieur d'une frontière définie (hyperplan).

En entraînement, nous récupérons les vecteurs avec les features de chaque image qu'on extrait à l'aide de ResNet pré entraîné, on normalise les données et on entraîne un modèle OneClassSVM afin d'englober toutes les données normales.

À la détection on vérifie si notre vecteur caractéristique se trouve à l'intérieur de la frontière (sans défaut) ou en dehors (avec défaut).

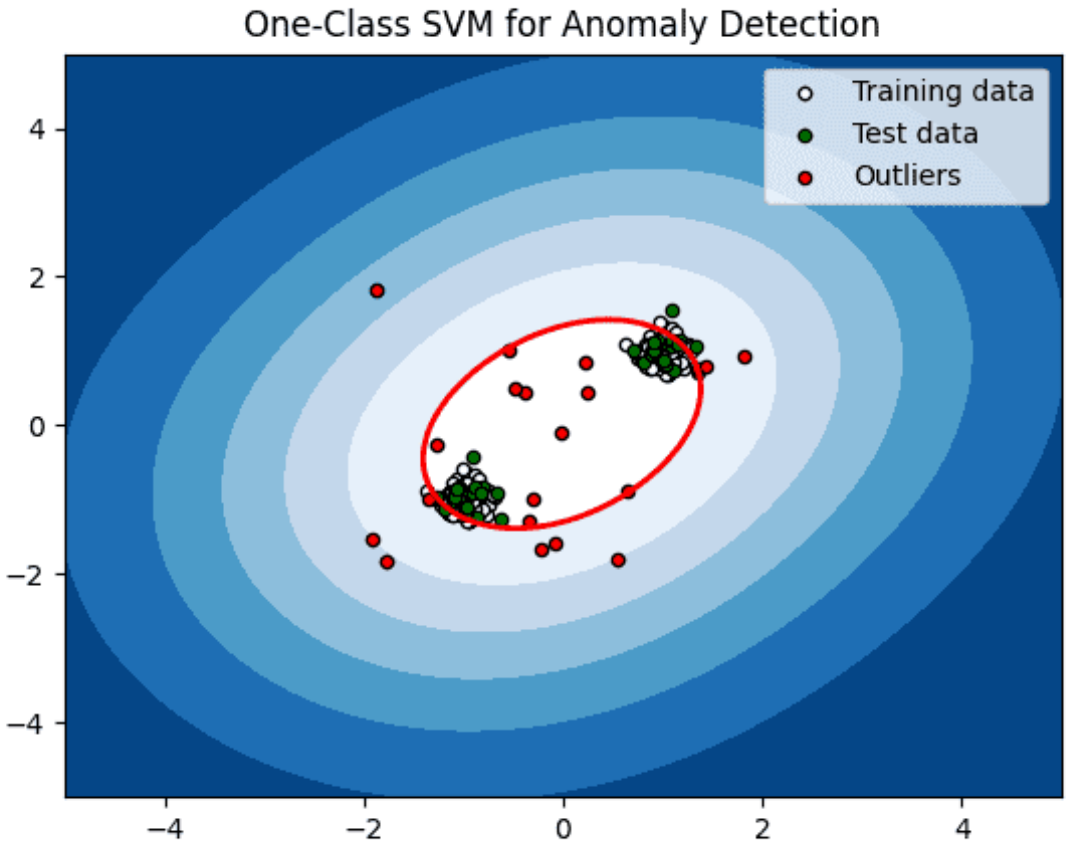


Figure 7 - One class SVM for anomaly detection by N. Van Otten [11]

2.6 – PatchCore [6]

PatchCore est une méthode qui performe incroyablement bien, combinant l'extraction de feautures avec des CNN pré entraînés, comparaison de patch et réduction de dimension.

Toutes les images utilisées en entraînement sont sans défauts.

En entraînement :

1. Nous extrayons les features de l'image à l'aide d'un modèle CNN pré entraîné tronqué (sans les couches de classification). Cela nous permet d'obtenir des représentations locales, correspondant à des patchs de l'image.
2. Chaque patch est représenté par un vecteur de caractéristiques.
3. Ces vecteurs sont ensuite projetés dans un espace plus faible dimension avec un projecteur aléatoire de type SparseRandomProjector.
4. Un sous échantillonnage aléatoire (coreset) est effectué afin d'éviter d'avoir des mêmes patches représentés à plusieurs reprises tout en conservant la couverture de tous les motifs.

Pour la détection :

1. Pour chaque image on extrait les caractéristiques de chaque patch.
2. On projette les features de ces patch.
3. Chaque patch est comparé à ceux du coreset via FAISS, une bibliothèque de recherche optimisée pour trouver le voisin le plus proche.
4. Le score d'anomalie est ensuite calculé. Si le voisin le plus proche se trouve très éloigné, nous sommes dans un cas de motifs jamais vus donc défaut.

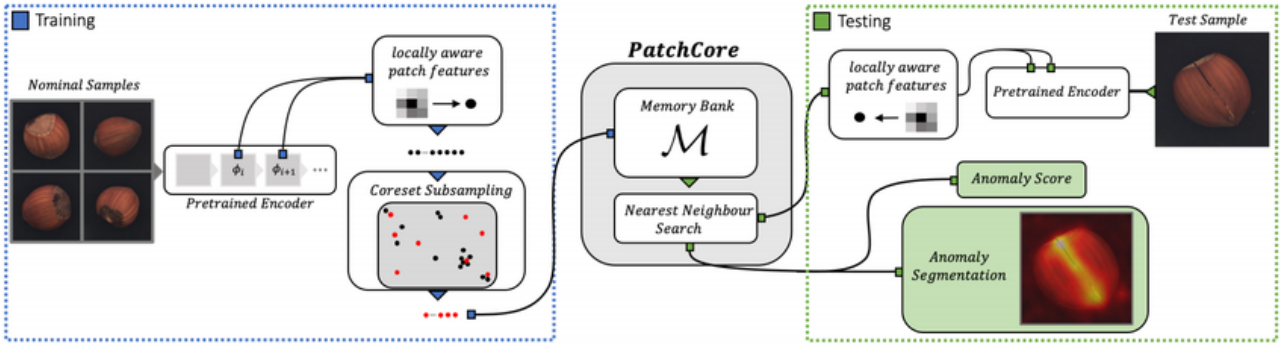


Figure 8 - Overall framework of PatchCore. During training, normal samples are decomposed into a memory bank of neighbourhood-aware patch-level features by Yajie Cui, Zhaoxiang Liu, Shiguo Lian [12]

2.7 – Benchmark sur le dataset MVTec

La phase exploratoire de notre projet nous a menés sur un benchmark préliminaire sur un dataset public nommé MVTec Anomaly Detection (MVTec AD)[13] afin d'évaluer les méthodes non supervisées. Ce dataset a été choisi car il est connu pour la détection d'anomalies visuelles en industrie. Il est composé de plus de 5000 images sur 15 catégories d'objets et textures, chaque catégorie comporte un ensemble de données d'entraînement (sans défauts) et un ensemble d'images avec plusieurs défauts et images dites saines (sans défaut). Les images sont réalistes et nous permettent de nous projeter dans le cœur de notre projet.

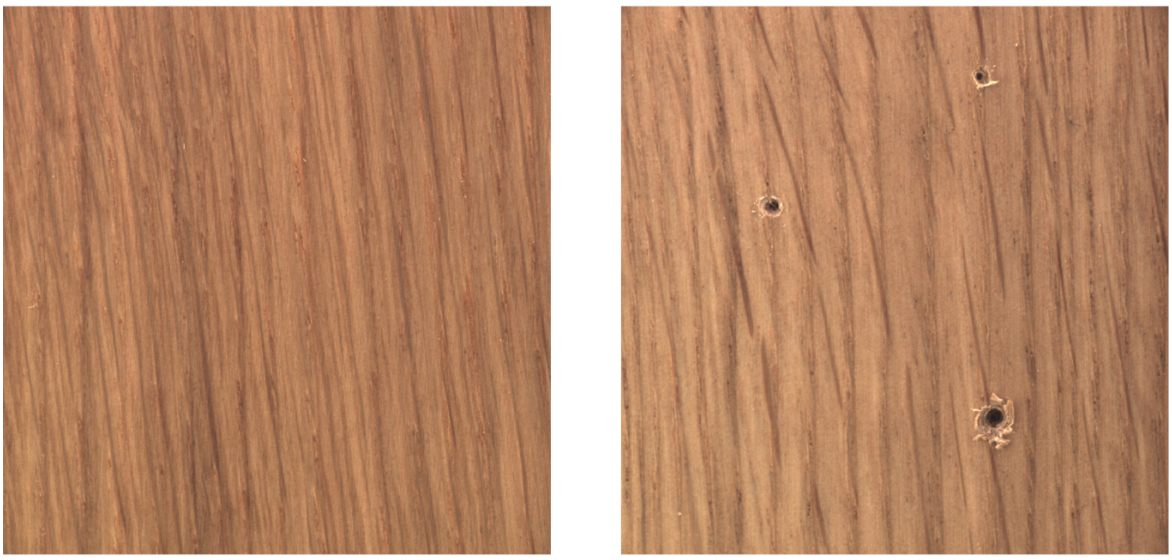


Figure 9 - Image prises du dataset MVtec Wood [13]. À **gauche** une image sans défaut à **droite** une image avec un défaut de type trou.

Nous avons pris pour référence le jeu d'images du bois pour ses caractéristiques de couleur unie, texture homogène et ses variations visuelles qui peuvent être comparées aux surfaces d'aluminium attendues sur notre ligne de production.

L'objectif de cette étude est de comparer les performances des approches explorées dans la partie État de l'art. Nous allons refaire ce benchmark sur les données collectées directement sur notre ligne de production.

Nous pouvons constater sur la Figure 10 que quatre méthodes sur six s'approchent et dépassent un AUROC de 90% ce qui montre la capacité à distinguer les défauts. D'autres métriques comme le F1 Score, Recall et précision sont notables, comme on peut le voir sur la Figure 11, PatchCore, One Class SVM et Kmeans se démarquent avec une bonne précision, rappel et vitesse d'inférence.

Les meilleurs candidats combinent haute performance de détection et faible latence d'inférence qui sont des critères essentiels pour notre intégration industrielle

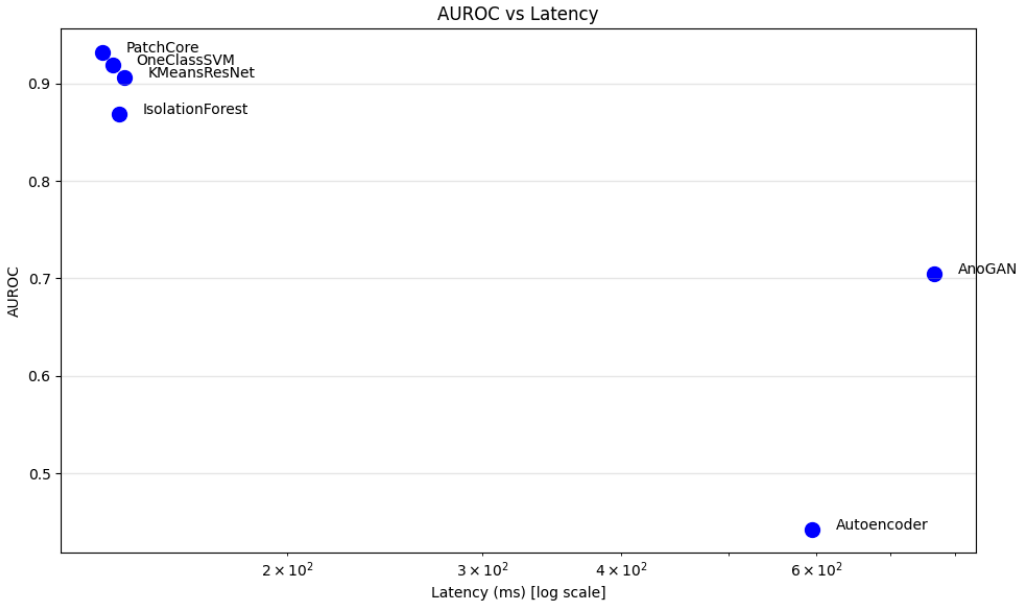


Figure 10 - AUROC benchmark MVTec

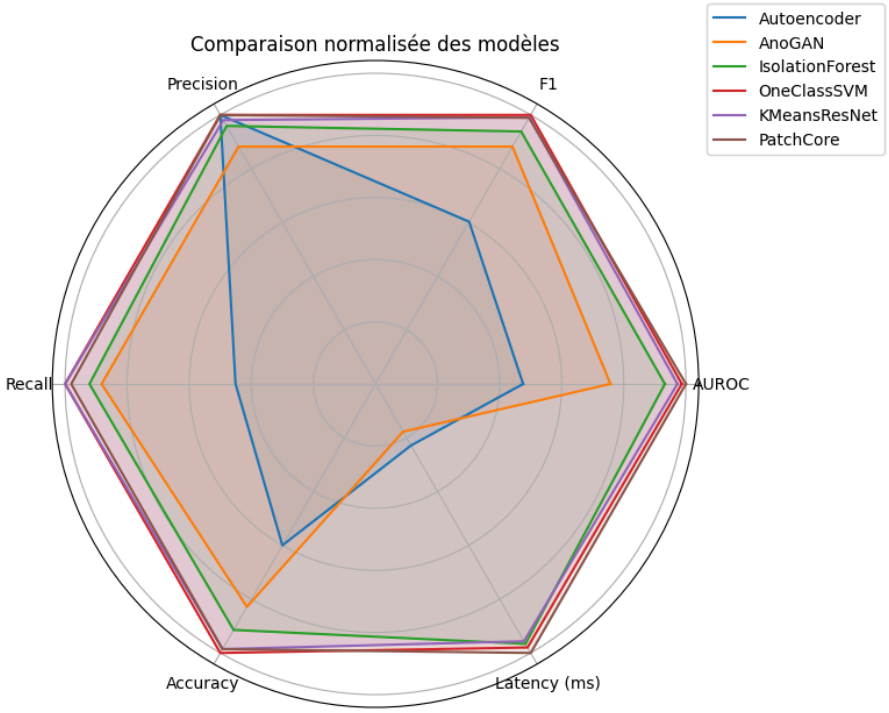


Figure 11 - Radar comparatif différentes méthodes

2.8 – Conclusion État de l'art

L'analyse de l'État de l'art nous a permis de parcourir et connaître les méthodes génératives (AutoEncodeur et AnogGAN) aux méthodes basées sur l'extraction de caractéristiques (Isolation Forest, One-Class SVM, K-means, PatchCore).

Nous pouvons voir que les performances sur le dataset MVtec AD sont de très bonne qualité. Ce dataset est de très bonne qualité et cela est démontré par les résultats.

Plutôt que choisir une méthode spécifique, nous avons opté pour une évaluation des ces algorithmes sur le dataset que nous obtiendrons de la ligne de production. Cette approche a pour idée de ne pas sous estimer des méthodes qui ont mal performé pour d'autres tâches qui sont différentes de la notre. Nous allons tester, toutes ces méthodes sur notre dataset pour identifier celui qui performe le mieux.

Chapitre 3 – Méthodologie

Dans ce chapitre nous allons explorer la méthodologie que nous avons utilisée pour couvrir les objectifs de ce projet. Nous allons dans un premier temps présenter l'architecture générale du système, depuis le choix des matériels jusqu'à leur intégration physique sur la ligne de production. Nous détaillerons ensuite le pipeline logiciel depuis l'acquisition d'image jusqu'au traitement de ces dernières.

3.1 – Analyse du problème

Mettre en œuvre un système de détection de défauts automatisé sur une ligne de production industrielle fonctionnant 24 heures sur 24, six jours sur sept, présente plusieurs défis. Contrairement à un laboratoire contrôlé, les conditions réelles sont bien différentes.

L'environnement de la ligne de production est poussiéreux et comporte souvent des particules d'aluminium (résidus de découpe). Ces particules peuvent s'accumuler sur les objectifs des caméras, dégradant la qualité des images au fil du temps.

L'un des principaux défis est le positionnement physique du système de vision (matériel). Comme l'objectif est d'inspecter la face inférieure des profilés, le système doit être installé dans un endroit étroit et difficile d'accès.

3.1.1 – Contraintes et exigences

Le projet doit respecter plusieurs contraintes imposées par l'environnement industriel et du contexte du travail de bachelor :

Contraintes budgétaires :
- Budget : à partir de 500 CHF demande la validation du Responsable de filière

Contraintes environnementales :
- Environnement poussiéreux avec copeaux d'aluminium
- Exposition à l'eau, huile et projections
- Espace d'installation restreint sous la ligne de production
- Fonctionnement 24h/24, 6j/7

Contraintes techniques :
- Détection en temps réel (vitesse 0.7 m/s)
- Faible luminosité ambiante
- Connectivité réseau limitée (proxy industriel)

3.1.2 – Étude comparative et choix de l'architecture

Alternatives considérées :

Micro-ordinateurs embarqués :
- Raspberry Pi 5 (156 CHF) : Bon rapport qualité/prix, GPIO disponibles
- NVIDIA Jetson (500+ CHF) : Plus puissant mais dépasse le budget
- PC industriel (1000+ CHF) : Hors budget

Caméras :
- Caméra USB étanche (74 CHF) : Compatible budget, IP67
- Caméra industrielle (500+ CHF) : Hors budget pour cette phase

Tous les composants doivent être logés dans un **boîtier étanche, résistant à l'eau, à l'huile et aux particules d'aluminium**.

En raison de la vitesse élevée des lignes d'extrusion, le système doit également être capable de faire de l'inférence en temps réel, sans ralentir la production.

Étant donné la faible luminosité ambiante sous la ligne, j'ai ajouté une bande LED gérée par le Raspberry Pi pour éclairer les images à capturer.

Ces défis physiques montrent bien la nécessité d'une solution compacte et demandant peu de maintenance.

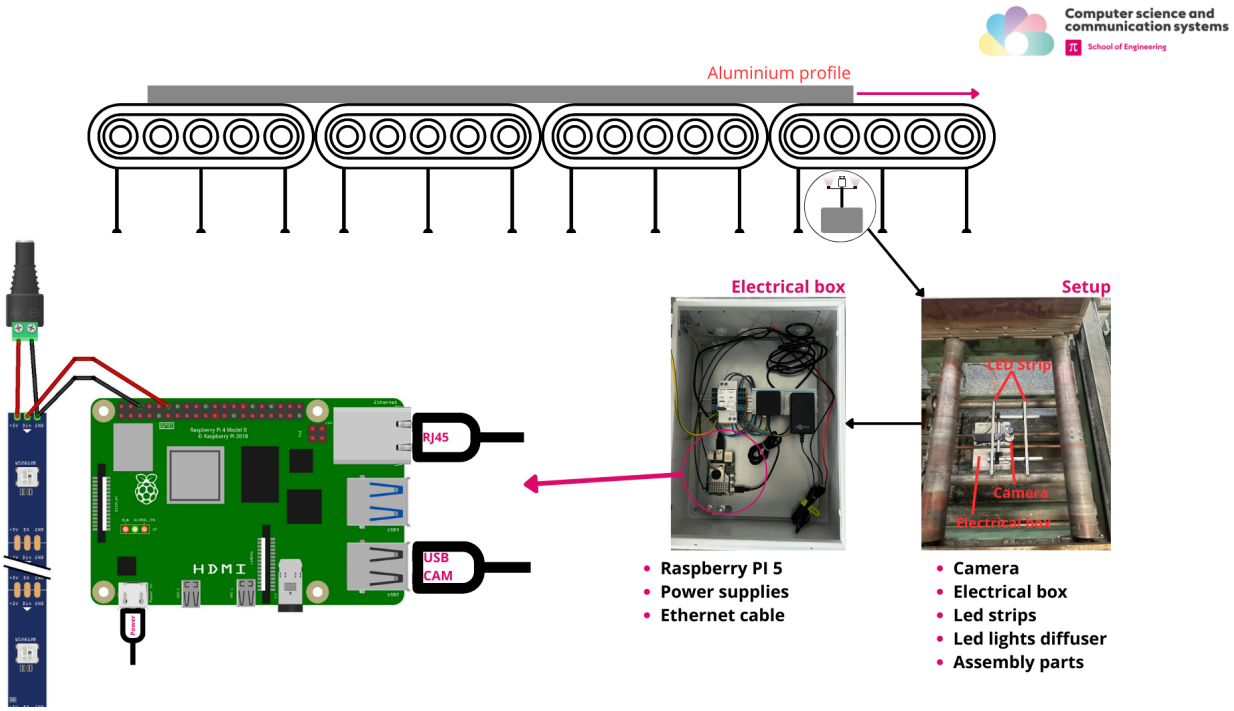


Figure 12 - Installation complète

3.2 – Architecture matérielle et intégration sur ligne de production

Le système que nous avons installé sur la ligne de production est composé des éléments suivants:

- **Micro-ordinateur embarqué :** Un **Raspberry Pi 5** (RPI5) 16Go sert d'ordinateur central pour contrôle de la caméra, de l'éclairage et exécution des différents codes. Notre choix s'est porté sur le RPI5 pour son bon rapport qualité/prix, ce qui nous a permis de l'installer dans un espace réduit également. Dessus nous pouvons connecter deux appareils USB 2.0 et deux appareils USB 3.0. Un autre point positif de ce RPI est d'avoir des ports **General Purpose Input/Output** (GPIO) que nous avons utilisés pour piloter notre bande Led.

Figure 13 - Raspberry Pi Foundation, « Raspberry Pi 5 – Hero Image, » Pi-Shop.ch.[Online]

[14]

- **Caméra industrielle**: Nous avons pu nous procurer une caméra compatible les RPI qui est haute résolution (1920x1080 pixels). Cette caméra nous avons orienté vers le haut pour capturer la face inférieure de nos profilés. Ce modèle a également été retenu pour son étanchéité (**IP67** qui veut dire complètement

étanche à la poussière et peut résister à une immersion dans l'eau douce jusqu'à 1 mètre pendant 30 minutes) qui correspond véritablement au besoin de protection de l'emplacement où nous installons notre système. Ce modèle est un dit low light qui est une contrainte que nous avons et essayons de corriger avec la LED pilotées par le RPI.

Ce dispositif s'est avéré suffisant pour la phase de test (proof-of-concept). Toutefois cette caméra est dotée d'un obturateur rolling shutter, ce qui provoque un flou de mouvement visible à haute vitesse. Nous devons éviter ce flou dans le contexte d'une production ou les profilés parcourent 17.5 m en 25 seconds ce qui correspond à une vitesse de 0,7 mètre par seconde.

Pour installation d'un système d'inférence en temps réel, une caméra plus avancée industrielle devrait être envisagée. Nous traitons cela dans le chapitre **Discussion**.

Cette caméra présente les caractéristiques principales suivantes :

- **Capteur:** Sony IMX291, CMOS couleur, 1/2.8″
- **Obturateur:** rolling shutter, ce qui induit flou de mouvement à haute vitesse
- **Alimentation & consommation :** 5 V DC USB
- **Objectif & FOV :** objectif fixe monté sur M12, ouverture f/2, diagonale 120°
  - **Vérification des objectifs**: L'objectif de la caméra doit pouvoir capturer la plaque au complet, afin de ce faire, nous devons calculer l'angle nécessaire pour la capturer connaissant:
  - Distance caméra à la plaque: ~38cm
  - Zone à capturer: Horizontal 75cm, Vertical 37cm

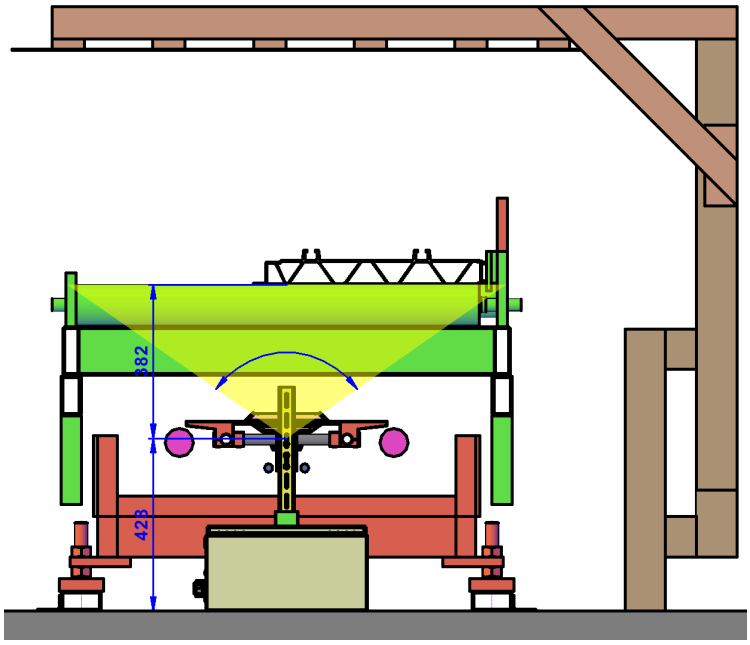


Figure 14 - Angle horizontal nécessaire , dessiné par Raoul Rey (Constellium)

$$\text{Angle nécessaire horizontal} = 2 * \arctan\left(\frac{\frac{75}{2}}{38}\right) = 2 * \arctan\left(\frac{32.5}{38}\right) \approx 89°$$

$$\text{Angle nécessaire vertical} = 2 * \arctan\left(\frac{\frac{37}{2}}{38}\right) = 2 * \arctan\left(\frac{18.5}{38}\right) \approx 52°$$

Figure 15 - UCTronics/Arducam, « 2 MP IMX291 low-light wide-angle USB camera module with waterproof metal case, » Pi-Shop.ch. [Online].[15]

- **Éclairage LED :** Afin d'avoir une solution pour notre problème de faible luminosité, nous avons deux bandes LED de type WS2812B. Sur ces bandes LEDs nous avons ajouté un matériau plastique qui permet que la lumière se diffuse et ne fasse pas des points lumineux blancs sur l'aluminium. Ces bandes LEDs sont pilotables via le RPI via le port GPIO 18. Afin de dimensionner l'alimentation que ces LEDs pouvaient consommer nous avons dû calculer le courant que ces dernières demandent. Nous savons que chaque LED consomme 0.3W à pleine luminosité. Nos 2 bandes réunies nous comptent 58 LEDs. La tension d'alimentation de nos bandes LEDs est de 5V.

  - $\text{Puissance totale} = 58 * 0.3W = 17,4W$

  D'ici nous pouvons calculer le courant consommé par notre bande LEDs à pleine luminosité.

  - $\text{Courant} = \frac{\text{Puissance}}{\text{Voltage}} = \frac{17,4}{5V} = 3,48A$

  Nous prévoyons une marge de sécurité d'environ 20% donc une alimentation 5V, 4A.

  Ayant une alimentation à 1A à disposition, nous l'avons utilisée pour ce prototype

  Nous préconisions une alimentation de 4A pour l'installation à utiliser en cas de poursuite de ce projet.

- **Coffret électrique et connectique :** L'ensemble (RPI, caméra, bande LEDs) est installé dans un coffret Rittal AX 1034.000 mis à disposition par Constellium.

  À l'intérieur de ce coffret, nous avons ajouté : deux prises 230V: la 1ère pour notre RPI, la 2ème pour notre transformateur 230V-5V pour alimenter les bandes LEDs.

  Tous les câbles passent également à l'intérieur de ce boîtier, comme le montre le schéma suivant. Nous avons toutes les sorties de câbles sont équipées avec des presse-étoupes afin de garantir l'étanchéité de l'installation.

Cette installation est compacte, modifiable et peut être installée sur la ligne de production. Le choix d'un ordinateur plus puissant de style Jetson pourrait être étudié.

Sur le tableau suivant nous pouvons voir les dépenses avec des valeurs réelles et approximatifs pour l'installation complète:

Matériel	**Coût (CHF)**
SanDisk Extreme PRO UHS-I V30 256G	29.70
Raspberry Pi 5	156.50
Camera USB étanche	73.80
Disjoncteur + 2 prises Hager T13 10A/250V	158.20
Câbles, connecteurs, visserie(approx.), LEDs	~30.00

Matériel	Coût (CHF)
Main d'œuvre estimée (3h à 120 CHF/h)	~360.00
Boîtier Rittal AX 1034.000	~105.00
Total estimé TTC	**~913.20**

Comme on peut voir pour une installation de type prototype on arrive vite à 1'000 CHF. En revanche d'un côté industriel ces montants sont raisonnable car une caméra industriel comme conseillé pour ce type de système coûterait plus cher que 1'000 CHF.

L'installation nécessite deux connexions externes :

1. Une alimentation en 230V pour fournir l'énergie aux composants (Raspberry Pi, LEDs, etc.)
2. Une connexion réseau via câble RJ45, permettant l'accès à distance au Raspberry Pi.

3.3 – Prototype v1

Notre système physique a pris forme, nous pouvons voir notre caméra ainsi que les deux bandes LEDs et le coffret électrique.

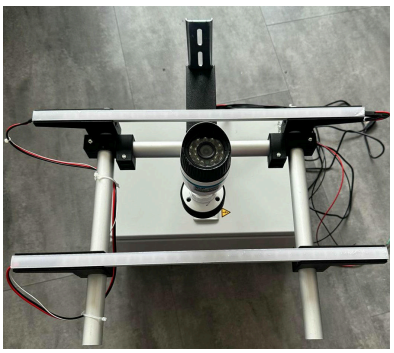


Figure 16 - Prototype installé sur la ligne de production

À l'intérieur de notre coffret électrique, nous retrouvons les composants (RPI, Transformateur 230-5V, prises électriques) ainsi que le câblage de notre système

Figure 17 - Intérieur de notre coffre électrique

3.4 – Prototype v2

Lors de la première semaine de fonctionnement, nous avons aperçu deux problèmes à régler:

1. Nous avions des copeaux provenant de la découpe qui se mettait devant la caméra ce qui dégradait la qualité des images prises. Afin d'éviter au maximum la chute de copeaux sur la lentille de la caméra,

nous avons prévu un déflecteur imprimé en 3D (pièce en rose clair ci-dessous). La pièce en rose foncé est la caméra, comme on peut le voir ce déflecteur va dévier une partie des copeaux qui vont essayer de s'introduire sur l'objectif de la caméra.

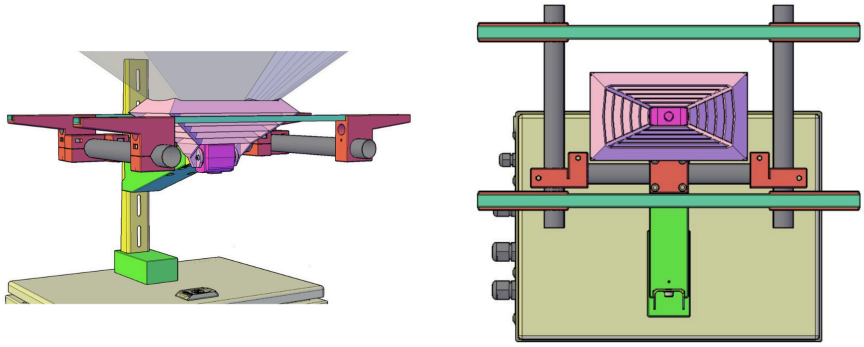


Figure 18 - Déflecteur, dessiné par Raoul Rey (Constellium)

Ici nous avons un exemple d'image avec un copeau devant :

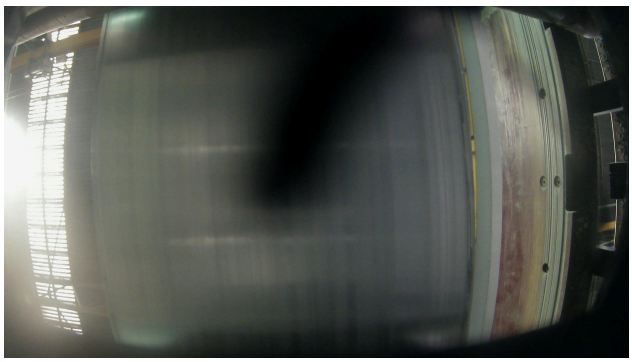


Figure 19 - Photo avec copeau d'allumnium

2. Un plafonnier LED très puissant qui permet d'éclairer la halle de production, crée une tâche blanche sur nos images sur les profilés moins larges. Comme on peut le voir sur l'image suivante

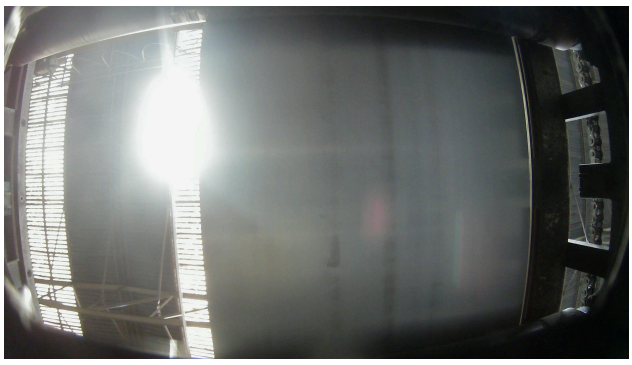


Figure 20 - Image sans cache

Nous avons essayé d'augmenter la luminosité de nos LEDs afin d'accroître le contraste ainsi que modifier les paramètres de la caméra. Nous n'avons pas réussi à avoir des résultats concluants. Quand nous effectuions le prétraitement nous obtenions les résultats suivants:

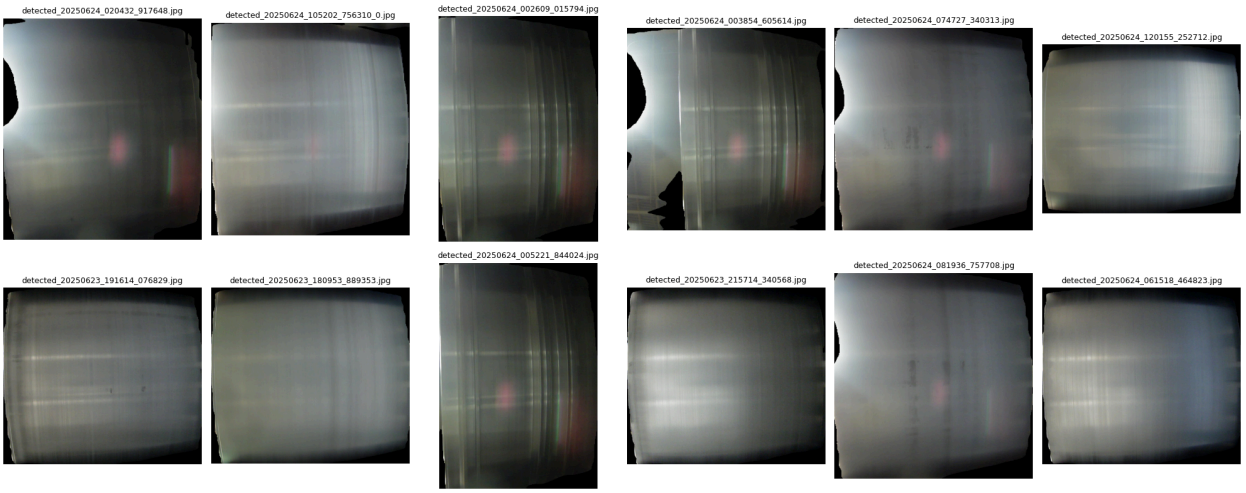


Figure 21 - Images pré traitées avant l'installation du cache

Afin de résoudre ce problème, nous avons installé un cache.

Initialement, ce cache devait être installé à 150 mm des rouleaux d'entraînement des profilés, mais le jour de l'installation, le responsable de production nous a informés que cela allait être impossible et qu'il fallait le remonter à 700mm (voir cote sur la Figure 22) des rouleaux. Nous avons donc pris notre matériel (bois, vis, etc) et remonté notre système de 550 mm.

Ce cache se fixe sur une partie rigide de notre ligne de production et empêche la lumière de se diriger sur notre caméra comme le montre la Figure 22:

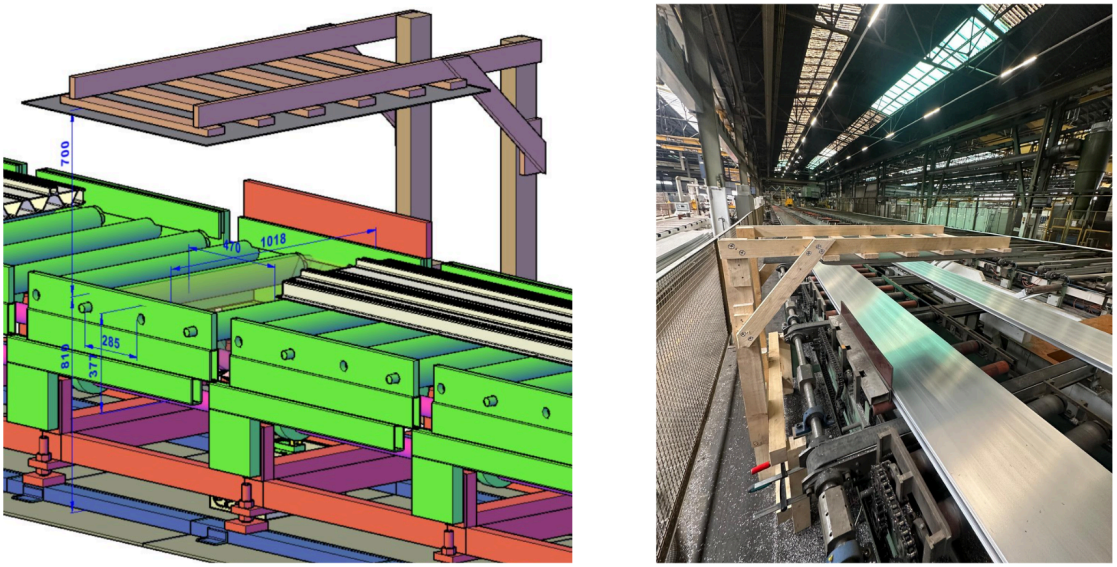


Figure 22 - Cache, schéma 3D exécuté par Raoul Rey (Constellium)

Après installation de ce cache, nous obtenons des images de meilleure qualité:

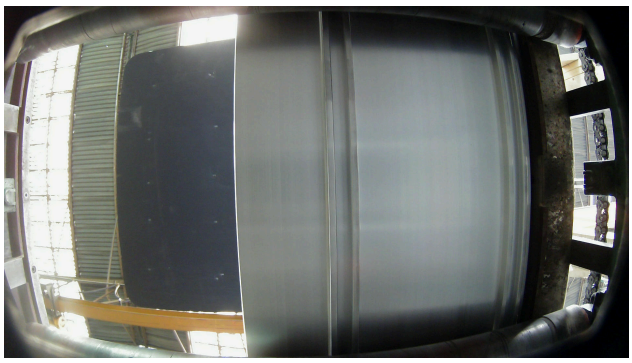


Figure 23 - Prise d'image avec le cache

Nous pouvons également constater que notre modèle de segmentation d'image performe beaucoup mieux.

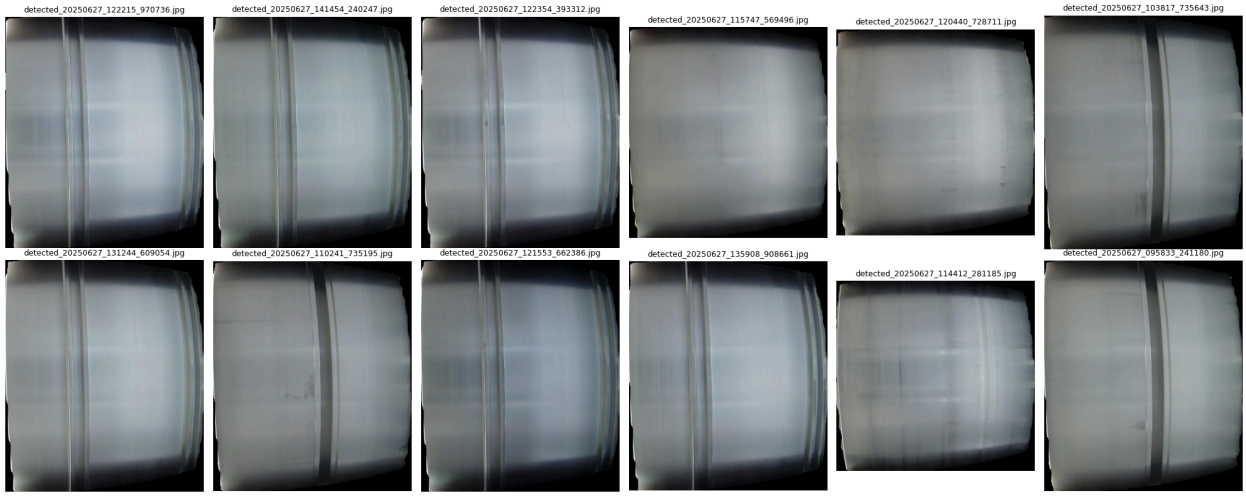


Figure 24 - Segmentation après installation du cache

3.5 – Système d'acquisition d'image

Nous avons développé plusieurs containers via docker afin de garantir une installation simple sur le RPI et une isolation des dépendances. Le but étant de pouvoir facilement les déployer sur un nouvel appareil si nécessaire.

Nous avons divisé en 3 containers principaux:

1. **Service de capture multi caméra :** Un service docker qui s'occupe de détecter les caméras connectées directement à notre RPI. La cadence est paramétrable via une variable CAPTURE_INTERVAL qui est en secondes. Nous avons effectué le code en sorte d'avoir plusieurs caméras car l'idée de base était d'avoir plusieurs caméras afin de capturer plusieurs angles. Afin de découpler les tâches et de ne pas bloquer la pipeline, nous avons décidé d'avoir 3 threads: le 1er s'occupe de capturer les images en continu, le 2ème s'occupe de détecter si une plaque est présente et si l'image est nette, dans ce cas il enregistre l'image, le 3ème s'occupe de sauvegarder la dernière image qui est pour l'afficher sur notre front.
2. **Service de contrôle des LEDs et sauvegarde :** Ce service utilise un backend implémenté en python à l'aide de Flask ainsi que les commandes pour gérer les bandes LEDs et le transfert les images. À partir d'un API REST, nous pouvons modifier la couleur et intensité des LEDs (cette API est utilisé via l'interface utilisateur Web ci-après). Ce service s'occupe aussi de vérifier l'espace disponible sur le RPI si un certain seuil est dépassé (ex: 40%), ce dernier envoie les images sur un serveur distant via le protocole SFTP. Ceci permet d'assurer que notre système fonctionne en continu sans saturer le stockage local. Un script planifié qui pilote ces transferts. La configuration du serveur distant (adresse, clés SSH, répertoire cible) est définie dans un fichier d'environnement `.env.example`.

3. **Interface utilisateur Web :** L'idée est d'avoir un aperçu rapide du système ainsi que permette dans le futur aux opérateurs ajuster les paramètre nécessaires, une interface web légère a été intégrée au système. Cette interface est accessible via l'adresse `://<IP_RPI>:3000`, cette interface affiche en temps réel la dernière image capturée par la caméra et permet d'allumer/éteindre les LEDs ainsi que ajuster la luminosité. Elle permet de déclencher manuellement une sauvegarde des images et de les envoyer sur le serveur distant.

   Pour accéder à Internet dans l'environnement industriel, un proxy de contournement (proxy bypass) a été mis en place.

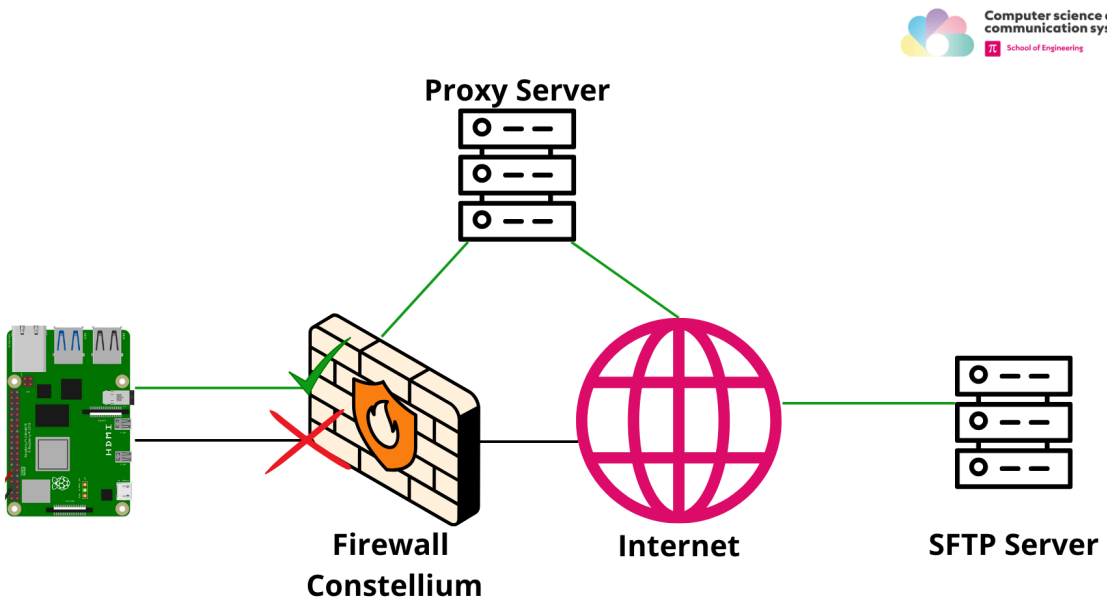


Figure 25 - Architecture réseau simplifiée

Vu que nous utilisons un proxy comme le montre la Figure 25 nous ne pouvons pas établir les connexion SSH classique utilisé pour notre transfert SFTP. Afin de contourner cette limitation, nous avons utilisé l'outil corkscrew [16], qui permet de tunneler une connexion SSH à travers un proxy HTTP. *Ceci a été un outil qui nous a été très utile !*

3.6 – Prétraitement et nettoyage des images

Les images brutes capturées sur la ligne de production sont difficiles à traiter, nous avons des pièces mécaniques tout autours de notre caméra qui capte ces dernières, des machines. Comme mentionné, certaines de nos images sont floues en raison des mouvement des profilés et des machines. Un prétraitement est crucial pour éliminer les images de mauvaise qualité pour ensuite passer à la partie détection de défauts.

Dans notre module *Data Cleaning*, nous supprimons les images qui sont floues et créons un hash pour chaque image afin de supprimer des doublons qui peuvent se créer quand la plaque est à l'arrêt par exemple.

Dans un deuxième temps, nous nous occupons d'utiliser un modèle de segmentation pour détecter la surface du profilé d'aluminium sur nos images brutes. Ce modèle segmente les plaques d'aluminium sur une image. Nous avons utilisés 385 images sur lesquelles avons segmentées nous mêmes les plaques. Ces annotations ont été réalisés à l'aide de la plateforme Roboflow [17] qui permet d'entraîner facilement un modèle de segmentation et qui m'a permis de récupérer du temps. Le modèle a atteint de très bonnes performances sur notre set de test avec les métriques suivantes :

- **mAP(mean Average Precision) à un seil IoU (Intersection over Union)@50 : 99.1 %** : IoU mesure à quel point le masque prédit par le modèle coïncide avec le masque réel. Un seuil de 50% est fixé si 50% du masque est prédit correctement nous considérons une bonne prédiction.
- **Précision: 95.7 %** : Ceci calcule de manière standard, taux de détection correctes parmis toutes les détection.

$$\text{Précision} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux positifs}}$$

- **Recall (Rappel) : 100 %** également pour cette métrique, capacité du modèle à trouver toutes les vrais pièces présentes dans les images.

$$\text{Rappel} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux négatifs}}$$

Dans le workflow de ce modèle nous détectons si l'image contient un profilé et la rognons en conséquence, comme on peut le voir sur l'image suivante :

Figure 26 - Segmentation Roboflow

Toutes les images segmentées sont ensuite enregistrées. Nous avons essayé des approches plus simples comme filtrage, masque, plus grande objet dans les tons gris mais aucune n'a donné des résultats satisfaisants car le gris et les ombres sont confondues.

3.7 – Génération de données synthétiques de défauts

Afin d'avoir des images avec des défauts réels, nous avons mis en place un processus d'ajout artificiel de défauts sur des images sans défauts.

Nous avons reçu quelques images de la part du responsable qualité de Constellium. Souvent ces défauts ont été photographiés à des fins d'information et non dans l'idée d'un post traitement pour nos fins.

L'utilisation de ces images

Figure 27 - Défaut sur un plaque d'aluminium

Nous avons récupéré les meilleures images et les avons rognées manuellement l'image afin d'avoir le défaut en évidence.

Dans un deuxième temps, nous avons utilisé Rembg qui est un outil python pour enlever les arrières plans sur les images. Nous pouvons voir sur l'image suivant les 7 défauts que nous avons retenus et enlevé l'arrière plan.

Figure 28 - Défaut après traitement

Dans l'idée, nous avons utilisé un tweaker, créé par un LLM, ChatGPT (prompt: « Write me a tewaker to tweak defects on a base image. » + code utilisé pour lire et visualiser les défauts). Ce tweaker, nous permet de faire en sorte que le défaut paraisse le plus réaliste possible. Ce dernier sauvegarde les paramètres en JSON, qui sont ensuite réutilisés lors de la génération des défauts. Nous effectuons les transformations telles que la taille, la rotation, le flou autour du défaut afin d'atténuer son incrustation sur les profilés, le contraste ou la brillance.

L'idée est de choisir une position plausible. Pour cela, on cherche une position aléatoire et on vérifie que nous ne sommes pas trop près des bords ou sur des zones noires qui sont l'arrière plan.

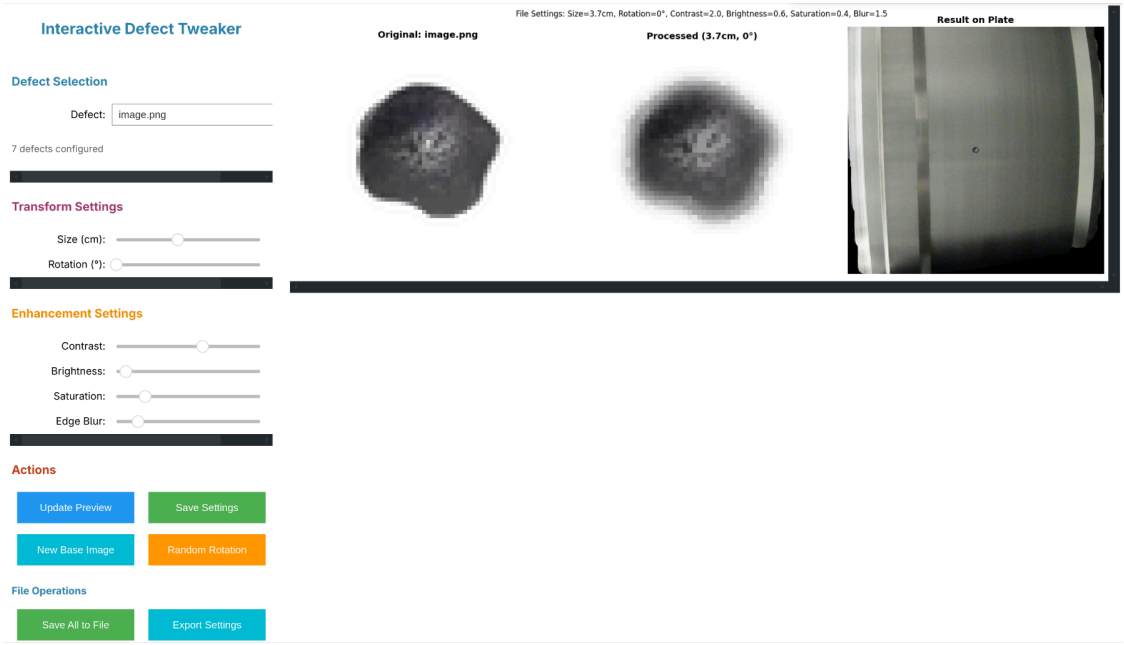


Figure 29 - Tweaker pour les défauts

Afin d'avoir un avis professionnel, nous avons généré tous les défauts et envoyé au responsable qualité qui nous a confirmé que nos défauts s'approchaient de la réalité du terrain.

Sur la rangée du haut, nous avons les images sans les défauts et sur la rangée du bas les images avec le défaut généré de manière synthétiques.

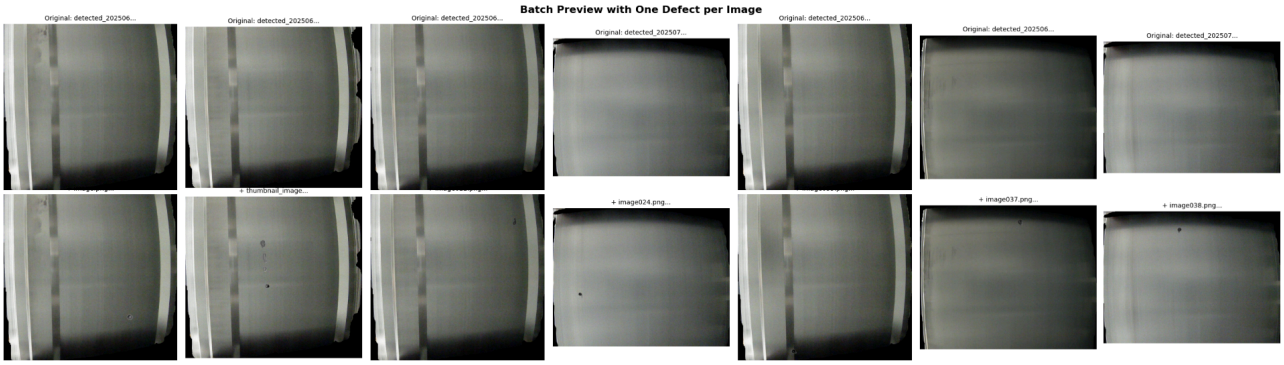


Figure 30 - Création de défaut sur les profilés

Nous utiliserons 20% de notre dataset pour validation de nos méthodes non supervisés. De ces 20%, nous allons générer 60% avec des défauts synthétique et 40% sans défauts.

Ce qui nous fait : **100'288** images d'entraînement et 20'058 en validation dont **12'071** avec défaut et **7'987** sans défaut.

3.8 – Détection non supervisé de défauts (anomaly detection)

La détection automatique de défauts est la deuxième grande tâche de notre travail de bachelor. En utilisant une stratégie non supervisée, nous avons entraîné et évalué plusieurs méthodes qui dont l'explication se trouve dans la partie « État de l'art ». L'objectif du système est que, pour chaque image, le système puisse déterminer si un défaut est présent ou non, sans les avoir vus au préalable. On va apprendre la normalité à notre modèle.

Toutes les méthodes ont utilisé le même set de données pour une avoir une parité au niveau de nos résultats.

Les métriques que nous avons utilisés sont l'AUROC (aire sous la courbe ROC), la précision, le rappel (recall), le score F1, l'exactitude équilibrée (Balanced Accuracy), ainsi que le temps d'inférence moyen par image (Latency). Un seuil optimal est calculé.

Chaque métrique a sa pertinence:

- Précision (Precision) : Proportion de prédiction défauts qui était réellement des défauts. Elle indique **la fiabilité**. Si on a une faible précision, on aura beaucoup de fausses alertes.

  $$\text{Précision} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux positifs}}$$

- Rappel (recall) : Proportion de vrai défauts qui ont été détectés correctement . Elle mesure la capacité à **ne pas rater des défauts**.

  $$\text{Rappel} = \frac{\text{Vrai positifs}}{\text{Vrai positifs+Faux négatifs}}$$

- Score F1: moyenne harmonique de la précision et du rappel. Compromis entre **la précision et le rappel.**

  $$\text{Score F1} = 2 * \frac{\text{Précision*Rappel}}{\text{Précision+Rappel}}$$

- Balanced Accuracy : Contrairement à l'accuracy, cette métrique va donner le même poids aux deux classes (défaut et normal). Ceci est intéressant si on a un un dataset déséquilibré.

  $$\text{Balanced Accuracy} = \frac{1}{2} * (\frac{\text{Vrai positifs}}{\text{Vrai positifs + Faux négatifs}} + \left(\frac{\text{Vrai négatifs}}{\text{Vrai négatifs + Faux positifs}}\right)$$

- AUROC (Area Under ROC curve) : aire sous la courbe ROC qui permet de voir la capacité de classement du modèle.

  $$\text{AUROC} = \int_0^1 \text{TPR}(\text{FPR}) d\text{FPR}.$$

  Utilisant la méthode des trapèzes selon la librairie python.

$$\text{TPR} = \text{True Positive Rate} = \text{Recall}$$

$$\text{FPR} = \text{False Positive Rate} = \frac{\text{Faux positifs}}{\text{Faux positifs} + \text{Vrai négatifs}}$$

Afin de mieux expliquer le fonctionnement de l'AUROC, on peut le comprendre comme suit: Imaginons, que l'on tire au hasard une plaque défectueuse et une plaque normal et on vérifie leurs scores.

- Si le modèle donne un score plus haut à la défectueuse il gagne
- L'AUROC c'est la proportion de fois où le modèle « gagne » à ce jeu.

La courbe ROC, trace le taux de vrais positifs (True Positive Rate) en fonction du taux de faux positifs (False Positive Rate) pour différents seuils de décision. L'aire sous cette courbe (entre 0 et 1) est ensuite mesurée.

AAfin de trouver le seuil optimal, on teste plusieurs valeurs et on retient celle qui maximise le F1-score. Ceci permet d'améliorer la qualité de notre décision binaire.

3.9 – Implémentation de PatchCore dans le système

Nous avons pu tester les 6 différentes méthodes Autoencoder, AnoGAN, Isolation Forest, OnceClass SVM, Kmeans Resnet et PatchCore, étant donnée que **PatchCore** a mieux performé que les autres, nous allons passer en détail le fonctionnement ce sa pipeline.

Nous avons utilisé la pipeline suivante pour effectuer l'entraînement (création de la mémoire) et l'inférence (calcul du score d'anomalie)

```
# Pseudocode PatchCore
model = ResNet18_Backbone(pretrained=True)
memory_bank = []

# "Training"
for image in training_images:
    feats = model(image)              # tenseur de caractéristiques CxHxW
    patches = reshape(feats)          # liste de vecteurs 512-D, de taille H*W
    patches_proj = random_proj(patches)   # projection aléatoire pour réduire la
dimension
    memory_bank.extend(patches_proj)

# Sélection du coreset (on garde 1% des patchs aléatoirement) :
memory_bank = select_random_subset(memory_bank, ratio=0.01)

# Inférence (pour une image de test) :
feats = model(test_image)             # CxHxW
patches = reshape(feats)             # vecteurs de patch de l'image test
patches_proj = random_proj(patches)
distances = []
for p in patches_proj:
    # distance au plus proche voisin dans la banque mémoire
    d = min(distance(p, mb) for mb in memory_bank)
    distances.append(d)
score = max(distances)               # plus grande distance = anomalie la plus marquée
```

Listing 1 - Code PatchCore sur 5.BenchmarkData/patchcore.py

Chaque image est d'abord redimensionnée et normalisée, ensuite nous la traitons à l'aide d'une architecture ResNet18 tronqué, qui extrait une carte de caractéristiques de (512x8x8) dans notre cas.

Comme nous le voyons dans le code ci-dessus, nous utilisons une projection aléatoire (Sparse Random Projection de sklearn) pour réduire la dimension des vecteurs de 512 à environ 300 ce qui accélère le calcul des distances tout en conservant essentiel de l'information.

Pour la banque mémoire, nous sélectionnons aléatoirement 1% des vecteur de patch extraits sur l'ensemble des données d'entraînement. Ce prélèvement aléatoire, s'avère suffisante en raison de la similitude des patch voisins, surtout sur notre type d'images qui a énormément de zones lisses.

Quand nous inférons, on extrait les patch de chaque image de test, on projette puis on calcule la distance au plus proche patch de la banque mémoire. Cette recherche est effectuée avec la bibliothèque **FAISS (Facebook AI Similarity Search)** qui est optimisée pour la recherche du voisin le plus proche. Le score se calcule ensuite par la plus grande de ces distances, c'est à dire du patch le plus anormal.

Ce score est ensuite comparé à un seuil pour décider si notre image est défectueuse ou pas.

Chapitre 4 – Résultats et analyse

Dans cette partie, nous allons analyser les résultats obtenus avec les différentes méthodes.

Nous observons que les méthodes reposant sur l'extraction de features sur des CNN pré entraînés (Isolation Forest, One-Class SVM, K-means, PatchCore) **performent mieux** que les méthodes entraînées de bout en bout (AnoGan et autoencodeur).

Modèle	AUROC	Latence (ms)	F1	Précision	Rappel	Acc. Équilibrée
Autoencoder	0.5	445.1	0.71	0.6	0.86	0.5
AnoGAN	0.5	240.1	0.73	0.6	0.91	0.51
IsolationForest	0.71	225.4	0.7	0.75	0.66	0.66
OneClassSVM	0.77	240.6	0.73	0.79	0.69	0.7
KMeansResNet	0.72	227.7	0.7	0.76	0.66	0.67
PatchCore	0.87	273.8	0.82	0.85	0.78	0.79

Figure 31 - Résultats benchmark avec données de la ligne de production

On peut voir que les méthodes entraînées de bout en bout performent de manière aléatoire. Par exemple l'autoencodeur et L'AnoGan n'obtiennent qu'un AUROC [18] d'environ 50%, ce qui nous montre l'incapacité de ces méthodes à distinguer réellement les défauts des normales.

Les approches classiques basées sur l'extraction de feautures ont obtenu des performances nettement meilleures comme par exemple Isolation Forest qui atteint ~70% d'AUROC, avec une précision ~74,6% et un rappel ~66,3% au seuil optimal, soit un F1 ≈ 70%. La One-Class SVM fait encore mieux (AUROC ~76%, F1 ~0,733, précision ~78,9%, rappel ~68,5%). Le clustering K-means sur features présente des résultats similaires (AUROC ~72%, F1 70%). Ces méthodes affichent des Balanced Accuracy de l'ordre de 66% à 70%, ce qui nous montre que nos méthodes d'extraction de features performent bien. On peut voir que la SVM une classe se démarque légèrement avec ~70% d'exactitude.

La meilleure méthode et celle qui se détache clairement de toute concurrence est **PatchCore**, en obtenant l'AUROC la plus élevée 87%, ce qui montre que cette méthode sépare bien les images défectueuses des images sans défauts. Son accuracy de 78,9 % est également supérieur, ce qui nous montre sa capacité à bien détecter les anomalies. PatchCore atteint un F1 de ~82% soit le plus élevé de nos benchmark qui est grâce à une précision de 85,3%.

PatchCore offre donc un gain notable en fiabilité de détection.

Notre graphique ci-dessous nous montre la métrique AUROC qui correspond l'aire en dessous de la ligne bleue qu'on essaie de maximer. La ligne grise nous montre le comportement d'une inférence aléatoire.

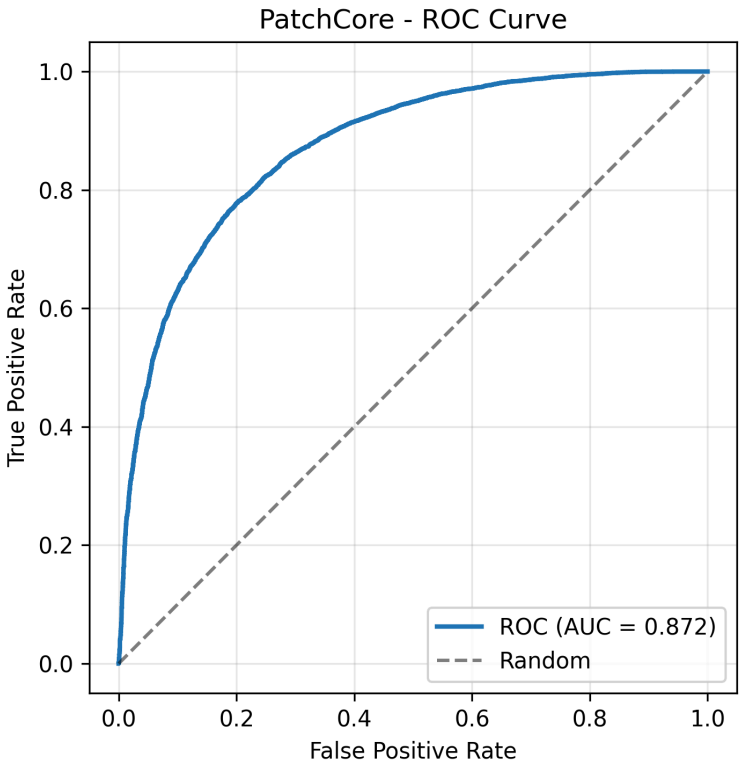


Figure 32 - ROC Curve PatchCore

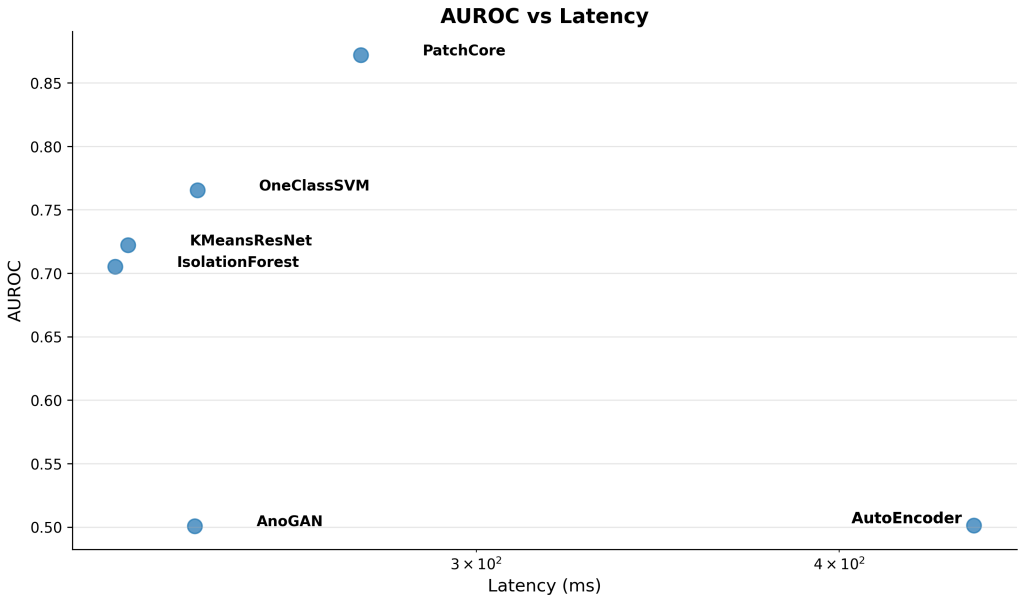


Figure 33 - AUROC vs latency

Le temps de calcul moyen montre que PatchCore malgré sa complexité reste dans des temps acceptables ~274 ms par image en moyenne sur le GPU de notre ordinateur. Ce n'est pas le plus rapide, mais reste du même ordre de grandeur que d'autres méthodes qui sont certes rapides mais ont de moins bonnes performances.

PatchCore demande peu de ressources, il peut traiter environ 3-4 images par seconde. Pour notre cadence cible de la ligne de production de ~5 fps, 274 ms par image est au dessus ce qui nous ferait avoir un décalage de 1 à 2 images. Ceci est ce qui est calculé sur notre ordinateur avec une carte graphique RTX 3050ti. Dans cette étude nous oublions que les images de test sont déjà segmentées lors du calcul des métriques de nos différentes modèles.

Nous avons lancé des tests préliminaires sur les RPI5 pour voir le temps que ce dernier met pour traiter les images.

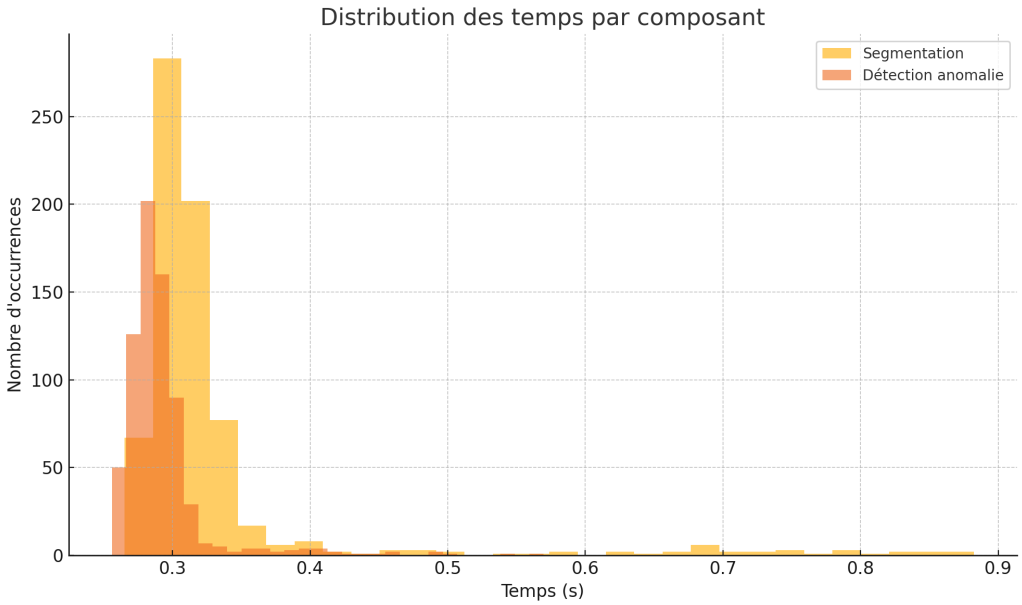


Figure 34 - Temps de segmentation + inférence

On peut voir que notre modèle prend facilement près de 600-700 ms pour segmenter et détecter l'anomalie. Ceci qui signifie qu'on aurait un retard plus conséquent sur notre ligne de production. Ceci pour une phase de prototypage pourrait aller car une fois le profilée débité un nouveau profilé doit être mise sur la table de transfert.

Avec PatchCore nous avons produit des heatmaps d'anomalie superposées ce qui nous permet de voir si le défaut se trouve bien ou notre algorithme le pense. La méthode pour le faire est la suivante :

On reconstruit une grille de patchs on chaque cellule prend la valeur de la distance du patch correspondant puis on interpole sur l'image originale.

Les zones de couleur rouge indiquent les régions de l'image qui ont le plus contribué au score d'anomalie (patch les plus éloignés)

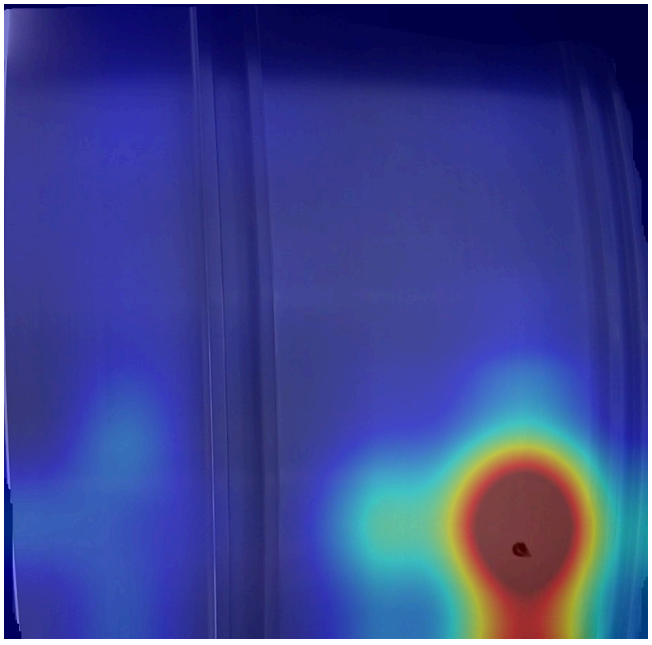


Figure 35 - Heatmap correcte

Chapitre 5 – Discussion

Dans cette section, nous revenons sur le déroulement concret du projet, depuis la planification initiale jusqu'à la mise en place sur la ligne de production. Nous analyserons les écarts entre ce que nous avions initialement prévu et la réalité du terrain comme les accès limités, les arrêts de presse ainsi que les leçons apprises. Pour finir, nous aborderons les limitations du système, les pistes d'amélioration et les considération concernant la durabilité de ce projet.

5.1 – Planning

Dans notre planning, nous avons sous estimé le travail de mettre le système en place. Dans notre cahier des charges certaines informations cruciales, telles que le fait que l'endroit n'est pas accessible sauf pendant les arrêts de presse, que le système doit être étanche. Cela nous a fait partir sur des pistes qui n'étaient pas alignées avec les besoins du système.

Pendant ce projet, nous avons eu plusieurs jours fériés donc plusieurs arrêts de presse n'ont pas eu lieu, ce qui nous a forcés à privilégier l'installation du système pendant le week-end. Dans notre planning initial, nous avions prévu finaliser le prototype durant la semaine 22 (26 mai au 01 juin), il s'est avéré que créer ce système a été plus compliqué, nous avons repoussé la **finalisation de notre prototype à la semaine 23 (2 au 8 juin)**.

L'installation sur la ligne de production a débuté la semaine **la semaine 23** et s'est terminé la semaine 26 (23 au 29 juin). Nous avons dû faire des modifications dans notre système que nous détaillerons sur la partie **Architecture matérielle et intégration sur ligne de production**.

Avec plus de recul, la création d'un prototype s'est révélée plus compliquée que nous l'imaginions.

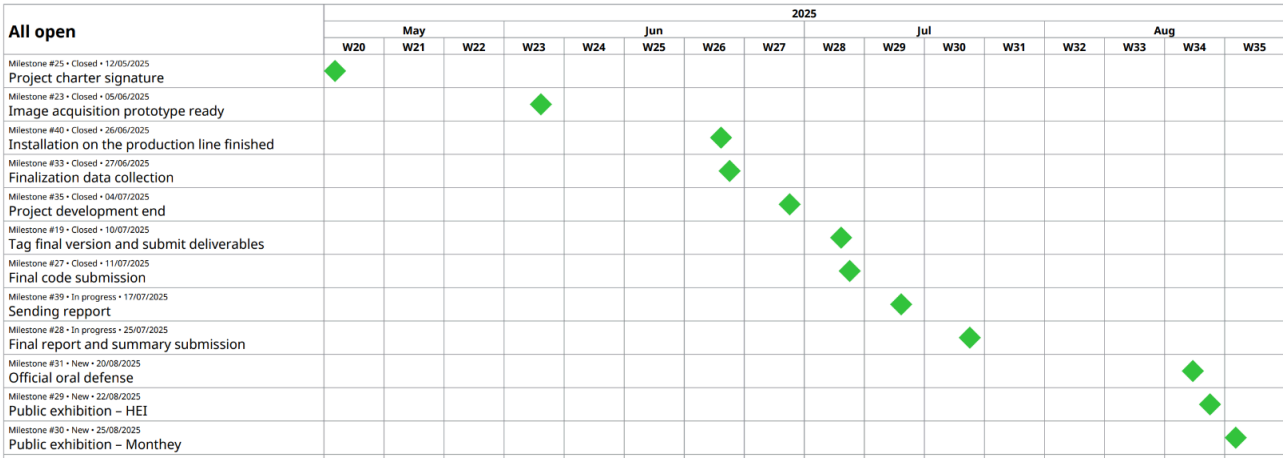


Figure 36 - Plan de nos milestones

Figure 37 - Plan des arrêt de presse

5.2 – Limitation actuelles du système

Malgré la réussite de la mise en place d'un prototype fonctionnel dans la ligne de production, le système présente plusieurs limitations qui affectent sa performance, robustesse et fiabilité sur le long terme.

1. **Temps d'inférence encore trop élevé** Les modèles pour l'analyse pour détection/segmentation de la plaque et pour la détection de défauts avec PatchCore reste relativement lente. Cela limite la capacité à suivre en temps réel.
2. **Le flou de mouvement lié à la caméra utilisée** La caméra sélectionnée, compatible avec notre budget et certaines de nos contraintes, est équipée d'un obturateur de type *rolling shutter*. Ce capteur provoque un flou directionnel quand les objets sont en mouvement rapide comme dans notre cas. Ce flou impacte la qualité de nos images et la précision.
3. **Sensibilité à l'environnement de production** Le système reste exposé aux copeaux de la découpe et à la poussière au niveau de l'objectif de la caméra. Le déflecteur mis en place a pu améliorer ce problème mais ne l'a pas totalement résolu, car des copeaux se sont déposés depuis cette installation.
4. **Absence d'intégration logicielle dans le système d'information de l'usine** Actuelle le système fonctionne en autonomie sans lien direct avec le système SAP de l'usine. Cela limite la traçabilité des données de production.

5.3 – Amélioration futures

5.3.1 – Vitesse d'inférence

Plusieurs pistes d'amélioration serait possible pour rendre le temps **d'inférence plus court**:

- **Alléger le modèle :** Utiliser un backbone plus petit comme par exemple MobileNet ou réduire la résolution des images, ce qui ferait aussi moins de patchs. À faire attention car cela pourrait aussi dégrader les performances actuelles de notre système.
- **Utiliser un hardware dédie :** Nous pourrions par exemple utiliser un NVIDIA Jetson qui permettrait les calculs de réseau neuronal plus rapides que le CPU que nous avons sur notre RPI.
- **Externaliser sur un serveur :** transmettre les images en temps réel à un serveur de traitement avec un GPU. L'inconvénient pourrait être la latence réseau
- **Créer un propre modèle de segmentation :** Nous avons utilisé Roboflow sur ce projet mais il serait possible d'utiliser **LabelStudio** pour faire un modèle plus rapide et qu'on peut contrôler plus simplement car nous avons utilisé la version gratuite où on peut pas récupérer le modèle.

5.3.2 – Caméra

Nous avons également remarqué un grand problème au niveau de la caméra actuelle. Nous avons utilisé un caméra low light, qui est optimisé pour une très faible luminosité mais compense cela par une exposition plus longue. Ceci nous a produit **des images floues lorsque la pièce est en déplacement**.

Concernant ce problème de **flou directionnel**, nous proposons :

- Installation d'une caméra industrielle avec type de capture **Global shutter** (ex: IDS UI-3280CP Rev. 2)

  Ce type de caméra permettrait de corriger le problème de flou directionnel

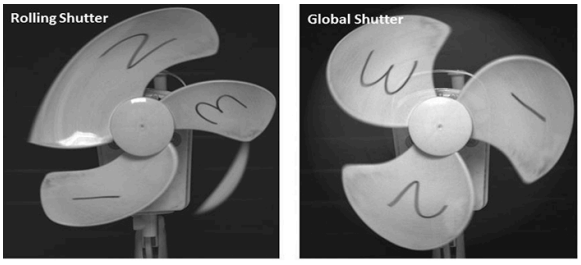


Figure 38 - Rolling Shutter vs Global Shutter [19]

- Protection IP67 comme demandé sur la ligne de production
- Alimentation PoE (Power over Ethernet) ou 12V
- L'objectif de la caméra doit avoir le bon angle pour pouvoir capturer la plaque au complet

5.3.3 – Déchets et environnement du système

Lors de notre utilisation, nous avons remarqué que les copeaux peuvent souvent se poser sur l'objectif de la caméra ou à des endroits où ils gênent la capture des images, nous recommandons de trouver une solution robuste et plus efficace que le déflecteur présenté dans Méthodologie de ce rapport.

5.3.4 – Intégration SAP

Pour la prochaine itération de ce projet, il serait intéressant de développer dans l'optique d'intégrer les informations sur la système SAP de l'usine. Ceci permettrai d'avoir le retour de détection du système intégré directement avec SAP.

5.4 – Considérations de durabilité du projet

Le développement de ce système de défection de défauts s'inscrit également dans une démarche de durabilité industrielle. Voici quelques points:

- **Réduction du gaspillage et optimisation des ressources:** En détectant les défauts, le système permet de réduire le nombre de pièces défectueuses expédiées aux clients ce qui diminue les coûts et les retour qualité. Sur le long terme une détection peut aider à l'amélioration du processus, par exemple, si l'on détecte plusieurs reprises les mêmes défauts on peut essayer de corréler les valeurs des différents capteurs de la machine afin de trouver la source du problème et trouver peut être une solution.
- **Efficacité énergétique**: Ce système s'inscrit dans la dynamique d'innovation et de l'industrie 4.0. en améliorant la qualité on augmente l'efficacité de tout le processus car on évite de devoir produire en urgence un nouvelle pièce. Notre système a une consommation électrique modeste comparée aux machine qui l'entourent.

Chapitre 6 – Conclusion

Ce travail de bachelor a permis de concevoir un prototype fonctionnel pour détecter automatiquement les défauts sur une ligne de production qui fonctionne 6 jours sur 7 et 24 heures sur 24. Le système montre des **bons résultats** sur les images où nous avons ajouté des défauts de manière synthétique avec l'utilisation de la méthode PatchCore pour la détection d'anomalies.

La mise en place du système a soulevé de nombreux défis importants:

- **Conditions industrielles :** la faible luminosité, les vibrations des machines, la chaleur de la halle, l'espace restreint pour installer le matériel et l'ajuster précisément pour la capture correcte de nos images. Installer un système dans une ligne de production déjà en fonctionnement implique de pouvoir garantir sa compatibilité avec les installation actuelles et de ne pas déranger son fonctionnement principal qui est la création de profilés en aluminium.
- **Contraintes de performance :** Le traitement d'image en temps réel, demande optimisation de modèle, du pipeline de traitement et la gestion des ressources du dispositif embarqué.
- **Manque de données réelles :** Le manque de données avec défauts, nous a malgré tout forcé à générer des anomalies de manière synthétique pour comprendre les performances de chaque méthode. Jusqu'à aujourd'hui, nous avons souvent travaillé avec des dataset publics, qui sont eux, souvent très propres et prêts à utilisation, dans ce projet tout le pipeline est assuré par nos soins ce qui est très compliqué.

Ce projet m'a permis de approfondir et développer de nombreuses compétences, en vision par ordinateur (traitement d'images d'industrie, détection d'anomalies non supervisé), en déploiement embarqué (RPI, scripts automatisés), mais aussi en grande partie la gestion de projet et communication avec les différents acteurs de la production industrielle chez Constellium.

Ce travail m'a appris à travailler dans un environnement industriel où on doit gérer des imprévus, construire de manière responsable et à bien réfléchir avant de prendre des décisions comme par exemple dans le choix de nos éléments hardware.

Bibliographie

[1] D. Gong *et al.*, « Memorizing Normality to Detect Anomaly: Memory-Augmented Deep Autoencoder for Unsupervised Anomaly Detection », in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. doi: arXiv:1904.02639.

[2] A. Singh et P. Reddy, « AnoGAN for Tabular Data: A Novel Approach to Anomaly Detection », in *Proc. 26th Int. Conf. Human–Computer Interaction (HCII 2024)*, 2024. doi: 10.48550/arXiv.2405.03075.

[3] R. T. Ionescu, S. Smeureanu, B. Alexe, et M. Popescu, « Unmasking the abnormal events in video », p. , 2017, doi: 10.48550/arXiv.1705.08182.

[4] F. T. Liu, K. M. Ting, et Z.-H. Zhou, « Isolation Forest », in *Proc. 8th IEEE Int. Conf. Data Mining (ICDM)*, 2008, p. 413-422. doi: 10.1109/ICDM.2008.17.

[5] V. Mounish, « A Comprehensive Guide For SVM One-Class Classifier For Anomaly Detection ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://www.analyticsvidhya.com/blog/2024/04/a-comprehensive-guide-for-svm-one-class-classifier-for-anomaly-detection/

[6] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, et P. Gehler, « Towards Total Recall in Industrial Anomaly Detection ». [En ligne]. Disponible sur: https://arxiv.org/abs/2106.08265

[7] C. Baur, S. Denner, B. Wiestler, S. Albarqouni, et N. Navab, « Autoencoders for Unsupervised Anomaly Segmentation in Brain MR Images: A Comparative Study », 2020. Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://arxiv.org/abs/2004.03271

[8] « Réseaux antagonistes génératifs ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://fr.wikipedia.org/wiki/R%C3%A9seaux_antagonistes_g%C3%A9n%C3%A9ratifs

[9] Y. Zhang, J. Li, l. Junfeng, et H. Pan, « Unsupervised bearing raceway surface defect detection based on improved f-AnoGAN », *Measurement Science and Technology*, vol. 36, p. , 2024, doi: 10.1088/1361-6501/ad8021.

[10] E. Safonova, A. Kravets, M. Scherbakov, A. Kizim, M. Al-Gunaid, et A. Echin, « Industrial Environmental Impact Assessment Method Based on Detection of Complex Anomalies in Time Series », *Applied System Innovation*, vol. 7, nº 5, p. 89, 2024, doi: 10.3390/asi7050089.

[11] N. Van Otten, « How To Implement Anomaly Detection With One-Class SVM In Python ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://spotintelligence.com/how-to-implement-anomaly-detection-with-one-class-svm-in-python/

[12] Anomalib Contributors, « PatchCore ». Consulté le: 17 juillet 2025. [En ligne]. Disponible sur: https://anomalib.readthedocs.io/en/v0.3.7/reference_guide/algorithms/patchcore.html

[13] MVTec Software GmbH, « MVTec Anomaly Detection Dataset (MVTec AD) ». Consulté le: 23 juillet 2025. [En ligne]. Disponible sur: https://www.mvtec.com/company/research/datasets/mvtec-ad/downloads

[14] R. P. Foundation, « Raspberry Pi 5 – Hero Image ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://www.pi-shop.ch/media/catalog/product/cache/1/image/650x/%E2%80%A6/pi_5_hero_1500x1500_1.png

[15] UCTronics/Arducam, « 2 MP IMX291 low-light wide-angle USB camera module with waterproof metal case ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://www.pi-shop.ch/2mp-imx291-low-light-120-wide-angle-usb-camera-module-with-waterproof-metal-case

[16] P. Padgett, B. Chan, et R. Sanchez, « Corkscrew: A tool for tunneling SSH through HTTP proxies ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://github.com/bryanpkc/corkscrew

[17] Roboflow, Inc., « Roboflow – Build and Deploy Computer Vision Applications ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://roboflow.com/

[18] Wikipédia contributors, « Courbe ROC — Wikipédia, l'encyclopédie libre ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://fr.wikipedia.org/wiki/Courbe_ROC

[19] C. Coates et I. Juvan-Beaulieu, « Rolling and Global Shutter sCMOS Camera Mode ». [En ligne]. Disponible sur: https://andor.oxinst.com/learning/view/article/rolling-and-global-shutter

[20] LEADRP Rapid Prototyping Blog, « A Complete Guide to Aluminum Extrusion ». Consulté le: 19 juillet 2025. [En ligne]. Disponible sur: https://leadrp.net/blog/a-complete-guide-to-aluminum-extrusion/

[21] Y. Cui, Z. Liu, et S. Lian, « A Survey on Unsupervised Anomaly Detection Algorithms for Industrial Images », *IEEE Access*, vol. 11, p. 55297-55315, 2023, doi: 10.1109/ACCESS.2023.3282993.

Annexes

Données du travail de bachelor

HES-SO Valais	SYND	ETE	TEVI	ISC
	X	X	X	X

Données du travail de bachelor

FO 1.2.02.07.FB
che/13.03.2024

Filière **ISC**	Année académique **2024-25**	No TB **ISC-ID-25-3**
Mandant ☐ HES—SO Valais-Wallis ☒ Industrie ☐ Etablissement partenaire	Etudiant·e **Francisco Ribeiro** Professeur·e **Francesco Carrino**	Lieu d'exécution ☒ HES—SO Valais-Wallis ☒ Industrie ☐ Etablissement partenaire
Travail confidentiel ☐ oui ☒ non	Expert·e *(données complètes)* **Emmanuel Senft, esenft@idiap.ch**	

Titre
Système de Surveillance en Temps Réel pour Détection de Défauts sur Aluminium

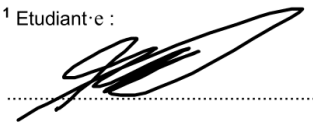
Description

Le projet vise à développer un système de détection automatique de défauts sur des produits en aluminium pendant la production (extrusion). Ces produits d'extrusion, mesurant plusieurs mètres de long, peuvent présenter des défauts visuels sur leur surface « cachée » (orientée vers le sol). Pendant la phase de surveillance, le produit se déplace à une vitesse d'environ 6 mètres par minute. Bien que cette vitesse ne soit pas élevée, les conditions de faible luminosité rendent la capture de photos de bonne qualité difficile. L'étudiant doit proposer un prototype pour la prise de mesures et la reconnaissance de défauts. La reconnaissance de défauts se fera d'abord par des méthodes non supervisées : le but sera de déterminer s'il y a un défaut (une anomalie) ou non. Ensuite, l'étudiant devra étudier la possibilité de classifier et d'indiquer le type de défaut.

Objectifs

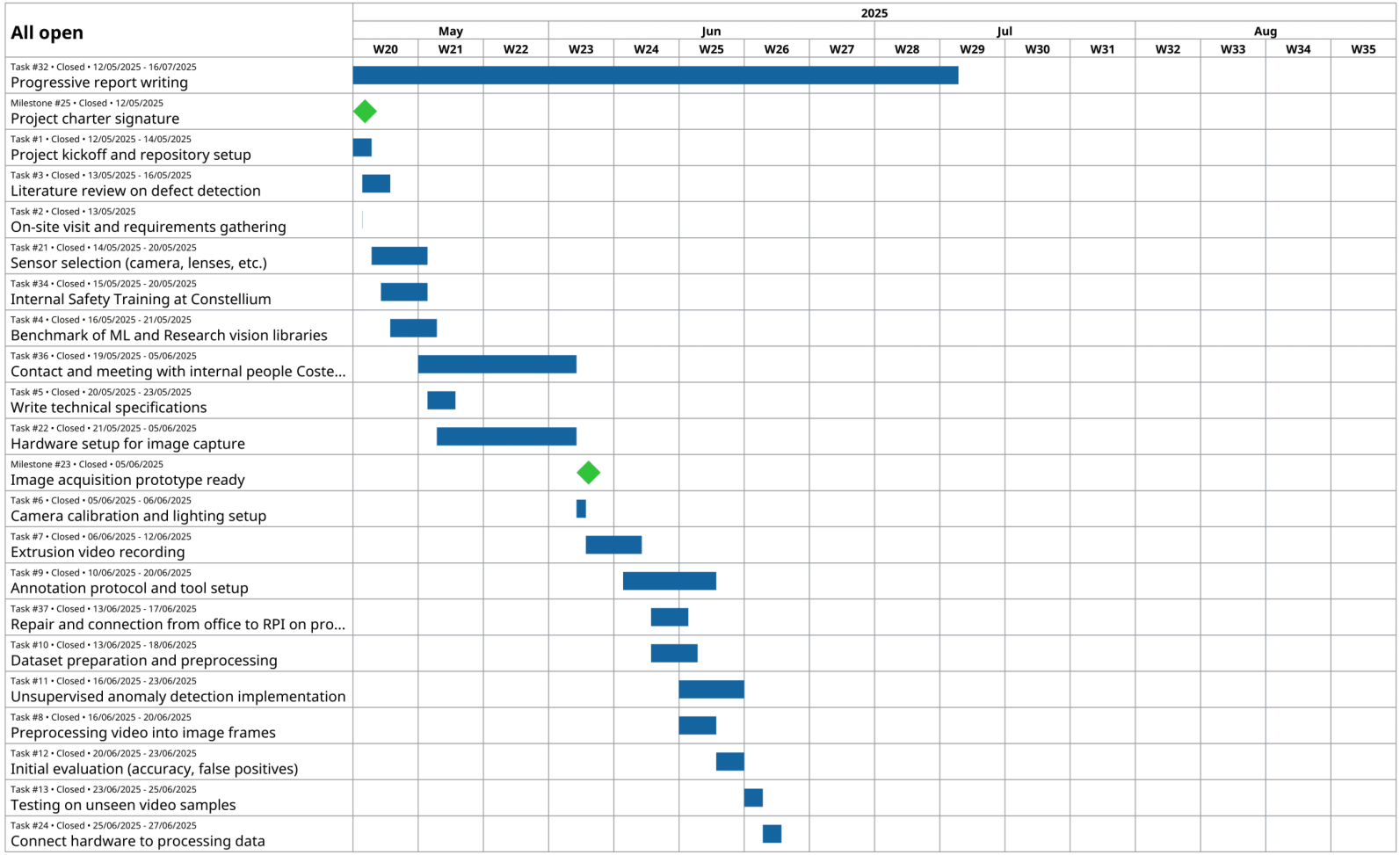
L'objectif principal de ce TB est le développement et l'intégration d'un « proof-of-concept » d'un système de détection de défauts selon les spécifications définies. Les tâches sont les suivantes :

- Prototype pour la prise de mesures
  - Concevoir et proposer un prototype pour la prise de mesures.
  - Développer un système de capture d'images capable de fonctionner dans des conditions de faible luminosité.
- Détection de défauts
  - Développer un système d'annotation de défauts pour fichier vidéo adapté au cas étudié.
  - Intégrer des méthodes non supervisées pour la détection des défauts (goal : déterminer s'il y a un défaut ou non).
  - Le prototype doit être conçu pour fonctionner en temps réel et être déployé dans l'infrastructure du partenaire. Le concept pourra cependant être démontré « offline » sur une vidéo pré-enregistrée.

Signature ou visa	Délais
Responsable de l'orientation *:* ..... [1] Etudiant·e : 	Attribution du thème *:* **14.04.2025** Début du travail de bachelor *:* **12.05.2025** Remise du rapport final *:* **25.07.2025, 12:00** Défense orale *:* **Semaine 34** (18 et 21 août 2025) Expositions et Pitch *:* **22.08.2025** – HEI **25.08.2025** – Monthey

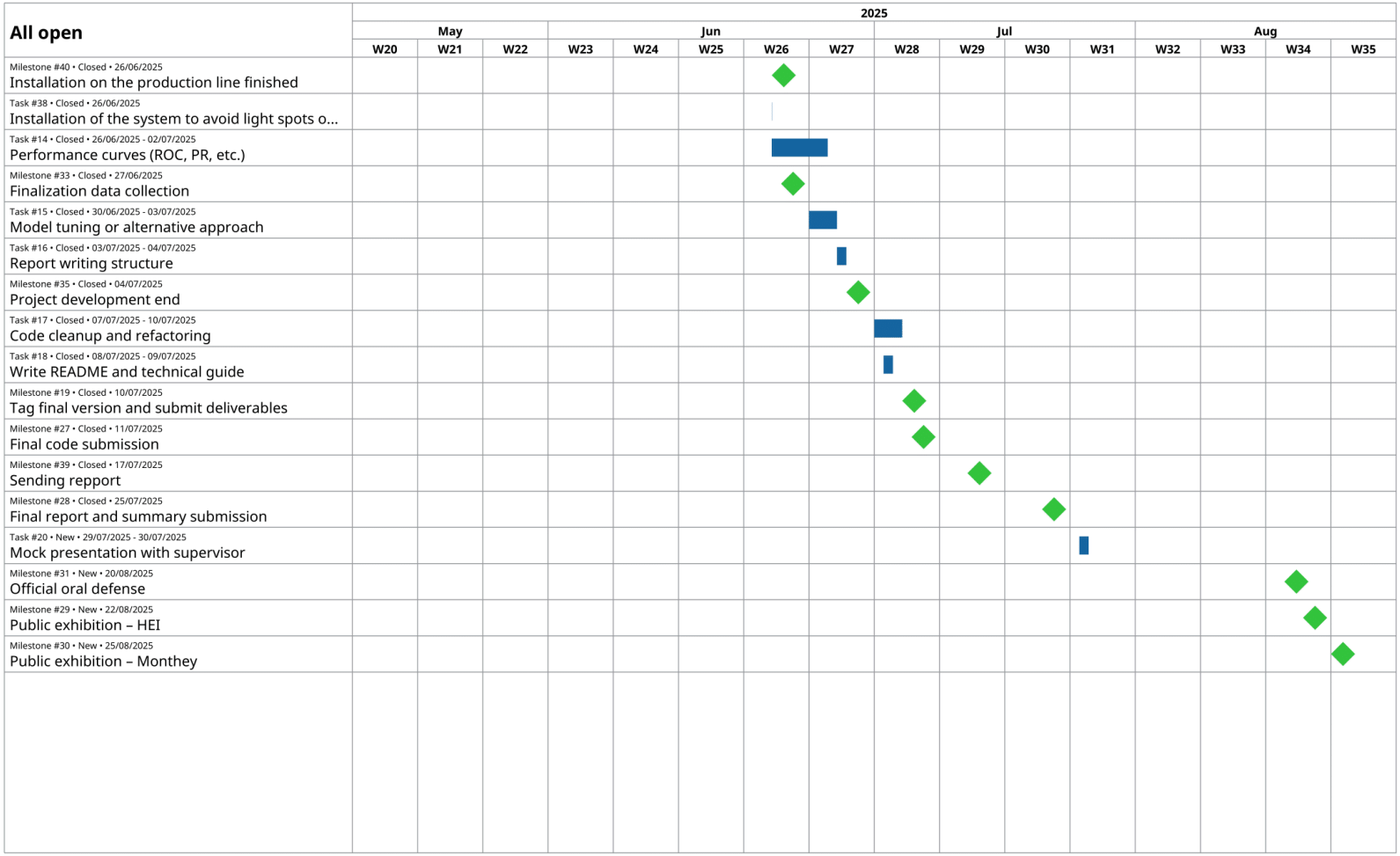
[1] *Par sa signature, l'étudiant·e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de bachelor.*

Planning Diagramme de Gantt



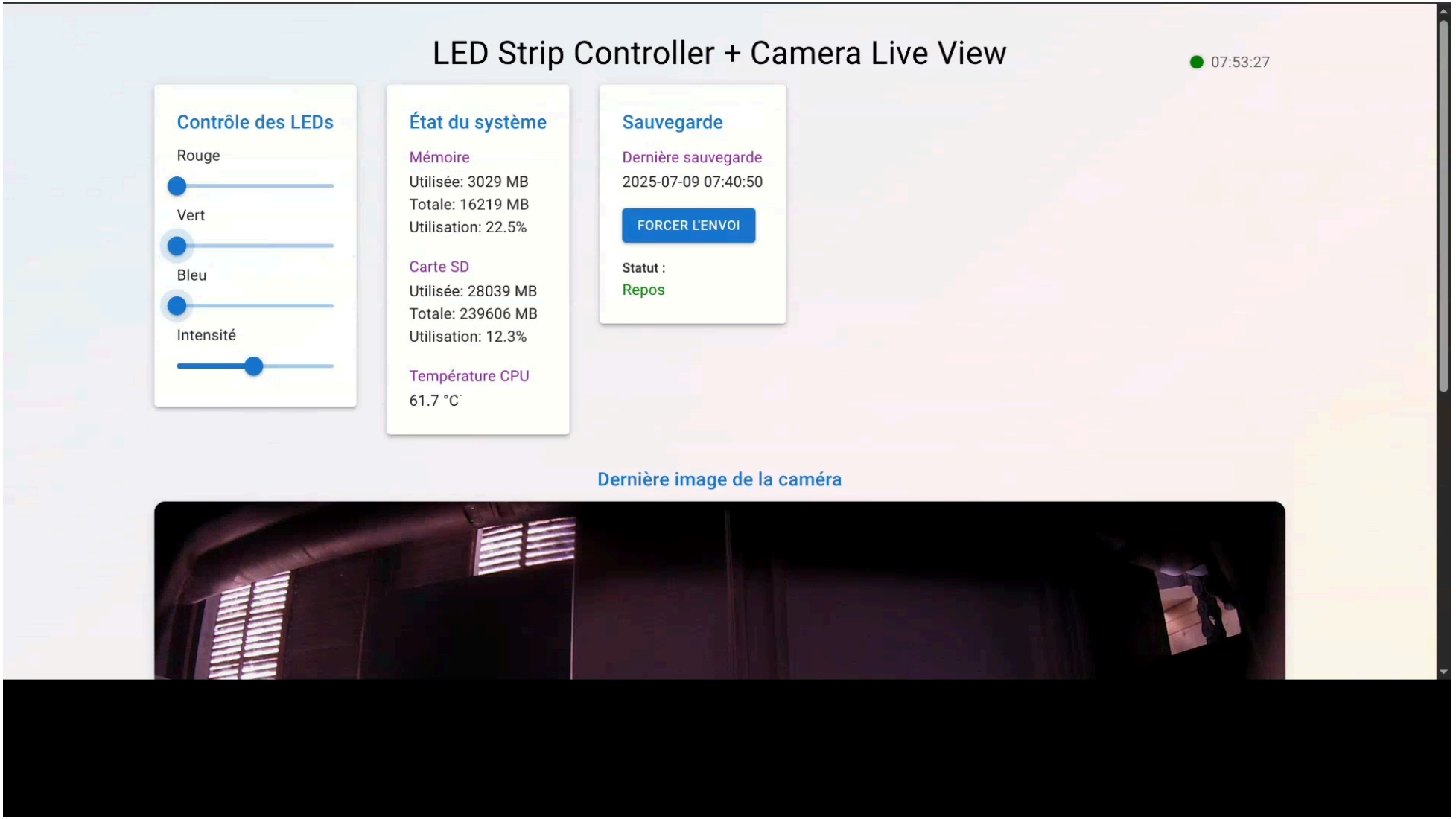
22/07/2025 08:14 1/2 All open

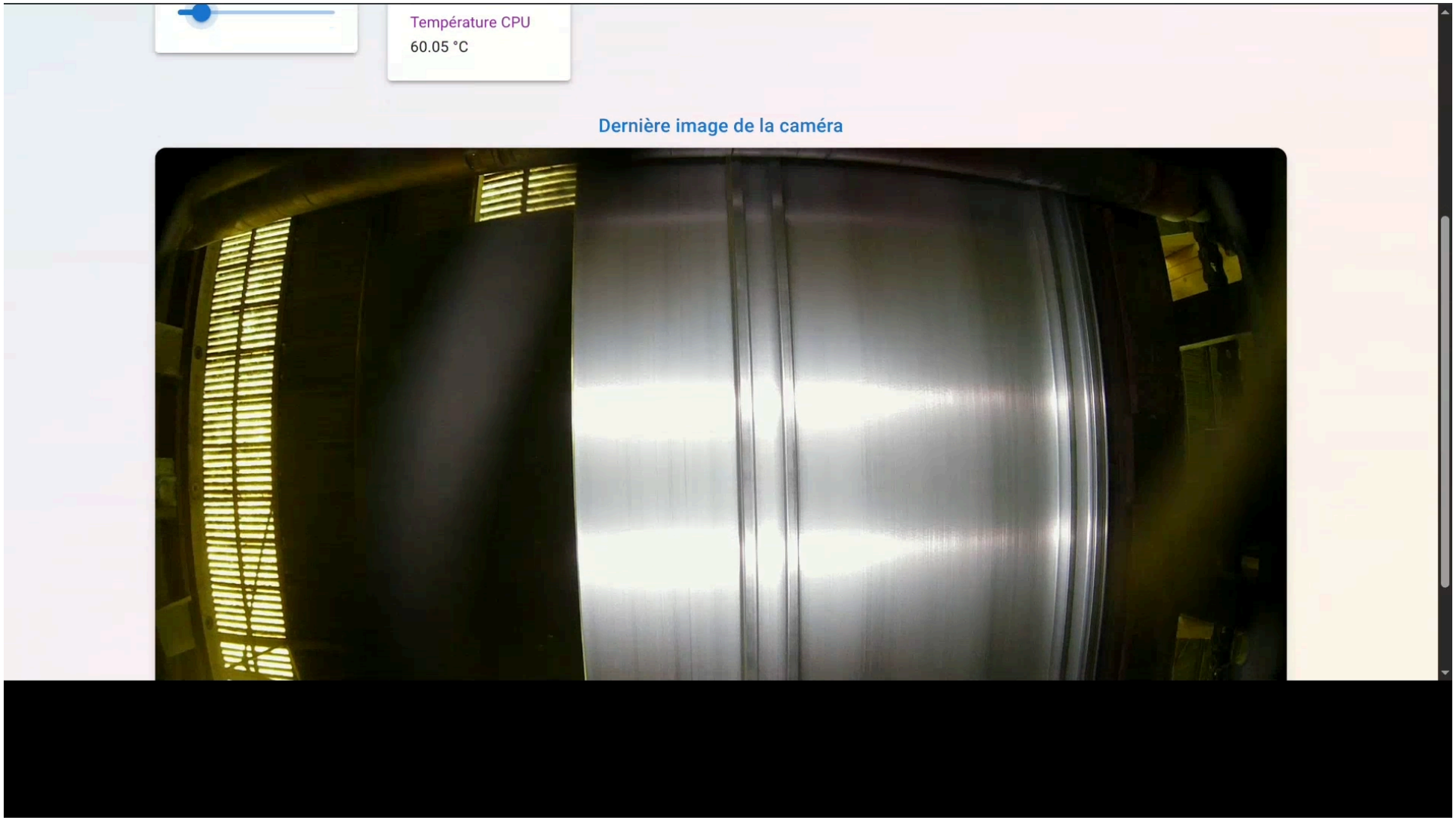
OpenProject



22/07/2025 08:14 2/2 All open

Front end


LED Strip Controller + Camera Live View
07:53:27
Contrôle des LEDs
Rouge
Vert
Bleu
Intensité
État du système
Mémoire
Utilisée: 3029 MB
Totale: 16219 MB
Utilisation: 22.5%
Carte SD
Utilisée: 28039 MB
Totale: 239606 MB
Utilisation: 12.3%
Température CPU
61.7 °C
Sauvegarde
Dernière sauvegarde
2025-07-09 07:40:50
FORCER L'ENVOI
Statut :
Repos
Dernière image de la caméra


Température CPU
60.05 °C
Dernière image de la caméra

Installation physique

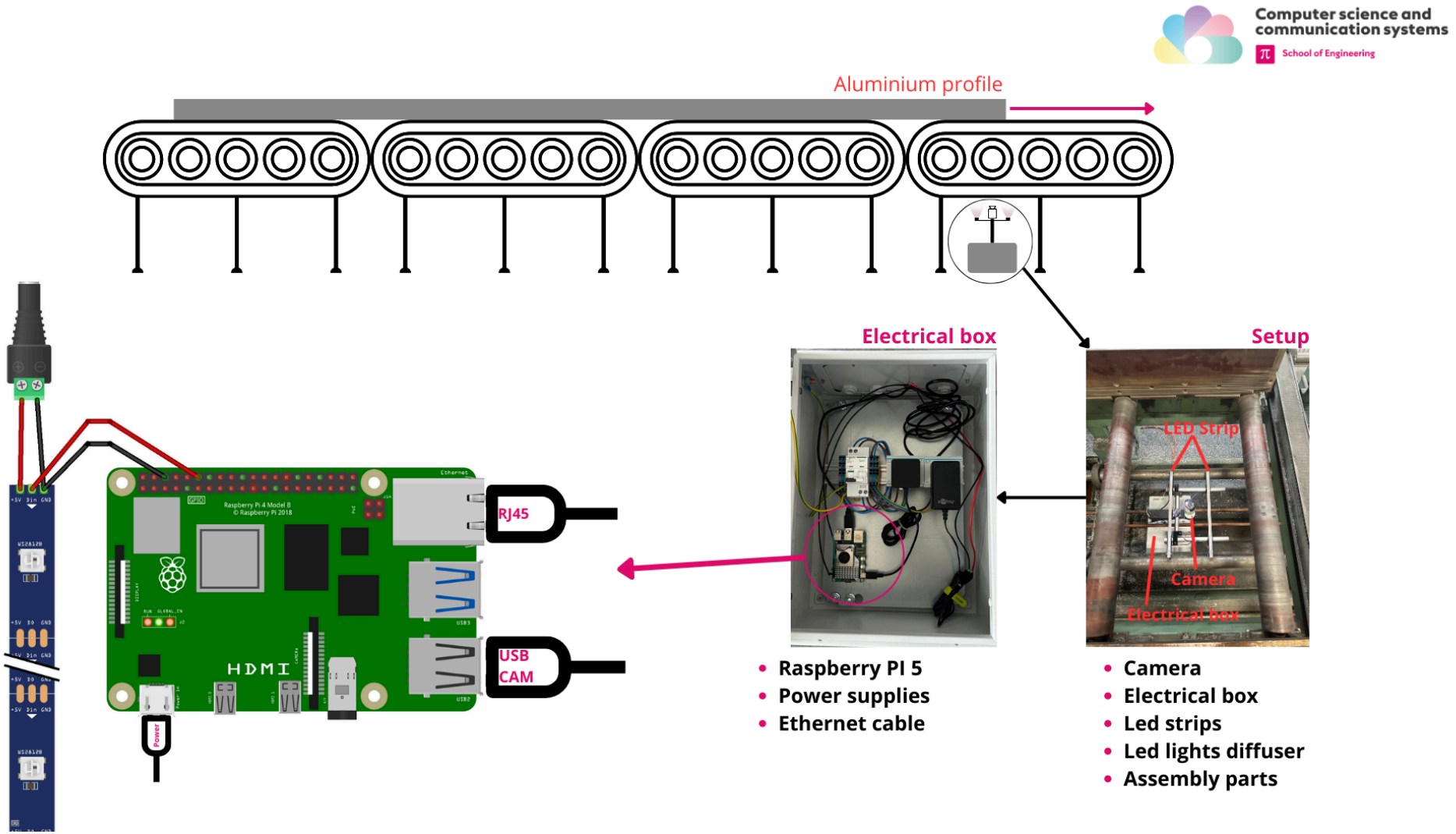


Table des acronymes

AUROC : Area Under Receiver Operating Characteristic
CNN : Convolutional Neural Network
CPU : Central Processing Unit
FAISS : Facebook AI Similarity Search
FPS : Frames per Second
GAN : Réseaux antagonistes génératifs
GPIO : General Purpose Input Output
GPU : Graphics Processing Unit
LED : Light Emitting Diode
PoE : Power over Ethernet
RPI : Raspberry Pi
SAP : Systèmes, Applications et Produits
SFTP : SSH File Transfer Protocol
SSH : Secure Shell

Table des figures