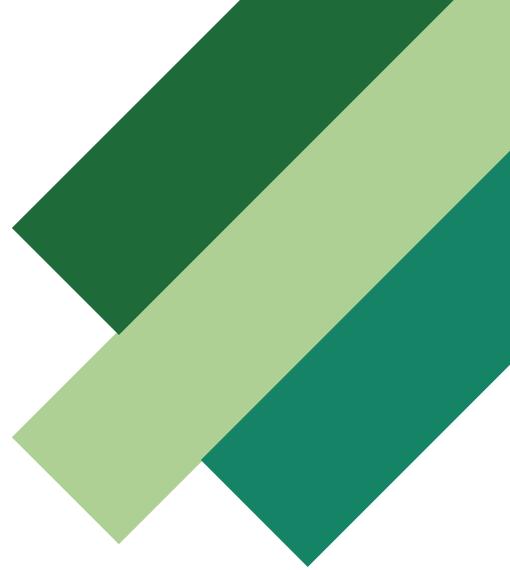


LOCHARD-CHAVEZ Oriana, LEBOEUF Elliott  
Tuteur ANDRA : HERMAND Guillaume  
Tuteurs école : SAINT-JORE Amaury, CIARLETTA Laurent



# ***Robotique et IA en environnements complexes : exploration de la galerie souterraine de l'ANDRA avec le robot et détection avancée de fissures***

Projet Industrie ANDRA 2024/2025



# Avant-Propos et Remerciements

Ce rapport a été rédigé à la suite du projet industrie de l'École des Mines de Nancy, en partenariat avec l'ANDRA, durant l'année scolaire 2024-2025. Il a été réalisé par les auteurs avec l'aide des ingénieurs du Techlab, de nos tuteurs ainsi que de l'intervenant de l'ANDRA. Nous tenons à remercier chaleureusement l'ensemble des personnes qui ont pu nous aider et conseiller tout au long de ce projet. L'ensemble de notre code se trouve à l'adresse github suivante : [https://github.com/mines-nancy/ANDRA\\_2024\\_2025](https://github.com/mines-nancy/ANDRA_2024_2025)

# Table des matières

<b>Avant-Propos et Remerciements</b>	<b>2</b>
<b>1 Introduction</b>	<b>4</b>
<b>2 État de l'art</b>	<b>5</b>
2.1 Détection de fissures . . . . .	5
2.2 Positionnement et évitement d'obstacle . . . . .	6
<b>3 Semestre 1</b>	<b>6</b>
3.1 Assemblage du robot . . . . .	6
3.1.1 Présentation des parties indépendantes . . . . .	6
3.1.2 Assemblage du robot . . . . .	7
3.2 Reprise du code de l'année dernière . . . . .	8
3.3 Reprise du code de Xinhao ZHAO . . . . .	9
3.4 Première descente . . . . .	10
<b>4 Semestre 2</b>	<b>12</b>
4.1 Fonctionnement basique de ROS . . . . .	12
4.2 Partie localisation . . . . .	13
4.2.1 Obtention de l'information des capteurs . . . . .	13
4.2.2 Fusion des données . . . . .	14
4.2.3 Algorithme de SLAM . . . . .	18
4.2.4 Algorithme AMCL . . . . .	18
4.3 Intégration de l'IA et affichage des fissures . . . . .	20
<b>5 Gestion de projet</b>	<b>21</b>
<b>6 Résultats</b>	<b>22</b>
<b>7 Conclusion et Ouvertures</b>	<b>24</b>

# 1. Introduction

Ce projet industrie est réalisé en collaboration avec l'ANDRA (Agence nationale pour la gestion des déchets radioactifs), une institution publique chargée du stockage des déchets nucléaires sur le moyen et long terme. Dans un premier temps, l'ANDRA a été chargée de développer un système de stockage en surface pour les déchets de faible et moyenne activité, en reprenant la gestion du Centre de stockage de la Manche et en établissant rapidement des règles pour encadrer et sécuriser ce stockage. Par la suite, l'agence a ouvert plusieurs sites dédiés à ce type de déchets, atteignant en 2020 un volume total de 1,3 million de mètres cubes stockés.

En parallèle, l'ANDRA a conduit plus de deux décennies de recherches sur le stockage géologique en profondeur. Ces travaux ont abouti au projet CIGEO, un site de stockage en profondeur, à environ 500 mètres sous terre, dans une couche géologique aux propriétés intéressantes, située dans la région de Bure, à cheval entre la Meuse et la Haute-Marne.

Cette infrastructure d'envergure comprendra plus de 270 kilomètres de galeries et d'alvéoles, avec une capacité de stockage d'environ 80 000 mètres cubes de déchets à haute activité et à longue durée de vie.

Cependant, en raison du cisaillement entre les jointures de la galerie, des fissures peuvent apparaître et se développer. Les galeries s'étendant sur plusieurs kilomètres, suivre l'évolution des fissures est une tâche répétitive et chronophage.

C'est dans ce contexte qu'est né le projet de ce parcours industrie, porté sur le thème suivant : “ Robotique et IA en environnements complexes : exploration de la galerie souterraine de l'ANDRA avec le robot et détection avancée de fissures.”



FIGURE 1 – Projet Cigéo, vue du laboratoire à Bure (Meuse)

## 2. État de l'art

### 2.1 Détection de fissures

Pour la détection de fissures, les technologies à l'état de l'art en termes de détection de fissures utilisent quasiment toutes des techniques de segmentation par IA. Cependant, cette tâche reste particulièrement difficile. En effet, les fissures peuvent prendre de nombreuses formes et l'environnement autour de ces dernières change très régulièrement (une fissure sur un fond uni blanc est plus facile à détecter qu'une fissure sur du crépi gris par exemple).

C'est pour cette raison qu'il n'existe aujourd'hui pas de base de données « universelle » pour la détection de fissures. C'est en particulier un problème sur le site de Bure car il y a de nombreux types de murs et donc de types de fissures différentes.

Nous avions tout de même essayé d'utiliser un modèle d'open data de fissures pour les détecter [8]. Cependant, les galeries possèdent des interstices et des câbles qui étaient souvent confondus avec de réelles fissures. Ces modèles étaient donc inutilisables en production (voir les figures 2 et 3).



FIGURE 2 – Un interstice reconnu comme une fissure



FIGURE 3 – Une barrière reconnue comme une fissure

## 2.2 Positionnement et évitement d'obstacle

Il est aujourd’hui possible d’obtenir à la fois un positionnement et un évitement d’obstacles très performants sur des modèles haut de gamme [1]. Ces technologies utilisent généralement un LiDAR (2D ou 3D) pour cartographier et d’autres capteurs comme des accéléromètres pour avoir une estimation de la vitesse et donc de la position. Cependant, pour éviter toutes formes de dérapage (« drift » en anglais), soit le fait de voir les erreurs de positions s’accumuler, le matériel doit être de grande précision, mais cet équipement est généralement très onéreux.

# 3. Semestre 1

## 3.1 Assemblage du robot

### 3.1.1 Présentation des parties indépendantes

Notre robot autonome est constitué de 5 parties indépendantes que nous allons détailler ici.

**Caméra PTZ** Une caméra PTZ (Pan-Tilt-Zoom) est une caméra capable de pivoter sur 360° et d’ajuster son angle et son zoom de manière dynamique. Il s’agit du même modèle utilisé l’année dernière, offrant une résolution de 1080p à 60 fps, mais elle ne dispose pas de stabilisateur d’image intégré. Il est donc crucial de minimiser les vibrations pendant son utilisation pour garantir la netteté de l’image. Autrement dit, le robot devra se déplacer à une vitesse modérée, ce qui ne pose pas de problème en soi, mais doit être pris en compte pour assurer des prises de vue claires et précises.

**LiDAR** Un LiDAR (Light Detection and Ranging) est une technologie qui utilise des impulsions laser pour mesurer des distances avec une grande précision. En émettant des faisceaux lumineux et en calculant le temps qu’ils mettent à revenir après avoir réfléchi sur des objets, un LiDAR est capable de reconstituer en deux ou trois dimensions (ici notre LiDAR est 2D) l’environnement, même dans des conditions de faible luminosité.

**ZED2** La caméra ZED2 est une caméra de profondeur, ce qui lui permet d’estimer si le robot se déplace en avant, en arrière ou s’il effectue une rotation. Ces informations seront essentielles lors de la phase de localisation. De plus, cette caméra possède une unité de mesure inertielle (IMU), un capteur capable de mesurer les accélérations linéaires et les vitesses angulaires grâce à des accéléromètres et des gyroscopes. L’IMU permet ainsi d’estimer la position, l’orientation et les mouvements du robot dans l’espace.



FIGURE 4 – PTZ Marshall CV-605



FIGURE 5 – YdLidar G4



FIGURE 6 – Stereolabs ZED2

**Jetson Orin Nano** La Jetson Nano est un microcontrôleur équipé d'un GPU intégré, offrant ainsi une capacité de traitement suffisamment puissante pour exécuter des tâches complexes, telles que la reconnaissance basée sur l'intelligence artificielle, directement sur le robot. Cette architecture permet d'éviter le recours à des solutions de edge ou cloud computing, qui pourraient introduire des latences.

**Agilex Scout Mini** Enfin, la base de notre robot est un Agilex Scout Mini. Ce dernier peut se contrôler à l'aide d'une télécommande ou à partir d'un microcontrôleur connecté à un convertisseur analogique numérique (CAN).



FIGURE 7 – Jetson Orin Nano



FIGURE 8 – Agilex Scout Mini

### 3.1.2 Assemblage du robot

La prise en main matérielle du robot a débuté fin septembre avec la réception de la Jetson Orin Nano et de la caméra PTZ. Les premières sessions ont été consacrées à la compréhension de leur fonctionnement et à la gestion des connexions réseau, notamment la nécessité d'utiliser deux entrées Ethernet pour accéder simultanément à la caméra et à Internet.

En novembre, la structure physique du Scout Mini a été finalisée avec la fabrication et le montage de la tourelle support. Parallèlement, un LiDAR 2D et une caméra ZED2 ont été intégrés au robot afin d'enrichir ses capacités de perception et de navigation. Les tests de

connexion entre ces capteurs et le Jetson Orin Nano ont été réalisés fin novembre, validant leur bon fonctionnement.

Faute d'alimentation autonome, la batterie n'étant pas encore soudée, des solutions temporaires via rallonge ont permis d'alimenter le système lors des essais en conditions réelles. L'ensemble du matériel a ainsi été progressivement assemblé et configuré pour offrir une base stable aux développements logiciels ultérieurs.



FIGURE 9 – Tourelle montée sur l'Agilex Scout Mini

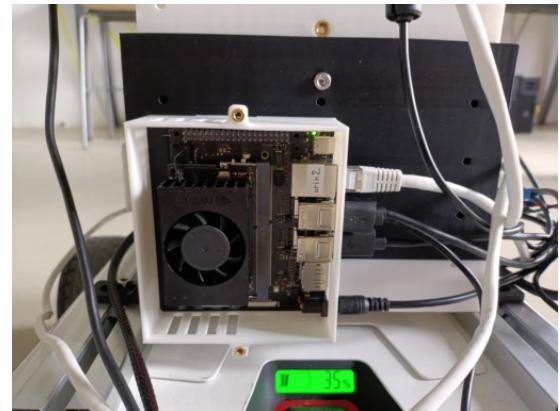


FIGURE 10 – Intégration de la Jetson Orin Nano

### 3.2 Reprise du code de l'année dernière

Bien que ce projet, en collaboration avec l'ANDRA, soit mené pour la quatrième année consécutive, nous avons constaté une absence de transmission des travaux réalisés au fil des années. En effet, seul le rapport final de l'année 2023/2024 nous a été accessible, sans aucune information concernant les contributions ou les résultats des équipes de 2021/2022 et 2022/2023. Face à cette grosse lacune dans la continuité et la gestion du projet, nous avons décidé de documenter rigoureusement l'ensemble de notre travail afin de faciliter la reprise du projet par les futures équipes (voir partie "Gestion de projet").

L'année précédente, les deux étudiants ont concentré leur travail sur la reconnaissance et la caractérisation de fissures par intelligence artificielle. Ils avaient créé un système pour scanner une fissure et estimer sa taille à partir d'un objet de référence dont la taille était connue.

Pour cela, ils utilisaient une caméra PTZ (Pan-Tilt-Zoom), que nous avons également pu exploiter cette année. La caméra effectuait des rotations verticales jusqu'à localiser un objet de référence, en l'occurrence une étiquette. Une fois cet objet détecté, l'algorithme d'intelligence artificielle chargé de repérer les fissures se déclenchaient automatiquement. En cas de détection, il était alors possible d'estimer la taille de la fissure en établissant une correspondance entre les pixels de l'image et une échelle en centimètres.

Cependant, les résultats présentés dans leur rapport présentaient une incertitude type trop élevée pour permettre une reprise fiable du travail sur l'estimation des dimensions. En effet, l'erreur atteignait  $\pm 4$  cm sur l'axe des abscisses et près de  $\pm 7$  cm sur l'axe

des ordonnées. Cette imprécision s'expliquait principalement par la qualité limitée de la caméra, les déformations optiques qu'elle générait, ainsi que par l'angle de prise de vue. En particulier, une prise de vue en biais réduisait significativement la précision de l'échelle de conversion pixel/cm. Après plusieurs essais, nous avons constaté que le programme détectait correctement les étiquettes. En revanche, il s'est avéré incapable d'identifier les fissures sur les images prises sur le site de Bure, fournies par l'équipe du TechLab lors d'une descente précédente. Face à ce constat, nous avons décidé de réaliser une évaluation plus globale de leur IA afin de vérifier la conformité des résultats obtenus par rapport à ceux présentés dans leur rapport.

Selon ce dernier, l'IA atteignait une précision de 73% et un rappel de 51%, des scores considérés comme acceptables. Toutefois, nos propres tests ont révélé des performances nettement inférieures. En effet, lorsqu'elle était confrontée à des images réelles issues de l'ANDRA, l'IA confondait fréquemment des éléments tels que des câbles, des interstices entre blocs ou des compteurs électriques avec des fissures (similaires aux images en section 2.1). Ces erreurs de classification remettent en question la fiabilité de l'algorithme dans des conditions réelles. Suite à ces tests, nous avons décidé de ne pas réutiliser leur IA, en raison de sa fiabilité insuffisante. De plus, leur protocole de détection nous semblait trop contraignant (pas toujours d'étiquettes) et peu précis.

### 3.3 Reprise du code de Xinhao ZHAO

En parallèle du travail de détection de fissures par IA, nous avons eu accès au travail de Xinhao ZHAO, un étudiant qui effectuait son PFE au Techlab sur l'évitement d'obstacle par robot. Son stage était également réalisé en partenariat avec l'ANDRA, et portait sur le même robot que celui utilisé dans notre projet : l'AgileX.

Ce PFE s'inscrivait dans la démarche de l'école de réaliser un robot et une solution de localisation/évitement d'obstacle propre au TechLab et commune à tous ses robots. Ainsi, Xinhao développait ses propres algorithmes en python afin de gérer la localisation et l'évitement d'obstacle du robot. Son travail se basait sur des April Tags et sur de la reconnaissance d'étiquettes (lui permettant de se situer dans le plan), ainsi que sur des instruments tels que le LiDAR 2D, une caméra PTZ et une ZED2.

L'architecture du code repose sur une organisation de 9 micro-services, chacun étant relativement indépendant des autres. Par exemple, un service est dédié au contrôle de la caméra PTZ, un autre à la gestion de la caméra ZED2, un troisième à la détection d'obstacles, et un quatrième à la navigation entre deux points. L'étudiant lançait chaque service manuellement dans un ordre précis afin de leur permettre de les connecter entre eux. Malheureusement, il n'a pas disposé du temps nécessaire pour réaliser la documentation de son code et indiquer comment le faire fonctionner.

Ainsi, malgré deux mois d'efforts, nous n'avons pas été en mesure d'utiliser correctement son programme afin de faire nos propres tests. En effet, nous sommes seulement parvenus à lancer les micro-services de façon indépendante.

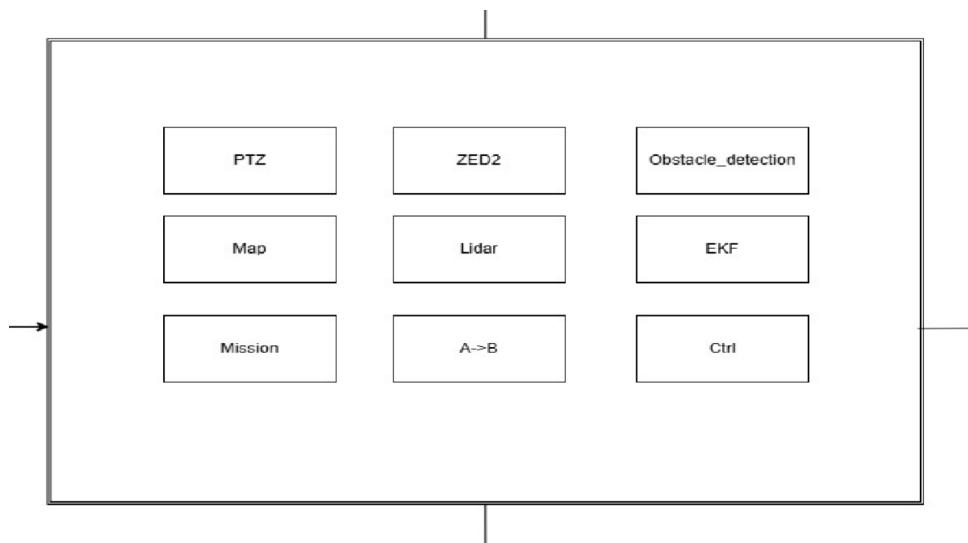


FIGURE 11 – Les différents services développés par Xinhao

### 3.4 Première descente

Notre première descente au laboratoire de Bure s'inscrit dans la continuité de la remise en question que nous avons menée en décembre 2024. À cette occasion, nous avons redéfini des objectifs concrets, le premier étant le développement d'une nouvelle IA de détection de fissures. En effet, comme mentionné précédemment, nous avions fait le choix de ne pas réutiliser celle conçue l'année précédente, afin de repartir sur une base de travail solide.

L'objectif principal de notre descente du 16 janvier 2025 était de constituer un jeu de données (dataset) représentatif des fissures présentes sur le site de l'ANDRA, en vue d'entraîner notre IA sur des données réellement adaptées au contexte du laboratoire de Bure. L'acquisition de nouvelles images s'est avérée indispensable, car les bases de données open source disponibles ne reflétaient pas les spécificités environnementales du site. Par ailleurs, nous souhaitions que les images d'entraînement soient capturées avec la même caméra que celle qui sera utilisée ultérieurement pour la détection, afin d'assurer une meilleure cohérence entre les phases d'apprentissage et d'application de l'IA.

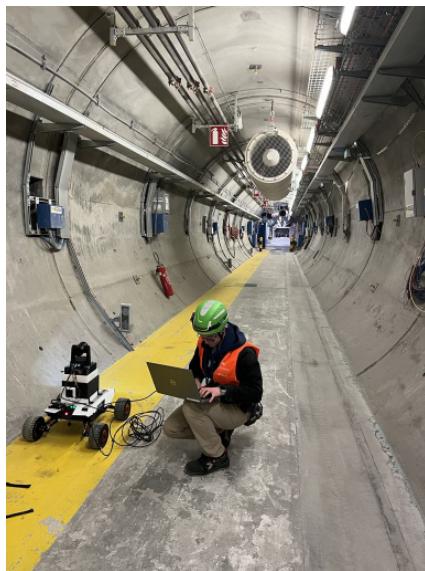


FIGURE 12 – Descente du 16 janvier 2025



FIGURE 13 – Prise d'image avec la tourelle

Afin de préparer notre descente, nous avons écrit et testé plusieurs algorithmes pour l'enregistrement semi-automatique de vidéos. Nous avons calculé les angles d'ouverture réels de la caméra afin de mieux cadrer nos images. Et nous avons fait des recherches au préalable sur l'entraînement d'IA afin de récolter des données de la meilleure qualité possible.

De retour au TechLab, nous avons ensuite découpé les vidéos en images et trié celles qui étaient nettes et intéressantes. Nous avons ainsi annoté nos images, ce qui, avec les images prises l'année dernière, nous a permis d'obtenir une base d'entraînement de près de 300 images.

L'entraînement terminé, nous avons testé notre modèle sur la base de validation, mais les résultats se sont révélés décevants. En y regardant de plus près, nous avons constaté un décalage entre les performances réelles du modèle et celles mesurées. Le problème venait du fait que YOLOv11, notre système de détection par "bounding box", interprétablait mal certaines fissures dites « discontinues », par exemple séparées par un tuyau. Lors de l'annotation, nous les avions considérées comme une seule fissure, alors que le modèle les traitait comme plusieurs entités distinctes.

Suite à la consultation avec l'entreprise, le phénomène de multi-détection a été considéré comme peu significatif, ce qui confirme la très bonne performance de l'IA (voir partie Résultats).



FIGURE 14 – Premier exemple d'une image sélectionnée pour l'entraînement



FIGURE 15 – Second exemple d'une image sélectionnée pour l'entraînement

## 4. Semestre 2

Après une réunion avec nos encadrants, il a été décidé d'abandonner l'API du TechLab pour se concentrer sur le framework robotique dominant dans l'industrie : ROS2 [2].

### 4.1 Fonctionnement basique de ROS

ROS (Robot Operating System) est un ensemble de logiciels open source conçu pour aider au développement de robots. Ce n'est pas un système d'exploitation au sens classique (comme Windows ou Linux), mais une infrastructure logicielle qui fonctionne généralement sur un système Linux (ici Ubuntu). Avant ROS, chaque entreprise de robotique pouvait décider de réaliser son interface de programmation comme elle le souhaitait. Cela offrait l'avantage de permettre à l'entreprise de garder un contrôle total sur les capacités et les limitations de son robot. Cependant, cela obligeait les développeurs à consulter de la documentation à chaque lancement de nouveau robot et à s'adapter constamment aux spécificités de chaque modèle, ce qui pouvait être chronophage et source de complexité. L'objectif de ROS est "Write once, Run everywhere", autrement dit qu'un code fonctionnant sur un robot peut fonctionner sur un autre robot si ce dernier possède les mêmes capteurs.

L'infrastructure de transfert de données par ROS est assez spécifique. En effet, chaque "programme" en ROS est appelé un noeud. Un noeud est un programme python ou C++ capable d'envoyer ou recevoir des informations diverses provenant de "topic". Un "topic" permet de faire transférer les informations entre les noeuds. Par exemple, on pourrait avoir un noeud qui calcule par des méthodes d'intégration la position du robot à partir des données de l'accéléromètre. Il y aurait alors deux topics : un premier topic s'occupera du transfert d'informations de l'IMU vers le noeud, le noeud effectuerait son traitement puis enverrait l'information de position sur un autre topic.

On appelle également "subscriber" un noeud abonné à un topic recevant des informations de ce dernier et "publisher" un noeud abonné à un topic envoyant des informations sur ce topic.

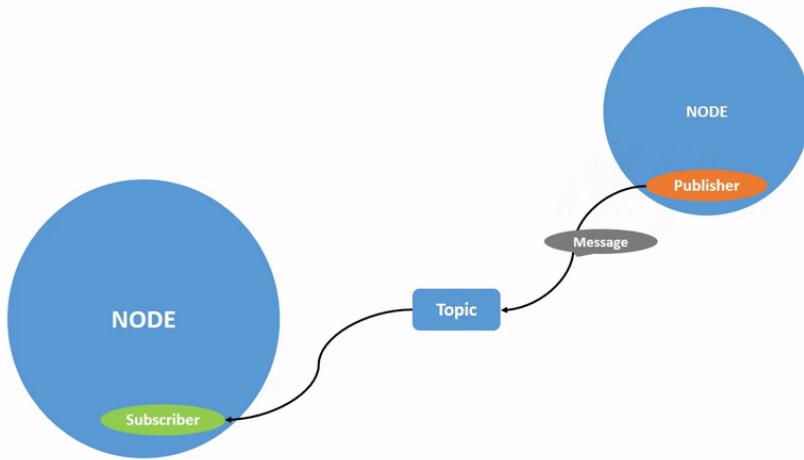


FIGURE 16 – Exemple d'un noeud publisher et d'un noeud subscriber sur le même topic

## 4.2 Partie localisation

### 4.2.1 Obtention de l'information des capteurs

A l'aide de ROS2, nous pouvons utiliser les différents modules fournis par les différents constructeurs pour récupérer les informations et les publier sur les bons topics.

#### LiDAR :

Notre LiDAR provient de la marque YDLIDAR, qui n'est pas particulièrement reconnue dans l'industrie, du moins son support logiciel semble limité [12]. Les modules n'ont pas reçu de mises à jour depuis plusieurs années, et ils ne sont plus compatibles avec les dernières versions de ROS2, notamment la version Humble que nous utilisons. En conséquence, nous avons dû adapter et modifier le code afin de résoudre les erreurs de compilation et rendre le système compatible avec les nouvelles versions de ROS2.

Le module publie les données du LiDAR sur le topic /scan qui est le topic par défaut pour les données des LiDARs. Pour autant, le LiDAR est “emboité” : il y a donc quatre piliers qui bloquent la vue du LiDAR.

De plus, le LiDAR étant tourné de 90° vers la gauche par rapport à l'avant du robot, il est nécessaire de préciser à ROS d'effectuer une rotation des données reçues pour correspondre à la réalité (plus de précisions dans le README sur github).

#### ZED2 :

Le module ZED2 officiel est bien plus complet et soutenu que celui du LiDAR [7]. Ce dernier publie de nombreux topics différents : pour récupérer les données de l'IMU (zed2 imu data), de l'odométrie (/zed2 odom)...

Il reste cependant à noter que la manière d'obtention de ces données par la ZED2 reste très opaque. Il n'y a que très peu d'informations constructeurs à ce sujet et l'appareil ressemble davantage à une boîte noire qu'à un capteur classique.

## Roues du robot :

Pour les roues du robot, nous utilisons le SDK officiel de l'Agilex Scout Mini pour ROS2 [6]. Ce dernier permet à l'aide d'un CAN d'obtenir la vitesse des roues sur un topic défini pour l'utilisateur (dans notre cas ce topic s'appelle /odom\_robot).

A l'aide de ce SDK, nous pouvons également contrôler le robot. Les commandes associées sont publiées sur le topic /cmd\_vel (topic classique pour les commandes).

## Caméra PTZ :

Pour ce qui est de la caméra, nous utilisons un protocole classique de transmission de flux vidéo appelé RSTP [10]. La PTZ est directement reliée à la Jetson Orin Nano qui peut ensuite redistribuer les images reçues sur un topic ROS dédié (voir 4.3)

## Observation des données :

Pour observer efficacement ces données, nous avons utilisé un logiciel faisant partie de l'environnement ROS appelé RViz2. ROS permet de lire les données des capteurs par le réseau, la Jetson Nano publie l'ensemble des topics sur le réseau local, ce qui permet à n'importe quel ordinateur connecté d'avoir une visualisation.

Pour obtenir une compatibilité maximum, nous avons créé une image docker permettant d'utiliser RViz2 sur n'importe quel ordinateur possédant une distribution linux avec un serveur X11. Cette image docker peut être trouvée sur le github.

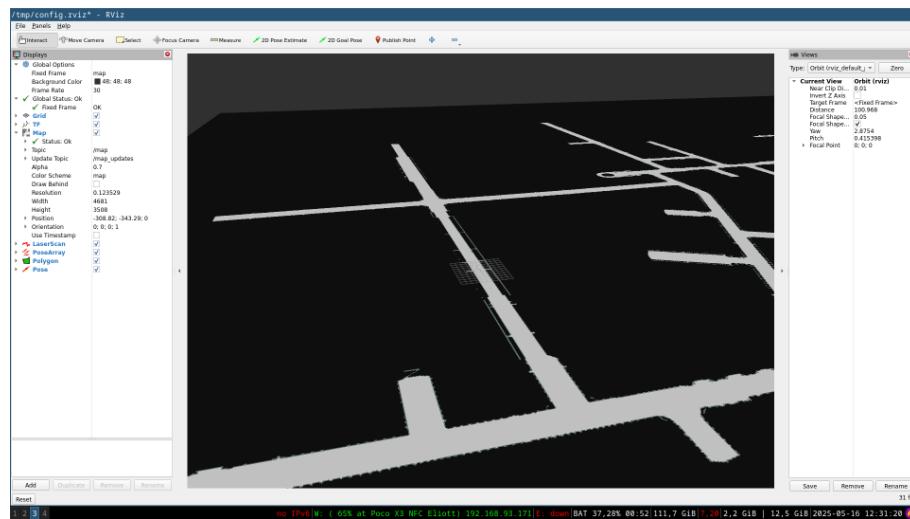


FIGURE 17 – Un exemple d'utilisation de RViz pour observer la carte utilisée (ici les galeries du site de Bure)

### 4.2.2 Fusion des données

Pour que la localisation soit précise, il faut fusionner les données des différents capteurs. En effet, la ZED2 possède un IMU et permet l'odométrie visuelle (grâce à ses caméras stéréoscopiques). On peut également ajouter les informations du LiDAR et de l'odométrie des roues pour obtenir un ensemble complet de données.

Pour cela nous utilisons un filtre de Kalman [9], du nom de l'ingénieur américano-hongrois Rudolf Kálmán. L'objectif d'un tel filtre est d'obtenir une estimation optimale de l'état du système (position, vitesse, orientation) en fusionnant les mesures de différents capteurs.

L'exemple typique est un véhicule roulant sur une route : si ce véhicule est équipé d'un GPS qui permet d'estimer sa position avec une précision de quelques mètres, on peut combiner les données du GPS avec l'odométrie du véhicule pour estimer plus précisément sa position. L'intérêt d'un tel dispositif est de combiner les avantages des différents capteurs : un GPS ne dérive pas dans le temps mais a une précision limitée à quelques mètres, à l'inverse l'odométrie peut fournir une estimation très précise sur de courtes distances mais accumule des erreurs (drift) au fil du temps.

Le filtre opère selon deux phases distinctes : la prédition et la correction. La phase de prédition consiste à estimer l'état futur à partir des mesures des capteurs proprioceptifs (IMU, odométrie des roues...) et de l'état précédemment estimé. La phase de correction permet de réduire les erreurs d'accumulation en recalibrant l'estimation à partir des mesures des capteurs extéroceptifs plus fiables (GPS, caméras, LiDAR).

Ces deux phases travaillent ensemble pour fournir une estimation optimale de la pose du robot. En plus de l'estimation de l'état, une matrice de covariance est calculée pour quantifier l'incertitude associée à chaque mesure capteur. Ainsi, en reprenant l'exemple du véhicule, on peut considérer que l'odométrie est précise à faible vitesse mais devient moins fiable à haute vitesse en raison du patinage des roues. Cette matrice de covariance permettra d'ajuster dynamiquement la confiance accordée à chaque capteur selon les conditions.

### Développement mathématique :

Dans cette sous-section, nous développons l'aspect mathématique du filtre de Kalman étendu (celui qui est prévu pour les systèmes non linéaires). Il nous faut d'abord un modèle physique du système, comme une intégration simple pour obtenir la position à partir de la vitesse.

### Modèle d'état (non linéaire) :

$$x_k = f(x_{k-1}, u_k) + w_k$$

où :

- $x_k$  est le vecteur d'état au temps  $k$ ,
- $u_k$  est le vecteur de commande,
- $f(\cdot)$  est une fonction non linéaire représentant la dynamique du système,
- $w_k \sim \mathcal{N}(0, Q_k)$  est le bruit de processus.

Dans notre cas,  $x_k$  représente le vecteur contenant la position, les angles (en quaternions) et les vitesses angulaires et linéaires.  $u_k$  représente la commande donnée au robot, par exemple, lorsque l'on demande au robot d'avancer, la commande moteur gauche et droite pourra être ajoutée à  $u_k$ .  $f$  est la fonction reliant nos données de commande à la position, elle est estimée automatiquement par le module ROS. Elle se base simplement sur de la mécanique classique.  $f$  représente le modèle physique sous jacent de nos commandes à partir d'une position précédente.  $w_k$  est le bruit du processus. Dans notre cas, on considère que  $Q_k$  est constant (les valeurs de  $Q$  sont données dans notre fichier de configuration).

### Modèle d'observation (non linéaire) :

$$z_k = h(x_k) + v_k$$

où :

- $z_k$  est le vecteur de mesure,
- $h(\cdot)$  est une fonction non linéaire représentant la relation entre l'état et l'observation,
- $v_k \sim \mathcal{N}(0, R_k)$  est le bruit de mesure.

$z_k$  représentera le vecteur avec l'ensemble des informations reçues par les différents capteurs, nous ne l'expliciterons pas ici en raison du nombre important de paramètres (accélération, repère polaire, vitesse...).  $h$  correspond à la fonction qui crée une relation entre la valeur mesurée et celle estimée dans la phase précédente, par exemple :

- $h_{encoders}(x_k)$  représente le lien entre la vitesse état et la vitesse mesurée par les roues
- $h_{imu}(x_k)$  représente la relation entre l'accélération de l'état et la valeur d'accélération mesurée
- $h_{lidar}(x_k)$  représente la projection des obstacles dans le repère

Encore une fois ces fonctions sont générées automatiquement par le module ROS et ne seront pas détaillées ici, mais il est possible de les modifier. De la même façon que précédemment  $R_k$  peut être ajusté à chaque pas mais nous l'avons désigné comme constant.

### Étape de prédiction :

En reprenant notre modèle physique, on obtient l'estimation de l'état actuel à partir de l'état précédent estimé.

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$$

A partir de maintenant, on essaye d'estimer la covariance à chaque étape. La covariance permet de savoir à quel point notre estimation est correcte. Comme  $f$  est non linéaire, on la linéarise en prenant son jacobien. En reprenant la définition de la covariance, on obtient :

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$$

où  $F_k$  est le Jacobien de  $f$  par rapport à  $x$  :

$$F_k = \left. \frac{\partial f}{\partial x} \right|_{x=\hat{x}_{k-1|k-1}, u=u_k}$$

### Étape de mise à jour (correction) :

On cherche maintenant à calculer le *gain de Kalman*, ce gain a pour but de répondre à la question : dois-je faire davantage confiance à mon estimation ou aux mesures ? Intuitivement, si  $K_k$  est proche de la matrice identité, on fait confiance à la mesure, si  $K_k$  est proche de 0 alors on fait davantage confiance à l'estimation.

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (z_k - h(\hat{x}_{k|k-1}))$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

où  $H_k$  est le Jacobien de  $h$  par rapport à  $x$  :

$$H_k = \left. \frac{\partial h}{\partial x} \right|_{x=\hat{x}_{k|k-1}}$$

### Pour notre robot :

Dans notre cas, nous avons accès à deux capteurs d'odométrie : la ZED2 et l'odométrie des roues du robot. Nous avons également accès à l'IMU de la ZED2. Ces capteurs sont des capteurs de précisions, ils sont donc précis sur une courte distance mais diverge rapidement (de l'ordre de la dizaine de mètres). Notre "capteur fiable" ne peut être un GPS en raison de l'incapacité de recevoir un signal sous terre.

La triangulation par bornes WiFi ou autres signaux radio s'avère en pratique peu précise, même avec les meilleures technologies actuelles du marché. L'approche la plus prometteuse, qui reste à implémenter, consiste à scanner les étiquettes métalliques disposées approximativement tous les mètres dans les galeries, à lire leur code d'identification et à recaler la position du robot à partir de ces références fixes. Cette approche ayant déjà été théorisée, le TechLab a développé un modèle d'intelligence artificielle capable de détecter ces plaques avec une grande précision. Cependant, le défi de la reconnaissance optique de caractères (OCR) pour lire les codes d'identification reste à résoudre.

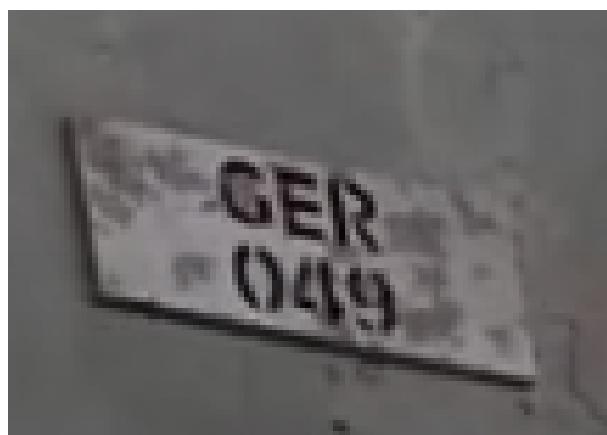


FIGURE 18 – Exemple d'une plaque en fer présente dans les galeries

Après estimation de la position, le package ROS publie son estimation sur le topic /odometry/filtered.

#### 4.2.3 Algorithme de SLAM

Un algorithme de SLAM (Simultaneous Localization and Mapping) est une méthode utilisée en robotique et en vision par ordinateur permettant à un robot ou à un dispositif mobile de se localiser dans un environnement inconnu tout en construisant une carte de cet environnement en temps réel. [11] [3]

L'idée principale derrière le SLAM est que, dans des environnements inconnus ou non prédéfinis, le robot doit simultanément résoudre deux problèmes :

1. **La localisation** : Où le robot se trouve-t-il dans son environnement ?
2. **La cartographie** : Quelle est la structure de cet environnement (la carte) ?

Le robot utilise le LiDAR, pour collecter des informations sur son environnement.

**Estimation de la position** : À partir des informations collectées par les capteurs et le filtre de Kalman, le robot tente de déterminer sa position actuelle en comparant les nouvelles données à celles qu'il a déjà recueillies.

**Construction de la carte** : En plus de se localiser, le robot génère une carte à l'aide du LiDAR (souvent une carte de points, ou une carte 2D ou 3D) représentant les objets et les structures qui l'entourent.

**Correction et mise à jour** : Comme la localisation du robot et la construction de la carte sont réalisées simultanément, chaque nouvelle mesure obtenue permet de réajuster l'estimation de la position et d'améliorer la carte.

Le problème de SLAM est complexe car il s'agit d'un problème non linéaire : les erreurs dans la localisation peuvent influencer la carte, et les erreurs dans la carte peuvent influencer la localisation. Il existe donc un compromis à gérer entre la précision de la carte et celle de la localisation.

Dans notre cas, nous appliquons le filtre de Kalman étendu présenté précédemment afin d'estimer l'état du robot et de corriger ses erreurs. Cette méthode, connue sous le nom d'EKF-SLAM (Extended Kalman Filter SLAM), permet de simplifier la complexité inhérente à la problématique de la localisation. En effet, le SLAM repose sur un paradoxe : pour se localiser, le robot a besoin d'une carte, mais pour construire cette carte, il doit connaître sa position. L'utilisation du filtre de Kalman permet de traiter le problème de localisation de manière quasi indépendante, atténuant ainsi cette interdépendance.

Lors de sa cartographie, le package publie sa carte sur le topic /map et la met à jour toutes les 10 secondes.

#### 4.2.4 Algorithme AMCL

Après avoir cartographié la carte en 2D (ou en avoir créé une manuellement), on peut utiliser l'algorithme AMCL (Adaptive Monte Carlo Localisation) pour estimer la position sur cette carte.

Voici en bref l'exécution de l'algorithme :

- On génère un grand nombre de particules uniformément, chaque particule représente une potentielle position et orientation du robot
- Toutes les particules se déplacent ensuite en suivant l'odométrie du robot
- Chaque particule reçoit un poids en fonction de la ressemblance entre les données de capteur LiDAR et ce que le robot devrait percevoir depuis la position hypothétique de la particule.
- Les particules avec des poids plus élevés sont rééchantillonnées plus fréquemment, celles avec des poids faibles sont éliminées.

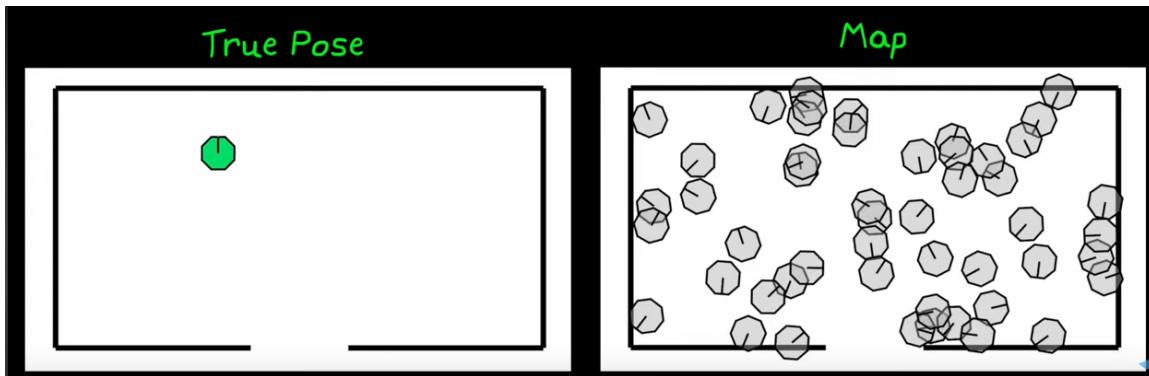


FIGURE 19 – Initialisation de l'algorithme, chaque particule est donnée le même poids [4]

Cet algorithme présente l'avantage de fonctionner correctement malgré les imperfections de notre odométrie. En effet, comme mentionné précédemment, le problème du SLAM réside dans le fait que l'algorithme repose sur une localisation fiable qui ne doit pas dériver excessivement.

Ainsi, avec AMCL, en fournissant une carte préétablie, il suffit de résoudre la partie "localisation" et les erreurs d'odométrie sont compensées par les données du LiDAR.

En pratique, pour la carte, nous utilisons un fichier au format .pgm, qui est un format d'image en niveaux de gris où chaque pixel est codé par une valeur entre 0 et 255. Dans notre cas, 0 représente un espace libre (navigable) et 255 représente un obstacle (mur). Nous utilisons un package ROS pour publier cette carte sur le topic /map. Ce package prend en paramètres un fichier de configuration YAML précisant notamment la résolution de l'image (une résolution de 0,05 signifie qu'un pixel représente 5 cm dans l'environnement)

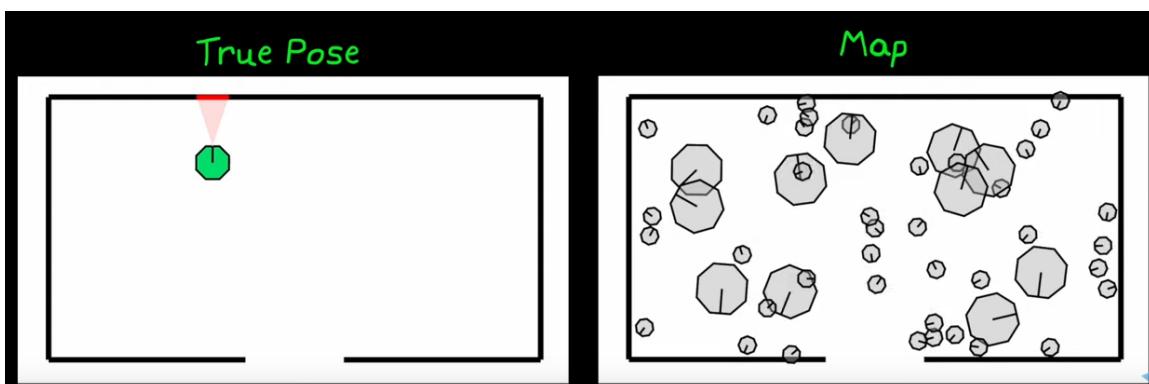


FIGURE 20 – Après comparaison des données du LiDAR et de celles perçues par chaque particule, les poids sont ajustés pour privilégier celles qui sont les plus cohérentes.

réel) et l'origine du repère de coordonnées (par défaut, l'origine est positionnée en bas à gauche de l'image). [5]

### 4.3 Intégration de l'IA et affichage des fissures

En complément de la partie localisation, nous avons intégré notre modèle d'intelligence artificielle de détection de fissures dans un nœud ROS (Robot Operating System), afin de rendre le robot plus autonome et efficace dans sa mission. Un premier noeud s'occupe de capturer automatiquement des images toutes les 10 à 20 secondes depuis la caméra PTZ montée sur le robot, puis publie ces images sur un topic ROS dédié. Le nœud contenant l'IA s'abonne à ce topic pour recevoir les images, qui sont transmises simultanément avec la position précise du robot au moment de la capture. Cette synchronisation garantit une correspondance directe entre l'image et la localisation. Le nœud de détection récupère donc chaque photo accompagnée de ses coordonnées, et applique le modèle d'intelligence artificielle pour identifier les fissures potentielles.

Dans un second temps, afin de rendre visuelle la détection des fissures sur une carte, nous avons développé un autre nœud ROS, report\_fissures.py. À la réception de cette position, une fonction dédiée trace un point rouge sur une carte statique de la galerie souterraine, fournie en amont sous forme d'image de référence. Cette carte du couloir ou de la galerie sert de fond, sur lequel sont superposées dynamiquement les annotations correspondant aux fissures détectées. Pour garantir la précision du placement, le système de localisation du robot fournit la position en mètres par rapport à un repère initial, dont les coordonnées sont définies manuellement. En effet, le robot ne peut pas cartographier son environnement ni se localiser de manière autonome, rendant indispensable la définition préalable de sa position de départ sur la carte. Un mécanisme de translation et d'ajustement d'échelle convertit ensuite ces coordonnées métriques en pixels, garantissant ainsi un positionnement exact sur l'image. Par ailleurs, l'origine de l'image est prise en compte en fonction du format de lecture utilisé (par exemple, IrfanView ou Matplotlib), afin d'assurer une correspondance rigoureuse entre la position réelle du robot et l'affichage graphique. Ce processus complet permet la génération en temps réel d'une carte annotée, précieuse pour le suivi, la maintenance et la planification des interventions sur les zones endommagées.

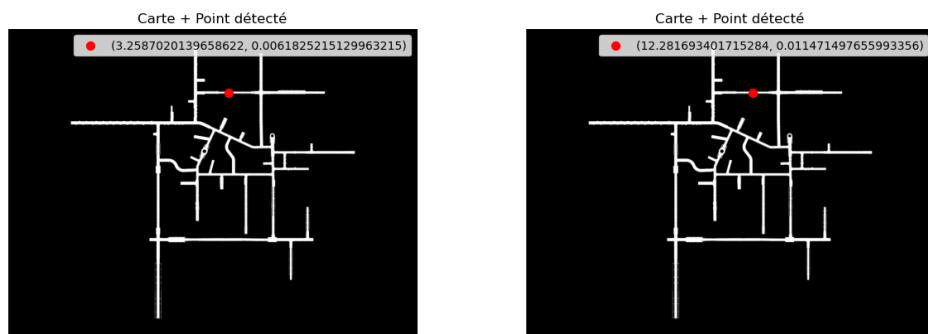


FIGURE 21 – Exemples de fissures détectées et placées sur la carte du laboratoire

Le système est ainsi conçu de manière modulaire, chaque composant fonctionnant de manière indépendante mais parfaitement coordonnée via les topics ROS, ce qui nous offre

une architecture flexible, robuste et évolutive. La synchronisation entre la prise d'image, l'analyse par l'intelligence artificielle et la localisation est gérée grâce aux timestamps ROS et à la gestion précise des messages, assurant une cohérence temporelle et spatiale dans l'ensemble du processus.

## 5. Gestion de projet

Pour gérer efficacement ce projet, nous avons mis en place une organisation basée sur une répartition claire des rôles et des tâches. Chaque membre prenait en charge des parties différentes du projet (par exemple la détection d'étiquettes, la navigation robotique, l'intégration de la caméra PTZ, etc.). Ensuite, nous mettions en commun nos avancées lors de réunions hebdomadaires pour discuter des résultats, des difficultés rencontrées et préparer les prochaines étapes ensemble. Cette coordination nous permettait d'ajuster le travail collectif et de nous assurer que les différentes parties du projet restaient cohérentes. Ces échanges fréquents étaient indispensables pour gérer les imprévus, comme les difficultés techniques liées au robot ou au code incomplet, ou encore les problèmes matériels. Par exemple, lors des réunions du 8 novembre et du 15 novembre, nous avons identifié des problèmes avec le modèle de détection d'étiquettes et ajusté nos objectifs en conséquence.

Pour structurer notre planification, nous avons réalisé un diagramme de Gantt par semestre. Ce choix s'est imposé notamment après la réunion du 16 décembre, durant laquelle les objectifs du projet ont été revus en collaboration avec l'entreprise. Ces diagrammes semestriels nous ont permis de mieux visualiser les différentes phases, d'adapter nos échéances et de réorganiser les tâches en fonction des nouveaux objectifs définis. Ils ont ainsi facilité le suivi de l'avancement et la gestion des priorités tout au long du projet.

On voit par exemple que lors du second semestre, notre travail s'est concentré sur la prise en main de ROS 2, puisque l'API propre au Techlab, qui devait faciliter l'intégration, n'était pas encore opérationnelle. Nous avons donc dû adapter notre approche et développer nos propres services pour la communication entre les différents modules du robot. Cette phase a été essentielle pour préparer la descente sur site à Bure du 16 mai, où l'objectif était d'avoir un robot capable de parcourir un couloir et de détecter les fissures tout en les plaçant précisément sur une carte.

Suite à cette descente, nous avons principalement focalisé nos efforts sur la rédaction collective du rapport, en tirant les enseignements des tests réalisés sur le terrain. Nous avons aussi commencé à préparer la suite du projet, notamment en anticipant une possible nouvelle descente le 13 juin, en vue d'améliorer les performances du robot et de l'IA sur site. Cette organisation nous a permis de maintenir une dynamique collective et de rester alignés sur les objectifs à moyen terme.

Après avoir présenté notre organisation, la répartition des tâches et la planification du projet, il est important de faire un point sur les forces et les faiblesses rencontrées ainsi que sur les opportunités et menaces qui ont influencé le déroulement de notre travail. Pour cela, nous avons réalisé une matrice SWOT, qui nous permet d'analyser de manière synthétique les différents aspects internes et externes impactant le projet.

Comme vu dans la matrice SWOT, l'un des points faibles identifiés tout au long du projet

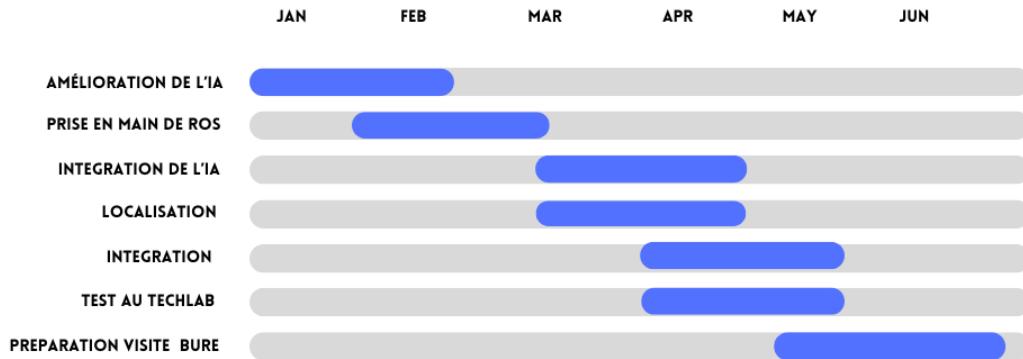


FIGURE 22 – Diagramme de Gantt du second semestre

a été le manque de continuité entre les différentes générations d'étudiants. Cette absence de transmission a constitué un frein important à notre progression. Pour y remédier et améliorer la gestion du projet pour les années à venir, nous avons décidé de rédiger un document récapitulatif et explicatif de l'ensemble de nos codes, détaillant leur fonctionnement et leur utilisation. Ce document a pour objectif de faciliter la reprise du projet par les futurs groupes. Malgré les difficultés techniques et matérielles rencontrées, c'est grâce à une bonne communication et une collaboration efficace au sein de notre équipe que nous avons pu avancer sur nos objectifs et maintenir une dynamique de travail constructive.

## 6. Résultats

Après avoir combiné l'ensemble des techniques décrites précédemment, nous obtenons un robot qui est capable de détecter les fissures et les placer sur une carte en 2D. Cela a notamment été démontré lors de notre seconde descente.

Du côté de l'IA, nous sommes particulièrement satisfait de ses performances (voir figure 24). En effet, dans la grande majorité des cas, les fissures sont correctement détectées et sont correctement délimitées. En revanche, la caméra a besoin d'être assez proche du mur (moins de deux mètres) pour réaliser une analyse correcte. En effet, étant donné que l'IA prend en entrée des images qui ont une résolution bien moindre (640x480), il faut que la compression de l'image n'empêche pas de pouvoir distinguer une fissure. Notons

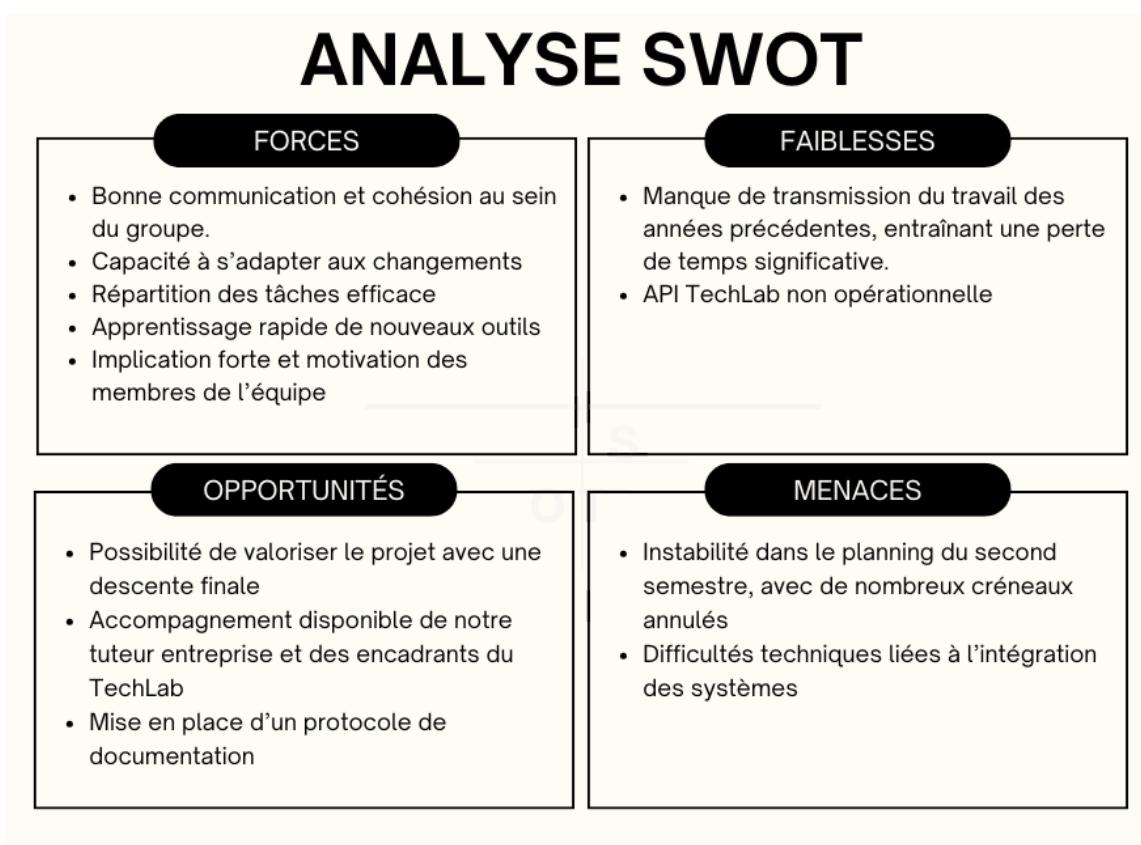


FIGURE 23 – Matrice SWOT du projet

également que l'auto-focus de la caméra est particulièrement sensible aux vibrations, il est donc nécessaire que le robot soit à l'arrêt lorsque la photo est prise. De plus, le modèle d'IA est relativement lent à analyser une image (5 sec pour le moment), ce qui n'est encore une fois pas particulièrement dérangeant si l'on suppose que le robot sera déployé pendant des temps où le site ne sera pas très occupé.

Du côté de la localisation, il y a encore beaucoup de progrès à réaliser. Malgré la fusion des données, le matériel n'étant pas de très haut de gamme, le robot est particulièrement sensible au drift. Ainsi, dépendant des poids donnés au filtre de Kalman, la localisation peut fonctionner de façon plus efficace suivant les situations. Par exemple, si l'on décide de donner beaucoup d'importance à l'odométrie des roues, le positionnement est précis lorsque le robot est en ligne droite mais dérape totalement lorsque ce dernier tourne. A l'inverse, un poids important sur la ZED2 donne généralement une meilleure précision sur la vitesse angulaire mais dérive rapidement sur de longues distances. L'amélioration décrite en section d'explication du filtre de Kalman semble être la piste la plus prometteuse pour le moment s'il est décidé de rester sur ce robot.

A cause du drift sur la localisation à l'aide du filtre de Kalman, la cartographie par SLAM est inexploitable. Cependant, tout à déjà été implémenté, il ne "reste" plus qu'à améliorer le positionnement.

Cependant, avec AMCL (donc avec une carte prédéfinie), la localisation se retrouve grandement améliorée, particulièrement si l'espace possède des asymétries. Le scan de l'environnement par le LiDAR empêche alors au robot de trop dériver. C'est cet algorithme qui

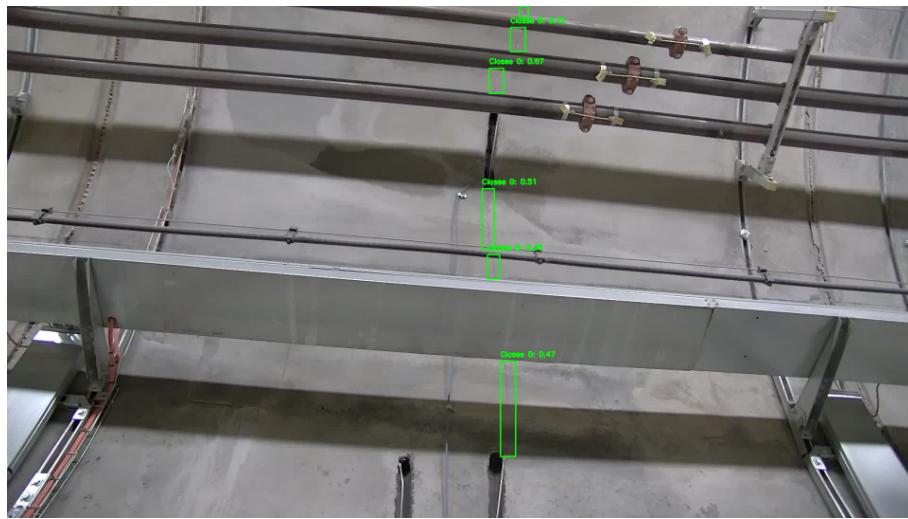


FIGURE 24 – Image prise directement durant la descente du 16/05 depuis le robot avec détections de fissures

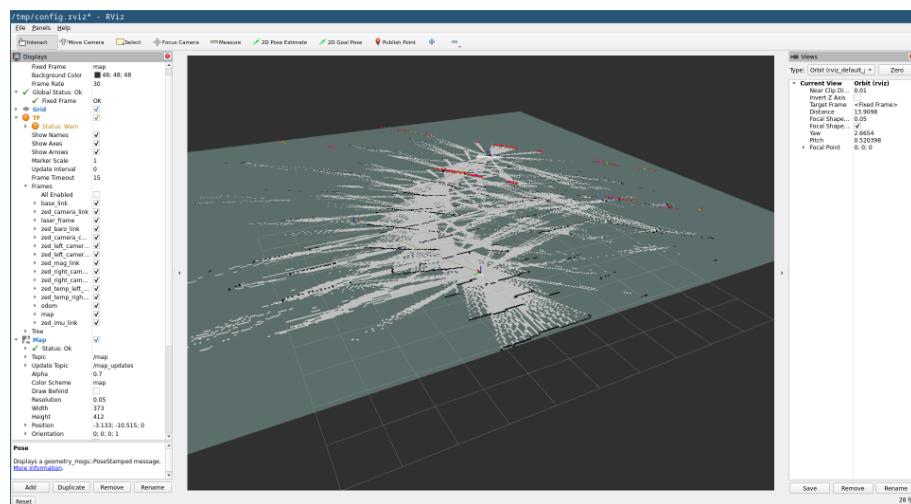


FIGURE 25 – Exemple de cartographie sur un des couloirs

avait été testé lors de la seconde descente et qui donnait des résultats très prometteur sur au moins 15 mètres.

## 7. Conclusion et Ouvertures

Pour aller plus loin et franchir un nouveau cap, plusieurs axes d'amélioration apparaissent comme prioritaires. Le premier concerne la localisation du robot, encore trop sujette au drift malgré les efforts de fusion de capteurs. Un changement de plateforme, avec un modèle mieux adapté à la navigation de précision, équipé par exemple de roues odométriques plus fiables ou d'un système de positionnement visuel, permettrait d'améliorer significativement la stabilité des estimations de position.

L'intégration d'un système de gestion de mission permettrait quant à elle d'automatiser des séquences complexes et de mieux structurer la répartition des tâches entre les modules

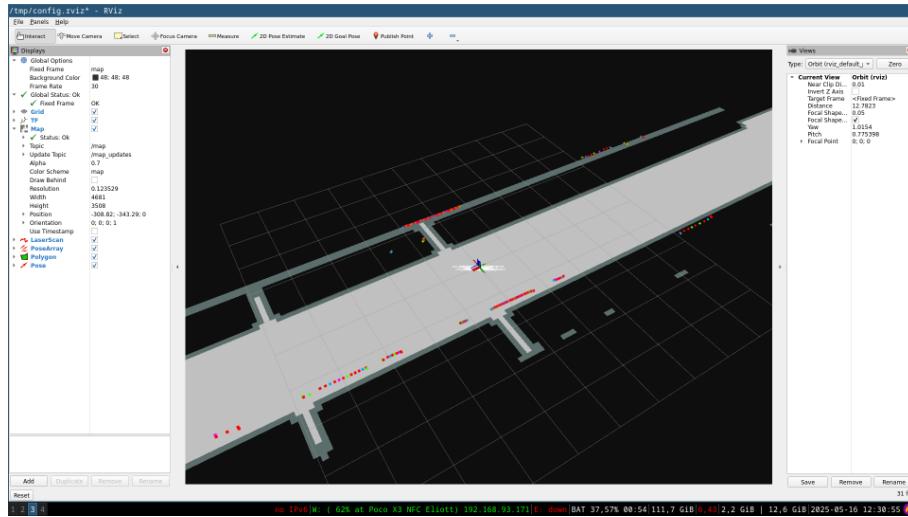


FIGURE 26 – Example de localisation en utilisant AMCL lors de notre descente

(navigation, détection, enregistrement, etc.), rendant le robot plus autonome et efficace.

Enfin, pour assurer la pérennité du projet, il est essentiel de garantir une vraie continuité entre les promotions. Cela passe par une documentation complète, un code maintenu proprement, et surtout par une transmission active entre équipes, avec des retours d'expérience partagés. Ce projet a le potentiel de devenir une référence pédagogique et technique, à condition d'être construit comme un héritage évolutif plutôt que comme un recommencement perpétuel.

## Références

- [1] Tim BAILEY et al. « Consistency of the EKF-SLAM Algorithm ». In : *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, p. 3562-3568. DOI : 10.1109/IROS.2006.281644.
- [2] Steven MACENSKI et al. « Robot Operating System 2 : Design, architecture, and uses in the wild ». In : *Science Robotics* 7.66 (2022), eabm6074. DOI : 10.1126/scirobotics.abm6074. URL : <https://www.science.org/doi/10.1126/scirobotics.abm6074>.
- [3] MATHWORKS. *What Is SLAM?* MathWorks Inc. 2024. URL : <https://www.mathworks.com/discovery/slam.html> (visité le 30/05/2025).
- [4] MATLAB. *Comprendre le filtre à particules / Navigation autonome, partie 2*. YouTube. 2020. URL : [https://www.youtube.com/watch?v=Nrzmh\\_yerBU&t](https://www.youtube.com/watch?v=Nrzmh_yerBU&t) (visité le 01/07/2020).
- [5] NAVIGATION2 PROJECT. *Configuring AMCL*. <https://docs.nav2.org/configuration/packages/configuring-amcl.html>. Accessed : May 30, 2025. 2024.
- [6] Agilex ROBOTICS. *scout\_ros2*. Accessed : 2025-05-30. 2023. URL : [https://github.com/agilexrobotics/scout\\_ros2](https://github.com/agilexrobotics/scout_ros2).
- [7] STEREO LABS. *zed\_ros2\_wrapper*. Accessed : 2025-05-30. 2023. URL : <https://github.com/stereolabs/zed-ros2-wrapper>.

- 
- [8] UNIVERSITY. *crack Dataset*. Open Source Dataset. visited on 2024-01-23. Déc. 2022. URL : <https://universe.roboflow.com/university-bswxt/crack-bphdr>.
  - [9] WIKIPÉDIA. *Filtre de Kalman — Wikipédia, l'encyclopédie libre*. [En ligne ; Page disponible le 4-décembre-2024]. 2024. URL : [http://fr.wikipedia.org/w/index.php?title=Filtre\\_de\\_Kalman&oldid=220858817](http://fr.wikipedia.org/w/index.php?title=Filtre_de_Kalman&oldid=220858817).
  - [10] WIKIPEDIA CONTRIBUTORS. *Real-Time Streaming Protocol — Wikipedia, The Free Encyclopedia*. [Online ; accessed 30-May-2025]. 2025. URL : [https://en.wikipedia.org/w/index.php?title=Real-Time\\_Streaming\\_Protocol&oldid=1285058072](https://en.wikipedia.org/w/index.php?title=Real-Time_Streaming_Protocol&oldid=1285058072).
  - [11] WIKIPEDIA CONTRIBUTORS. *Simultaneous localization and mapping — Wikipedia, The Free Encyclopedia*. [Online ; accessed 30-May-2025]. 2025. URL : [https://en.wikipedia.org/w/index.php?title=Simultaneous\\_localization\\_and\\_mapping&oldid=1282391263](https://en.wikipedia.org/w/index.php?title=Simultaneous_localization_and_mapping&oldid=1282391263).
  - [12] YDLIDAR. *ydlidar\_ros2\_driver*. Accessed : 2025-05-30. 2023. URL : [https://github.com/YDLIDAR/ydlidar\\_ros2\\_driver](https://github.com/YDLIDAR/ydlidar_ros2_driver).