POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

Documentation du projet répondant à l'appel d'offres no. A2024-INF3995 du département GIGL.

Conception d'un système aérien d'exploration

Équipe No **105**

Marc Jodel Dumesle ALCINDOR	2000081
Bailly Jonathan Clovis YARO	2128882
Zakarya KHNISSI	1989641
Samuel KOUAKOU	2068795
Axelle LETIEU	2152361

Septembre 2024

1. Vue d'ensemble du projet

1.1 But du projet, porté et objectifs (Q4.1)

Le but de ce projet est de concevoir et de développer une solution technique adaptée à l'appel d'offres proposé par l'Agence Spatiale, visant à mettre en place un système d'exploration multi-robot autonome. Ce système permettra à une équipe de deux robots d'explorer une pièce d'un bâtiment de taille moyenne en utilisant une caméra et un capteur laser, tout en rapportant les informations recueillies à une interface opérateur basée sur le Web. Les membres de l'équipe projet auront pour objectif de développer cette solution durant la période impartie, en s'appuyant sur des concepts clés acquis dans les projets intégrateurs précédents, notamment en gestion de projet, développement logiciel embarqué et robotique. La planification sera principalement contrainte par les livrables à produire selon l'échéancier établi, tout en intégrant les nouvelles technologies nécessaires à l'implémentation des fonctionnalités demandées.

Un autre objectif majeur de ce projet est d'acquérir et approfondir les connaissances techniques liées à l'interaction entre la partie embarquée des robots, la station au sol et la simulation des systèmes. Cela comprend l'intégration d'un simulateur pour tester les fonctionnalités avant leur déploiement sur les robots physiques. Le projet permettra également de professionnaliser le travail à travers la documentation, la gestion du code, l'estimation des coûts et la mise en place d'un calendrier de développement détaillé, suivant les exigences de l'appel d'offres. Les livrables attendus comprendront une version partielle du système avec des fonctionnalités de base, suivie d'une version finale où toutes les fonctionnalités demandées seront implémentées et prêtes à être déployées sur des robots physiques.

1.2 Hypothèse et contraintes (Q3.1)

On peut identifier plusieurs hypothèses et contraintes liées à ce projet. Sur le plan technique on a posé les hypothèses suivantes:

- Pour la simulation et le déploiement physique, on a supposé que le terrain sera plat et sans élévations.
- Pour la simulation, un ensemble de mur est utilisé pour simuler les obstacles
- Pour le déploiement physique, les obstacles sont des boîtes ainsi que des arbres en plastique.
- L'environnement de la simulation ainsi que du déploiement physique sont tout les deux éclairés
- Finalement, on a posé l'hypothèse que lors de la simulation et du déploiement physique, les robots ainsi que la station au sol seront sur le même réseau WIFI

Ensuite, sur les autres plans, il est attendu que l'équipe de développement, bien que confrontée à des défis techniques et à de nouvelles technologies, soit capable de proposer une solution satisfaisante pour l'Agence. Cela demandera aux membres de l'équipe d'acquérir rapidement des compétences techniques spécifiques, notamment en ce qui concerne l'utilisation des robots AgileX Limo et des technologies associées. De plus, les spécifications fournies par l'Agence doivent être suffisamment claires et détaillées pour éviter toute ambiguïté lors du développement. Les exigences sont énoncées dans un document officiel, garantissant ainsi que l'équipe se concentre sur les aspects cruciaux du projet sans perdre de temps en clarifications inutiles. Le temps imparti à l'équipe pour réaliser ce projet a été jugé suffisant, à condition qu'elle adopte une organisation efficace, incluant un travail collaboratif en dehors des heures de cours et des laboratoires afin de respecter les délais.

En ce qui concerne les contraintes, elles sont nombreuses et strictes. Les contraintes qui suivent sont imposées par l'appel d'offre. Le prototype doit impérativement être implémenté en utilisant les deux robots AgileX Limo fournis par l'Agence, sans possibilité de déroger à cette règle. Ensuite, la communication entre la station au sol et les robots physiques doit obligatoirement passer par un réseau WiFi fourni par l'Agence, ce qui impose une contrainte technique importante. Les robots ne peuvent utiliser que les capteurs installés par l'Agence, à savoir l'IMU, la caméra 3D, la caméra RGB et le lidar, pour accomplir leurs tâches. La station au sol, pour sa part, devra être un ordinateur portable ou un PC, sans autres options matérielles.

Sur le plan logiciel, les robots doivent obligatoirement être programmés sous Ubuntu, installé sur leur ordinateur de bord, bien que l'usage de machines virtuelles, comme Docker, soit autorisé pour faciliter le développement. L'interface utilisateur utilisée sur la station au sol doit être identique, que ce soit pour les robots physiques ou pour les

robots simulés, avec des ajustements uniquement pour les fonctionnalités spécifiques à chaque système. Le contrôle des robots se fera entièrement à bord de ces derniers, et la station au sol ne devra envoyer que des commandes de haut niveau, telles que le lancement de missions ou des mises à jour, sans dicter les mouvements précis des robots. Enfin, toutes les composantes logicielles, à l'exception de celles embarquées sur les robots physiques, devront être conteneurisées avec Docker, afin de garantir la reproductibilité du système, d'éviter les problèmes de compatibilité, de faciliter le déploiement, et d'assurer une évaluation standardisée et équitable pour tous les étudiants.

Ces contraintes imposent un cadre strict au projet, mais garantissent que le produit final sera robuste, reproductible et adapté aux exigences de l'Agence.

En dehors des contraintes imposées par l'appel d'offre, en tant que contracteur nous sommes aussi notamment soumis à des contraintes, notamment:

- L'impossibilité de sortir les robots physique de la salle M7703
- L'espace de la salle M7703 devant être partagé avec d'autres contracteurs
- Les pannes des robots physiques nécessitant l'intervention des chargés
- La nécessité de réserver la salle M7703 avant d'y avoir accès.
- La limite de 3H pour les réservations de la volière

1.3 Biens livrables du projet (Q4.1)

Le projet sera divisé en trois livrables, conformément aux directives de l'Agence. Le premier livrable, attendu pour le vendredi 20 septembre 2024, correspondra à la Preliminary Design Review (PDR). À ce stade, l'équipe devra remettre un prototype préliminaire, basé sur la proposition initiale soumise en réponse à l'appel d'offres. Ce prototype démontre les fonctionnalités de base du système, mais ne sera pas encore complet.

Ensuite, le vendredi 1er novembre 2024, le deuxième livrable marquera la Critical Design Review (CDR). Ce livrable présente un système partiellement fonctionnel, tel que décrit dans le document "Système d'exploration multi-robot : Exigences techniques". Le système devra inclure certaines fonctionnalités clés, mais ne sera pas encore entièrement finalisé.

Enfin, le mardi 3 décembre 2024, le dernier livrable sera évalué lors de la Readiness Review (RR), marquant ainsi la fin du projet. À cette étape, l'équipe devra remettre un produit final entièrement fonctionnel, incluant toutes les fonctionnalités décrites dans le document d'exigences techniques. Ce livrable devra être complet et prêt pour une utilisation réelle, conformément aux attentes de l'Agence.

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Pour le projet de cette session, après plusieurs réunions d'équipe, nous avons opté pour une structure d'organisation collaborative et flexible. En effet, afin d'assurer une répartition équitable des tâches, mais aussi pour veiller à ce que toute l'équipe contribue pleinement au projet, nous avons séparé les tâches en deux sous catégories: techniques et administratives.

Certains membres de l'équipe seront chargés de la coordination et de la gestion du projet. L'objectif avec ce rôle est de nous assurer que les réunions hebdomadaires, les échéances et la répartition des tâches soient respectées. Ils auront également la responsabilité des rapports d'avancement hebdomadaires à communiquer aux responsables tous les lundis avant 9h. Il s'agit d'un rôle important afin d'assurer la progression efficace du projet.

Une autre partie des membres de l'équipe sera en charge du développement de la station au sol. Cela inclut le développement de la partie client, serveur, la création et la gestion de la base de données associée, mais également la liaison avec le logiciel embarqué.

D'autres membres travaillent en parfaite collaboration sur le code de la partie embarquée afin d'implémenter les fonctionnalités demandées mais aussi afin de veiller à optimiser continuellement les algorithmes. Enfin, toutes les fonctionnalités implémentées aussi bien sur la partie logiciel embarquée seront testées et validées par certains membres de l'équipe.
Le tableau ci-dessous illustre précisément et en détail les assignations des rôles et les responsabilités qui en découlent.

Rôle	Assignation	Responsabilité
Coordination et Gestion	● Alcindor	Gérer les tâches et les échéances, documenter les rapports.
Développement de la station au sol	● Yaro ● Alcindor ● Kouakou ● Khnissi ● Letieu	Développer le client et le serveur. Créer de la base de données. Gérer les interactions client-serveur, serveur et logiciel embarqué.
Développement de la partie embarquée	● Yaro ● Alcindor ● Kouakou ● Khnissi ● Letieu	Implémenter des noeux pour la navigation et le bon fonctionnement de la mission du robot.
Test et validation	● Alcindor ● Kouakou ● Yaro	Implémenter les tests pour les algorithmes. Assurer le bon fonctionnement des requis.

Figure 2: Assignations des rôles des membres de l'équipe

2.2 Entente contractuelle (Q11.1)

Pour ce projet, nous avons opté pour un contrat de livraison clé en main avec un prix fixe. Ce choix signifie que notre équipe prend en charge la livraison complète du produit, et le paiement final est effectué à la réception et validation du produit par le client. Ce type de contrat présente plusieurs avantages : il assure le contrôle du budget et limite les risques de dépassement de coûts, ce qui sécurise le client face aux imprévus financiers. En outre, le client bénéficie d'une gestion simplifiée, car le suivi du développement est allégé. Si des ajustements ou fluctuations de coûts surviennent, le client est immédiatement informé pour limiter tout retard.

Bien que ce type d'entente exige une planification détaillée et un suivi rigoureux de notre part, il garantit une livraison conforme aux délais et aux coûts prévus (voir les sections **Échéancier** et **Budget**).

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)

La figure ci-dessous montre l'architecture générale du système. L'interface utilisateur, développée en Angular, permet aux utilisateurs de démarrer des missions et de suivre en temps réel l'état des robots via une communication en HTTP et WebSocket avec le serveur NestJS. Ce serveur joue le rôle central, gérant les missions et assurant la connexion directe avec les nœuds ROS 2 des robots physiques et de la simulation. Les informations sur l'état des robots et les logs de mission sont gérées et diffusées en temps réel via ce canal WebSocket, assurant ainsi une communication continue et réactive.

Le choix de retirer Rosbridge pour se connecter directement aux nœuds ROS 2 offre plusieurs avantages. Cette approche réduit la latence et simplifie l'architecture en supprimant une couche intermédiaire, ce qui améliore la réactivité et la fiabilité du système. La connexion directe permet une meilleure maîtrise des données échangées et optimise l'accès aux services ROS 2.

Tous les composants, notamment l'interface utilisateur, le serveur NestJS et les nœuds ROS 2, sont déployés dans des conteneurs Docker orchestrés par Docker Compose. Cette approche assure un environnement standardisé et reproductible.

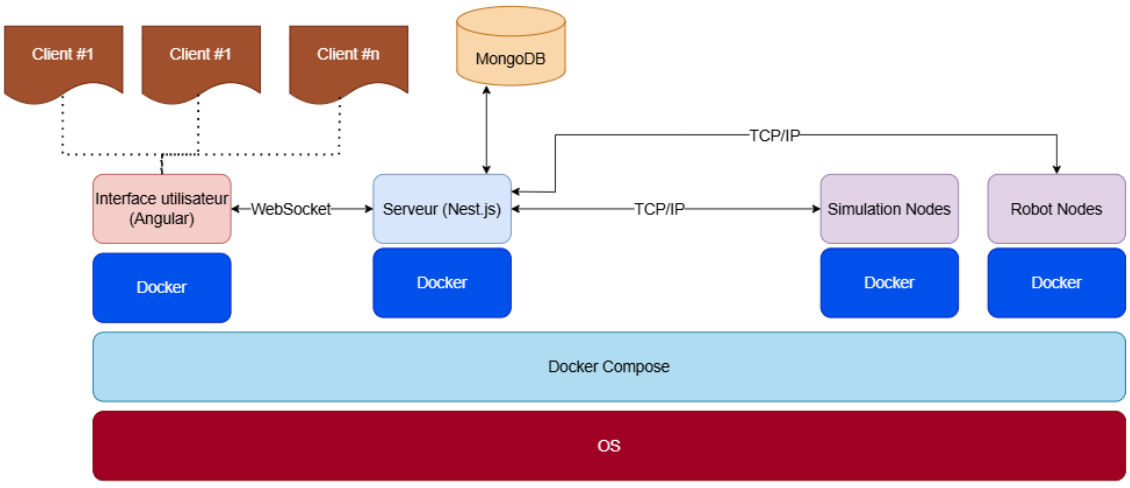

Figure 2: Diagramme de l'architecture logicielle générale

3.2 Station au sol (Q4.5)

Pour le système de contrôle des robots physiques et simulés, nous avons adopté une architecture modulaire reposant sur des contrôleurs indépendants, chacun responsable d'une fonctionnalité spécifique, facilitant ainsi la gestion des missions, la communication avec les robots et la surveillance des logs.

Chaque contrôleur assure un rôle spécifique :

- **Mission Controller** gère l'initiation, le suivi et la gestion des missions.
- **Robot Controller** centralise le contrôle des robots physiques.
- **Simulation Controller** gère les interactions spécifiques à l'environnement de simulation
- **Logs Controller** est dédié à la collecte et au stockage des journaux de mission, permettant un suivi historique des événements importants.

Ces contrôleurs communiquent directement avec **RosService**, qui se connecte aux nœuds ROS 2 sans passer par un serveur Rosbridge. Ce choix de conception optimise la performance en supprimant les couches intermédiaires, offrant ainsi une communication plus rapide et une gestion directe des services ROS.

Les données des missions et les journaux sont centralisés dans MongoDB à travers le **MongoDB Service**, qui offre une interface pour stocker et récupérer les informations critiques telles que les cartes de navigation et les logs des missions. Cette centralisation permet de consulter l'historique des missions et les événements de suivi via l'interface Angular.

Nous avons choisi **MongoDB** pour sa flexibilité et sa capacité à gérer des données non structurées, ce qui est idéal pour les besoins variés du système de contrôle des robots. De plus, l'ensemble de l'équipe a eu l'occasion de l'utiliser dans le cadre de précédents projets comme le projet intégrateur 2. **MongoDB** permet de stocker efficacement les données de mission, les journaux, et les cartes de navigation sous forme de documents **JSON**, facilitant ainsi l'intégration avec les services **ROS** qui génèrent des données semi-structurées. Grâce à sa nature orientée document, **MongoDB** offre une grande flexibilité dans l'évolution du modèle de données, ce qui est essentiel pour un système en constante évolution comme le nôtre, où de nouvelles fonctionnalités et types de données peuvent être ajoutés sans avoir à modifier les structures existantes.

De plus, **MongoDB** est conçu pour supporter des requêtes rapides et évolutives, ce qui est particulièrement avantageux pour les applications en temps réel. En centralisant les

données critiques dans **MongoDB**, le **MongoDB Service** peut facilement répondre aux demandes de l'interface **Angular**, permettant une consultation rapide de l'historique des missions et des événements de suivi. Cette structure de données centralisée assure ainsi une consultation rapide des informations historiques et des journaux de mission, optimisant la performance globale du système et répondant aux besoins de gestion et de surveillance en temps réel.

Les informations collectées sont diffusées en temps réel via **SyncGateway**, qui utilise **Socket.IO** pour assurer une communication fluide avec l’interface utilisateur basée sur le protocole WebSocket. Cela permet à l'utilisateur, via une interface Angular, de visualiser l'état des missions, d'interagir avec les robots et de consulter les journaux en temps réel.

En résumé, cette architecture modulaire et découplée, facilitée par l’utilisation des contrôleurs, garantit une extensibilité et une maintenance simplifiée, tout en assurant des performances optimales et une synchronisation fluide entre les composants du système.

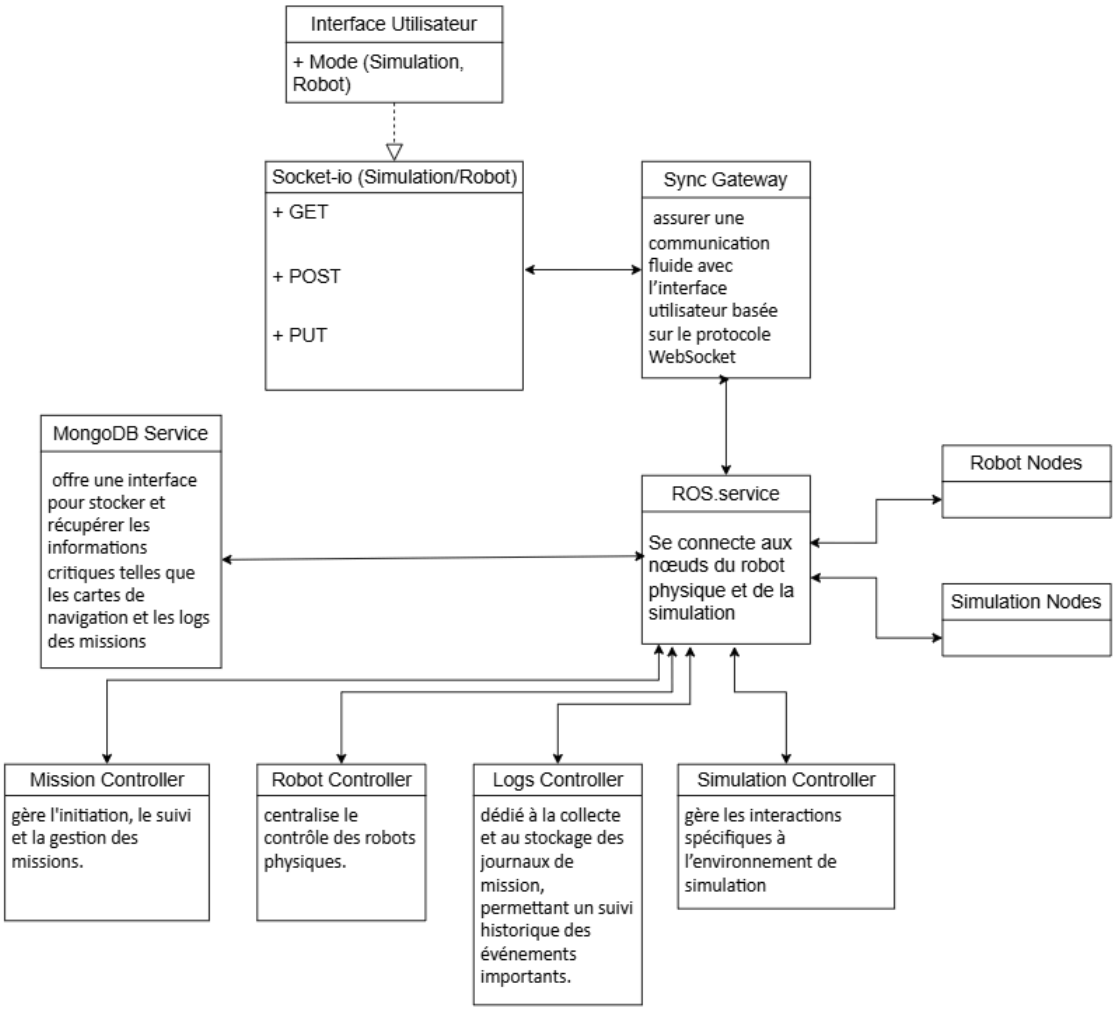

Figure 3: Diagramme de l'architecture de la station au sol

3.3 Logiciel embarqué (Q4.5)

L'architecture est organisée autour de plusieurs nœuds, chacun ayant des responsabilités distinctes pour gérer les capteurs, les missions, la navigation et les communications. Ces nœuds communiquent entre eux via le middleware ROS 2 pour coordonner les opérations et s'assurer que le robot exécute ses missions de manière efficace.

Nœuds de Capteurs

- **IMU Node** : Ce nœud reçoit les données de l'accéléromètre et du gyroscope pour surveiller l'orientation et la vitesse angulaire du robot.

- **RGB/3D Camera Node** : Capture des images en couleur et des données de profondeur 3D, permettant au robot d'avoir une perception visuelle de son environnement.
- **LiDAR Node** : Fournit une détection d'obstacles à travers des balayages laser, essentiels pour la navigation et l'évitement d'obstacles.

Ces capteurs transmettent leurs données au **SensorManager**.

SensorManager (Gestionnaire de Capteurs)

- **Fonction principale** : Le SensorManager centralise, filtre et transforme les données de capteurs avant de les transmettre aux autres nœuds qui en ont besoin. Cela garantit que les données sont prêtes pour une utilisation efficace dans la navigation et la planification des missions.

Nœud Map Merge

- **Fonction principale** : Ce nœud fusionne les maps publiées par chacun des robots afin de produire une map globale qui sera affiché sur le frontend.

Nœud SLAM Toolbox

- **Fonction principale** : **SLAM Toolbox** (Simultaneous Localization and Mapping) utilise les données des capteurs traitées par le SensorManager pour générer une carte en temps réel et déterminer la position du robot dans cet espace. Cela permet au robot de s'auto-localiser pendant qu'il explore l'environnement.

Nœuds Navigation2

- **Fonction principale** : Il s'agit d'un ensemble de nœud utilisé pour toutes les opérations de déplacement. Ces nœuds servent à l'exploration et l'évitement d'obstacles.

Nœuds de Commande et Mission

- **Limo_Base_Node** : C'est le nœud de base du robot, responsable de la gestion des commandes de mouvement **(cmd_vel)** et de la publication des données d'odométrie dans un namespace unique propre à chaque robot. Cela permet de gérer plusieurs robots sans conflits.
- **Identify Service** : Ce service permet au robot de signaler sa présence. Cela aide la station de contrôle à identifier et gérer plusieurs robots simultanément.

- **Mission Service** : Gère l'initialisation, le suivi et la fin des missions. Il surveille l'état du robot et ajuste les commandes en fonction de l'avancement de la mission.

Nœud Random Walker

- **Fonction principale** : Ce nœud initie des déplacements autonomes vers des points aléatoires en prenant en compte les obstacles détectés. Il envoie des positions aux nœuds de Navigation2 qui s'occupent d'effectuer le déplacement vers ceux-ci. Il s'agit d'un algorithme de type "Momentum Based Movement"

Nœud Explore_lite

- **Fonction principale** : Ce nœud initie des déplacements autonomes vers des points avec pour objectif se diriger vers les zones inexplorées. Il communique directement avec Navigation2 pour le déplacement. Il s'agit d'un algorithme de type "Frontier exploration"

ROSService

- **Fonction principale** : Ce service connecte tous les nœuds au middleware ROS 2, assurant la communication bidirectionnelle avec la station de contrôle. Cela permet de transmettre des commandes au robot et de recevoir des mises à jour en temps réel sur son état.

Les dépendances externes, notamment le **SLAM_TOOLBOX**, **Navigation2, Explore_Lite** et **Map Merge**. Ceux-ci ont été choisies pour leur efficacité éprouvée dans leurs domaines respectifs. Ces composants sont essentiels pour la navigation, la localisation et la fusion de cartes, et permettent au système de bénéficier d'une performance optimisée en termes de traitement des données de capteurs et de navigation, tout en restant compatibles avec notre infrastructure ROS 2. Leur intégration garantit que le système répond aux exigences de missions complexes sans nécessiter de développement de solutions équivalentes en interne, permettant ainsi un gain de temps significatif.

Cette architecture modulaire autour de ROS 2 assure une gestion robuste des capteurs, des missions et de la navigation, permettant au robot d'exécuter des missions complexes dans divers environnements tout en assurant une interaction fluide avec la station de contrôle.

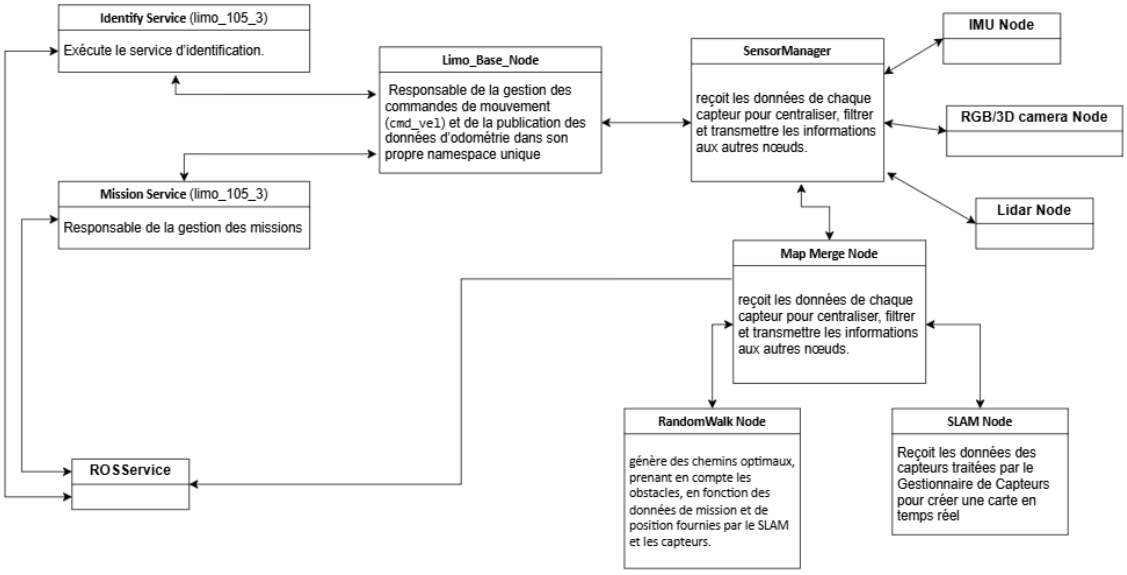

Figure 4: Diagramme de l'architecture du logiciel embarqué

3.4 Simulation (Q4.5)

L'architecture de simulation des robots repose sur ROS 2 et est conçue pour refléter l'architecture du logiciel embarqué des robots physiques. Cette configuration permet d'effectuer des tests en environnement virtuel qui seront facilement transposables sur les robots réels.

Structure de la Simulation

La simulation utilise plusieurs nœuds ROS 2, chacun étant responsable de tâches spécifiques :

- **Gazebo Simulation Node** : Lance l'environnement Gazebo avec un fichier SDF personnalisé pour la simulation. Cet environnement contient les modèles de robots et les éléments de l'environnement (modèle **modified_world.sdf**).
- **Robot State Publishers** : Publient les états des deux robots simulés (**limo_105_3** et **limo_105_4**) en fonction de leurs descriptions SDF respectives, permettant à d'autres nœuds d'accéder à leurs états dans le monde simulé.

- **Bridge Node** : Utilise **ros_gz_bridge** pour assurer la communication entre les nœuds ROS 2 et Gazebo, facilitant l'échange de données entre les deux environnements.

Gestion des Missions et Services Robotiques

Les services de mission et d'identification sont intégrés directement dans la simulation pour chaque robot, avec les nœuds **Identify Service** et **Mission Service** qui fonctionnent sous les espaces de noms dédiés (**limo_105_3** et **limo_105_4**). Ces services permettent de lancer et de gérer les missions depuis la simulation de manière similaire aux robots physiques.

Traitement de la Localisation et Navigation

Les robots utilisent un système de cartographie et de navigation avec les modules **SLAM** et **Nav2**, lancés respectivement après des délais de 5 et 10 secondes. Le **SLAM Toolbox** est configuré pour chaque robot afin de générer des cartes en temps réel en se basant sur les données des capteurs, tandis que **Nav2** assure le calcul des trajectoires et la navigation autonome, même dans des environnements simulés.

Exploration Automatisée et Fusion de Cartes

Les robots exécutent des missions d'exploration via le nœud **Explore Lite**, lancé après un délai, et sont configurés pour explorer leur environnement et produire une carte globale. Un nœud **Map Merge** est également lancé pour fusionner les cartes produites par chaque robot, permettant une vue d'ensemble cohérente de l'environnement.

En utilisant une combinaison de composants internes et de dépendances externes (comme **SLAM, Explore Lite, Nav2,** et **Map Merge**), cette architecture modulaire assure une gestion robuste des missions, de la localisation, de la navigation et de la fusion de cartes dans un environnement simulé. L'ensemble de ces nœuds permet aux robots d'exécuter des missions complexes dans des environnements variés, offrant ainsi une simulation précise et adaptable à des environnements réels.

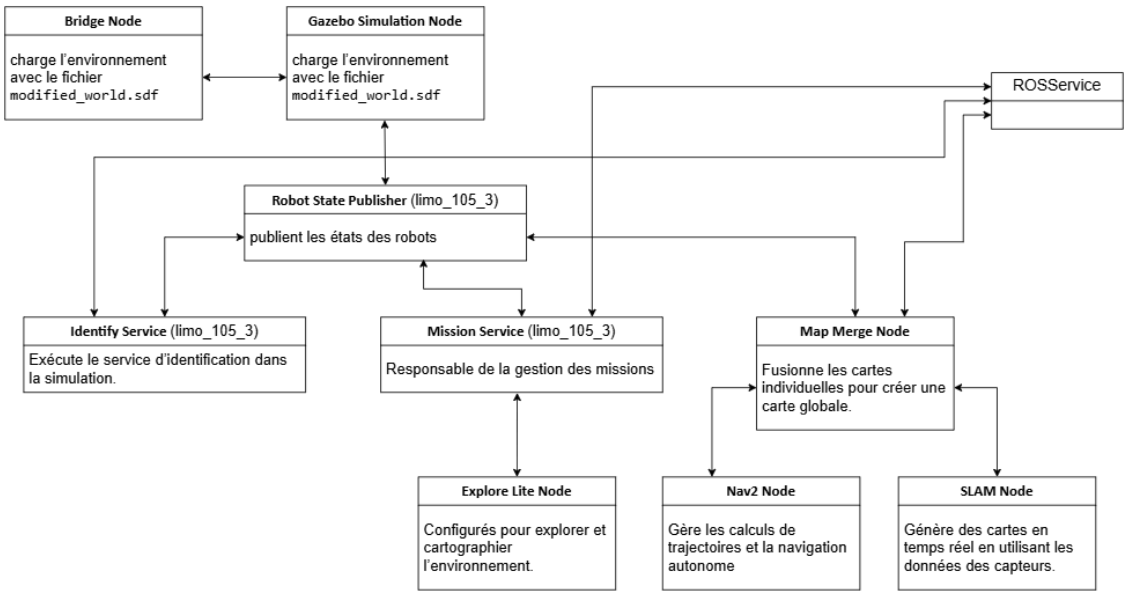

Figure 5: Diagramme de l'architecture de la simulation (limo_105_3)

3.5 Interface utilisateur (Q4.6)

Pour la conception de l'interface utilisateur pour le contrôle des robots physiques et simulés, nous avons opté pour une architecture modulaire basée sur Angular, en séparant les responsabilités entre les différents composants et services. Cette approche permet de gérer de manière claire et intuitive les interactions avec le robot physique ou la simulation via une interface unifiée. Le composant **HomePageComponent** sert de point d'entrée et permet à l'utilisateur de basculer entre le mode simulation et le mode robot physique grâce à un bouton de sélection.

Les composants (**SimulationPageComponent**) et (**RobotPageComponent**) sont dédiés respectivement à la gestion de la simulation et du robot physique. Chaque composant permet à l'utilisateur d'envoyer des commandes spécifiques, telles que démarrer ou arrêter une mission, ou encore consulter l'état du robot ou de la simulation. Ces actions sont accompagnées de commandes additionnelles comme l'identification du robot (via un signal visuel ou sonore) et la mise à jour des informations en temps réel.

Pour assurer la communication avec le backend, un service central appelé (**Simulation/RobotService**) interactions via des requêtes HTTP et WebSocket (Post, Get,

Put) permettant de centraliser les communications avec le système ROS 2. Le composant (***SimulationPageComponent***) s'occupe des commandes pour le mode simulation en utilisant le service pour démarrer, arrêter, identifier et mettre à jour le statut des robots simulés. De même, le composant (***RobotPageComponent***) effectue ces actions pour les robots physiques en interagissant avec le service pour gérer les commandes et récupérer leur statut

Le backend centralise toutes les requêtes envoyées par les composants frontend. Il gère les commandes, qu'il s'agisse de démarrer une mission ou de récupérer l'état actuel des capteurs, et transmet ces informations a ROS 2. Ce dernier joue un rôle crucial en permettant la communication entre le backend et les nœuds du robot physique ou simulé, assurant ainsi une transmission fluide des données à travers un protocole WebSocket.

Pour la conception visuelle et la gestion des composants interactifs de l'interface utilisateur, nous avons choisi d'utiliser **Angular Material** et **Tailwind CSS**. **Angular Material** fournit un ensemble de composants UI prêts à l'emploi et bien intégrés avec Angular, tels que des boutons, des formulaires et des barres de navigation, ce qui simplifie le développement de l'interface utilisateur et garantit une cohérence visuelle. En complément, **Tailwind CSS** est utilisé pour la personnalisation des styles, en nous permettant d'appliquer facilement des classes utilitaires pour une mise en page flexible et réactive. Cela garantit que l'interface est non seulement fonctionnelle, mais également intuitive et moderne, tout en réduisant le temps nécessaire à la conception CSS.

L'utilisation d'Angular Material et Tailwind CSS assure également une grande adaptabilité de l'interface aux différents types d'appareils (ordinateurs, tablettes, etc.), en offrant une conception qui s'ajuste automatiquement à la taille de l'écran. Cela permet aux utilisateurs de contrôler les robots de manière fluide, que ce soit sur un poste de travail ou un appareil mobile.

Cette architecture modulaire et découplée assure non seulement une extensibilité facile, mais également une gestion efficace des services. Chaque service encapsule une responsabilité bien définie, tandis que les contrôleurs centralisent les interactions avec le backend. Cela permet une gestion claire des commandes envoyées à la simulation ou au robot physique, tout en garantissant une communication fluide et réactive entre les différents composants du système, le tout avec une interface utilisateur moderne et intuitive grâce à Angular Material et Tailwind CSS.

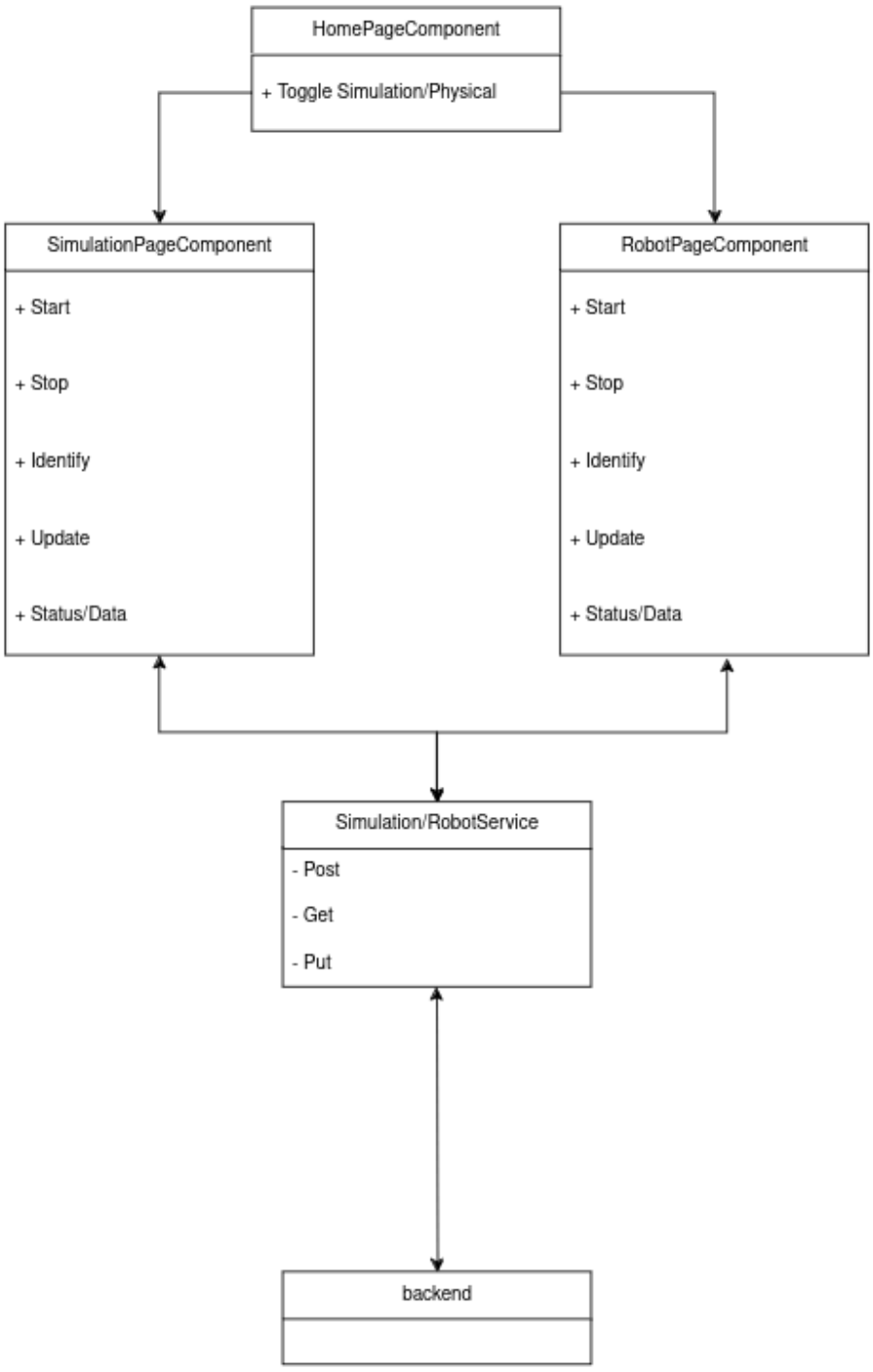

Figure 5: Diagramme d'architecture de l'interface utilisateur

3.6 Fonctionnement général (Q5.4)

Pour faire fonctionner le système à partir du code développé, il faut cloner le dépôt Git et avoir les permissions nécessaires pour le serveur NestJs, le client Angular et les

composants logiciels MongoDB et Docker à l'aide des commandes appropriées (*npm install, docker compose up*). Ensuite pour l'initialisation de la base de données il est nécessaire de configurer MongoDB à l'aide des scripts fournis dans le répertoire database/ du projet pour créer et initialiser la base de données qui s'occupera du stockage des informations liés aux missions des robot.
L'interface utilisateur est démarré en accédant au client Angular et se dirigeant vers l'URL fournie par le serveur NestJS et à partir de cette interface on peut lancer des missions, gérer les opérations et surveiller l'état des robots. Lorsqu'on envoie des commandes aux robots, elles transitent via le serveur NestJS vers le serveur Rosbridge et sont dirigées vers les nœuds ROS qui contrôlent les robots.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

Pour développer le système des robots contrôlés par la station au sol, nous avons établi un budget en fonction des lots de travail de chaque membre de l'équipe et en fonction de leurs rôles. Le tableau ci-dessous résume l'estimation des coûts.

Rôle	Taux horaire	Nombres d'heures	Coût total ($ CAD)
Coordonnateur	145 $/h	126 heures (100 en gestion + 26 en développement)	**18270$**
Développeur	130$ / h	504 heures (126 heures de développement par membres)	**65520$**

Figure 6: Tableau d'estimation des coûts

En respectant les 630 heures-personnes demandées, nous concluons à un total estimé à **84450$**.

4.2 Planification des tâches (Q11.2)

Le tableau ci-dessous fournit un aperçu des principales fonctionnalités à développer ainsi que la durée estimée nécessaire pour les faire.

Tâche	Description	Durée estimée	Délai	Responsables
Amélioration l'interface de la station au sol	Rendre l'interface utilisateur plus intuitive. Placer les boutons manquants.	*1 semaine*	**27/09/24**	• Alcindor • Yaro
Évitement d'obstacles	Implémentation de la détection d'évitement d'obstacles	*1 semaine*	**03/10/24**	• Alcindor • Khnissi
Exploration autonome du robot	Implémentation de l'algorithme pour l'exploration autonome du robot (physique et simulation	*2 semaines*	**08/10/24**	• Yaro • Letieu • Kouakou
Retour à la base et gestion de la batterie	Programmation du retour à la base du robot(physique et simulation) lorsque la batterie est faible	*1 semaine*	**11/10/24**	• Kouakou • Letieu
Cartographie en temps réel	Génération des cartes à partir des données du capteur	*2 semaines*	**15/10/24**	• Alcindor • Yaro • Letieu
Affichage de la position du robot	Affichage en temps réel de la position du robot sur la carte	*2 semaines*	**19/10/24**	• Khnissi • Kouakou
Interface multi-appareils	Le système est accessible depuis plusieurs appareils	*1 semaine*	**21/10/24**	• Alcindor • Letieu
Rédaction du CDR	Rédaction du rapport à remettre	*En continu*	**30/10/24**	Tous les membres de l'équipe
Cartes 3D	Génération de cartes 3D à partir des capteurs du robot	*2 semaines*	**15/11/24**	• Khnissi • Yaro
Détection des chutes	Implémenter une prévention du robot pour éviter qu'il chute.	*2 semaines*	**22/11/24**	• Letieu • Khnissi • Kouakou

Base de données	Création et gestion de la base de données- Enregistrement des missions et des cartes générées	*2 semaines*	**05/11/24**	• Alcindor • Yaro • Khnissi • Kouakou
Finalisation et validation des fonctionnalités	Validation et tests des fonctionnalités implémentées	*2 semaines*	**29/11/24**	Tous les membres de l'équipe
Rédaction du RR	Rédaction du RR et soumission final du projet	*En continu*	**02/12/24**	Tous les membres de l'équipe

Figure 7: Tableau de planification des tâches

4.3 Calendrier de projet (Q11.2)

Le tableau ci-dessus représente le calendrier de projet que nous avons élaboré. Il fait état des différentes clés que nous avons définies afin de garantir le bon déroulement du projet mais également le respect des échéances et des différents livrables.

Phase	Date	Description
Preliminary Design review (PDR)	***20 Septembre 2024***	• Réponse à l'appel d'offre • Connexion entre la partie embarquée et la station au sol • Démonstration vidéo du bon fonctionnement du robot
Développement de la station au sol	***20 Octobre 2024***	• Développement du serveur et du serveur • Assurer de façon continue une bonne connexion entre le serveur et le logiciel embarquée • Création de la base de données
Développement de la partie embarquée	***25 Octobre 2024***	• Développement des différents nœuds du système. • Intégration en continue de la

		liaison serveur et système embarquée
Critical Design Review (CDR)	***1 Novembre 2024***	● Test et validation du logiciel embarqué et de la station au sol. ● Remise du système avec des fonctionnement partiels
Readiness Review (RR)	***3 Décembre 2024***	● Test et validation en continue du logiciel. ● Remise d'un système complet avec l'ensemble des fonctionnalités du système.

Figure 8: Tableau de calendrier du projet

4.4 Ressources humaines du projet (Q11.2)

Notre équipe se compose de 5 ingénieurs dont les compétences humaines, organisationnelles et techniques représentent un atout majeur pour une atmosphère conviviale au sein de l'équipe. En effet, monsieur Alcindor notre responsable de projet, qui a la charge de la gestion globale du projet et du développement de certaines parties critiques du code, possède de l'expérience dans la gestion de projet et a une grande capacité à coordonner les équipes.

Spécialisé en systèmes embarqués, il possède les compétences en développement logiciel intégré à des systèmes embarqués. D'un autre côté, tous nos développeurs-analystes, chargés d'une grande partie de la conception logicielle du système ont des compétences avancées en conception logiciels avec une expertise accrue dans des langages comme Python, ROS, Javascript, mais aussi avec des framework comme angular en développement d'application web. Ils possèdent des connaissances solides en systèmes embarqués et dans des environnements de simulation comme avec gazebo.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Le contrôle de la qualité est une étape essentielle dans notre processus de développement, et nous y accordons une grande importance au sein de notre équipe.

Afin d'assurer un haut niveau de qualité pour chaque bien livrable, nous avons mis en place un processus de révision rigoureux. Chaque livrable doit passer par une révision systématique basée sur des lignes directrices précises. Les tâches sont réparties entre les membres de l'équipe, chacun étant responsable de certaines tâches spécifiques. La gestion des projets se fait via GitLab, où les *issues* sont attribuées aux membres de l'équipe en fonction de leurs responsabilités.

Chaque membre de l'équipe est responsable d'écrire un code de bonne qualité et compréhensible par les autres, comprenant des commentaires (lorsque nécessaire), des noms pertinents de variables et de fonctions et une organisation claire du code et des fichiers qu'il code. Nous encourageons également des **commits** fréquents, des **merges requests** de petite taille et des **pulls** réguliers pour que chacun des membres puissent avoir une mise à jour régulière de l'évolution des autres. Avant que le code ne soit intégré dans le projet, des *merge requests* sont créées. Ces *merge requests* sont examinées par d'autres membres de l'équipe avant son intégration finale. Ce processus garantit que chaque contribution est soigneusement vérifiée et que les normes de qualité sont respectées.

Étant donné que les vérifications manuelles ne sont pas toujours suffisantes, des tests automatiques sont également mis en place pour s'assurer du bon fonctionnement des fonctionnalités et de la cohérence du code. Nous utilisons principalement des **tests unitaires** pour vérifier chaque fonction individuellement, et des **tests d'intégration** pour évaluer l'interaction entre les différentes parties du code.

5.2 Gestion de risque (Q11.3)

Notre projet comporte plusieurs risques que nous avons classés par ordre de priorité pour mieux anticiper et y répondre efficacement.

Le risque principal est le **retard dans le développement**, que nous considérons comme **critique**. Un retard dans la réalisation des fonctionnalités pourrait affecter toutes les étapes du projet et compromettre les délais finaux. Pour prévenir ce risque, nous avons mis en place une planification détaillée avec des échéances intermédiaires et une répartition claire des tâches entre les membres de l'équipe.

Les **problèmes d'interactions entre les composantes** (frontend, backend, services) représentent un autre risque **critique**. Des dysfonctionnements ou des incompatibilités entre ces parties du projet peuvent ralentir le développement et compliquer les tests. Nous organisons donc des réunions d'équipe pour synchroniser les travaux et réalisons des tests d'intégration fréquents pour détecter et résoudre les problèmes rapidement.

Le **manque de communication** est un autre risque **important**. Dans une équipe, une mauvaise communication peut mener à des malentendus et des erreurs dans l'exécution des tâches. Pour garantir une bonne coordination, nous privilégions des échanges réguliers via Discord et des réunions hebdomadaires pour maintenir la transparence et assurer que chacun est aligné sur les objectifs du projet.

Les **incompatibilités entre les technologies** (comme Tailwind, Angular et Angular Material) constituent un risque **important**. Des conflits techniques entre ces outils peuvent nécessiter des ajustements imprévus. Nous utilisons des outils de validation dès le début du projet pour identifier rapidement les incompatibilités et les résoudre avant qu'elles ne posent problème.

Enfin, l'**expérience utilisateur de l'interface Web** est un risque que nous considérons comme **faible**. Une interface peu intuitive pourrait limiter l'efficacité des utilisateurs finaux. Pour assurer une bonne expérience, nous effectuons des tests d'usabilité réguliers, en simulant des situations réelles, pour ajuster l'interface en fonction des retours des utilisateurs.

5.3 Tests (Q4.4)

Comme nous l'avons mentionné dans la section précédente, les tests sont très utiles pour le développement du projet. Ils doivent être fait à tous les niveaux pour mener à bien notre évolution. Les tests sont fait au niveau du robot physique, au niveau de la simulation et de l'interface utilisateur.

Au niveau du robot physique:
Nous avons des tests d'intégration pour vérifier que le robot se connecte au backend, nous devons également tester le moteur pour guider la direction et les mouvements du robot via des tests fonctionnels.

Tâches et/ou requis	Tests
Détection des chutes	Tester les capteurs pour prévenir les chutes
Évitement d'obstacles	Vérifier les capteurs pour détecter et éviter les obstacles
Exploration autonome du robot	Vérifier le bon fonctionnement de l'algorithme d'exploration lors de la mission

Figure 9: Tableau des tâches et des tests relatifs sur le robot physique

Au niveau de la simulation:

Nous devons vérifier que les simulations sont correctes et que les données des capteurs s'accordent avec la réalité.

Tâches et/ou requis	Tests
Exploration autonome du robot	Simuler plusieurs scénarios d'exploration pour voir le comportement virtuellement.
Évitement d'obstacles	Tester la réactivité du robot devant un obstacle.
Cartographie à temps réel	Vérifier que la carte générée renvoie les bonnes informations et reflète l'environnement.

Figure 10: Tableau des tâches et des tests relatifs sur le robot simulé

Au niveau de l'interface:
Nous devons tester l'interface utilisateur et la mise à jour synchronisée des données et de la simulation sur l'interface.

Tâches et/ou requis	Tests
Interface multi-appareils	Tester si plusieurs appareils peuvent se connecter sans interférence
Affichage du déplacement du robot	Vérifier l'affichage à temps réel du robot et la qualité de
Validation des fonctionnalités sur l'interface.	Faire des essais pour s'assurer que toutes les fonctionnalités sont intégrées et fonctionnent avec différents utilisateurs.

Figure 11: Tableau des tâches et des tests relatifs sur la station au sol

Par ailleurs, pour chaque partie du code front-end, des tests sont créés avec **Jasmine** et **Karma**. Jasmine aide à écrire des tests pour vérifier que le code fonctionne bien, et Karma exécute ces tests automatiquement dans différents navigateurs. Cela permet de vérifier que tout marche correctement et de repérer les erreurs rapidement.

5.4 Gestion de configuration (Q4)

Notre projet utilise GitLab comme système de gestion de version pour suivre les modifications du code source. Nos branches portent le nom des membres de l'équipe et

sont mises à jour selon leur progression individuelle.Chaque membre travaille dans une branche dédiée, et avant toute intégration dans la branche principale, un *Merge Request* est créé afin de valider les modifications. Cela permet d'assurer un contrôle de qualité et une révision par un autre membre de l'équipe.

Les *Issues* sont créées pour chaque tâche à réaliser et sont classées avec des étiquettes correspondant au dépôt ou à la remise concernée, facilitant ainsi le suivi des développements. Le code source est organisé de manière à séparer le développement du front-end (développé avec Angular) du back-end (développé en ROS2 et Python), pour avoir une architecture modulaire et facilement maintenable.

Les informations sur l'évolution des tâches et des modifications apportées au code sont directement suivies via GitLab, notamment à travers les *Issues* et les pipelines d'intégration continue. Cela permet de documenter automatiquement les changements et de garantir que les différentes étapes du projet sont correctement suivies et tracées.

5.5 Déroulement du projet (Q2.5)

Dans notre équipe, concernant la remise du PDR (Preliminary Design Review), nous avons respecté nos délais. L'avancée du travail était en accord avec notre échéancier, et nous avions déjà programmé un rendez-vous pour réaliser des simulations avec le robot physique. Depuis le début du projet, plusieurs aspects se sont déroulés de manière satisfaisante. L'organisation de l'équipe et la proactivité de certains membres ont permis de maintenir un travail fluide, et nous avons réussi à adopter des normes et des règles de communication qui ont grandement facilité l'avancement du projet.

Cependant, nous rencontrons également des difficultés. Il n'est pas toujours facile de trouver des horaires de réunion convenant à tout le monde. De plus, étant au début du projet, il est nécessaire de prendre le temps de justifier certains choix techniques pour la structuration globale. Enfin, l'adaptation aux nouveaux outils et la compréhension de leur fonctionnement posent des défis à plusieurs membres de l'équipe.

Actuellement, nous sommes en pleine phase de développement de notre projet, entre la remise du Preliminary Design Review (PDR) et celle du Critical Design Review (CDR). Nous rencontrons particulièrement des difficultés de coordination pour les réunions en raison de la période intra-semestre mais. Nous avons finalement pu couvrir les différents requis obligatoires prévus pour cette remise. Réaliser l'exploration autonome du robot a été notre principal.

Dans cette architecture, bien que nous ayons rencontré certaines difficultés, notamment avec l'utilisation de **Explore Lite**, qui n'a pas fonctionné comme prévu, nous avons décidé d'adopter une approche alternative pour assurer l'exploration autonome du robot.

Pour pallier ces limitations, nous avons intégré un **algorithme basé sur l'utilisation du LiDAR**. Cet algorithme utilise les données du LiDAR pour générer des trajets sécurisés et optimiser l'exploration de l'environnement. Plutôt que de dépendre uniquement d'un module externe comme Explore Lite, l'algorithme de LiDAR permet au robot de calculer les trajectoires en temps réel, d'identifier les obstacles, et d'explorer l'espace de manière autonome. Cette solution s'intègre naturellement avec le **Random Walk Node** et le **SLAM Node**, offrant ainsi une meilleure maîtrise de l'exploration en exploitant pleinement les données de capteurs.

Cette alternative a amélioré la fiabilité du système, tout en offrant une flexibilité accrue pour ajuster les comportements d'exploration en fonction des spécificités de l'environnement du robot. L'utilisation directe des données du LiDAR a permis de contourner les limitations initiales et de renforcer l'indépendance de l'architecture logicielle embarquée, en minimisant les dépendances externes.

Nous avons aussi une rencontre HPR pour parler de la dynamique du groupe et de la gestion du projet

6. Résultat des tests de fonctionnement du système complet

Après l'intégration de toutes les branches du projet, nous avons effectué des tests de fonctionnement pour évaluer le comportement global du système. Ces tests nous ont permis de vérifier si les fonctionnalités en simulation et avec les robots physiques répondent aux exigences des requis.

En ce qui concerne la simulation, la génération de l'environnement pour les missions fonctionne. Les robots explorent l'environnement de manière autonome et l'évitement des obstacles est géré efficacement

Au niveau du robot physique, l'exploration autonome se fait bien. Par contre, son comportement n'est pas aussi optimal en environnement réel car les capteurs lidar utilisés semblent moins performants qu'en simulation et très sensibles à la réflexion de la lumière et aux obstacles trouvés.

Quant à l'interface utilisateur, il affiche en temps réel l'état des robots et leur progression dans l'environnement simulé, avec une compatibilité sur différents appareils (ordinateur, tablette, mobile), facilitant le suivi de mission. L'interface est aussi capable d'afficher les logs de débogage. La carte de l'environnement est générée à partir des données des capteurs lidar, offrant une visualisation en temps réel de l'espace exploré, avec un niveau de précision satisfaisant pour visualiser l'environnement.

7. Références (Q3.2)

Navigation 2. (s.d.). *Navigation 2 Documentation*. https://docs.nav2.org/index.html

Macenski, S. (s.d.). *slam_toolbox*. GitHub. https://github.com/SteveMacenski/slam_toolbox

Robo-friends. (s.d.). *m-explore-ros2*. GitHub. https://github.com/robo-friends/m-explore-ros2

ANNEXES