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:

- **Disponibilité des ressources matérielles** : Les ressources matérielles nécessaires au développement et au fonctionnement du système seront disponibles en continu, permettant une progression fluide du projet sans interruptions majeures.
- **Fiabilité des capteurs** : Les capteurs installés sur les robots seront suffisamment précis et fiables pour permettre une exploration autonome de l'environnement, même dans des conditions simulées ou réelles.
- **Stabilité des réseaux WiFi** : La communication entre la station au sol et les robots physiques sera limitée par la stabilité du réseau WiFi, et des solutions alternatives (comme l'utilisation du réseau à des heures creuses) devront être envisagées en cas de surcharge.
- **Compatibilité de l'interface utilisateur** : L'interface utilisateur sera accessible via un navigateur web compatible, garantissant une utilisation fluide et intuitive pour les opérateurs.
- **Fiabilité du système grâce aux tests** : La mise en place de tests unitaires approfondis pour le backend et le client est essentielle pour assurer un fonctionnement fiable du système, bien qu'un temps supplémentaire puisse être requis pour maximiser cette fiabilité.

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 M-7703
- L'espace de la salle M-7703 devant être partagé avec d'autres contracteurs
- Les pannes des robots physiques nécessitant l'intervention des chargés
- La nécessité de réserver la salle M-7703 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

Nous avons mis en place des processus de communication réguliers pour assurer une coordination optimale. Parmi ces processus, nous organisons trois rencontres hebdomadaires où nous discutons des progrès réalisés, des obstacles rencontrés et des prochaines tâches à entreprendre. En complément, nous maintenons une discussion continue sur Discord, favorisant une communication fluide entre les membres en dehors de ces réunions.

Pour faciliter la gestion des tâches, nous avons choisi GitLab comme plateforme principale, exploitant son système de gestion des tickets. Chaque tâche est détaillée dans un ticket dédié, permettant à chaque membre de s'assigner ses responsabilités. Cette approche encourage une répartition claire des rôles tout en assurant une collaboration transparente, où chacun peut suivre et contribuer aux discussions et aux progrès relatifs à chaque tâche.

De plus, nous avons enrichi notre environnement de travail collaboratif en intégrant des plugins à Discord. Ces extensions permettent de visualiser en temps réel l'engagement de chaque membre dans le projet, notamment à travers leurs commits sur GitLab. Cette fonctionnalité nous offre une vue d'ensemble dynamique sur l'avancement du projet et renforce la transparence et la responsabilité collective. L'utilisation conjointe de GitLab et Discord, avec leurs outils respectifs, constitue ainsi un pilier central de notre organisation, garantissant une gestion fluide, une traçabilité précise et une collaboration continue.

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.

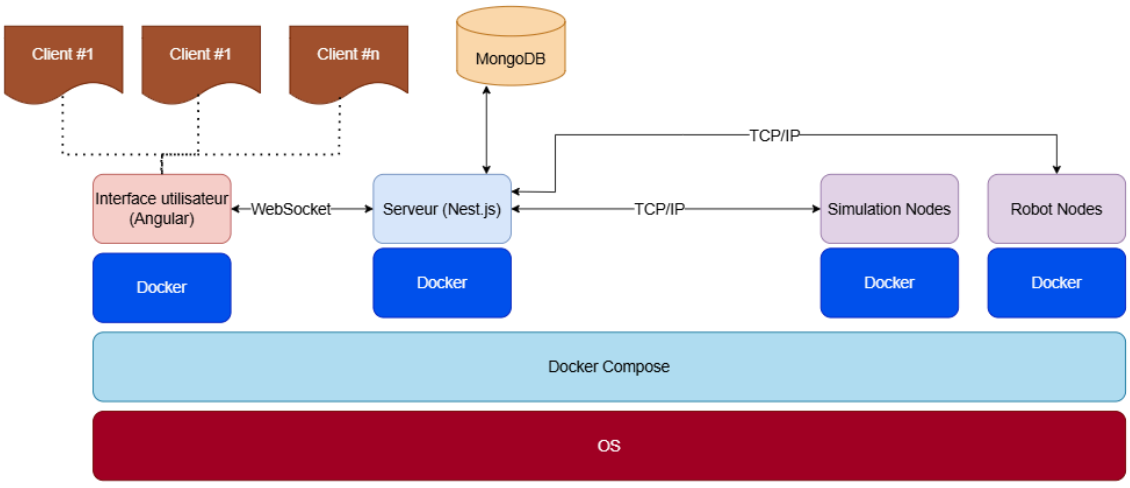

Figure 2: Diagramme de l'architecture logicielle générale

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.

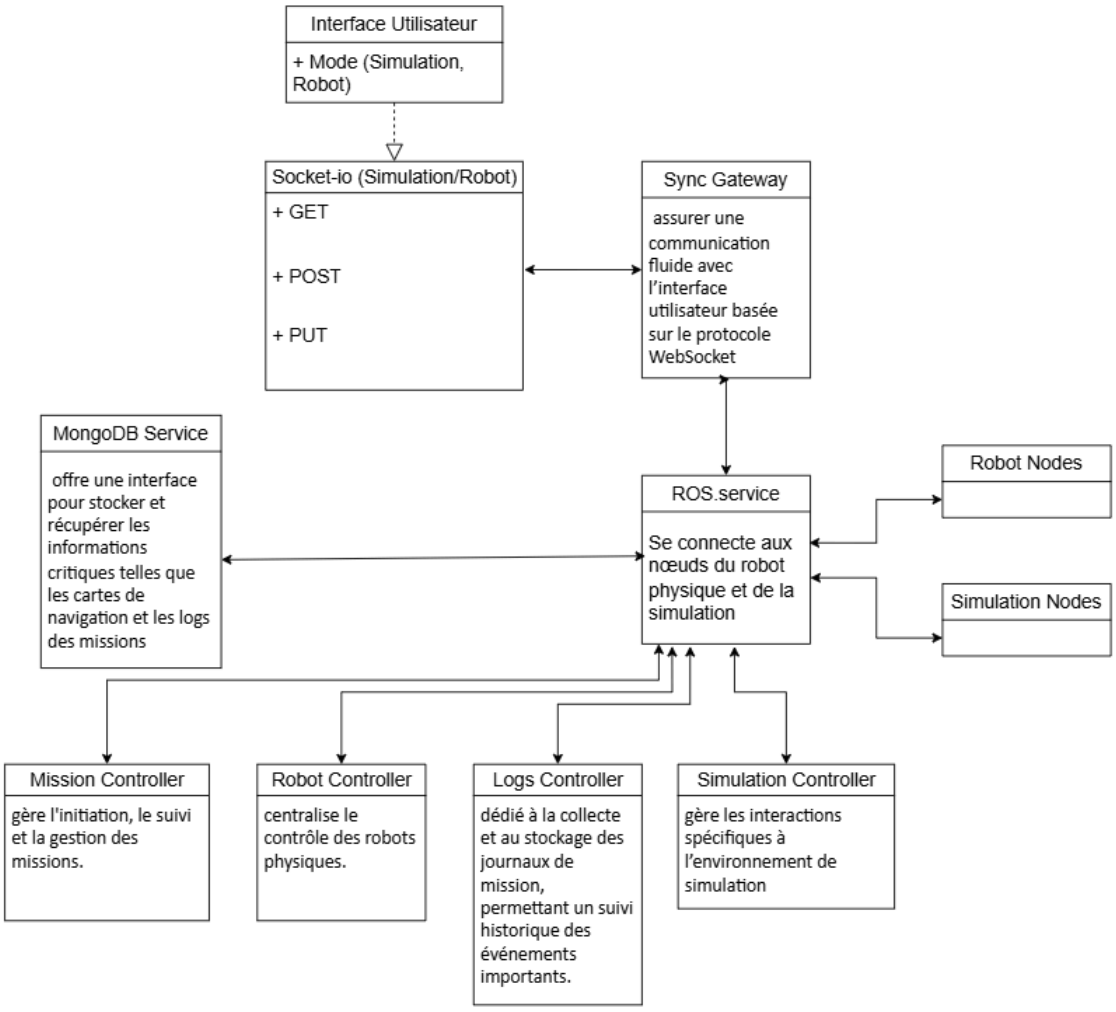

Figure 3: Diagramme de l'architecture de la station au sol

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 affichée 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 Nav2 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 Nav2 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.

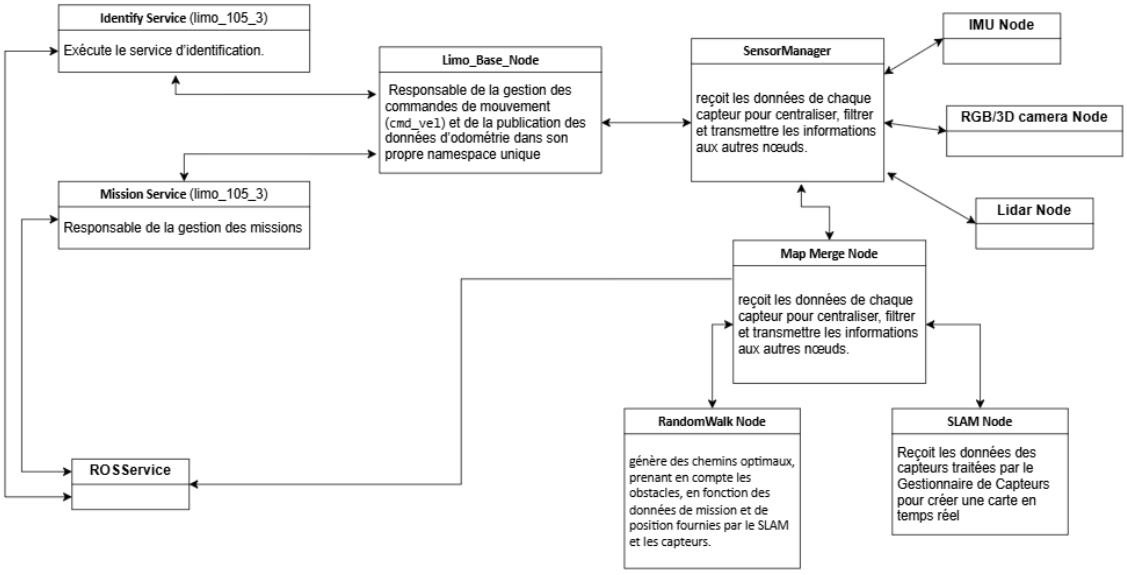

Figure 4: Diagramme de l'architecture du logiciel embarqué

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.

Noeud Octomap

- **Fonction principale :** Ce noeud s'occupe de traiter le nuage de points envoyé par le LIDAR des robots, ce noeud est couplé avec une fonction définie sur ROSService qui permet de traiter davantage le nuage de points afin de permettre l'utilisateur voir une carte tridimensionnelle.

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.

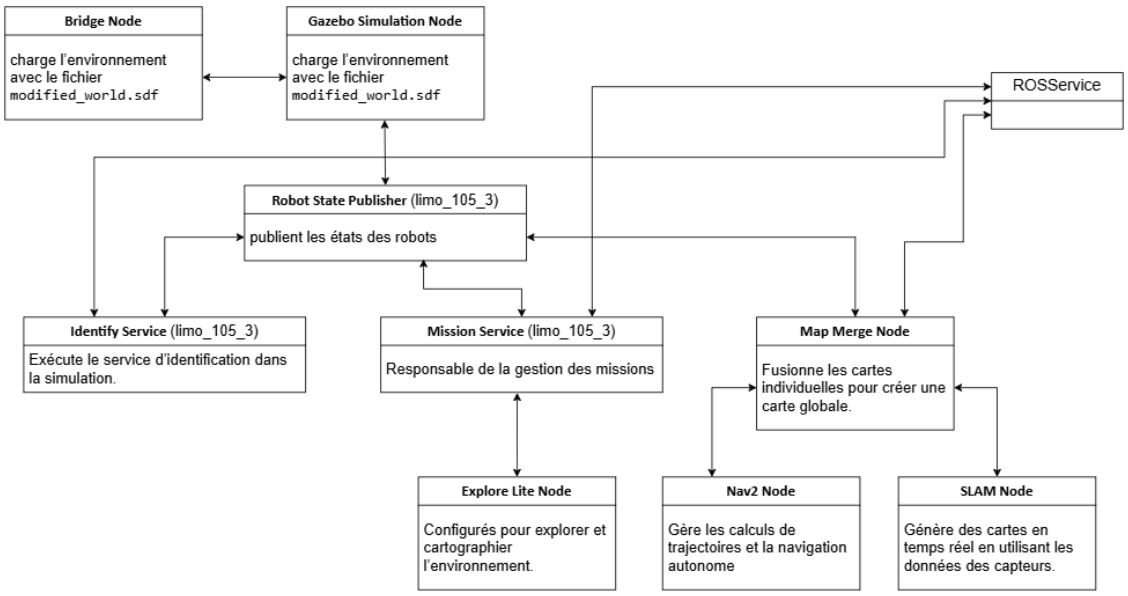

Figure 5: Diagramme de l'architecture de la simulation (limo_105_3)

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.

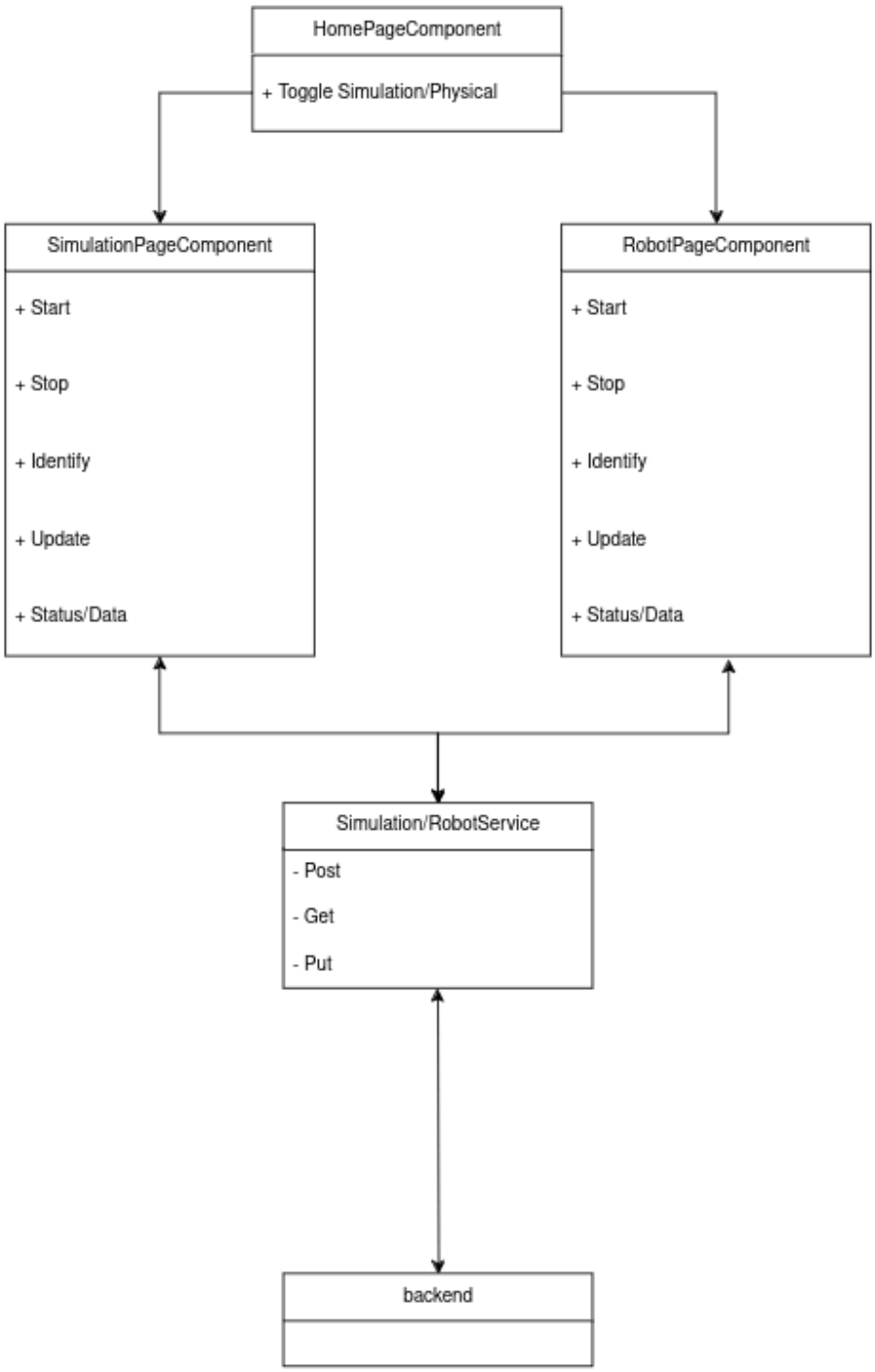

Figure 5: Diagramme d'architecture de l'interface utilisateur

3.6 Fonctionnement général (Q5.4)

Afin de lancer le projet, assurez vous de suivre les étapes suivantes:

Simulation

- Mettre la variable SIMULATION à 1 dans le fichier .env
- À la racine du repo, executer la commande source launch_station.sh
- Une fois la station au sol lancée, sélectionnez le bouton "Simulation".
- Puis dans l'interface qui s'affiche cliquer sur "Start ROS" pour lancer la simulation Gazebo.
- Patientez 1 minute le temps que le système complet soit lancé.

Robots Physiques

Assurez vous que la station au sol et les robots soient sur le même réseau WIFI puis suivre les étapes suivantes:

- À la racine du repo, executer "source launch_robot.sh 1" sur l'un des robots
- Puis, à la racine du repo sur l'autre robot, executer "source launch_robot.sh 2"
- Sur la station au sol, mettre la variable SIMULATION à 0 dans le fichier .env
- À la racine du repo, executer la commande source launch_station.sh

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	*2 semaines*	**19/10/24**	• Khnissi • Kouakou

	sur la carte			
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 (2D/3D) 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	Faire des essais pour s'assurer que toutes les fonctionnalités

fonctionnalités sur l'interface.	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 et organiser les modifications du code source. Les branches sur GitLab sont structurées de manière à refléter l'organisation de l'équipe, chaque branche étant dédiée à un membre spécifique. Cette division permet à chaque membre de travailler de manière indépendante sur sa propre branche, selon sa progression individuelle. Avant toute intégration dans la branche principale, un Merge Request est créé pour soumettre les modifications à une révision par un autre membre de l'équipe. Cette procédure garantit un contrôle de qualité rigoureux et facilite la détection et la correction des erreurs potentielles avant la fusion.

Les Issues sont créées pour chaque tâche à réaliser et sont soigneusement classées avec des étiquettes correspondant au dépôt ou à la remise concernée. Cela permet de suivre efficacement les développements et de prioriser les tâches selon les besoins du projet. Le code source est organisé selon une architecture modulaire pour simplifier la maintenance, en séparant clairement le développement du front-end (développé avec Angular) de celui du back-end (développé en ROS2 et Python).

L'évolution des tâches et les modifications apportées au code sont suivies directement via GitLab, notamment à travers les Issues et les pipelines d'intégration continue. Cette intégration permet de documenter automatiquement les changements, de surveiller le statut des tâches en temps réel et de garantir que toutes les étapes du projet sont correctement planifiées, suivies et tracées. Cette organisation structurée, basée sur les branches individuelles et une gestion rigoureuse des Merge Requests, contribue à une collaboration efficace et à un développement fluide.

5.5 Déroulement du projet (Q2.5)

Ce qui a été bien réussi :

- Nous avons respecté les délais des remises tels qu'ils étaient planifiés, grâce à l'attribution d'échéances claires à chaque étape du projet et à des suivis réguliers. Cette rigueur dans la gestion des délais nous a permis de livrer un produit final en temps voulu, conformément aux attentes.
- Notre équipe a maintenu une communication ouverte et transparente, favorisant une collaboration efficace entre les membres. Cela a permis de résoudre les problèmes et de prendre des décisions rapidement tout au long du projet.
- Face aux changements imprévus survenus en cours de projet, comme la nécessité d'intégrer le changement de modes de contrôle des roues, nous avons pu ajuster nos plans et stratégies grâce à notre flexibilité. Par exemple, cette adaptation nous a amenés à considérer l'abandon temporaire de la conteneurisation du projet afin de simplifier le déploiement et de respecter les délais. Ces ajustements nous ont permis de surmonter les obstacles sans retarder les livrables.
- Nous avons accordé une grande importance à la qualité de nos livrables et du code en maintenant un haut niveau d'attention aux détails et en effectuant des validations continues, ce qui a permis de garantir un produit final de qualité supérieure.
- Nous avons complété tous les requis obligatoires, ainsi que plusieurs requis optionnels, démontrant notre engagement à dépasser les attentes initiales.

Ce qui n'a pas été bien réussi :

- Bien que nous ayons fait preuve de flexibilité, la gestion de certains changements imprévus a posé des défis. Par exemple, l'intégration des nouveaux modes de contrôle a nécessité des modifications importantes qui ont parfois perturbé notre planification et notre exécution, entraînant des problèmes de coordination.
- Notre capacité à prioriser efficacement n'a pas toujours été optimale. Nous avons rencontré des difficultés à déterminer quelles tâches devaient être accomplies en priorité, ce qui a parfois retardé l'avancement du projet.
- Nous n'avons pas clairement défini les rôles et responsabilités de chaque membre de l'équipe. Ce manque de clarification a entraîné des confusions quant aux tâches à accomplir et aux responsabilités individuelles, compliquant ainsi la coordination générale.

6. Résultat des tests de fonctionnement du système complet

Pour évaluer le système dans son ensemble, nous avons réalisé des tests ciblés sur les différentes composantes. Les fonctionnalités côté client et serveur ont été vérifiées à l'aide de tests unitaires. Concernant le code destiné aux robots physiques et à la simulation, nous avons défini une procédure de test permettant de valider chaque fonctionnalité de manière manuelle. Ci-dessous, les résultats des tests du système pour chaque fonctionnalité requise dans le cadre du CDR :

- RF1 → Fonctionnel : Chaque robot doit individuellement répondre à la commande "Identifier" disponible dans l'interface utilisateur.
- RF2 → Fonctionnel : L'équipe de robots doit répondre aux commandes suivantes, disponibles sur l'interface utilisateur :
    - "Lancer la mission" : début de la mission
    - "Terminer la mission" : arrêt immédiat
- RF3 → Fonctionnel : Pour chaque robot, l'interface utilisateur doit montrer le robot et son état (attente, en mis sion, etc.), mis à jour avec une fréquence minimale de 1 Hz.
- RF4 → Fonctionnel : Après le départ, les robots doivent explorer l'environnement de façon autonome.
- RF5 → Fonctionnel : Les robots doivent éviter les obstacles détectés par leurs capteurs.
- RF6 → Fonctionnel : Le retour à la base doit rapprocher les robots de leur position de départ pour qu'ils soient à moins de 0,3 m de celle-ci.
- RF7 → Fonctionnel : Le retour à la base doit être activé automatiquement dès que le niveau de batterie devient de moins de 30%. Le niveau de batterie des robots doit être affiché sur l'interface utilisateur.
- RF8 → Fonctionnel : La station au sol doit collecter les données générées par les robots et produire une carte de l'environnement. Cette carte doit être affichée en continu lors de la mission dans l'interface utilisateur. La carte générée doit ressembler à l'environnement exploré
- RF9 → Lors d'une mission, la position des robots dans la carte doit être affichée en continu.
- RF10 → Fonctionnel : L'interface utilisateur pour l'opérateur doit être disponible comme service Web et visualisable sur plusieurs types d'appareils (PC, tablette, téléphone) via réseau. Au moins deux appareils

doivent pouvoir être connectés en même temps pour la visualisation des données lors d'une mission.

- RF11 → Fonctionnel (simulation) : La carte générée représente l'environnement en 3D et en couleur.
- RF13 → Fonctionnel : Le système doit pouvoir détecter une élévation négative (p.ex. une marche) et empêcher le robot de tomber.
- RF15 → Fonctionnel (simulation) : Deux modes de contrôle des roues différents sont utilisés sur les robots (contrôle Ackerman, contrôle différentiel 4 roues)
- RF17 → Fonctionnel : Une base de données doit être présente sur la station au sol et enregistrer au minimum les attributs suivants pour chaque mission : date et heure de la mission, durée, robots, physique/simulation et distance totale parcourue par les robots.
- RF18 → Fonctionnel : La carte générée lors d'une mission doit être enregistrée sur la station au sol.
- RC1 → Fonctionnel : L'opérateur du système doit pouvoir vérifier que le système fonctionne correctement et avoir les informations nécessaires pour résoudre les problèmes. À cet effet, des logs de débogage doivent être disponibles en continu lors de chaque mission. Ces logs comprennent toutes les informations de débogage nécessaires au développement, dont au minimum : les lectures des senseurs (distance et position) de chaque robot (à minimum de 1 Hz) ainsi que les commandes envoyées par la station au sol.
- RC2 → Fonctionnel : Le logiciel complet de la station au sol doit pouvoir être lancé avec une seule commande sur un terminal Linux.
- RC3 → Fonctionnel : L'environnement virtuel pour les tests dans Gazebo doit pouvoir être généré aléatoirement. L'environnement simulé doit avoir un minimum de 3 murs.
- RC5 → Fonctionnel : Le système doit être conçu pour fonctionner avec 1 ou 2 robots. La station au sol et l'interface utilisateur doivent s'adapter automatiquement et se connecter à tous les robots disponibles pour la mission, sans que l'utilisateur n'ait à en spécifier le nombre.

7. Travaux futurs et recommandations (Q3.5)

Futurs travaux :

- Compléter les requis optionnels qui n'ont pas pu être finalisés, en raison de leurs exigences nécessitant un investissement de temps supplémentaire.
- Mener des tests approfondis en conditions réelles pour valider les performances du système dans des environnements variés, et identifier d'éventuelles améliorations.
- Ajouter un mode de simulation multi-robots plus complexe, pour évaluer les interactions entre plusieurs robots dans un environnement partagé.
- Mettre en place une interface utilisateur plus intuitive et interactive, afin de simplifier la prise en main du système par des utilisateurs non techniques.

Recommandations :

- Améliorer la performance globale du projet en optimisant le code, en réduisant les temps de latence et en augmentant l'efficacité de la communication entre les différents modules.
- Renforcer la résilience du système face aux pannes, notamment en intégrant des mécanismes de récupération automatique et des systèmes de redondance.
- Finaliser la documentation technique et utilisateur pour garantir une utilisation, une maintenance et une évolution aisées du système à long terme.
- Identifier des applications potentielles du projet dans d'autres domaines, tels que la logistique, la santé ou l'éducation, afin d'élargir les cas d’utilisation et de maximiser son impact.
- Prévoir des sessions de formation ou des guides pratiques pour les futurs utilisateurs, afin de faciliter leur adoption et utilisation du système.

8. Apprentissage continu (Q12)

Samuel : Au cours de ce projet, j’ai su apprendre à me réajuster en permanence en vue d’accomplir une tâche donnée, j’ai appris que souvent il fallait tenter différentes hypothèses en vue de trouver la bonne. Et j’ai aussi compris qu’il fallait que j’apprenne à y aller étape par étape en vue de résoudre un problème plutôt que de vouloir tout faire en une fois. En bref, ce projet m’a surtout aidé à travailler mon organisation et ma manière de découper en petites parties une tâche en vue de pouvoir la compléter.

Axelle : Au cours de ce projet, j’ai pris conscience de mes difficultés, tant sur le plan technique qu’en matière de planification. J’ai rencontré des obstacles pour produire des codes fonctionnels et pour intégrer les différentes composantes du système. Ces défis ont mis en évidence mes lacunes en développement et ma tendance à abandonner rapidement face aux difficultés.

Pour y remédier, j'ai appris à décomposer les problèmes en tâches plus simples et à approfondir mes recherches pour mieux comprendre les fonctionnalités à développer. Grâce à une équipe bienveillante, j'ai bénéficié de précieux conseils, notamment sur la structuration des fichiers et sur certains concepts techniques. Cette expérience m'a permis d'améliorer ma méthodologie de travail et de mieux persévérer face aux défis techniques.

Jodel : Travailler sur le projet cette session m'a offert une expérience riche en apprentissage continu.

J'ai été confronté à plusieurs défis, notamment la configuration du robot pour qu'il se déplace de manière autonome tout en évitant les obstacles. Cela m'a poussé à rechercher des solutions, tester différentes approches, et ajuster ma méthode lorsque les résultats n'étaient pas ceux attendus. J'ai appris à analyser les causes des problèmes, explorer de nouvelles idées, et m'adapter rapidement. Pour pallier à mes lacunes, je me suis appuyé sur des ressources en ligne mais aussi des échanges avec mes collègues. Ce projet m'a permis de renforcer ma capacité à résoudre des problèmes de manière autonome et méthodique et à travailler de manière plus structurée. J'ai également appris à mieux gérer les imprévus et à rester plus serein.

Jonathan : Au cours de ce projet, j'ai pu constater un manque de précision dans mes estimations de date limite d'implémentation. Cela a plusieurs fois donné lieu à des heures de travail supplémentaires ou des retards dans les pires cas.

Afin d'y remédier, j'ai décidé de subdiviser chaque tâche m'ayant été attribuée en sous-tâches, chacune avec sa deadline intermédiaire. Cela m'a permis de fournir des deadlines plus précises et de finir mes tâches dans les temps. De plus, j'ai sollicité l'avis de mon équipe par rapport à mes dates limites de sorte à m'assurer d'avoir considéré chaque tâche dans son ensemble.

Zakarya : Pendant le projet, j'ai identifié des lacunes dans mes compétences organisationnelles, principalement en raison d'un emploi du temps chargé. Cela a parfois affecté ma capacité à structurer efficacement mon travail et à respecter certaines priorités. De plus, j'ai rencontré des difficultés à communiquer clairement mes idées et les défis que j'observais au sein du projet aux autres membres de l'équipe. Cette situation a occasionnellement freiné la collaboration et la résolution collective des problèmes.

Pour y remédier, j'ai commencé à utiliser des outils de gestion du temps, tels que des plannings hebdomadaires et des rappels, pour mieux structurer mes tâches et prioriser les plus importantes. J'ai également pris l'habitude de documenter mes idées et observations dans un format clair et synthétique avant de les partager avec l'équipe, ce qui a facilité leur compréhension et encouragé des échanges constructifs.

9. Conclusion (Q3.6)

Maintenant que le système est finalisé, nous pouvons affirmer que notre démarche de conception a été, dans l'ensemble, efficace et aboutie. Voici quelques éléments clés à souligner concernant nos hypothèses initiales et la vision globale du système :

- Disponibilité des ressources matérielles : Les ressources matérielles se sont avérées disponibles en continu, confirmant ainsi nos hypothèses et permettant un développement fluide du système sans interruptions significatives.
- Fiabilité des capteurs : Les capteurs installés ont démontré une précision et une fiabilité adéquates, permettant une exploration autonome de l'environnement et validant ainsi nos hypothèses initiales.
- Stabilité des réseaux WiFi : Les réseaux WiFi ne se sont pas révélés suffisamment fiables pour assurer une communication efficace entre la station au sol et les robots physiques, en raison du débit limité et du nombre élevé d’utilisateurs. Pour contourner ce problème, nous avons dû effectuer nos tests à des heures creuses, lorsque le réseau était moins sollicité.
- Interface utilisateur : L'opérateur a pu utiliser un navigateur web compatible avec l'interface prévue, garantissant ainsi une expérience fluide et une utilisation optimale du système.
- Tests unitaires : Des procédures de test approfondies ont été mises en place pour garantir la fiabilité du système, avec des tests unitaires réalisés tant côté backend que côté client. Cependant, il sera nécessaire de consacrer davantage de temps pour renforcer la fiabilité globale du système.
- Conteneurisation: Nous avons été contraints d’opter pour une implémentation sans conteneurisation du fait de la complexité de notre système final. Nous avons cependant implémenté des scripts de lancement afin de garantir la compatibilité maximale de notre système ainsi qu’une simplicité d'exécution.

En conclusion, la réussite du système valide notre démarche de conception ainsi que la pertinence de nos hypothèses initiales et de notre vision du projet. Les résultats obtenus démontrent notre capacité à surmonter les défis techniques et organisationnels, aboutissant à un système fonctionnel répondant aux attentes de l'Agence Spatiale de Polytechnique Montréal.

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