



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 d'exploration martien**

Équipe No 107

Ely Cheikh Abass

Omar Benzekri

Abdul-Wahab Chaarani

Loïc Nguemegne

Thomas Rouleau

Ivan Samoylenko

2 décembre 2024

## Table des matières

1.	Vue d'ensemble du projet .....	4
1.1	But du projet, porté et objectifs (Q4.1).....	4
1.2	Hypothèse et contraintes (Q3.1).....	4
1.3	Biens livrables du projet (Q4.1) .....	6
2.	Organisation du projet.....	6
2.1	Structure d'organisation (Q6.1) .....	6
2.2	Entente contractuelle (Q11.1).....	7
3.	Description de la solution.....	8
3.1	Architecture logicielle générale (Q4.5).....	8
3.2	Station au sol (Q4.5).....	10
3.3	Logiciel embarqué (Q4.5) .....	12
3.4	Simulation (Q4.5).....	14
3.5	Interface utilisateur (Q4.6) .....	15
3.6	Fonctionnement général (Q5.4).....	16
4.	Processus de gestion.....	17
4.1	Estimations des coûts du projet (Q11.1).....	17
4.2	Planification des tâches (Q11.2).....	18
4.3	Calendrier de projet (Q11.2).....	20
4.4	Ressources humaines du projet (Q11.2).....	21
5.	Suivi de projet et contrôle .....	22
5.1	Contrôle de la qualité (Q4) .....	22
5.2	Gestion de risque (Q11.3) .....	23
5.3	Tests (Q4.4) .....	24
5.4	Gestion de configuration (Q4) .....	25
5.5	Déroulement du projet (Q2.5).....	27
6.	Résultats des tests de fonctionnement du système complet (Q2.4) .....	28
6.1	Requis fonctionnels .....	28
6.2	Requis Matériels.....	30
6.3	Requis logiciel .....	30
6.4	Requis de conception .....	30
6.5	Requis de qualité .....	31
7.	Travaux futurs et recommandations (Q3.5) .....	31
8.	Apprentissage continu (Q12) .....	33
8.1	Ely Cheikh Abass .....	33
8.2	Omar Benzekri.....	34
8.3	Abdul-Wahab Chaarani .....	35
8.4	Loïc Nguemegne .....	36
8.5	Thomas Rouleau .....	37

8.6	Ivan Samoylenko .....	37
9.	Conclusion (Q3.6) .....	39
10.	Références (Q3.2) .....	41
	ANNEXES.....	42
	Annexe A – Résumé des requis .....	42
	<b>Annexe B – Architecture logiciel .....</b>	44
	Annexe C – Échéanciers de projet.....	45

## 1. Vue d'ensemble du projet

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

Le présent devis a pour objet le développement d'une preuve de concept d'exploration planétaire par un système multirobot, équipé des capteurs minimaux requis par l'Agence Spatiale Canadienne (ASC). Ce projet, orienté vers la recherche et l'enseignement en exploration spatiale, a pour objectif de démontrer l'efficacité et la viabilité d'un système multirobot autonome simple dans un environnement intérieur contrôlé, simulant des conditions d'exploration planétaire. Les robots devront explorer de manière autonome une zone inconnue, de la taille d'une pièce, en utilisant exclusivement les capteurs définis par l'ASC : l'IMU (« Inertial Measurement Unit »), la caméra 3D, la caméra RGB et le LiDAR (« Light Detection and Ranging »). L'opérateur pourra suivre les données en temps réel via une interface web, avec un contrôle limité au démarrage et à l'arrêt des opérations. Le système devra générer une cartographie précise de la zone explorée.

Ce projet vise à fournir une preuve de concept de niveau de maturité 4 (NMS 4), démontrant des robots entièrement autonomes, capables de gérer les obstacles et les imprévus avec une intervention humaine minimale, tout en collectant les données nécessaires à la cartographie.

Le projet doit suivre un calendrier de développement rigoureux, incluant les biens livrables suivants :

- **Preliminary Design Review (PDR)** : Présentation des premiers requis R.F.1 et R.F.2 ([Annexe A](#)), dont les démonstrations vidéo du serveur web pour la station au sol et du robot physique, ainsi que la simulation Gazebo avec deux robots.
- **Critical Design Review (CDR)** : Documentation révisée du projet et démonstrations vidéo des requis R.F.1 à R.F.5, R.F.10, R.C.1, ainsi qu'un prototype de R.F.8. Les requis R.C.3, R.Q.1 et R.Q.2 doivent également être satisfaits ([Annexe A](#)).
- **Readiness Review (RR)** : Présentation du produit final, incluant toutes les fonctionnalités prévues ([Annexe A](#)). Démonstrations vidéo des requis finaux complétés et remise du code source complet, des tests automatisés et des instructions de compilation.

### 1.2 Hypothèse et contraintes (Q3.1)

Le projet repose sur plusieurs hypothèses techniques et non techniques qui influencent la réalisation et la planification des tâches. Les hypothèses principales de la création de la preuve de concept de NMS 4 sont les suivantes :

1. **Accès aux ressources nécessaires** : Les ressources nécessaires pour le développement, incluant les robots AgileX Limo et les équipements associés, seront disponibles en bon état et dans les délais requis. Le local prévu à leur utilisation sera disponible régulièrement.

2. **Conformité des outils de développement** : Les outils et les plateformes de développement, tel qu'Ubuntu et Docker, fonctionneront comme prévu et seront compatibles avec les exigences du projet.
3. **Conditions de tests prévisibles** : Les conditions de test en environnement contrôlé simuleront de manière réaliste les conditions d'exploration planétaire, sans variations imprévues pouvant affecter les résultats.

En plus des hypothèses influençant le projet, plusieurs contraintes internes et externes déterminent la forme finale du bien livrable, certaines issues des exigences de l'appel d'offres, d'autres spécifiques au contexte de travail :

1. **Robots** : Les robots utilisés seront les AgileX Limo fournis par l'Agence, sans modifications matérielles ou ajout de capteurs.
2. **Batteries** : Les robots fonctionnent sur batteries, ce qui limite leur temps d'utilisation continue. Il est donc nécessaire d'assurer que toutes les batteries disponibles soient toujours chargées, en charge ou en cours d'utilisation. De plus, l'utilisation des batteries doit être planifiée en considérant les besoins des autres équipes.
3. **Environnement de tests** : L'utilisation des robots sera limitée à la salle dédiée fournie par l'ASC (surnommée la volière), avec réservation préalable (maximum de deux équipes simultanément) et simulant des conditions d'exploration planétaire.
4. **Capteurs des robots** : Les robots utiliseront exclusivement les capteurs fournis par l'ASC (IMU, caméra 3D, caméra RGB, et LIDAR).
5. **Communication** : La communication avec les robots se fera exclusivement par Wi-Fi (2,4 GHz et 5 GHz)

### **1.3 Biens livrables du projet (Q4.1)**

Les principaux livrables et artéfacts correspondants de ce projet et à ses requis ([Annexe A](#)) sont présentés dans le tableau suivant (tableau 1).

Livrable	Artéfact	Date de livraison
Prototype préliminaire (PDR)	1. Documentation initiale du projet (PDF) 2. Démonstration vidéo du web (R.F.1 et R.F.2) 3. Démonstration vidéo du robot (R.F.1) 4. Démonstration vidéo de la simulation Gazebo (R.F.2) 5. Fichiers Docker pour le serveur web et la simulation 6. Code source initial	20 septembre 2024
Système avec fonctionnement partiel (CDR)	1. Documentation révisée du projet (PDF) 2. Présentation technique 3. Démonstrations vidéo des requis R.F.1 à R.F.5, R.F.10, R.C.1 et du prototype du requis R.F.8 4. Validation des requis R.C.3, R.Q.1 et R.Q.2 5. Fichier Docker mis à jour pour tous les services 6. Code source mis à jour	1 <sup>er</sup> novembre 2024
Système complet (RR)	1. Documentation finale du projet (PDF) 2. Présentation orale du produit final 3. Démonstrations vidéo des fonctionnalités de tous les requis obligatoires et optionnels sélectionnés 4. Système complet et conteneurisé 5. Code source complet et instructions de compilation 6. Tests automatisés et fichiers Docker Compose associés 7. Document « Tests.pdf » avec les étapes de tests fonctionnels non automatisés	3 décembre 2024

Tableau 1 : livrables, artéfacts et leurs dates prévues de livraison

## **2. Organisation du projet**

### **2.1 Structure d'organisation (Q6.1)**

L'équipe de projet, composée de six membres, fonctionne en mode agile sous la supervision d'un coordinateur de projet. Cette structure d'organisation vise à maximiser l'efficacité, la flexibilité et la collaboration au sein de l'équipe et avec l'Agence Spatiale Canadienne (ASC). Le coordinateur de projet est responsable de la vision globale du projet, de la définition des priorités et de la gestion des tâches. Il travaille en étroite collaboration avec l'équipe pour assurer que les objectifs du projet sont atteints. Les cinq développeurs-analystes sont chargés de

la conception, du développement et des tests des fonctionnalités du produit. Ils travaillent en étroite collaboration pour livrer des incrémentums de produit fonctionnels à chaque sprint. Les responsabilités sont partagées, permettant au coordinateur de participer au développement et aux développeurs-analystes de contribuer à la gestion des priorités et des tâches. Cette structure collaborative permet aux membres de l'équipe de travailler avec des rôles bien définis tout en conservant une grande flexibilité permettant de s'adapter aux besoins du projet.

La communication au sein de l'équipe est essentielle pour le succès du projet. Plusieurs canaux de communication sont établis : des réunions bihebdomadaires (*stand-up*) pour discuter des progrès, des obstacles et des plans jusqu'à la prochaine réunion, des revues de sprint pour présenter les fonctionnalités développées et recueillir des retours, et des rétrospectives de sprint pour discuter des points positifs et des améliorations possibles pour le prochain sprint, particulièrement après les jalons importants (« post-mortem »). L'équipe utilise divers outils pour faciliter la collaboration, la communication et la gestion de projet : GitLab pour la gestion du code source, la documentation et la gestion des tâches, git pour le contrôle des versions et Discord pour la communication en temps réel.

Le coordinateur de projet est le principal point de contact pour l'ASC et les parties prenantes. Les membres de l'équipe doivent communiquer leurs décisions importantes et leurs avancements aux autres membres de l'équipe. Chaque semaine, un rapport d'avancement est produit pour communiquer à l'ASC les avancements de la semaine et la planification de la semaine suivante.

Les décisions sont prises de manière collaborative, avec une forte implication de tous les membres de l'équipe. En cas d'urgence, le coordinateur de projet a le dernier mot sur les priorités et les exigences, mais les décisions techniques sont prises par consensus au sein de l'équipe de développement ou par les membres impliqués dans la tâche concernée si l'impact y est restreint.

## **2.2 Entente contractuelle (Q11.1)**

Nous proposons à l'ASC une entente contractuelle de type « Contrat à terme – Temps plus frais ». Ainsi, nous nous engageons à consacrer 630 heures au développement du projet ([Section 4.1](#)), garantissant un niveau d'effort proportionnel aux résultats attendus pour atteindre les exigences du projet ([Annexe A](#)). Nos obligations se termineront à l'échéance du contrat, indépendamment de l'état final du produit, de sorte que nous garantissons un investissement en temps et en effort, sans engagement quant à la forme finale du produit.

Ce type de contrat offre plusieurs avantages pour l'ASC. D'abord, il procure une flexibilité maximale, permettant d'ajuster les spécifications et priorités en cours de route afin d'utiliser au mieux les ressources pour atteindre un produit fonctionnel ([Section 4.1](#)). Cette approche permet également de réduire les coûts initiaux et simplifie de processus de démarrage du projet, permettant à l'ASC d'obtenir une preuve de concept plus rapidement.

En revanche, ce contrat n'offre aucune garantie du coût final, ce qui peut entraîner un risque de dépassement budgétaire. Dans le cadre de ce projet, nous considérons que ce risque est limité par le nombre d'heures et de membres d'équipe maximum fixé dans la demande de soumission de l'ASC ([Section 4.4](#)). Par ailleurs, il n'y a aucun incitatif financier à réduire le temps et les coûts, ce qui peut affecter l'efficacité. Cependant, les échéanciers et les objectifs établis par l'ASC permettent de maintenir un haut niveau de qualité et de respecter le temps prévu dans le contrat ([Annexe C](#)). Dans ce contexte, une efficacité accrue n'apporterait pas de bénéfice direct, tandis qu'un travail de qualité est plus avantageux pour l'ASC.

Ce type de contrat est donc idéal pour ce projet. Il offre à la fois la flexibilité nécessaire pour ajuster les spécifications et priorités en fonction des besoins et permet de commencer rapidement, assurant ainsi des résultats préliminaires pour l'ASC dès que possible.

### 3. Description de la solution

#### 3.1 Architecture logicielle générale (Q4.5)

La phase de planification et de conception est déterminante pour la réussite de tout projet. En définissant précisément l'architecture de l'application, on garantit une mise en œuvre efficiente, une maintenabilité optimale et, par conséquent, la pérennité du système. L'architecture proposée a été élaborée avec un souci de rigueur, en tenant compte des exigences du présent devis:

- **Serveur web client** : Ce serveur permet de rendre le tableau de bord du robot accessible aux utilisateurs via une interface conteneurisée à l'aide de Docker, assurant portabilité et scalabilité. Développé avec le cadre Angular, il offre une interface utilisateur interactive et dynamique. Le serveur web communique avec les clients via HTTP pour la transmission des données et s'appuie sur WebSocket pour établir une communication bidirectionnelle en temps réel avec le serveur backend.
- **Serveur backend** : Central dans l'architecture, le serveur backend assure l'intermédiation entre les utilisateurs et les robots. Il relaie les requêtes des utilisateurs et traite les données reçues des robots. Conteneurisé avec Docker et développé avec NestJS, il offre une structure modulaire, facilitant la gestion des communications réseau. Il communique avec la base de données via HTTP et interagit avec ROS (« Robot Operating System ») via ROSBridge et WebSocket, permettant la gestion des données en temps réel et la publication/souscription aux « topics » ROS nécessaires à l'opération des robots ([R.L.2](#)).

- **Serveur de base de données**: Ce composant gère le stockage des données générées par les robots. Basé sur MongoDB, une base de données NoSQL, il assure la fiabilité et l'accessibilité des informations critiques via des requêtes HTTP avec le serveur backend.
- **Rosbridge websocket** : Ce composant assure la communication entre les robots, la simulation et le backend en utilisant WebSocket. Il permet au backend de publier et de souscrire aux « topics » ROS, garantissant ainsi une transmission en temps réel des données essentielles à la gestion des robots.
- **Simulation gazebo** : Le simulateur Gazebo est utilisé pour reproduire le comportement des robots dans des environnements virtuels. Conteneurisée pour simplifier son déploiement, la simulation communique avec le backend via ROSBridge et WebSocket, garantissant une synchronisation et une interaction en temps réel avec l'ensemble du système.
- **Les robots** : déployés dans un environnement contrôlé, les robots communiquent avec le backend via ROSBridge en utilisant WebSocket, assurant une transmission rapide et fiable des données pour un contrôle et une analyse en temps réel de leur comportement ([R.L.3](#)).

L'architecture du projet est présentée dans la figure ci-dessous (figure 1).

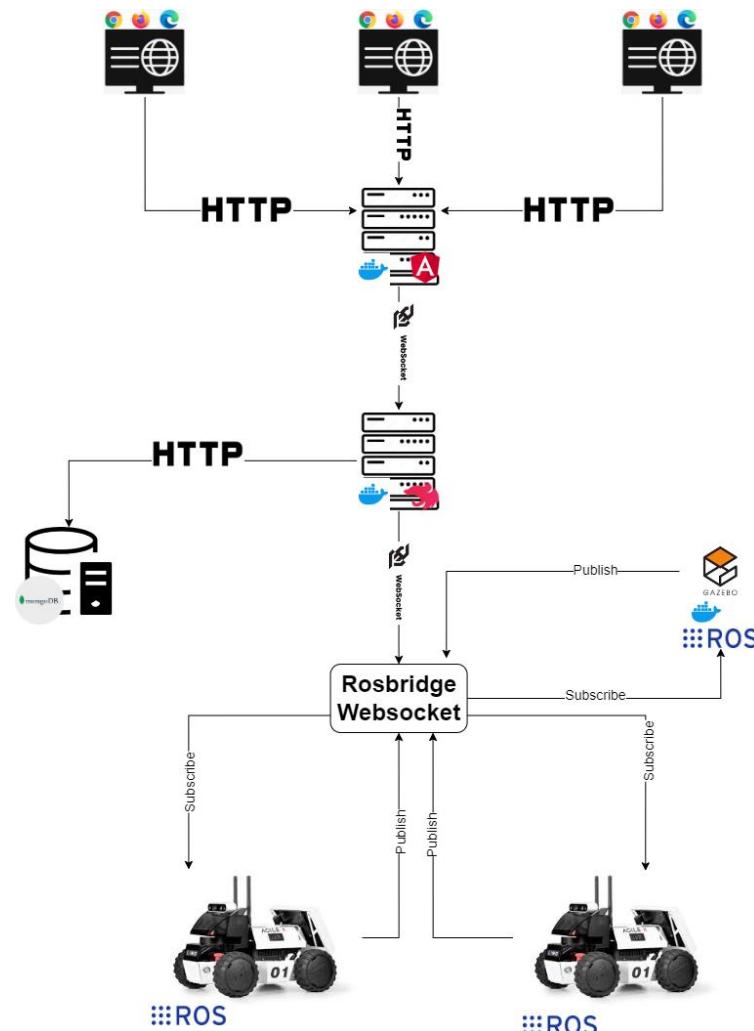


Figure 1 : Vue haut niveau de l'architecture du système d'exploration multirobots

### 3.2 Station au sol (Q4.5)

La station au sol est un composant central du système, constitué de trois modules principaux : une interface utilisateur, un serveur et une base de données. Chacun de ces éléments est conçu pour répondre aux exigences fonctionnelles et techniques du projet, en assurant une communication fluide avec les robots et une gestion efficace des données. La station au sol prend la forme d'un ordinateur portable sur lequel roulent les trois modules tels que requis par R.M.4 ([Annexe A](#)).

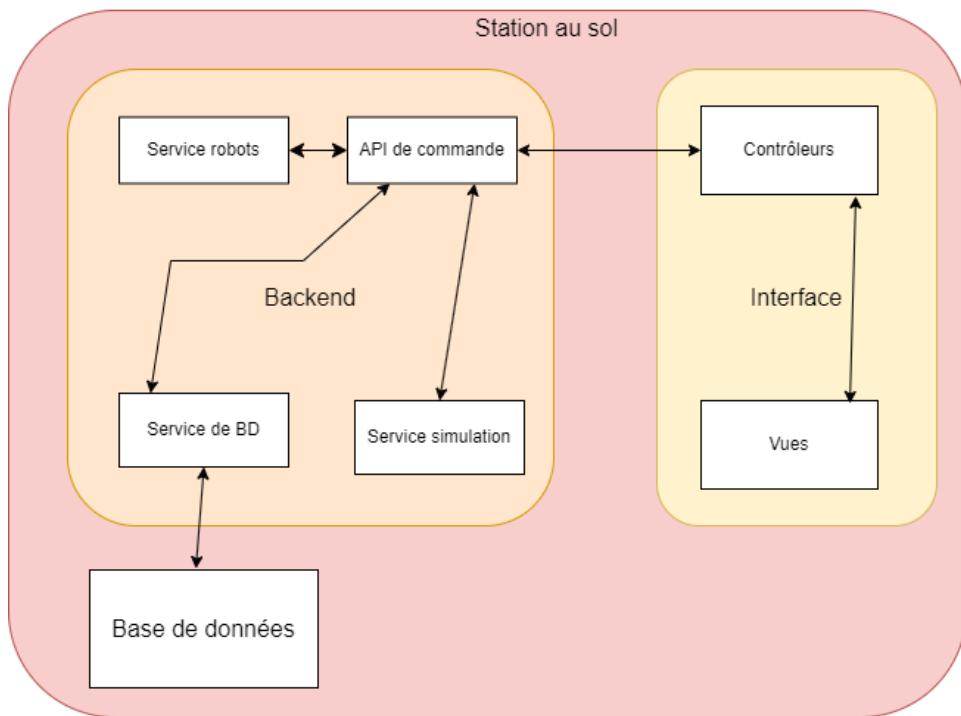
L'interface utilisateur (UI) constitue le point d'entrée du système, permettant aux opérateurs de contrôler les robots physiques et d'interagir avec la simulation comme requis par R.L.2 et R.L.4 ([Annexe A](#)). Développée avec HTML, CSS, Angular, et TypeScript, cette interface web est privilégiée pour sa simplicité de déploiement et sa compatibilité multiplateforme. Le choix d'Angular repose sur l'expérience de l'équipe avec ce cadre, acquise lors d'un projet antérieur, garantissant une maîtrise optimale de l'outil et une courbe d'apprentissage réduite.

L'interface est divisée en deux parties : la vue, qui représente la partie visible et interactive pour l'utilisateur, et les contrôleurs, qui relaient les commandes vers l'API du système. Cette approche modulaire améliore la maintenabilité et permet une séparation claire des responsabilités. L'architecture de l'interface utilisateur est décrite en plus grand détail dans la [section 3.5](#).

L'API est le cœur de la communication entre les différentes composantes du système. Elle repose sur des modèles de données et des services structurés pour relayer efficacement les commandes de l'interface vers les robots et la simulation. Une classe dédiée est chargée d'établir la communication via WebSocket, permettant une transmission en temps réel des commandes et des réponses ([R.M.2](#)). Deux autres classes gèrent les échanges avec les robots et la simulation, garantissant une faible latence, critère essentiel dans les systèmes embarqués en temps réel. Bien que le déploiement de l'application ne soit pas prévu à grande échelle, cette architecture modulaire permet une évolution future aisée.

La base de données, de type NoSQL (MongoDB), est choisie pour sa flexibilité et son adéquation avec le format des données manipulées (objets JavaScript). Ce choix est justifié par la nature relativement simple et peu volumineuse des données à stocker. MongoDB permet une gestion efficace de ces données tout en assurant une bonne évolutivité, notamment en prévision d'une externalisation possible de la base de données pour des extensions futures. Cette solution optimise également l'utilisation de la bande passante en minimisant la complexité des échanges de données.

Les trois sous-systèmes de la station au sol sont conçus pour fonctionner de manière autonome, en ne dépendant que des données d'entrée reçues. Cela permet une flexibilité accrue, car toute modification apportée à un sous-système n'affectera pas les autres, favorisant ainsi une approche modulaire et évolutive. La communication entre les sous-systèmes est standardisée grâce à l'utilisation d'un format d'échange JSON, garantissant une interopérabilité fluide et une gestion cohérente des messages échangés. La figure 2 ci-dessous illustre l'architecture globale de la station au sol, mettant en évidence les interactions entre les différents sous-systèmes.

**Figure 2 :** Architecture globale de la station au sol

### 3.3 Logiciel embarqué (Q4.5)

La figure B.1 ([Annexe B](#)) décrit l'architecture du logiciel embarqué choisie. Cette architecture repose sur une division atomique des fonctionnalités, où chaque nœud ROS est spécifiquement dédié à une tâche unique. Ce choix présente plusieurs avantages en termes de clarté, de modularité et de facilité de maintenance. En attribuant une tâche distincte à chaque nœud, qu'il s'agisse de la gestion des missions, du retour à la base ou de la mise à jour du code, chaque composant reste isolé et indépendant des autres. Cela simplifie considérablement le développement et les tests, car il devient plus facile de diagnostiquer et corriger les problèmes de manière ciblée sans impacter l'ensemble du système. Certains de ces nœuds proviennent de « package » et de « frameworks » de tiers, tels que **Slam Toolbox** [1] pour les nœuds liés à la cartographie et **Nav2** [2] pour la gestion de l'exploration, en publiant directement sur le "Topic" de vitesse. En complément, une dépendance importante est **orbbec\_camera** [5], utilisée pour la détection des élévations négatives et la prévention des chutes (R.F.13). En plus de ces intégrations, des nœuds spécialisés ont été développés pour répondre aux besoins spécifiques de l'ASC, tels que les nœuds d'identification, de mise à jour du logiciel, de serveur de mission et de gestion de la batterie.

L'architecture de communication entre le serveur et les robots repose sur **Rosbridge WebSocket**, évitant ainsi la création d'un module de communication par socket spécifique et facilitant une intégration directe avec ROS. En traitant directement les messages JSON envoyés par le serveur, ROS gère efficacement les échanges sans ajout de complexité. Le serveur est chargé de relayer les tâches critiques reçues de l'opérateur, telles que l'identification des robots (R.F.1), le lancement et l'arrêt des missions (R.F.2), le retour à la base (R.F.6), ainsi que la supervision des actions des robots en temps réel avec une fréquence minimale de 1 Hz (R.F.3). Par ailleurs, le serveur coordonne également la mise à jour des logiciels (R.F.14) et l'édition de code à distance via l'interface utilisateur pour ajuster les comportements des robots avant ou entre les missions (R.F.16).

Les robots, une fois les missions lancées, explorent l'environnement de manière autonome (R.F.4), évitent les obstacles détectés par leurs capteurs (R.F.5), et génèrent des cartes en temps réel grâce à des nœuds spécialisés connectés à **Slam Toolbox**. Ces cartes sont affichées sur l'interface utilisateur de la station au sol et enregistrées pour consultation ultérieure (R.F.8, R.F.9, et R.F.18). Lors de la mission, si le niveau de batterie d'un robot devient inférieur à 30 %, une commande automatique de retour à la base est déclenchée, rapprochant le robot de sa position initiale à moins de 0,3 mètre (R.F.7).

Une fonctionnalité notable est la communication directe entre robots. Grâce au **P2P Node**, les robots partagent leur position et leur distance par rapport à leur point de départ ou à la station au sol, permettant une collaboration en temps réel sans passer par le serveur (R.F.19). Cela est déclenché par une commande dédiée, mais la communication reste indépendante du serveur central.

Enfin, une base de données sur la station au sol enregistre les informations clés de chaque mission, notamment la date, la durée, les robots impliqués, et la distance parcourue (R.F.17). Ces données sont accessibles via une interface utilisateur intuitive permettant de consulter l'historique des missions et d'ouvrir les cartes générées précédemment.

En conclusion, cette architecture modulaire et intégrée repose sur des nœuds spécialisés, des dépendances tierces comme **Nav2**, **Slam Toolbox**, et **orbbec\_camera**, ainsi que sur une infrastructure de communication robuste via **Rosbridge WebSocket**, garantissant une réponse efficace aux requis fonctionnels du projet.

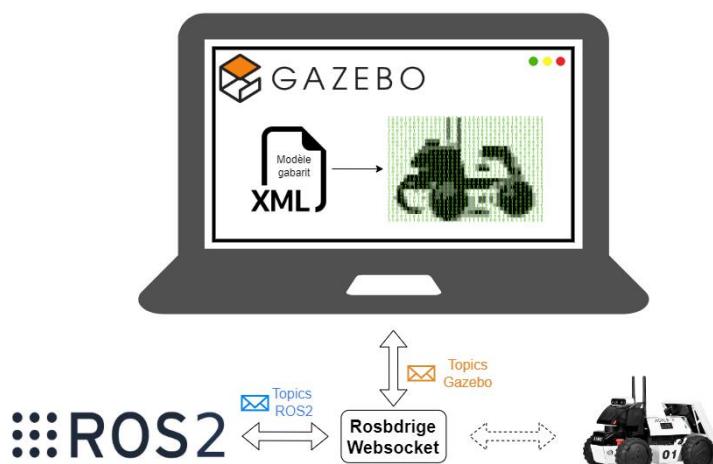
### 3.4 Simulation (Q4.5)

Le simulateur Gazebo a été retenu pour son intégration efficace avec ROS2 [3]. En effet, ROS2 peut communiquer avec Gazebo via la même structure de messages que celle utilisée pour interagir avec des robots physiques. Cela signifie que les nœuds ROS2 peuvent envoyer et recevoir des messages (« topics ») avec Gazebo, exactement comme ils le feraient avec un robot réel. De plus, Gazebo est capable de publier des données de capteurs virtuels, tels que des lasers, des caméras, et d'autres senseurs, en suivant la même syntaxe réseautique que ROS, garantissant ainsi une interaction fluide et réaliste. La Figure 4 illustre l'architecture de simulation envisagée.

Les capteurs embarqués prévus pour la conception de notre système, tels que le LiDAR, les caméras et les IMUs, sont pris en charge nativement par Gazebo. Ce simulateur permet la gestion de multiples robots, rendant possible la simulation concurrente de plusieurs entités. De plus, Gazebo offre une simulation physique précise. La gravité, l'inertie, les collisions et le frottement sont paramétrables, ce qui permet un comportement réaliste des robots et de leur interaction avec l'environnement simulé, pouvant également être visualisé en 3D.

Les éléments simulés peuvent être définis via un modèle XML, permettant de personnaliser les caractéristiques de chaque objet dans la simulation, telles que la géométrie ou les capteurs intégrés. Ce modèle XML offre la flexibilité nécessaire pour définir précisément les propriétés de chaque objet et générer autant d'instances que nécessaire.

En résumé, cette uniformité dans la structure de message permet de développer et de tester des algorithmes dans un environnement simulé sans changer la logique de communication utilisée par un robot réel. Cela simplifie considérablement le processus de développement et l'assurance qualité. En outre, les options qu'offre Gazebo faciliteront la simulation des requis nécessitant la détection d'obstacles et la génération d'une carte 3D, tels les requis R.F.5, R.F.8, R.F.11 et R.F.15 ([Annexe A](#)).



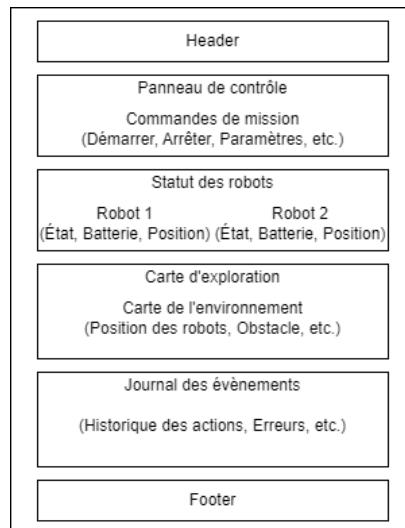
**Figure 4 : Architecture de simulation**

### 3.5 Interface utilisateur (Q4.6)

L'interface utilisateur est conçue en tant que Single Page Application (SPA) pour garantir une navigation fluide et réactive, tout en étant intuitive et simple d'utilisation. Elle est conçue en accord avec les exigences fonctionnelles et de conception établies pour le projet. Elle est structurée en plusieurs sections principales : le panneau de contrôle, l'affichage du statut, la carte de l'environnement et les journaux d'activités.

Le panneau de contrôle permet de gérer les commandes de mission, telles que « Lancer la mission » et « Terminer la mission » (R.F.2), ainsi que la commande « Identifier » pour les robots (R.F.1). Il est conçu pour répondre aux exigences de réponse en temps réel aux commandes (R.F.3) et pour afficher les informations de contrôle nécessaires à la gestion des robots (R.F.10). L'affichage du statut, quant à lui, offre une vue en temps réel de l'état des robots (R.F.3) ainsi que de leur niveau de batterie (R.F.7), avec une mise à jour minimale à une fréquence de 1 Hz, répondant ainsi aux exigences fonctionnelles critiques du système.

La carte de l'environnement constitue une autre section essentielle de l'interface. Elle affiche en continu la carte et la position des robots (R.F.8, R.F.9), avec une représentation en 3D et en couleur (R.F.11). Cette carte est mise à jour en temps réel pendant les missions et permet de spécifier des zones de sécurité (R.F.20), ce qui est essentiel pour garantir le bon déroulement des opérations. Enfin, la section des journaux permet d'afficher les journaux d'activité et de collecter les logs de débogage (R.C.1). Elle offre un accès en continu aux logs et permet la consultation des missions précédentes, assurant ainsi une traçabilité et une capacité de diagnostic complète, conformément aux exigences de collecte des logs (R.C.1).



**Figure 5 :** Architecture de l'interface utilisateur

Les choix de conception de l'interface utilisateur, représentée dans la figure 5, sont justifiés par les exigences fonctionnelles et les besoins de l'application. Le panneau de contrôle répond aux exigences R.F.2, R.F.1, et R.F.6 en permettant aux utilisateurs de lancer et d'arrêter les missions, d'identifier les robots, et de gérer les commandes en temps réel. Cette section offre un accès rapide et efficace aux fonctions critiques pour la gestion des missions. L'affichage du statut, conçu pour répondre aux exigences R.F.3 et R.F.7, fournit une vue en temps réel sur l'état des robots et leur niveau de batterie. Cela permet une gestion proactive des missions, en assurant que les opérateurs soient constamment informés des conditions des robots.

La carte de l'environnement, quant à elle, satisfait les exigences R.F.8, R.F.9, et R.F.11. Elle permet de visualiser l'environnement et la position des robots en temps réel avec une représentation en 3D et en couleur, facilitant ainsi la compréhension et l'interaction avec l'espace simulé ou réel. Enfin, la section des journaux garantit la collecte des logs (R.C.1), en fournissant une traçabilité complète des activités. Cette fonctionnalité est essentielle pour diagnostiquer les problèmes et analyser les missions en cours ou les missions passées, renforçant ainsi la robustesse et la fiabilité du système.

### **3.6 Fonctionnement général (Q5.4)**

Le fonctionnement général du système comprend le téléchargement du code source, la configuration de Docker pour le déploiement et la connexion aux robots. Les étapes principales sont présentées ci-dessous, et les actions détaillées pour chaque étape sont présentes dans le fichier [README.md](#) à la racine du projet.

#### **a. Télécharger le code**

Le projet est hébergé sur GitLab et peut être cloné en utilisant Git. Cette étape permet de récupérer l'ensemble des fichiers du dépôt et de préparer le projet pour la suite des opérations. La commande, incluant le lien vers le répertoire, est la suivante :

```
git clone https://gitlab.com/polytechnique-montr-al/inf3995/20243/equipe-107/gepetto.git
```

#### **b. Utilisation de Docker**

Le projet utilise Docker pour simplifier la gestion des services, des dépendances et le déploiement. Les trois services non embarqués (Client : application front-end Angular, Server : API back-end NestJS, et Gazebo : environnement de simulation) sont configurés dans le fichier « compose.yaml » pour un lancement simultané. Cette approche permet de centraliser la construction et le démarrage des services, avec possibilité de lancer chaque service indépendamment.

Vous trouverez l'information nécessaire à l'installation de Docker dans leur [documentation officielle](#) et les modifications nécessaires pour accorder les bonnes permissions dans leur documentation des [étapes post-installation](#).

### c. Démarrer le système

Pour démarrer le système, exécutez la commande suivante dans le répertoire « launch » :

```
./start.sh
```

Cette commande permet d'effectuer deux actions : elle récupère l'adresse IP de l'ordinateur pour permettre aux utilisateurs du réseau de se connecter au serveur lancé dans un conteneur Docker et exécute docker-compose pour construire les conteneurs nécessaires au client, au serveur et à la simulation. Une fois lancé, le serveur est accessible à l'adresse <http://localhost:3000/> et l'interface utilisateur à l'adresse <http://localhost:4200/>. La simulation Gazebo démarre au même moment et une exploration est prête à être lancée.

### d. Lancer une mission

Avant de lancer une mission, assurez-vous que les robots sont bien connectés. Si ce n'est pas le cas, connectez-les en appuyant sur « Robot 1 » et « Robot 2 », et vérifiez que la simulation est bien démarrée. Cliquez ensuite sur le « Lancer la mission » pour commencer l'exploration en sélectionnant entre le mode simulation et le mode physique et sur « Terminer la mission » pour arrêter les robots. Vous pouvez accéder aux logs des missions à partir de la section « Historique des logs » de l'en-tête.

## 4. Processus de gestion

### 4.1 Estimations des coûts du projet (Q11.1)

Le coût total du projet est estimé en fonction du temps de travail alloué, des taux horaires définis pour chaque rôle au sein de l'équipe, du nombre de membres de l'équipe responsable du projet et des coûts fixes de production incluant les outils logiciels et matériel nécessaire à l'accomplissement des requis.

La répartition du travail est basé sur une charge de travail maximale de 630 heures-personnes, conformément aux exigences du projet. L'équipe est composée de six membres : cinq développeurs-analystes et un coordonnateur de projet. Le taux horaire estimé des développeurs-analystes est de 130 \$/heure et celui du coordonnateur est de 145 \$/heure. Le tableau suivant présente les heures de travail et les coûts associés pour chaque rôle (tableau 2).

Rôle	Taux horaire (\$CAD/h)	Temps alloués (h)	Coût total (\$CAD)
Développeurs	130	520	67 600
Coordonnateur	145	110	15 950
Total estimé		630	83 550

Tableau 2 : Coût des ressources humaines

Cette estimation inclut la planification, le développement, les tests, la validation et la documentation du système multirobots.

Deux robots AgileX Limo Pro sont nécessaires pour l'accomplissement du système demandé par l'ASC. Les coûts associés sont présentés dans le tableau suivant (tableau 3) :

Matériel	Coût unitaire (\$CAD)	Nombre d'unités	Coût total (\$CAD)
Robot AgileX Limo	4 351,39 [4] <sup>1</sup>	2	8 702,78
Total estimé			8 702,78

**Tableau 3 :** Coûts matériels fixes nécessaires pour l'accomplissement du projet

Les outils organisationnels, logistiques et de développement n'induiront aucun coût supplémentaire au projet par leur nature gratuite et open source. La salle de tests fonctionnels, soit le local M-7703 du pavillon Lassonde de Polytechnique Montréal, et ses installations sont gracieusement fourni par l'ASC et n'encourront donc aucun coût supplémentaire de notre part. Le tableau suivant résume les coûts du projet tel que défini dans ce devis (tableau 4) :

Ressource	Coût (\$CAD)
Ressources humaines	83 550
Matériel	8 702,78
Total estimé	92 252,78

**Tableau 4 :** Résumé des coûts du projet

Le coût total prévu pour la réalisation du projet est donc de 92 252,78\$. Cette estimation prend en compte les ressources humaines, la durée nécessaire pour compléter les tâches requises pour chaque phase de développement et l'acquisition du matériel nécessaire au projet.

## 4.2 Planification des tâches (Q11.2)

La planification des tâches pour ce projet suit une approche rigoureuse, mais flexible de gestion du temps et des ressources, répartie entre trois principaux jalons et les membres de l'équipe. La planification est structurée sur trois axes : l'allocation du temps des requis et des tâches par un diagramme de Gantt, l'identification et la définition des jalons clés et la répartition des responsabilités au sein de l'équipe.

---

<sup>1</sup> Coût en \$CAD obtenue par conversion à valeur de 1.36 \$CAD pour 1\$USD en date du 18 septembre 2024.  
Le prix originel est de 3200 \$USD.

La figure C.1 ([Annexe C](#)) est le diagramme de Gantt du projet représentant l'allocation du temps pour chaque groupe de tâches majeures du projet. Les requis sont considérés comme l'équivalent d'une tâche et ont été rassemblé en plusieurs groupes afin de faciliter la visualisation et le suivi. Ces groupes ont été divisés en plusieurs phases suivant les jalons clés afin de couvrir les principaux aspects, de la recherche et développement à la validation et la documentation :

L'utilisation de GitLab nous permet de structurer et de suivre efficacement l'avancement du projet et les tâches attribuées à chaque membre de l'équipe. La planification du projet est organisée autour de trois jalons critiques, chacun correspondant à une étape importante du développement du produit :

1. **Preliminary Design Review (PDR)** : Prototype préliminaire, à livrer le 20 septembre 2024
2. **Critical Design Review (CDR)** : Système partiel fonctionnel, prévu pour le 1<sup>er</sup> novembre 2024
3. **Readiness Review (RR)** : Système final complet avec toutes les fonctionnalités, à livrer le 3 décembre 2024

Concernant la répartition des tâches des membres de l'équipe, nous avons opté pour une attribution flexible des responsabilités plutôt que l'attribution rigide de l'accomplissement des tâches. Cela permet une plus grande adaptabilité au sein de l'équipe, tout en assurant que chaque domaine du projet est couvert et que toutes les tâches critiques sont réalisées dans les temps. Chaque membre de l'équipe est donc responsable de la supervision et de la gestion des tâches dans leur domaine de compétence, tout en ayant la possibilité de déléguer ou de collaborer selon les besoins du projet. Cela garantit une répartition dynamique des tâches et favorise la continuité en cas de surcharge de travail ou d'imprévus. Les responsabilités de chacun sont présentées dans le tableau suivant (Tableau 5) :

Membre de l'équipe	Responsabilité générale	Requis et tâches associées
Thomas Rouleau	Coordinateur du projet, gestion des ressources et des livrables	Recherche et Analyse des Exigences, Documentations de Conception et rapports d'avancement, démos vidéo, présentations du produit, requis généraux R.M et R.L, R.F.9, R.F.20, R.C.1 ( <a href="#">Annexe A</a> )
Ely Cheikh Abass	Responsable du système embarqué	R.F.1, R.F.2, R.F.4, R.F.6, R.F.7, R.F.13, R.F.15 ( <a href="#">Annexe A</a> )
Loïc Nguemegne	Responsable de l'intégration des systèmes et Docker	Configuration de l'Environnement de Développement, R.L.4, R.F.8, R.F.11, R.F.14, R.F.12 ( <a href="#">Annexe A</a> )
Abdul-Wahab Chaarani	Responsable de la simulation et des tests et validations	R.F.4, R.F.5, R.C.2, R.C.3, R.Q.2 ( <a href="#">Annexe A</a> )
Ivan Samoylenko	Responsable du backend	R.F.2, R.F.17, R.F.18, R.F.19, R.Q.1 ( <a href="#">Annexe A</a> )
Omar Benzekri	Responsable du frontend et de l'interface utilisateur	R.F.3, R.F.9, R.F.10, R.F.16, R.F.20, R.C.4 ( <a href="#">Annexe A</a> )

**Tableau 5 :** Répartitions des responsabilités des tâches aux membres de l'équipe

### 4.3 Calendrier de projet (Q11.2)

Le calendrier du projet est structuré autour des phases clés du développement, avec des dates jalons pour chaque livraison. Ce calendrier, présenté dans le tableau suivant, est conçu de manière à fournir une vue d'ensemble des délais prévus pour l'achèvement des différentes étapes du projet (tableau 6).

Phase du projet	Date de début	Date de fin	Jalon clé
Prototype préliminaire (PDR)	26 août 2024	20 septembre 2024	Soumission PDR, R.F.1, R.F.2
Développement partiel du système (CDR)	21 septembre 2024	1 <sup>er</sup> novembre 2024	Soumission CDR, Présentation technique, R.F.1 à R.F.5, R.F.8, R.F.10, R.C.1, R.C.3, R.Q.1, R.Q.2 ( <a href="#">Annexe A</a> )
Développement final (RR)	2 novembre 2024	3 décembre 2024	Livraison finale, Présentation du produit final, Tous les requis obligatoires et sélectionnés ( <a href="#">Annexe A</a> ), Tests et instructions de compilation et de lancement

**Tableau 6 :** Calendrier des dates cibles de terminaison des phases importantes

Ce calendrier garantit que les phases importantes sont achevées dans les temps impartis et que les jalons définis sont respectés. Chaque membre de l'équipe est informé des dates limites et des périodes de développement cruciales. Le suivi en temps réel de l'avancement via GitLab permettra de maintenir une coordination efficace entre les différents acteurs.

#### **4.4 Ressources humaines du projet (Q11.2)**

L'équipe est composée de six membres, chacun ayant des compétences spécifiques qui contribuent à la réalisation du projet. Chaque membre apporte une expertise spécifique, garantissant ainsi que les compétences nécessaires sont disponibles pour couvrir tous les aspects du projet, de l'intégration logicielle aux tests en simulation. Le coordonnateur est responsable de la gestion du temps et des ressources, ainsi que de la communication avec l'Agence. Le suivi des tâches et des responsabilités sera effectué via GitLab, garantissant une gestion transparente et efficace. Le tableau ci-dessous présente la répartition des rôles ainsi que les qualifications des membres de l'équipe (tableau 7).

Rôle	Membre de l'équipe	Qualifications	Responsabilités principales
Coordonnateur de projet	Thomas Rouleau	Supervision générale, ROS 2, Gestion de projet	Supervision générale, Développement ROS, validation de code
Développeur embarqué	Ely Cheikh Abass	ROS 2, Protocoles de communication sans fil	Développement ROS, protocoles de communications
Développeur embarqué	Loïc Nguemegne	Docker, ROS 2, Intégration système	Conteneurisation, Développement ROS
Développeur embarqué	Abdul-Wahab Chaarani	Gazebo, Tests de validation, ROS 2	Développement Gazebo, Tests et validation unitaire et en simulation, validation de code
Développeur Full stack	Ivan Samoylenko	Protocoles de communication, Serveurs, Base de données (DB)	Serveur et base de données, protocoles de communication, validation de code
Développeur web	Omar Benzekri	Développement web, Protocoles de communication	Interface web, protocoles de communications

**Tableau 7 :** Rôles, expertises et qualifications des membres de l'équipe de développement du projet de multirobots d'exploration

## 5. Suivi de projet et contrôle

### 5.1 Contrôle de la qualité (Q4)

Le contrôle de la qualité du projet repose sur un ensemble de lignes directrices strictes et des processus de vérification rigoureux afin de garantir la fiabilité et la performance des livrables. Chaque composant, fonctionnalité ou changement de l'application suit un processus de révision en trois étapes : développement, tests manuels et revue de code.

**Revue systématique du code :** Chaque contribution au code, qu'il s'agisse de la partie « front-end » (Angular), « back-end » (NestJS), embarqué (ROS) ou simulation (Gazebo), est développée sur une branche tirée de la branche de développement « develop ». Une fois terminée, elle est soumise à une révision par au moins deux développeurs expérimentés de l'équipe. Les réviseurs sont choisis en fonction de l'aspect du projet concerné : un ou deux experts spécifiques par partie, ainsi qu'un réviseur général ayant une connaissance approfondie de la structure globale du projet. Cette révision porte sur les aspects suivants :

- Conformité aux normes de codage : Vérification des normes définies dans le fichier pour garantir la lisibilité et la maintenabilité du code.
- Couverture des tests unitaires : Vérification que chaque fonction critique est testée à travers des suites de tests définies dans les dossiers `src` respectifs des dossiers `client` et `server` (dans le cas d'une fonctionnalité web).
- Performance et efficacité du code : Contrôle de la performance, notamment pour le module de communication robotique dans `/embedded_ws`.

**Tests unitaires et d'intégration :** L'exécution des tests se fait automatiquement à l'aide de bibliothèques, telles que Karma (pour le « front-end ») et Jest (pour le « back-end »). Ces tests sont organisés dans les fichiers de configuration `karma.conf.js` et `tsconfig.spec.json`, respectivement. Ils garantissent que chaque composant individuel fonctionne comme prévu et que l'intégration de ceux-ci dans le système global ne provoque aucune régression.

**Validation des livrables :** Avant chaque livraison, une session de test manuel est effectuée en environnement Gazebo pour valider les interactions entre les robots simulés et l'interface utilisateur. Les tests couvrent :

- Actions critiques : Effectuer les actions critiques, telles que l'initialisation des missions, l'enregistrement des logs, et la mise à jour en temps réel des statuts.
- Robustesse des communications inter-robots : Vérification via le module `com_bridge` dans `/embedded_ws`.

Les résultats des revues et des tests sont documentés et archivés dans le dossier `/rapports-hebdomadaires`, afin d'assurer une traçabilité complète.

## **5.2 Gestion de risque (Q11.3)**

Il existe plusieurs scénarios, qui, s'ils ne sont pas gérés correctement, pourraient compromettre le bon déroulement du projet, voire nuire à son achèvement.

Le risque principal au bon déroulement du projet est d'effectuer une mauvaise analyse des exigences ou une conception initiale erronée de la solution. Si l'équipe analyse mal les besoins et les objectifs attendus par l'ASC, alors des efforts et du temps peuvent être investis pour un développement invalide. De même, des erreurs majeures de conception initiales peuvent provoquer une modification et une restructuration majeure du code. Cette perte de productivité peut mettre à risque la bonne réalisation du projet et limiter notre capacité à le livrer dans son intégralité. Ayant connaissance de ce risque, nous nous sommes assurés de mettre le temps et l'attention nécessaire à la bonne analyse et conception initiale de notre solution. Nous avons également utilisé le lien de communication avec les représentants de l'ASC pour assurer la bonne compréhension de toutes les spécificités du projet. La gestion et la mise en place de solutions efficaces pour mitiger ce risque sont notre priorité.

L'introduction de bogues lors de la gestion des branches Git est fréquente dans les grands projets. Pour limiter ce risque, notre équipe applique une politique de « commits » atomiques et de « merge requests » réguliers, afin de minimiser la confusion entre les différentes versions du code et de prévenir les erreurs de résolution de conflits de fusion. Bien que des risques pour l'intégrité du code subsistent, leur impact est atténué grâce aux outils à notre disposition qui facilitent la résolution de problèmes. Les erreurs de code sont courantes, et bien que la plupart aient un impact minime, certaines peuvent causer d'importants retards. En raison de la probabilité élevée de ce risque et des dommages potentiels, il s'agit d'une priorité élevée pour notre équipe.

Dans les projets de système embarqué, des dommages aux composants matériels peuvent survenir, et leur impact est significatif. La fonction primaire de ce projet ne peut plus être accomplie si une pièce du robot est perdue ou endommagée. Pour prévenir de tels incidents, notre équipe a établi une politique interdisant de sortir le robot ou ses pièces du local réservé à son utilisation, de ne pas désassembler le robot, et d'exercer une grande prudence lors de sa manipulation. En cas d'accident, un plan d'action a été développé. Celui-ci consiste à solliciter l'aide des chargés du projet pour prévoir toute réparation nécessaire, à demander au professeur responsable du projet l'accès à un robot de remplacement. Tous les accidents seront financièrement couverts par un dépôt versé par chaque membre avant le début des manipulations des robots. Enfin, notre équipe s'est engagée à mettre en place rapidement des tests système de bout en bout pour vérifier le bon fonctionnement du système en cas de remplacement de matériel. Si le temps de remplacement ou de mise hors d'état d'un robot perdure, le développement nécessaire se poursuivra sur le simulateur et avec le robot fonctionnel. Ce risque

est d'une priorité moyenne pour notre équipe de par sa faible probabilité d'occurrence.

Un autre risque, bien que faible, est la possibilité du départ d'un membre de notre équipe ou de son impossibilité à travailler pour une période donnée (maladie grave, etc.). Nous comprenons que des imprévus de ce type font partie intégrante des projets de longue durée et sommes conscients de la charge de travail additionnelle qui pourrait en résulter si notre équipe se réduisait à cinq membres, même temporairement. Pour faire face à ces imprévus, nous avons opté pour la méthode Agile, jugée la plus adaptée pour maintenir la stabilité de l'équipe. À chaque sprint de deux semaines, nous redistribuons les tâches pour le sprint à venir de sorte que chaque tâche et responsabilité soit couverte. En outre, nous prenons en considération la possibilité qu'une tâche prenne plus de temps que prévu. La méthode Agile démontre ici son utilité : lors des réunions d'équipe hebdomadaires, nous discutons à tour de rôle de nos avancements passés et nos actions futures. Si une tâche est identifiée comme étant en retard, nous pouvons décider d'affecter davantage de ressource à celle-ci, en fonction des priorités que nous révisons constamment. Ce risque est donc d'une priorité faible pour notre équipe.

Le dernier risque principal identifié concerne les conflits internes, pouvant résulter des méthodes de travail divergentes, d'opinions variées et d'un manque de cohésion au sein de l'équipe. Tout conflit ou climat de tension peut affecter le rendement global de l'équipe, c'est pourquoi nous avons mis en place un canal de conversation « hors-sujet » sur le serveur discord de l'équipe pour désamorcer l'accumulation de stress. De plus, nous organisons des soirées de cohésion afin de favoriser les échanges et développer des relations amicales entre les membres. Dans l'éventualité où une situation de conflit surviendrait malgré nos efforts, nous tenterons de résoudre le conflit en interne, en utilisant les compétences acquises dans le cadre des cours HPR à Polytechnique Montréal. Si la situation persiste, une consultation avec les responsables du projet sera envisagée. Ce dernier risque est donc aussi d'une priorité faible pour notre équipe.

### **5.3 Tests (Q4.4)**

Conformément aux exigences du R.Q.2, des tests unitaires ou des procédures de test sont demandés pour chaque fonctionnalité. Quelques tests spécifiques seront définis pour chaque sous-système.

Comme décrit précédemment, les fonctionnalités web seront testées à travers des tests unitaires. D'abord, la station au sol sera testée en envoyant des messages prédéfinis, par exemple, « lancer la mission ». On pourra donc observer les réponses du système via les logs produits. De plus, l'interface utilisateur sera testée en se connectant à travers plusieurs navigateurs pour tester sa compatibilité et sa robustesse lorsqu'elle est soumise à plusieurs tâches.

En ce qui concerne le sous-système embarqué, nous opterons pour le développement de procédures de tests. Ceci testera une fonctionnalité de bout en

bout, ce qui permet de valider le comportement attendu du système. Ces procédures seront détaillées dans un fichier nommé « Tests.pdf ». Une telle procédure serait de lancer une mission complète, du lancement à la réalisation, en passant par le retour à la base. Les artefacts produits par cette mission seront comparés par ceux d'une simulation sur Gazebo ayant les mêmes paramètres. On vérifiera ensuite que tous les composants interagissent comme attendu.

#### **5.4 Gestion de configuration (Q4)**

La gestion de la configuration du projet repose sur l'utilisation de Git pour le contrôle de version, permettant une organisation claire du code source et une gestion efficace des versions et des branches. Le dépôt est structuré en plusieurs répertoires, chacun jouant un rôle bien défini :

Organisation du code source :

- **Application** : Ce répertoire regroupe l'ensemble de l'application web :
  - **Client** : Le code source Angular est organisé dans le répertoire *application/client/src*, avec des modules séparés dans des sous-répertoires tels que *app*, *assets*, et *environments*, favorisant la modularité et la réutilisation du code. Les fichiers de configuration sont situés à la racine du répertoire *client* et le reste se trouve dans *src*.
  - **Serveur** : Le code source NestJS se trouve dans *application/server*, avec des sous-dossiers pour organiser les différents aspects de l'application. Le répertoire *app* contient le code principal, y compris les modules, contrôleurs et services. Le répertoire *assets* regroupe les ressources statiques, alors que *e2e* contient les fichiers de test « end-to-end ». Les fichiers de configuration se trouvent à la racine du répertoire *server*.
  - **Commun** : Les interfaces et énumérations communes aux deux environnements (client et serveur) sont stockées dans *application/common*, incluant des énumérations dans *enums* et des interfaces *interfaces* pour assurer une harmonisation des types et structures de données.
- **Docker** : Ce répertoire contient les fichiers nécessaires pour la configuration des conteneurs Docker, dont les *Dockerfile*s pour le client et le serveur, ainsi qu'un fichier *compose.yaml* pour la configuration des services Docker.
- **Embarqué** : Regroupe les fichiers relatifs aux systèmes embarqués, incluant *examples\_msgs* pour les messages d'exemple et *src* pour le code source.
- **Gazebo** : Contient des fichiers relatifs à la simulation Gazebo, avec un répertoire *project\_ws* pour le code source associé.
- **Rapports hebdomadaires** : Contient les rapports hebdomadaires du projet.
- **Gabarits** : Contient les modèles de communication.

Tests et gestion des fichiers de données :

- Les tests sont isolés dans des fichiers `.spec.ts` dans chaque répertoire `src`, garantissant que chaque module dispose de tests dédiés.
- Les fichiers de description des messages échangés entre robots sont contenus dans `embedded_ws/examples_msgs`, utilisés pour simuler et valider les communications.

Conteneurisation:

- Le projet utilise Docker pour simplifier la configuration des environnements et assurer la portabilité. Les fichiers Docker (`Dockerfile_client`, `Dockerfile_server` et `Dockerfile_gazebo`) définissent des conteneurs isolés pour chaque composant, permettant de lancer des instances autonomes du front-end, du back-end, et des simulateurs.
- Le fichier `compose.yaml` gère l'intégration de ces conteneurs, assurant leur coordination.

Documentation :

- La documentation relative au code source, aux tests, et aux processus de configuration est maintenue à jour dans le fichier `README.md` ainsi que dans nos rapports.

## 5.5 Déroulement du projet (Q2.5)

Notre équipe a atteint les objectifs établis par l'ASC pour les étapes du PDR et du CDR ([Section 4.3](#)) dans les échéances établies. L'organisation efficace des responsabilités, la collaboration et la séparation des tâches ([Section 4.2](#)) ont permis de paralléliser les efforts, favorisant le développement d'expertise pour chaque membre de l'équipe.

Notre réussite, tant en matière de respect des requis que de fonctionnement du système d'exploration multi-robots, repose principalement sur la rigueur de nos méthodes de collaboration et de communication. Des revues de code régulières ont permis aux membres de l'équipe de s'entraider en identifiant des erreurs potentielles, en harmonisant les pratiques et en assurant une qualité de code uniforme. Ces revues ont aussi été l'occasion pour tous d'enrichir leurs connaissances sur les techniques et technologies utilisées dans le projet, tout en offrant une visibilité continue sur l'avancement des différentes parties du projet. En parallèle, l'approche d'intégration continue adoptée a été essentielle pour la stabilité et la fiabilité du code, exigeant que chaque modification passe par des tests automatisés. En intégrant fréquemment les améliorations, nous avons pu détecter rapidement les erreurs et réagir de façon proactive, favorisant ainsi une mise en place rapide de solutions. De plus, les réunions de synchronisation hebdomadaires ont été tenues pour partager les progrès, aligner les tâches, ajuster les dates butoirs et anticiper les défis. Ces rencontres ont permis de garder une vue d'ensemble sur l'avancement du projet et de maintenir une bonne dynamique d'équipe, ce qui a renforcé la capacité des membres à solliciter l'aide d'autres membres en cas de risque de non-respect d'une échéance. La réponse rapide et proactive des autres membres à ces demandes a favorisé la cohésion de l'équipe et a permis de maintenir un progrès constant. Cette collaboration fluide entre les membres de l'équipe a été critique pour le partage de connaissances, l'entraide et l'efficacité du projet lors de la résolution de problèmes complexes rencontrés, et ce, malgré les expertises acquises. Enfin, le maintien d'une documentation commune consignant les principales étapes de développement et les bonnes pratiques a assuré une cohérence dans le code, facilité le suivi des tâches et garanti que chacun puisse se référer aux mêmes standards tout au long du projet.

Cependant, plusieurs aspects ont été plus complexes et difficiles que prévu. D'abord, la prise en main et la maîtrise de ROS2, Gazebo et d'autres technologies complexes ont requis plus de temps qu'estimé, rallongeant ainsi la phase de démarrage du projet. Un autre délai inattendu a été noté lors du mauvais fonctionnement du Lidar d'un de nos deux robots, nécessitant son échange pour un autre robot fonctionnel. Bien que cette période d'apprentissage ait renforcé nos compétences techniques, elle a temporairement ralenti l'avancement initial du projet. Ceci a poussé à une certaine ultraspecialisation des membres créant un petit débordement dans la connaissance du fonctionnement du code. De plus, le processus de révision des « Merge Request » (MR) s'est avéré plus chronophage que prévu, impactant notre capacité à implémenter certaines

fonctionnalités rapidement. Ce problème n'est pas dû au temps de révision lui-même, mais au délai avant qu'un membre soit disponible à faire la revue. Pour y remédier, nous avons réorganisé nos priorités et instauré des créneaux hebdomadaires spécifiques pour les révisions, afin de mieux fluidifier le développement. Enfin, une période particulièrement chargée en mi-session a causé un retard par rapport au calendrier ([Annexe C](#)). Néanmoins, la confiance mutuelle et la synergie développées au sein de l'équipe nous ont permis de surmonter ces défis en adaptant notre rythme de travail et en redistribuant les efforts selon les priorités du projet et les compétences de chacun.

## 6. Résultats des tests de fonctionnement du système complet (Q2.4)

### 6.1 Requis fonctionnels

- R.F.1 : Lorsqu'un utilisateur appuie sur le bouton d'identification d'un robot via l'interface web, le robot sélectionné reçoit la commande « Identifier » et émet une mélodie extraite de la chanson Galactic de Maes, célèbre artiste du 21<sup>e</sup> siècle. Aucun autre robot n'émet de son, confirmant l'identification unique.
- R.F.2 : Depuis l'interface web, la commande « Lancer la mission » active tous les robots connectés. Ces robots commencent leur déplacement en suivant un algorithme d'exploration aléatoire, évitent les obstacles détectés et progressent vers des zones libres. La commande « Terminer la mission » arrête immédiatement l'exploration.
- R.F.3 : Une fois un robot connecté, l'interface utilisateur affiche son état parmi les options suivantes : « EN ATTENTE », « EN MISSION », ou « BATTERIE FAIBLE ». Le niveau de batterie est également affiché, mis à jour à une fréquence inférieure à 1 Hz, et toutes les informations sont visibles sur une seule page.
- R.F.4 : Dès réception de la commande « Lancer la mission », les robots adoptent un algorithme d'exploration aléatoire à l'aide du « Framework » Nav2. Leurs capteurs LIDAR détectent les espaces accessibles à proximité, vers lesquels ils se dirigent pour maximiser l'exploration.
- R.F.5 : L'algorithme d'exploration permet aux robots d'éviter des obstacles, qu'ils soient statiques ou mobiles, détectés par le lidar. Les robots n'incluent pas les zones obstruées dans leur trajectoire, contournant ainsi les obstacles rencontrés.
- R.F.8 : Les données du lidar génèrent une carte de l'environnement du robot sous forme matricielle à l'aide du « package » Slam Toolbox. Mise à jour en temps réel, cette carte est envoyée au serveur pour affichage continu sur l'interface utilisateur. La carte représente les zones libres (blanc), les obstacles (noir) et les zones non explorées (gris). Actuellement, cette fonctionnalité est limitée à un seul robot en mission.

- R.F.10 : L'interface utilisateur est disponible en tant que service web, accessible sur tout appareil connecté au même réseau que la station au sol, via un navigateur web. Bien que théoriquement accessible par un nombre illimité d'appareils, la fonctionnalité a été testée avec un nombre limité de connexions. Tous les utilisateurs peuvent visualiser les données de mission, mais le contrôle est restreint au premier appareil connecté. Les tests unitaires confirment le bon fonctionnement de l'interface.
- R.F.12 : Avant de commencer une mission, l'interface utilisateur permet de définir manuellement ou automatiquement la position et l'orientation initiales de chaque robot. Cette configuration garantit une synchronisation optimale entre les robots et leurs trajectoires prévues. Les coordonnées définies apparaissent sur la carte de l'interface utilisateur avant le lancement de la mission.
- R.F.13 : Les robots sont équipés de capteurs qui leur permettent de détecter une élévation négative (ex. : chute ou descente de marche). Si une telle situation est détectée, le robot interrompt immédiatement sa progression, prévenant ainsi tout risque de chute. Par exemple, en cas de placement sur une table, le robot reconnaît les bords et ajuste ses mouvements en conséquence.
- R.F.14 et R.F.16 : L'interface inclut un éditeur de code pour personnaliser les configurations du robot. Les utilisateurs peuvent, par exemple, modifier la musique associée à l'identification du robot physique ou ajuster les variables de simulation, comme la consommation de batterie. Une fois les changements effectués, une commande « Mettre à jour » applique les modifications et redémarre automatiquement la simulation, garantissant un environnement à jour.
- R.F.17 et R.F.18 : Chaque mission exécutée est automatiquement enregistrée dans une base de données centralisée. Les utilisateurs peuvent consulter l'historique des missions via une interface dédiée, qui inclut des options de filtrage par date, durée et état de la mission. Les cartes générées sont également sauvegardées et accessibles pour une analyse post-mission ou une réutilisation.
- R.F.19 : Les robots disposent d'un protocole de communication pair-à-pair (P2P), activé dès le début d'une mission. Cette communication leur permet d'échanger des données comme la distance relative ou la position. L'écran des robots affiche une icône indiquant leur statut :  (en attente),  (le plus proche) ou  (le plus éloigné). Cette fonctionnalité, limitée aux robots physiques, améliore la coordination et réduit les dépendances au serveur central.
- R.F.20 : L'interface utilisateur permet de définir une zone de sécurité rectangulaire en entrant les coordonnées des coins opposés. Cette zone est visuellement représentée en rouge sur la carte et empêche les robots de s'aventurer à l'extérieur. Si un robot dépasse cette limite, il retourne immédiatement dans la zone autorisée et reprend son exploration. Cette

fonctionnalité est actuellement implémentée uniquement en simulation, où elle s'affiche également sur l'environnement virtuel (Gazebo).

## **6.2 Requis Matériels**

- R.M.1 : L'installation du code et de ses dépendances est complète et fonctionnelle sur les deux robots AgileX Limo fournis par l'ASC.
- R.M.2 : Toutes les communications avec les robots physiques passent par la station au sol, via Rosbridge WebSocket, en utilisant le réseau WiFi de l'Agence.
- R.M.3 : Aucun capteur additionnel n'a été installé, ni sur les robots physiques ni dans l'environnement de simulation.
- R.M.4 : La station au sol peut être n'importe quel ordinateur, fixe ou portable.

## **6.3 Requis logiciel**

- R.L.1 : Les robots sont programmés dans un conteneur Docker sous Ubuntu 22.04.
- R.L.2 : Aucune différence n'existe au niveau de l'interface web entre la connexion avec les robots physiques et la simulation.
- R.L.3 : Tous les contrôles des robots sont effectués depuis la station au sol via Rosbridge WebSocket. Les commandes sont transmises au conteneur Docker qui gère les robots.
- R.L.4 : Toutes les composantes logicielles — simulation, serveur de la station au sol, serveur web de l'interface, et logiciel embarqué — sont conteneurisées avec Docker. Un fichier Docker Compose permet de lancer plusieurs composants simultanément, sous Ubuntu 22.04.

## **6.4 Requis de conception**

- R.C.1 : Les logs importants de niveau minimal « INFO » et les commandes envoyées par la station au sol sont affichés en continu sur l'interface utilisateur, ces derniers sont également enregistrés sur la station au sol pour analyse. L'historique des missions est accessible, identifié par la date et heure de lancement. Actuellement, les données des capteurs ne sont pas encore disponibles sur l'interface utilisateur. L'historique est consultable depuis l'en-tête de l'interface.
- R.C.2 : Le système de la station au sol et l'environnement Gazebo peuvent être lancé avec la commande « docker-compose -f compose.yaml up » depuis le répertoire contenant le fichier compose.yaml. La fonctionnalité a été vérifiée sous WSL2 et Linux Ubuntu 22.04.
- R.C.3 : L'environnement virtuel Gazebo génère un nombre aléatoire de murs obstacles entre trois et dix, en plus des murs de délimitation de la zone d'exploration.

- R.C.4 : L'interface utilisateur respecte les dix heuristiques d'utilisabilité, est lisible et facile à utiliser. Elle offre aussi deux thèmes de couleurs pour répondre aux préférences des utilisateurs.

## 6.5 Requis de qualité

- R.Q.1 : Le code est uniformisé dans chaque composant (logiciel embarqué, simulation, station au sol, interface utilisateur), selon des conventions reconnues et adaptées aux langages et objectifs des composants.
- R.Q.2 : Les composantes de la station au sol et de l'interface utilisateur sont couvertes par des tests unitaires avec une couverture minimale de 95 %. Pour le système embarqué et la simulation, des procédures de test permettent de vérifier le comportement attendu. Ces procédures sont connues à l'interne seulement pour le moment.

Les résultats des tests de fonctionnement démontrent que le système répond aux requis fonctionnels, matériels, logiciels, de conception, et de qualité, validant ainsi la capacité de l'architecture proposée à remplir les exigences de l'ASC. Les tests ont confirmé le bon fonctionnement du système face aux obstacles, la cohérence de l'interface utilisateur et la modularité du code grâce à Docker. Certaines limitations subsistent tout de même, notamment l'accès limité aux données des capteurs dans les logs de l'interface utilisateur et l'indisponibilité de la cartographie lors de la simulation multirobot. Ces résultats démontrent que le système est bien conçu pour la navigation autonome, mais qu'il y a place à de l'amélioration dans des conditions d'opérations réelles.

## 7. Travaux futurs et recommandations (Q3.5)

Le prototype de système d'exploration multirobots développé par notre équipe satisfait l'ensemble des requis obligatoires ainsi qu'une majorité des requis optionnels. Cependant, les requis R.F.11, exigeant la génération d'une carte en 3D et en couleur, et R.F.15, permettant deux modes distincts de contrôle des roues, n'ont pas été intégrés dans le cadre du prototype. Ce choix visait à prioriser la qualité globale du livrable dans les délais impartis, tout en capitalisant sur les points forts de notre équipe. Ces requis constituent néanmoins des pistes intéressantes pour de futurs développements, car ils pourraient améliorer la précision des capacités de cartographie et permettre une meilleure adaptation des modes de déplacement des robots à des terrains variés. Compte tenu de la nature prototypale du projet, des efforts supplémentaires devraient être consacrés à l'amélioration de la maintenabilité et de la robustesse du code, afin d'en garantir la pérennité et la résilience face à des événements imprévus. En outre, l'interface utilisateur actuelle nécessiterait des ajustements pour gérer simultanément plus de deux robots, une capacité déjà implémentée dans notre solution. De plus, la simulation devrait inclure la génération dynamique des fichiers d'exploration et de navigation, renforçant ainsi l'évolutivité du système et permettant un support pour un nombre illimité de robots en environnement simulé.

Par ailleurs, notre équipe a déterminé des recommandations et des pistes d'amélioration. Nous suggérons l'intégration d'une fonctionnalité exploitant les caméras embarquées des robots pour visualiser leur champ de vision, ce qui serait particulièrement utile pour le diagnostic, la surveillance à distance et l'exploration, à l'image des robots martiens tels que *Curiosity*. En complément, le système pourrait intégrer de nouvelles commandes dédiées à chaque robot, telles que le retour individuel à la base ou l'exploration partitionnée, optimisant ainsi la gestion des missions. Enfin, une migration vers une infrastructure reposant sur une base de données et un serveur distant, plutôt que localisés au centre de contrôle, améliorera la scalabilité et la flexibilité du système, répondant ainsi à des exigences opérationnelles plus complexes.

## 8. Apprentissage continu (Q12)

### 8.1 *Ely Cheikh Abass*

#### 8.1.1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

Une des principales lacunes rencontrées durant le projet a été liée à la documentation fournie. Elle s'est avérée très générale, sans indications précises ou exemples clairs pour l'application des concepts nécessaires. Cette situation a engendré une difficulté importante dans l'interprétation des informations et dans la mise en œuvre des solutions attendues. En pensant initialement posséder les connaissances requises pour affronter le projet, j'ai sous-estimé l'importance de maîtriser l'art de lire, analyser et exploiter efficacement des documentations techniques. Cette lacune a eu pour conséquence que certaines tâches, qui semblaient simples au premier abord, ont pris beaucoup plus de temps que prévu pour être réalisées, affectant ainsi la gestion globale du projet.

#### 8.1.2 Méthodes prises pour y remédier.

Pour remédier à cette situation, j'ai adopté plusieurs approches. Tout d'abord, j'ai intensifié mes recherches en consultant non seulement la documentation fournie, mais également des ressources complémentaires comme des forums spécialisés, des tutoriels, des vidéos explicatives et des communautés en ligne dédiées à ROS et aux frameworks utilisés. Ensuite, j'ai structuré mon processus de lecture en établissant une méthodologie systématique : lire en profondeur, prendre des notes, résumer les points-clés, et appliquer immédiatement les concepts appris sur des cas pratiques.

Pour surmonter ces difficultés, j'ai travaillé en étroite collaboration avec mes collègues. Nous avons partagé nos découvertes, nos interprétations des documents, et nos méthodes de résolution. Ce travail d'équipe m'a permis d'enrichir ma compréhension et d'avancer plus efficacement sur les tâches qui demandaient une interprétation approfondie de la documentation.

#### 8.1.3 Identifier comment cet aspect aurait pu être amélioré.

Cet aspect aurait pu être amélioré à plusieurs niveaux. Tout d'abord, il aurait été bénéfique que la documentation initiale soit mieux structurée, plus détaillée et qu'elle contienne des exemples concrets ou des guides pas-à-pas directement liés aux tâches spécifiques du projet. Cela aurait permis d'accélérer la compréhension et l'application des concepts nécessaires.

Ensuite, une meilleure planification en amont, incluant du temps dédié à l'exploration approfondie de la documentation et des outils, aurait permis de limiter les imprévus liés au temps d'apprentissage. Enfin, une approche collaborative plus formalisée, avec des sessions régulières de partage des connaissances et d'évaluation des progrès réalisés, aurait encore renforcé la capacité collective à surmonter les défis liés à la documentation. Cela aurait permis de s'assurer que toutes les lacunes étaient identifiées et traitées de manière proactive et concertée.

## **8.2 Omar Benzekri**

### **8.2.1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.**

Pendant le développement du projet j'ai rencontré certaines lacunes dans mes connaissances, notamment dans la gestion des communications en temps réel entre le front-end (Angular) et le back-end (NestJS). Bien que j'aie une solide expérience en Angular, l'implémentation d'une synchronisation efficace avec les robots via des WebSockets était un défi initial. De plus, l'intégration de cartes interactives pour suivre les déplacements des robots en temps réel a révélé une connaissance limitée des bibliothèques de cartographie avancées. Enfin, sur le plan organisationnel, j'ai réalisé que je pouvais améliorer ma capacité à planifier des cycles de développement plus courts et à collaborer plus efficacement avec l'équipe de back-end pour résoudre rapidement les dépendances.

### **8.2.2. Méthodes prises pour y remédier.**

Pour pallier ces lacunes, j'ai entrepris plusieurs initiatives :

1. **Renforcement des connaissances en communication temps réel** : J'ai suivi des tutoriels et exploré la documentation officielle de Socket.IO pour mieux comprendre et implémenter des WebSockets entre Angular et NestJS. J'ai également collaboré étroitement avec notre équipe back-end pour aligner nos protocoles de communication.
2. **Approfondissement des outils de cartographie** : J'ai consulté des ressources en ligne et expérimenté des implémentations pratiques avec Leaflet. Cela m'a permis de créer des cartes dynamiques et conviviales pour afficher les missions des robots en temps réel.
3. **Amélioration de la planification** : J'ai adopté la méthodologie agile en proposant des sprints hebdomadaires spécifiques au front-end. Cette approche m'a aidé à mieux prioriser les tâches et à réduire les dépendances bloquantes.

### 8.2.3. Identifier comment cet aspect aurait pu être amélioré.

Une meilleure préparation initiale aurait permis d'atténuer plusieurs de ces défis. Par exemple, un *workshop* technique au début du projet sur l'utilisation des WebSockets et des outils de cartographie aurait été très bénéfique. De même, un plan de communication plus détaillé avec l'équipe back-end aurait permis d'éviter certains retards dus aux dépendances non prévues. Enfin, la mise en place d'outils collaboratifs comme Jira ou Asana dès le départ aurait optimisé la gestion des tâches et renforcé la transparence au sein de l'équipe.

En résumé, ce projet m'a permis de renforcer mes compétences techniques et organisationnelles tout en découvrant de nouvelles méthodes pour résoudre efficacement des défis complexes.

## 8.3 **Abdul-Wahab Chaarani**

### 8.3.1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

Une première lacune que j'ai identifiée, trop tard durant le projet à mon avis, est le fait que je travaillais en silo sur certains requis du projet. Par exemple, j'ai pris en charge la simulation des requis sur Gazebo. Néanmoins, plusieurs requis étaient entrelacés. En effet, la génération de la carte de l'environnement allait de pair avec l'exploration autonome et l'évitement d'obstacles. Ce travail isolé a parfois causé des décalages ou des incompatibilités lorsqu'est venu le moment d'intégrer les différentes parties du projet.

Une deuxième lacune importante que j'ai identifiée est le fait de toujours travailler en local sur ma machine, étant moins à l'aise avec les conteneurs. Nous avons rapidement rencontré des problèmes de dépendances et de chronologie lors de l'intégration et de l'automatisation avec Docker, ce qui a retardé certaines étapes clés du projet.

### 8.3.2. Méthodes prises pour y remédier.

Pour diminuer le travail en silo, j'ai participé activement à des séances de travail communes. J'ai partagé mon horaire de disponibilités et, lorsque je travaillais sur le projet, je me plaçais dans un canal vocal de Discord et invitais mes collègues à se joindre à moi. J'ai remarqué une amélioration importante sur l'efficacité du développement de certains requis. Par exemple, pour le développement de la zone de sécurité (geofence), en travaillant directement avec mes collègues du backend et du frontend, nous avons réussi à livrer ce requis complexe en peu de temps et à l'intégrer au reste du code de manière fluide. Ensuite, pour remédier à

ma difficulté avec les conteneurs, j'ai sollicité l'aide de collègues expérimentés pour valider mes configurations et pour m'aider à intégrer mon travail.

#### 8.3.3. Identifier comment cet aspect aurait pu être amélioré.

Concernant le travail en silo, organiser des séances de travail communes, et non seulement faire des rencontres de mise à niveau aurait aider à trouver les problèmes plus tôt. Travailler également en présentiel, ou même à la volière aurait favoriser les échanges sur nos obstacles et comment les contourner. Pour ma difficulté avec Docker, je pense qu'un meilleur encadrement initial ou une session de formation en équipe aurait permis de me familiariser plus rapidement avec cet outil. Également, la mise en place de lignes directrice pour l'intégration des conteneurs, en plus de celles du code, dès le début du projet aurait pu prévenir certains de mes problèmes rencontrés.

### **8.4 Loïc Nguemegne**

#### 8.4.1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

L'une des lacunes principales identifiées au cours du projet concernait ma capacité à rechercher et exploiter efficacement l'information dans la documentation technique. Le projet reposait en grande partie sur l'utilisation de packages ROS 2 existants. Cependant, j'ai constaté que la majorité des ressources documentaires disponibles portaient sur ROS 1, ce qui a mis en lumière mes difficultés à identifier et extraire des informations pertinentes dans un contexte de transition technologique.

#### 8.4.2. Méthodes prises pour y remédier.

Pour surmonter cette difficulté, j'ai adopté une méthodologie rigoureuse afin de structurer mes recherches d'information. Dans un premier temps, j'ai établi une approche par étapes :

- Consulter les sources officielles des packages ROS 2, telles que les documentations des développeurs et les dépôts officiels.
- Explorer les forums communautaires reconnus comme StackOverflow.
- Enfin, élargir mes recherches à des pages collaboratives telles que GitHub ou d'autres dépôts publics, tout en validant la fiabilité des informations trouvées.

Cette méthodologie m'a permis d'avancer par élimination et de maximiser la pertinence des informations collectées.

#### 8.4.3. Identifier comment cet aspect aurait pu être amélioré.

Pour améliorer cet aspect, il aurait été judicieux de signaler dès le début du projet l'existence d'incompatibilités entre certains packages ROS 1 et ROS 2, afin de mieux orienter les recherches documentaires. Une formation introductory ou un guide spécifique sur les différences majeures entre ROS 1 et ROS 2 aurait

également été bénéfique pour accélérer la prise en main des outils nécessaires et éviter les pertes de temps liées à des ressources obsolètes.

## **8.5 Thomas Rouleau**

### **8.5.1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.**

La principale lacune identifiée durant ce projet a été mon manque d'implication technique au début de celui-ci. En me concentrant exclusivement sur la gestion organisationnelle, j'ai négligé des opportunités d'apprentissage sur des aspects techniques, notamment ROS 2 et ses fonctionnalités. Cela a limité ma compréhension des outils utilisés et freiné ma progression dans les tâches techniques. Une seconde lacune a été ma réticence initiale à demander de l'aide rapidement lorsque je rencontrais des obstacles. Cette attitude a entraîné des pertes de temps significatives et prolongé certains blocages, impactant la fluidité du développement et la productivité globale de l'équipe.

### **8.5.2. Méthodes prises pour y remédier.**

Pour pallier mon manque d'implication technique, après la première remise, je me suis fixé l'objectif de prendre en charge un ou deux requis spécifiques et d'en assurer l'implémentation complète. Cette démarche m'a permis de renforcer mes compétences techniques et de contribuer de manière plus directe à l'implémentation du projet. Concernant ma réticence à solliciter du soutien rapidement, l'équipe a discuté de l'impact négatif de ce comportement sur le projet. Nous avons mis en place des mesures pour améliorer la communication interne. J'ai pris l'initiative de créer un canal de communication spécifique, intitulé « bugs et bloquants », sur notre serveur Discord. Ce canal offre un espace dédié pour signaler rapidement les problèmes rencontrés et faciliter leur résolution collective.

### **8.5.3. Identifier comment cet aspect aurait pu être amélioré.**

Pour éviter cette situation, il aurait été judicieux d'établir dès le départ un équilibre clair entre les responsabilités techniques et organisationnelles. Cela aurait permis de mieux répartir les charges de travail et de garantir que chaque membre de l'équipe progresse dans sa compréhension des technologies utilisées, tout en respectant les objectifs de gestion du projet. Concernant les blocages causés par un manque de communication rapide des problèmes critiques, il aurait été bénéfique de mettre en place dès le début un protocole clair pour le signalement et la résolution des obstacles. Par exemple, l'instauration précoce d'un canal de communication dédié, comme celui mis en place ultérieurement, aurait permis d'encourager des échanges proactifs dès les premières phases du projet, réduisant ainsi les délais liés aux blocages techniques.

## **8.6 Ivan Samoylenko**

### **8.6.1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.**

Une première lacune identifiée concernait mon manque de familiarité avec l'utilisation de Docker. Bien que j'aie une solide expérience en développement backend, je n'avais jamais utilisé cet outil auparavant. Cela m'a empêché de

conteneuriser l'API et la base de données sans l'aide d'un collègue. Cette situation a mis en évidence un besoin de compétences dans les outils de conteneurisation et de déploiement. Une autre lacune concernait mon manque d'expérience en systèmes embarqués, particulièrement dans l'utilisation de ROS. Après avoir travaillé sur la partie web, j'ai contribué aux tâches liées aux robots physiques, mais plusieurs concepts fondamentaux de ROS m'étaient inconnus, rendant mes débuts laborieux et ralentissant mes contributions au projet.

#### 8.6.2. Méthodes prises pour y remédier.

Concernant Docker, j'ai tiré profit du cours INF8480 - Systèmes répartis et infonuagique suivi en parallèle, qui m'a permis de comprendre progressivement la base théorique et les commandes essentielles à l'utilisation de cet outil. J'ai également étudié la configuration réalisée par mon collègue pour renforcer mes connaissances pratiques. Pour combler mon retard en ROS, j'ai lu de la documentation officielle et suivi des tutoriels pratiques, en réalisant des expérimentations même si certaines n'étaient pas directement utiles au projet. Par ailleurs, j'ai sollicité régulièrement mes collègues pour clarifier des concepts spécifiques.

#### 8.6.3. Identifier comment cet aspect aurait pu être amélioré.

Concernant Docker, il aurait été avantageux d'avoir une formation préalable sur les bases des conteneurs avant le début du projet ou de planifier les tâches de conteneurisation pour un moment où je disposais d'une meilleure compréhension grâce au cours INF8480. Par rapport à ROS, il aurait été bénéfique de débuter mon apprentissage en parallèle avec mes collègues dès les premières phases du projet, ce qui aurait permis une meilleure coordination et une prise en charge plus progressive des requis des robots physiques. Une initiation spécifique à ROS avant ou au début du projet aurait également permis d'éviter des pertes de temps et des expérimentations inutiles.

## 9. Conclusion (Q3.6)

Lors de la réalisation de ce projet, notre équipe a rencontré plusieurs défis inhérents à la complexité et à la nouveauté des technologies utilisées. Bien que nous ayons respecté l'ensemble des requis obligatoires et optionnels choisis, de nombreuses leçons ont été tirées tout au long du processus.

Au début du projet, notre vision globale du système était encore floue, et l'absence de certaines ressources spécifiques, comme une documentation claire ou une formation structurée sur certains outils, a rendu la planification initiale difficile. En conséquence, certaines tâches ont nécessité beaucoup plus de temps que prévu, notamment en raison de la nécessité d'acquérir des compétences pratiques sur ROS2, Gazebo, Docker et d'autres outils critiques. Ce manque de préparation initiale a également conduit à la sous-estimation de la complexité de certaines fonctionnalités, comme l'exploration autonome et l'intégration des systèmes de communication.

Au fur et à mesure de l'avancement, plusieurs solutions initialement proposées ont été ajustées ou complètement revues. Par exemple, l'algorithme d'exploration autonome, qui devait initialement être basé sur des trajectoires préprogrammées, a été remplacé par une solution utilisant **Nav2** et un comportement exploratoire aléatoire, en raison de sa meilleure adéquation avec les requis. Ce type d'ajustement a permis d'améliorer la robustesse et la performance globale du système, mais a également mis en lumière des lacunes dans notre démarche de conception initiale. Un autre point clé a été la communication en temps réel via **Rosbridge WebSocket**, qui s'est avérée plus complexe que prévu. Alors que nous pensions qu'il s'agirait d'une tâche relativement simple, de nombreux défis ont émergé, notamment en ce qui concerne la synchronisation des données entre les robots et la station au sol. Cette fonctionnalité a nécessité plusieurs heures supplémentaires de développement et d'intégration pour garantir sa fiabilité.

En termes de méthodologie, nous avons constaté que certaines erreurs dans notre démarche auraient pu être évitées avec une approche de conception et de test plus structurée. Par exemple, l'intégration tardive de certaines fonctionnalités a parfois révélé des incompatibilités ou des problèmes non anticipés, retardant ainsi leur mise en œuvre. Une autre faiblesse identifiée a été notre tendance à tester les fonctionnalités uniquement une fois complètes, ce qui a parfois retardé la découverte de problèmes critiques. À l'avenir, rédiger un plan de test clair dès le début et tester régulièrement les incrémentations partiels auraient permis de gagner du temps et d'assurer une meilleure stabilité du système.

Malgré ces défis, ce projet a été une expérience d'apprentissage enrichissante. Chaque membre de l'équipe a renforcé ses compétences dans des domaines clés tels que le développement de systèmes embarqués, l'intégration logicielle et la collaboration en équipe. De plus, la flexibilité et la capacité d'adaptation

démontrées par l'équipe ont permis de surmonter les obstacles et de livrer un système fonctionnel répondant aux exigences de l'ASC.

En conclusion, ce projet nous a permis de comprendre l'importance d'une planification initiale détaillée et d'une collaboration continue. Nous recommandons, pour de futurs projets similaires, une phase préliminaire consacrée à la formation sur les outils utilisés, l'élaboration d'un plan de tests systématique et une évaluation régulière des solutions mises en œuvre. Ces ajustements contribuerait à améliorer la qualité et l'efficacité des développements tout en réduisant les risques liés aux ajustements tardifs.

## 10. Références (Q3.2)

- [1] S. Toolbox. "README.md." [https://github.com/SteveMacenski/slam\\_toolbox/](https://github.com/SteveMacenski/slam_toolbox/) (accessed 31 octobre 2024).
- [2] Nav2. "README.md." <https://github.com/ros-navigation/navigation2> (accessed 31 octobre 2024).
- [3] Gazebo. "ROS 2 Integration." [https://gazebosim.org/docs/fortress/ros2\\_integration/](https://gazebosim.org/docs/fortress/ros2_integration/) (accessed 9 septembre, 2024).
- [4] Agilex. "limo-pro." <https://global.agilex.ai/products/limo-pro> (accessed 18 septembre, 2024).
- [5] Orbbec\_camera. "README.md" [https://github.com/orbbec/OrbbecSDK\\_ROS2/tree/main/orbbec\\_camera/la](https://github.com/orbbec/OrbbecSDK_ROS2/tree/main/orbbec_camera/la) unch accessed 18 septembre, 2024).

## ANNEXES

### ***Annexe A – Résumé des requis***

#### Requis Généraux

Les requis suivants sont obligatoires. En cas de non-respect, la soumission et/ou le prototype pourra être rejetés par l'Agence.

#### Requis matériels :

- **R.M.1** : Utiliser deux robots AgileX Limo fournis par l'Agence.
- **R.M.2** : Communication entre la station au sol et les robots via WiFi fourni par l'Agence.
- **R.M.3** : Utiliser uniquement les capteurs installés par l'Agence (IMU, caméra 3D, caméra RGB, lidar).
- **R.M.4** : La station au sol doit être un laptop ou PC.

#### Requis logiciels :

- **R.L.1** : Programmer les robots avec Ubuntu, machine virtuelle possible (Docker).
- **R.L.2** : Interface utilisateur identique pour simulation et robots physiques.
- **R.L.3** : Contrôle des robots à bord, station au sol envoie des commandes de haut niveau.
- **R.L.4** : Conteneuriser les composantes logicielles avec Docker, sauf celles embarquées.

#### Requis spécifiques :

Les requis spécifiques sont classifiés par niveau de criticité et accompagnés de points. Les soumissionnaires doivent accomplir un minimum de 100 points, avec un maximum de 120 points possible. Certains de ces requis sont obligatoires, tandis que d'autres sont optionnels, laissant le choix aux soumissionnaires. Les requis optionnels que notre équipe a décidé de ne pas mettre en œuvre sont indiqués en rouge.

#### Requis fonctionnels :

- **R.F.1** : Robots doivent répondre à la commande "Identifier" (3 points sur robots physiques).
- **R.F.2** : Commandes "Lancer la mission" et "Terminer la mission" (2 points en simulation, 2 points sur robots physiques).
- **R.F.3** : Interface utilisateur montre l'état des robots (2 points).
- **R.F.4** : Robots explorent l'environnement de façon autonome (2 points en simulation, 2 points sur robots physiques).
- **R.F.5** : Robots évitent les obstacles (4 points en simulation, 4 points sur robots physiques).
- **R.F.6** : Retour à la base à moins de 0,3 m de la position de départ (4 points en simulation, 6 points sur robots physiques).

- **R.F.7** : Retour à la base automatique à moins de 30% de batterie (2 points).
- **R.F.8** : Station au sol collecte les données et produit une carte de l'environnement (5 points en simulation, 5 points sur robots physiques).
- **R.F.9** : Position des robots affichée en continu sur la carte (3 points).
- **R.F.10** : Interface utilisateur disponible comme service Web (4 points).
- **R.F.11\*** : Carte générée en 3D et en couleur (5 points).
- **R.F.12** : Position et orientation initiales des robots spécifiables par l'opérateur (1 point).
- **R.F.13** : Détection d'élévation négative pour éviter les chutes (4 points sur robots physiques).
- **R.F.14** : Mise à jour du logiciel de contrôle via l'interface utilisateur (5 points sur robots physiques).
- **R.F.15\*** : Deux modes de contrôle des roues différents (5 points).
- **R.F.16** : Éditeur de code pour modifier le comportement des robots (5 points).
- **R.F.17** : Base de données sur la station au sol pour enregistrer les missions (5 points).
- **R.F.18** : Carte générée enregistrée et accessible pour inspection (5 points).
- **R.F.19** : Communication P2P entre robots pour afficher le robot le plus éloigné (5 points sur robots physiques).
- **R.F.20** : Zone de sécurité (geofence) spécifiable dans l'interface utilisateur (5 points).

Requis de conception :

- **R.C.1** : Logs de débogage disponibles en continu et sauvegardés (5 points).
- **R.C.2** : Logiciel de la station au sol lancé avec une seule commande (4 points).
- **R.C.3** : Environnement virtuel généré aléatoirement dans Gazebo (1 point en simulation).
- **R.C.4** : Interface utilisateur facile d'utilisation et lisible (5 points).
- **R.C.5** : Système conçu pour fonctionner avec 1 ou 2 robots (5 points en simulation).

Requis de qualité :

- **R.Q.1** : Code standardisé suivant des conventions reconnues (5 points).
- **R.Q.2** : Tests unitaires ou procédures de test pour chaque fonctionnalité (5 points).

**\*Requis optionnels rejeté**

## Annexe B – Architecture logiciel

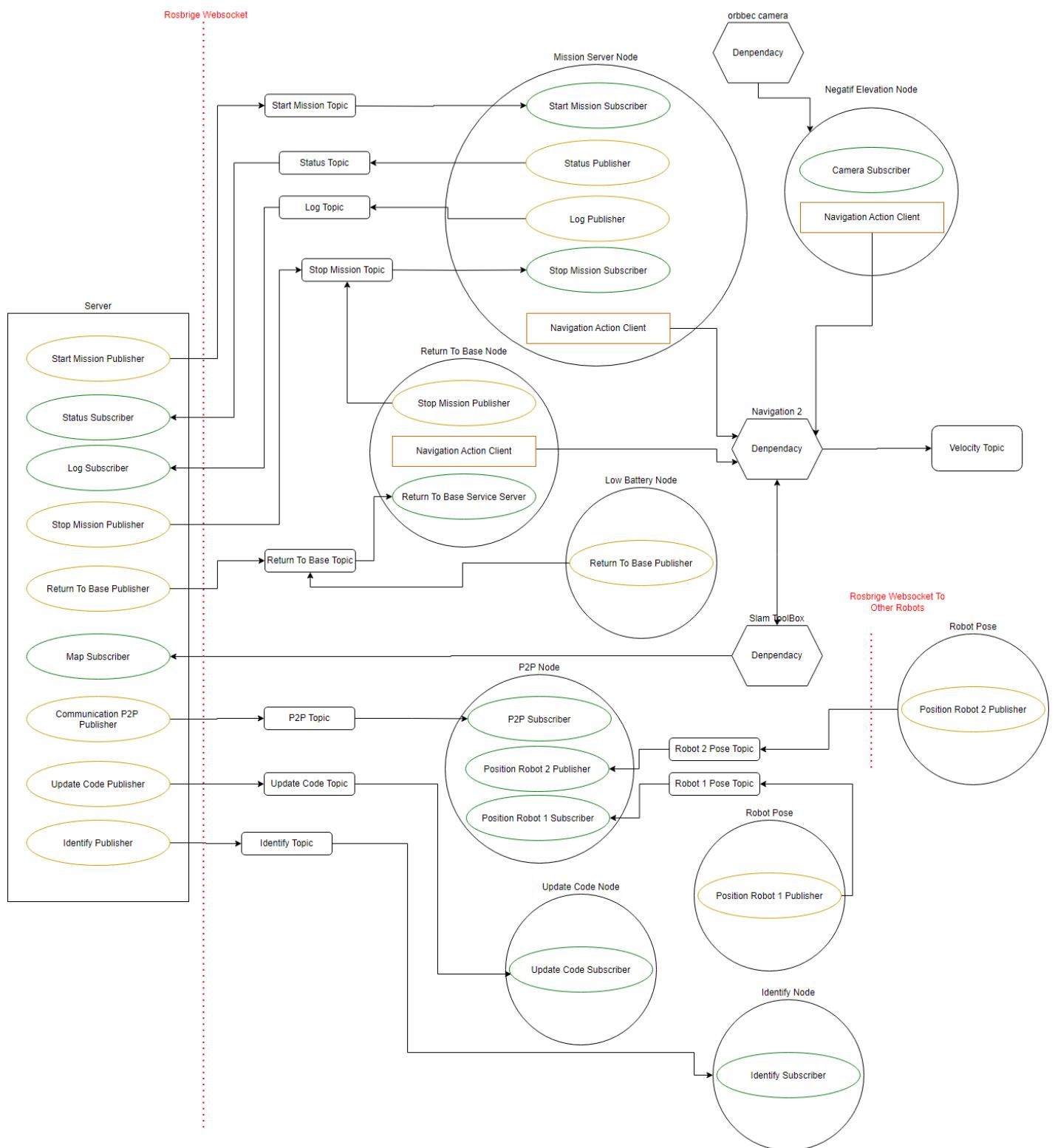


Figure B.1 : Architecture du logiciel embarqué

## Annexe C – Échéanciers de projet

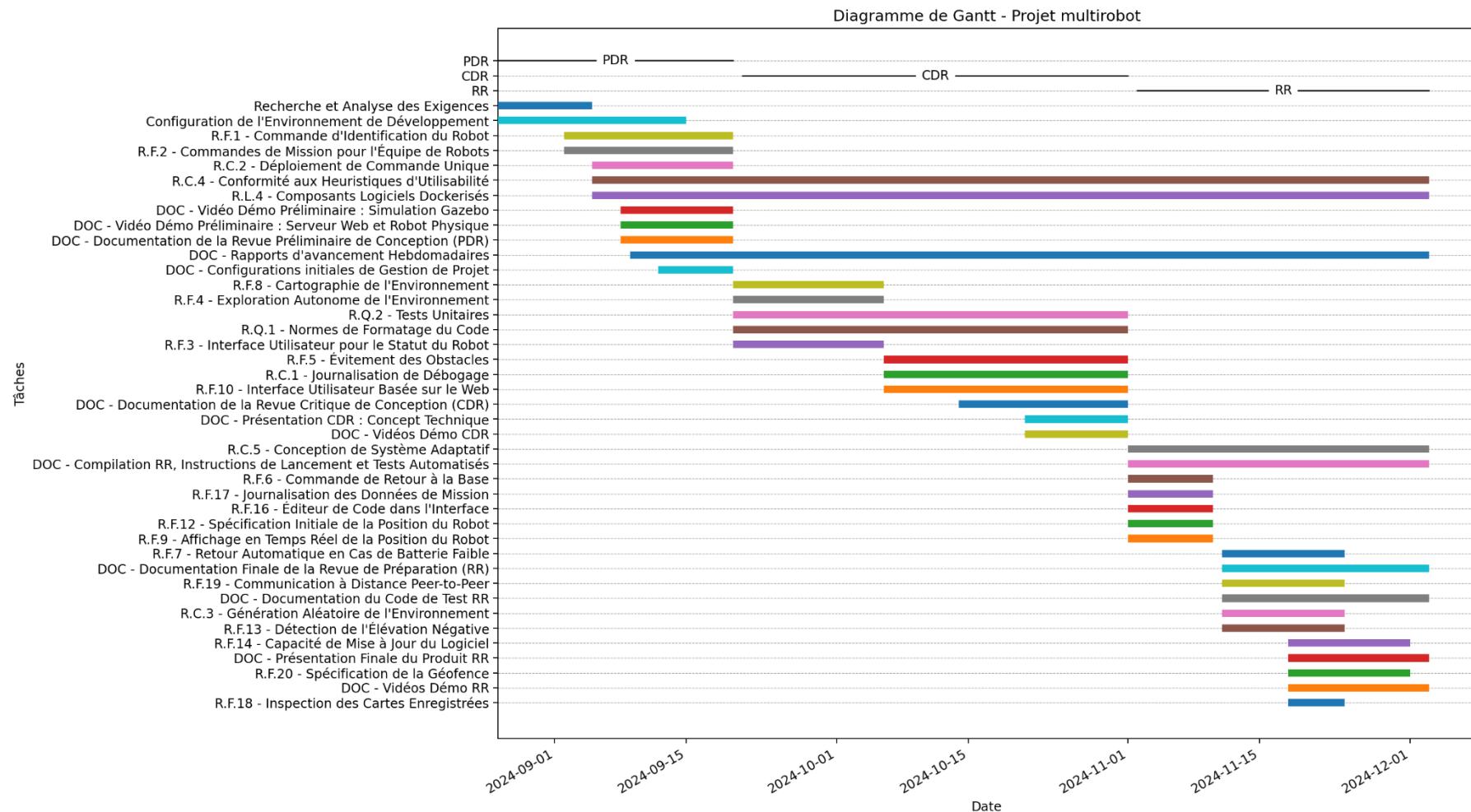


Figure C.1 : Diagramme de Gantt du projet d'exploration par multirobots