



**POLYTECHNIQUE  
MONTRÉAL**

UNIVERSITÉ  
D'INGÉNIERIE

Département de génie informatique et génie logiciel


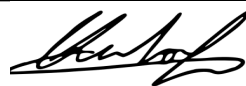
**INF3995**

**Projet de conception d'un système informatique**

Documentation du projet répondant à l'appel d'offres  
no. A2023-INF3995 du département GIGL.

***Conception d'un système aérien d'exploration***

Équipe No **104**

<i>Aymane Chalh</i>	A.C
<i>Anass El Kettani</i>	A E K
<i>Abderrahim Zebiri</i>	A.Z
<i>Thibault Demagny</i>	
<i>Nada Alami Chentoufi</i>	
<i>Lina Khial</i>	L.K

29 Septembre 2023

# Table des matières

<b>1. Vue d'ensemble du projet</b>	<b>3</b>
1.1 But du projet, porté et objectifs (Q4.1)	3
1.2 Hypothèse et contraintes (Q3.1)	4
1.3 Biens livrables du projet (Q4.1)	5
<b>2. Organisation du projet</b>	<b>6</b>
2.1 Structure d'organisation (Q6.1)	6
2.2 Entente contractuelle (Q11.1)	8
<b>3. Description de la solution</b>	<b>10</b>
3.1 Architecture logicielle générale (Q4.5)	10
3.2 Station au sol (Q4.5)	11
3.3 Logiciel embarqué (Q4.5)	13
3.4 Simulation (Q4.5)	16
3.5 Interface utilisateur (Q4.6)	18
3.6 Fonctionnement général (Q5.4)	19
<b>4. Processus de gestion</b>	<b>20</b>
4.1 Estimations des coûts du projet (Q11.1)	20
4.2 Planification des tâches (Q11.2)	20
4.3 Calendrier de projet (Q11.2)	23
4.4 Ressources humaines du projet (Q11.2)	24
<b>5. Suivi de projet et contrôle</b>	<b>24</b>
5.1 Contrôle de la qualité (Q4)	24
5.2 Gestion de risque (Q11.3)	26
5.3 Tests (Q4.4)	27
5.4 Gestion de configuration (Q4)	28
5.5 Déroulement du projet (Q2.5)	29
● 5.5.1 Résultats des tests de fonctionnement du système complet (Q2.4)	31
<b>6. Références (Q3.2)</b>	<b>35</b>

## 1. Vue d'ensemble du projet

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

Ce projet consiste à concevoir un système robotique qui permettra l'exploration et la cartographie d'un terrain ainsi qu'assurer une communication et un contrôle à distance de robots AgileX Limo.

L'utilisateur pourra contrôler le robot via une interface Web. L'accès à l'interface peut se faire à partir de plusieurs types d'appareils (PC, tablette, téléphone).

L'application Web permettra de visualiser des données lors d'une mission et transmettre une série de commandes à des robots physiques ainsi qu'un environnement simulé de Gazebo, en temps réel.

Au fur et à mesure que les robots explorent un terrain, ils transmettent continuellement les informations nécessaires pour la cartographie.

#### **Objectifs:**

- **Système de contrôle et de communication :**

Assurer une communication bidirectionnelle entre l'interface utilisateur Web et les robots.

- **Interface utilisateur Web :**

Concevoir une interface Web conviviale qui prend en charge la visualisation en temps réel des données des robots. Cette plateforme peut être accédée par différents types d'appareils (PC, tablette, téléphone).

- **Contrôle à distance :**

L'interface utilisateur doit permettre à l'utilisateur de contrôler les robots à distance. En conséquence, l'utilisateur peut contrôler des mouvements, gérer la vitesse, lancer ou arrêter une mission.

- **Intégration avec Gazebo :**

Afin de réaliser un système robotique fiable et tester les fonctionnalités avant de les déployer sur le robot, nous avons opté pour le simulateur Gazebo. Le système doit communiquer avec Gazebo pour simuler les mouvements des robots physiques.

- **Navigation autonome et Cartographie :**

Grâce aux capteurs (IMU, caméra 3D, caméra RGB) dont disposent les robots, ils doivent cartographier leur environnement en temps réel et assurer une navigation autonome.

- **Transmission de données en temps réel :**

Mettre en place un système de transmission de données en temps réel entre les robots et l'interface utilisateur pour visualiser les données de capteurs, les cartes générées et l'emplacement actuel des robots sur le terrain.

- **Stockage et gestion des données :**

Stocker et gérer les données collectées par les robots pour la cartographie.

Notre projet sera structuré en trois livrables clés :

- Preliminary Design Review (PDR) : Dans cette documentation initiale du projet, nous répondrons à l'appel d'offres du client, en fournissant une description de haut niveau de l'architecture et de l'infrastructure employée à la fois pour les robots et l'application web.
- Critical Design Review (CDR) : Ce document développe les aspects décrits dans le PDR avec plus de profondeur et de précision. Il présente un prototype plus avancé du projet, en expliquant plus en détail les choix de conception effectués - qu'ils aient été modifiés ou conservés - à la suite du lancement du processus de développement.
- Readiness Review (RR) : Le rapport final comprend le prototype fonctionnel ultime du projet, intégrant tous les retours d'information et les idées recueillies dans les rapports précédents.

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

### Hypothèses:

- Python est assez performant pour être implémenté dans notre projet pour le back-end, capable de tenir des connexions avec le client et avec le robot avec un transfert d'information à une fréquence de 1Hz.
- MongoDB est capable de stocker toutes nos données, et est accessible avec des temps de réponse raisonnables.
- L'utilisation des websockets va nous permettre d'avoir plusieurs missions, plusieurs connexions simultanées et de communiquer en temps réel avec les robots

### Contraintes:

- Fragilité du robot :  
Les composants du robot peuvent se casser facilement.

- Présence physique obligatoire :  
La présence physique au laboratoire est impérative pour avancer sur le robot. La nécessité d'être présent physiquement au travail peut entraîner une baisse de la productivité.
- Nouvelles technologies :  
En raison de la complexité et l'étendue des fonctionnalités à implémenter, nous avons utilisé plusieurs technologies dont certaines jamais vues (ROS). En conséquence, nous devons faire preuve d'adaptation rapide.
- La coordination des horaires :  
La synchronisation des plannings afin de déterminer un créneau horaire commun peut s'avérer une tâche complexe.
- La communication et l'efficacité :  
En raison du manque de communication et d'harmonie entre les membres de l'équipe, l'efficacité est compromise.
- L'assignation des tâches :  
Le manque de clarté dans la répartition et la description des tâches entraîne un manque d'organisation

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

#### 1.3.1 PDR :

- Présentation de la documentation initiale du projet en réponse à l'appel d'offres.
- Démo vidéo du serveur web sur la station au sol et du robot physique (requis R.F.1).
- Démo vidéo de la simulation Gazebo avec deux robots (requis R.F.2).
- Possibilité de démonstration en direct sur demande.
- Démonstrations possibles lors du suivi hebdomadaire.

#### 1.3.2 CDR :

- Présentation du concept presque final du projet avec des changements et précisions depuis la PDR.
- Révision complète de la documentation du projet.
- Présentation technique (max. 10 minutes) suivie de vidéos de démos.
- Démos vidéos de plusieurs requis, y compris un prototype (R.F.8).
- Possibilité d'interruption pour des questions et des commentaires.

#### 1.3.3 RR:

- Présentation du concept final et complet du projet.
- Mise à jour de la documentation du projet pour refléter les progrès.
- Présentation détaillée du produit final (max. 45 minutes).
- Démos vidéos de tous les requis complétés.

- Fourniture du code, des tests automatiques et des instructions de compilation et de lancement.

## 2. Organisation du projet

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

La structure d'organisation de l'équipe de projet pour la réalisation du projet avec des robots munis d'un minimum de capteurs est la suivante :

- **Coordinateur du Projet : Anass El Kettani**

Le coordinateur de projet joue un rôle essentiel dans la gestion globale du projet. Sa mission consiste à superviser toutes les activités, à coordonner les membres de l'équipe, à prendre des décisions stratégiques pour orienter le projet, et à veiller à ce que le projet avance conformément aux objectifs établis. Il est responsable de la prise de décisions cruciales pour garantir que le projet progresse dans la bonne direction tout en optimisant l'utilisation des ressources.

Le coordinateur de projet attribue des tâches spécifiques à chaque membre de l'équipe en fonction de leurs compétences et de leurs domaines d'expertise. Cette allocation précise des responsabilités contribue à exploiter pleinement les talents de l'équipe et à assurer l'efficacité opérationnelle.

Par ailleurs, le coordinateur de projet maintient une communication fluide avec toutes les parties prenantes du projet, y compris les clients, les membres de l'équipe, les fournisseurs et les intervenants externes. Cette communication ouverte et transparente favorise la compréhension mutuelle et permet de résoudre rapidement les problèmes potentiels.

- **Premier développeur-analyste: Nada Alami Chentoufi**

Le premier développeur est un expert technique dans la programmation des robots, intégration des capteurs et des instruments, tests et débogage. Ses principales responsabilités sont les suivantes :

- Concevoir, développer et mettre en œuvre le code logiciel nécessaire pour le fonctionnement des robots.
- Sélectionner et intégrer les capteurs et les instruments appropriés pour permettre au robot de collecter des données pertinentes.
- S'assurer que les robots sont programmés pour exécuter leurs tâches de manière efficace et précise.
- Veiller à ce que le robot soit capable de collecter diverses données de son environnement à l'aide des capteurs intégrés.

- **Deuxième développeur-analyste : Abderrahim Zebiri**

Similaire au premier développeur-analyste, ce membre de l'équipe a des responsabilités techniques similaires, avec un accent sur des aspects spécifiques de la conception des algorithmes de contrôle. Ses principales responsabilités sont les suivantes :

- Participer activement à la programmation des robots en implémentant les algorithmes de contrôle et les comportements spécifiques requis.
- Assurer que les capteurs fonctionnent de manière cohérente et que les communications sont stables et fiables.
- Élaborer des procédures de test pour évaluer la performance et la précision du robot.
- S'occuper de la simulation des robots utilisés en utilisant Gazebo.
- Mettre en place des processus de collecte, de stockage et de transmission des données.

- **Troisième développeur-analyste: Lina Khial**

Ce membre est responsable du développement de la partie backend du système de contrôle à distance, y compris la gestion des données et des communications avec les robots. Ses principales responsabilités sont les suivantes :

- Concevoir, développer et maintenir la logique back-end du système de contrôle à distance des robots.
- Concevoir et maintenir la structure de la base de données pour stocker les données critiques liées au contrôle des robots, ainsi que les données nécessaires à la création des cartes géographiques.
- L'intégration des fonctionnalités de contrôle à distance en coordonnant avec les membres qui s'occupent de la programmation du robot.
- Mise en place de la communication avec les robots.

- **Quatrième développeur-analyste : Thibault Demagny**

Ce développeur est responsable du développement de l'interface utilisateur frontend du système de contrôle à distance des robots. Ses principales responsabilités sont les suivantes :

- La conception et développement de l'interface utilisateur.
- Veiller à ce que la partie liée à l'interface utilisateur reçoive toutes les informations du backend.
- Optimiser les performances de l'interface utilisateur pour garantir une réactivité optimale, même dans des conditions de charge élevée.

- **Cinquième développeur-analyste : Aymane Chalh**

Polyvalent, ce membre de l'équipe peut contribuer aux aspects frontend et backend du projet. Il peut également rejoindre l'équipe de développement du robot en fonction des besoins. Ses principales responsabilités sont les suivantes:

- Collaborer étroitement avec les membres de l'équipe frontend pour contribuer à la conception, au développement et à l'amélioration de l'interface utilisateur.
- Travailler avec les développeurs backend pour assister à la mise en place de la logique backend et de la gestion des données, au besoin.
- Adapter ses compétences en fonction des exigences actuelles du projet.

En résumé, l'équipe du projet est composée d'un total de six membres, chacun ayant des rôles et des responsabilités clairement définis pour assurer le succès de la réalisation de la preuve de concept d'exploration avec des robots munis d'un minimum de capteurs.

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

### **1. OBJET DU CONTRAT**

#### **1.1 Description du Projet**

Ce présent contrat engage notre entreprise, Poly-Robot, dans le cadre du projet d'exploration planétaire en qualité de prestataire, ci-après dénommée "le Prestataire", pour le compte de l'Agence Spatiale, ci-après dénommée "le Client"

### **2. APPROCHE CONTRACTUELLE**

Conformément à l'appel d'offres émis par le Client, nous avons choisi un modèle de Contrat Livraison Clé en Main - Prix Ferme. Le choix de cette approche découle des motifs suivants :

- **Besoin de Livraison à Date Précise**

Le Client exige la livraison d'un produit final à une date précise (5 décembre 2023) pour répondre à ses besoins spécifiques et aux objectifs du projet. Le contrat à terme - temps plus frais ne répond pas à cette exigence, car il ne garantit pas une livraison à date fixe.

- **Non-Applicabilité du Contrat de Partage des Économies**

Étant donné que le produit final est destiné principalement à des études des bénéfices d'un système exploratoire utilisant des robots, l'intention n'est pas de créer un produit destiné à être vendu et profitable sur une longue période. Par conséquent, le contrat de partage des économies ne s'applique pas à ce projet.

- **Clarté des Livrables et Estimation des Coûts**

Le produit final à livrer est clairement défini, et les différents livrables sont spécifiés avec des dates de remise fixes. De plus, il est possible d'estimer les coûts finaux pour l'ensemble du projet, ce qui prouve que le contrat à prix ferme est le plus approprié pour cette situation.



### **3. ENGAGEMENTS DES PARTIES**

- **Responsabilités du Prestataire**

Le Prestataire s'engage à concevoir, développer et livrer le prototype fonctionnel de NMS 4 conformément aux spécifications fournies par le Client. Le Prestataire est responsable de la gestion de projet, de la coordination des ressources, de la réalisation des tests nécessaires et de la livraison du produit final dans les délais convenus.

- **Responsabilités du Client**

Le Client s'engage à fournir au Prestataire toutes les informations et les ressources nécessaires pour la réalisation du projet. Il collaborera activement avec le Prestataire pour garantir le succès du projet.

### **4. DURÉE DU CONTRAT**

Ce contrat entre en vigueur à la date de sa signature et demeure en vigueur jusqu'à la livraison du prototype fonctionnel de NMS 4 conforme aux spécifications convenues.

### **5. PRIX ET PAIEMENT :**

Le prix total convenu pour ce contrat est de 91 625 dollars tel que détaillé dans la section 4.1. Estimation des coûts du projet. Les modalités de paiement sont définies conformément aux termes convenus dans cette offre.

### **6. LIVRAISON :**

Le Prestataire s'engage à livrer le prototype fonctionnel de NMS 4 à l'emplacement désigné par le Client (École Polytechnique Montréal) conformément au calendrier de livraison convenu dans l'appel d'offres.

### **7. PROPRIÉTÉ INTELLECTUELLE :**

Tous les droits de propriété intellectuelle relatifs au prototype développé dans le cadre de ce contrat seront détenus par le Client.

### **8. RÉSILIATION DU CONTRAT :**

En cas de non-respect grave des obligations contractuelles par l'une des parties, l'autre partie se réserve le droit de résilier le contrat conformément aux dispositions légales applicables.

### **9. LOI APPLICABLE ET JURIDICTION :**

En cas de litige découlant de l'interprétation ou de l'exécution de ce contrat, les parties conviennent que la juridiction compétente sera le tribunal de Montréal.

Le Prestataire reconnaît avoir pris connaissance de l'ensemble des termes et des conditions de ce contrat et les accepte en le signant.

### 3. Description de la solution

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

L'image ci-dessous représente le système dans son ensemble, ainsi que ses principaux flux de communication :

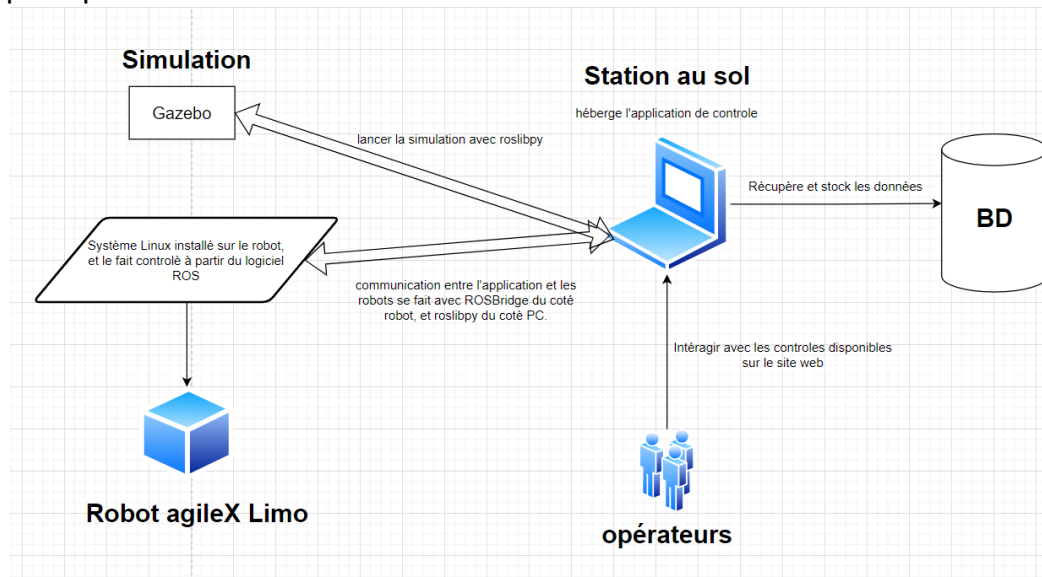


Figure 1: Image décrivant l'architecture globale du projet

L'application web sera hébergée sur la station au sol. En outre, pour répondre aux exigences du projet, une base de données sera utilisée pour stocker les données de la mission et conserver un historique. Ces données comprennent l'horodatage de la mission, la durée de son exécution, les coordonnées de l'environnement scannées - reçues des robots - et d'autres informations connexes. MongoDB a été sélectionné en raison de sa simplicité et de sa fiabilité, comme en témoignent les projets précédents menés dans le cadre de notre programme d'études. Tous les autres éléments représentés dans le schéma (système embarqué et station au sol) seront présentés plus en détail dans les sections suivantes.

En ce qui concerne la communication entre l'application et les robots, elle s'effectue via ROSBridge du côté des robots et roslibpy du côté de l'application. ROSBridge agit comme un "middleware" pour les systèmes ROS, permettant la transmission et la réception de commandes basées sur JSON via des websockets [5]. En parallèle, roslibpy, une bibliothèque Python, facilite cette communication du côté serveur en fournissant les fonctionnalités ROS

nécessaires pour communiquer via ce websocket, de manière identifiable pour le robot lors de l'exécution [4].

Lorsque l'opérateur sélectionne une commande dans l'interface utilisateur, l'application la traduit en une requête HTTP que le serveur doit traiter. Ensuite, avec l'aide de `roslibpy`, le serveur peut envoyer des commandes au robot sous forme de JSON via le websocket établi avec `ROSBridge`. En conséquence, le robot recevra les fichiers de lancement appropriés à exécuter et répondra à la commande initialement choisie.

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

La station au sol sert de centre de commande pour nos robots, via une application web accessible sur toute d'appareil, tels que les ordinateurs personnels, les smartphones et les tablettes. Cette application doit respecter les exigences logicielles spécifiées dans la documentation relative aux exigences logicielles.

Cette application web est structurée en deux composantes principales :

- **L'interface utilisateur** (frontend): Nous avons choisi d'exploiter les capacités du framework Angular pour le créer, étant donné que nous sommes déjà familiers avec celui-ci, et qu'il facilite le développement d'applications web dynamiques et réactives [2]. Nous expliquerons plus en détail la structure du code et sa mise en œuvre que nous avons générée dans la section 3.5 "interface utilisateur".
- **Le serveur** (backend) :

#### ***Choix de framework et langage de programmation:***

- Développé à l'aide de Flask, un framework python connu pour sa facilité d'utilisation, il fournit les outils et les fonctionnalités essentiels nécessaires à la création d'applications web et d'API [3].
- Python était le langage de programmation le plus optimal que nous pouvions utiliser, car il possède de nombreuses bibliothèques mathématiques que nous pouvons exploiter pour traiter les grandes quantités de données que nous recevons de nos robots au cours de chacune de leurs missions.

#### ***Fonctionnement:***

L'architecture de notre serveur est structurée en plusieurs sous-dossiers. Le fichier "run.py" est chargé de l'initialisation et du démarrage du serveur. À partir de ce point d'entrée, les divers sous-dossiers du backend coopèrent pour gérer les requêtes entrantes et manipuler efficacement les données. Plus précisément, dans la structure du backend, le dossier

"controllers" abrite un fichier "controller.py" qui définit tous les points d'accès HTTP pour traiter les requêtes provenant du frontend. Ensuite, ce fichier coordonne les interactions avec les services situés dans notre dossier "services", où s'effectuent les opérations complexes de logique et de traitement des données. Ces services, le cas échéant, s'intègrent avec la base de données pour stocker ou récupérer des informations, établissant ainsi une architecture globale cohérente.

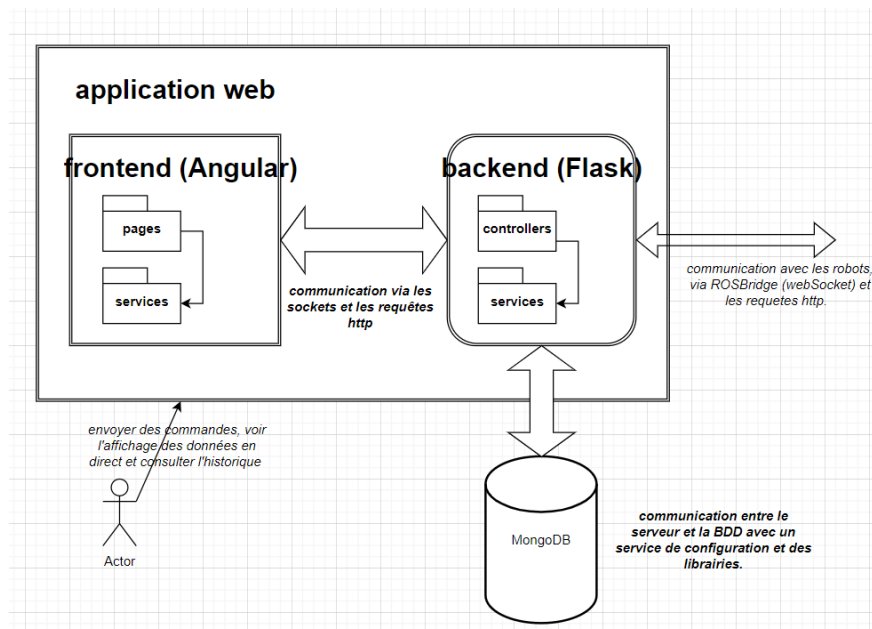


Figure 2: Image décrivant l'architecture de l'application web

### **3.3 Logiciel embarqué (Q4.5)**

Les commandes de l'application Web sont transmises au robot en utilisant ROS Bridge via le topic "launch\_command". Ce topic reçoit à la fois la commande à exécuter et le nom du fichier associé. Le robot prend ensuite en charge le traitement de cette commande. Le package "environment\_setting" est responsable de l'exécution de ces commandes sur le robot. Il le fait de deux manières principales : en utilisant des fichiers au format .launch et en lançant des nœuds ROS.

La communication entre le serveur Web et les robots s'effectue en utilisant ROS Bridge. Par exemple, lorsque l'utilisateur souhaite identifier un robot, une commande est publiée sur le topic /launch\_command via ROS Bridge. Le nœud "launch" du package "environment\_setting" souscrit à ce même topic. Il reçoit la commande et l'exécute directement dans un terminal sur le robot, ce qui permet d'effectuer l'identification.

De même, pour lancer une mission, le nœud ROS reçoit le chemin du fichier .launch correspondant, et il procède à son exécution sur le robot. Cette approche facilite la gestion des commandes et des missions depuis l'application Web, en assurant une communication fluide et efficace avec les robots.

Dans le cadre de la communication entre les deux robots, nous avons opté pour une approche où un robot fait office de maître (robot master) et un autre le suit de près. Cette configuration repose sur l'utilisation de l'export ROS URI, permettant de regrouper tous les sujets (topics) des deux robots sous un même espace ROS. Cette décision a été prise dans le but de simplifier la communication entre les deux robots et de faciliter leur repérage mutuel.

Dans cette architecture, nous avons mis en place des espaces de noms (namespaces) distincts pour chaque robot, et nous avons effectué des remappages (remap) sur certains topics pour les différencier. Par exemple, un topic "/map" est transformé en "/robotn/map" pour garantir une séparation claire. Cette approche présente de nombreux avantages, notamment en termes de débogage, de maintenance et de possibilité d'ajouter de nouveaux robots simultanément.

Cette architecture a grandement simplifié et amélioré le processus de cartographie du terrain, car nous pouvons fusionner les cartes de chaque robot en utilisant le package ROS `multirobot_map_merge`. Cette fusion de cartes est essentielle pour assurer une représentation précise et cohérente de l'environnement, ce qui contribue à la réussite des missions multi-robot.

Dans le contexte de la navigation, de la cartographie et de la fusion de cartes, nous avons choisi les packages suivants pour les robots :

- **move\_base** et **explore\_lite** est utilisé pour la navigation autonome. Il offre une modularité et une efficacité avérée dans la planification de trajectoires, tout en intégrant des mécanismes de récupération en cas d'échec de navigation.
- **gmapping** est le choix pour la cartographie et la localisation simultanée (SLAM).
- **multirobot\_map\_merge** est employé pour l'intégration cartographique. Il assemble les cartes des différents robots, générées par gmapping, en une représentation unifiée.

Plus de détails sur l'utilisation de ces packages sont disponibles dans la section **3.4** de la simulation. Le schéma ci-dessous illustre l'architecture globale de l'environnement du logiciel embarqué, englobant la communication entre l'application Web et le robot, ainsi que le fonctionnement interne des packages, y compris leur mode de communication et leur interaction.

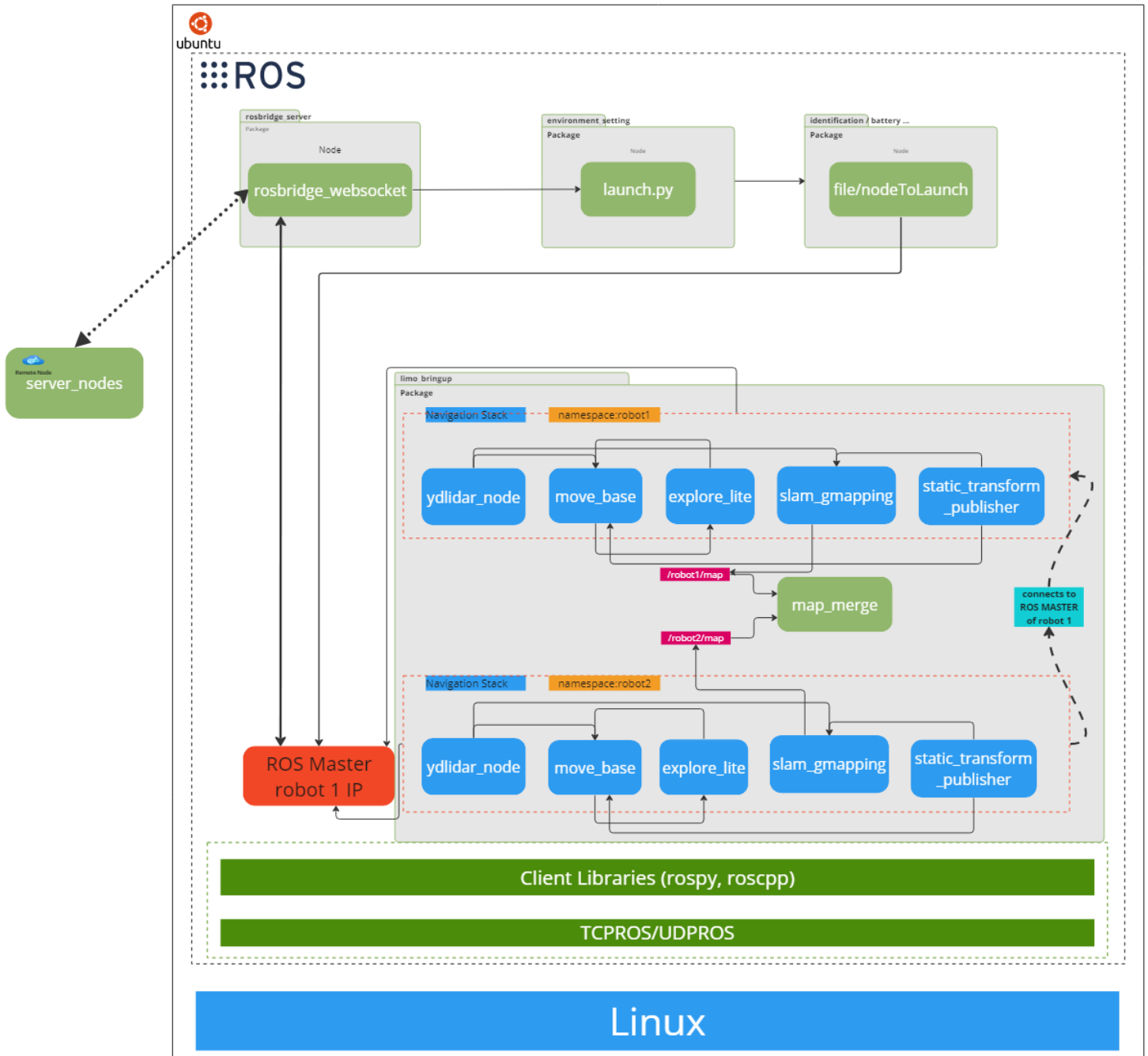


Figure 3 : Schéma décrivant l'architecture générale de l'environnement du logiciel embarqué

### 3.4 Simulation (Q4.5)

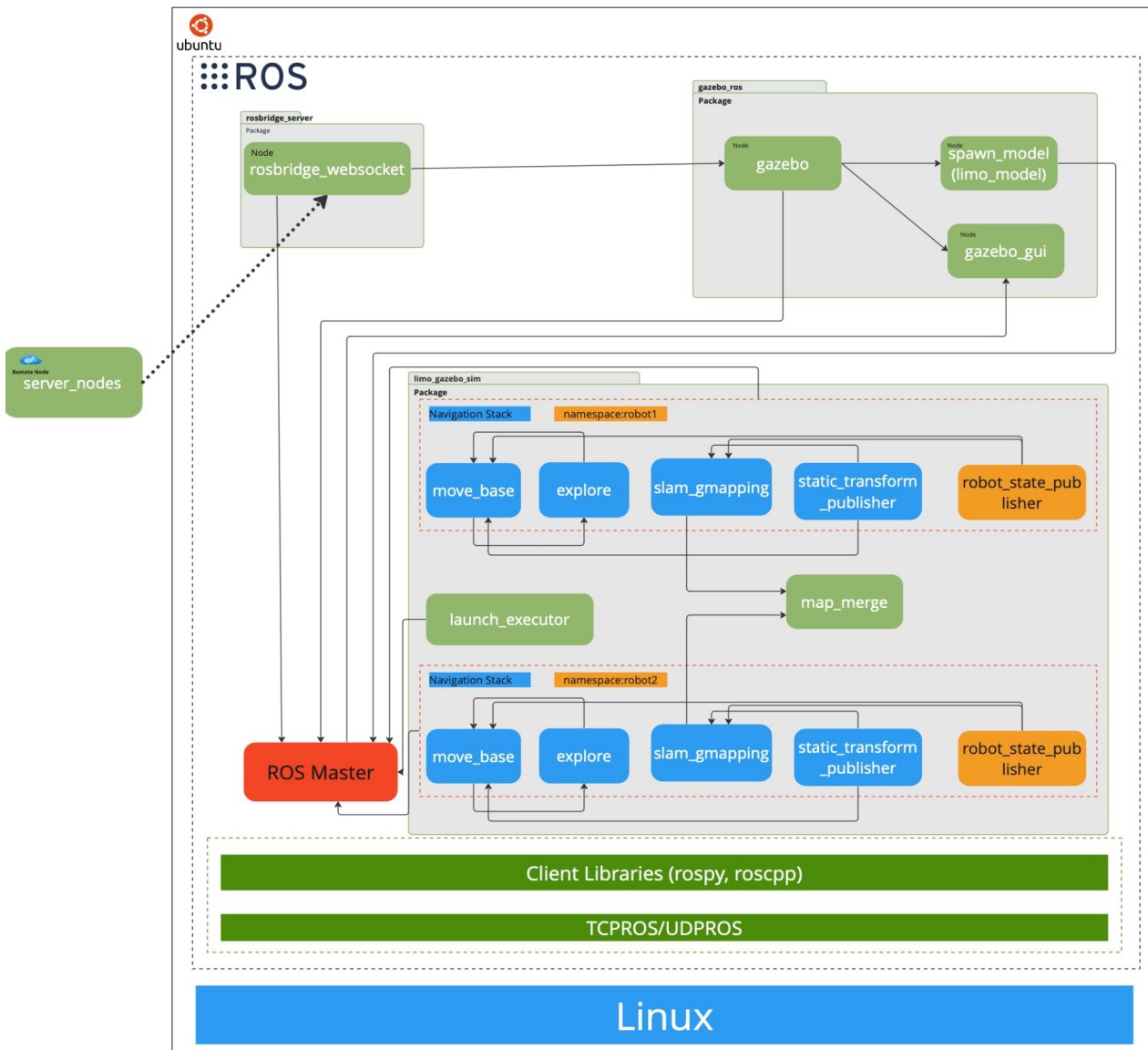


Figure 4 : Schéma décrivant l'architecture de l'environnement de simulation

#### Choix de Gazebo comme Simulateur :

Le choix de Gazebo comme simulateur pour la simulation robotique est justifié par sa capacité à fournir une simulation physique réaliste, son intégration





temps réel une carte précise via le message `nav_msgs/OccupancyGrid`. `Explore_lite` exploite cette carte actualisée pour détecter les frontières inexplorées et diriger le robot vers ces zones vierges. En planifiant de manière autonome des itinéraires vers de nouvelles frontières, `explore_lite` optimise le processus d'exploration, rendant superflue toute intervention manuelle. Cette combinaison de `gmapping` et `explore_lite` équipe les robots avec la capacité d'élaborer une cartographie autonome d'environnements inconnus, augmentant ainsi l'efficacité et l'efficacité de la mission d'exploration.

#### **Choix de `multirobot_map_merge` pour l'Intégration Cartographique :**

Le package `multirobot_map_merge` est sélectionné pour sa capacité spécifique à assembler des cartes de `nav_msgs/OccupancyGrid` issues de différents robots en une représentation unifiée. Cette fonctionnalité est cruciale dans notre scénario où deux robots effectuent simultanément la cartographie SLAM d'un même environnement. `multirobot_map_merge` travaille avec les cartes générées par `gmapping` (`ns/map`) de chaque robot, en ajustant les cartes individuelles pour corriger les écarts et les superposer avec précision. Ce processus produit une vue d'ensemble complète, facilitant la coordination des robots et l'efficacité collective dans des missions complexes.

### **3.5      *Interface utilisateur (Q4.6)***

Le côté front-end, comme mentionné précédemment dans la section 3.2 'Station au sol', a été développé en utilisant Angular. La logique derrière les composants front-end (pages, boutons, etc.) est gérée à travers notre dossier de services, où différents fichiers de services `“.ts”` communiquent avec le back-end via des requêtes HTTP et récupèrent les données lorsque nécessaire.

L'interface utilisateur offre aux opérateurs un ensemble restreint d'actions. La page d'accueil présente les noms et les pourcentages de batterie de tous les robots disponibles. À partir de là, l'utilisateur peut choisir de lancer une mission ou de déclencher un signal sonore d'identification pour un robot sélectionné. En outre, les utilisateurs ont la possibilité d'accéder à une liste des missions précédemment effectuées, qui comprend tous les détails correspondants.

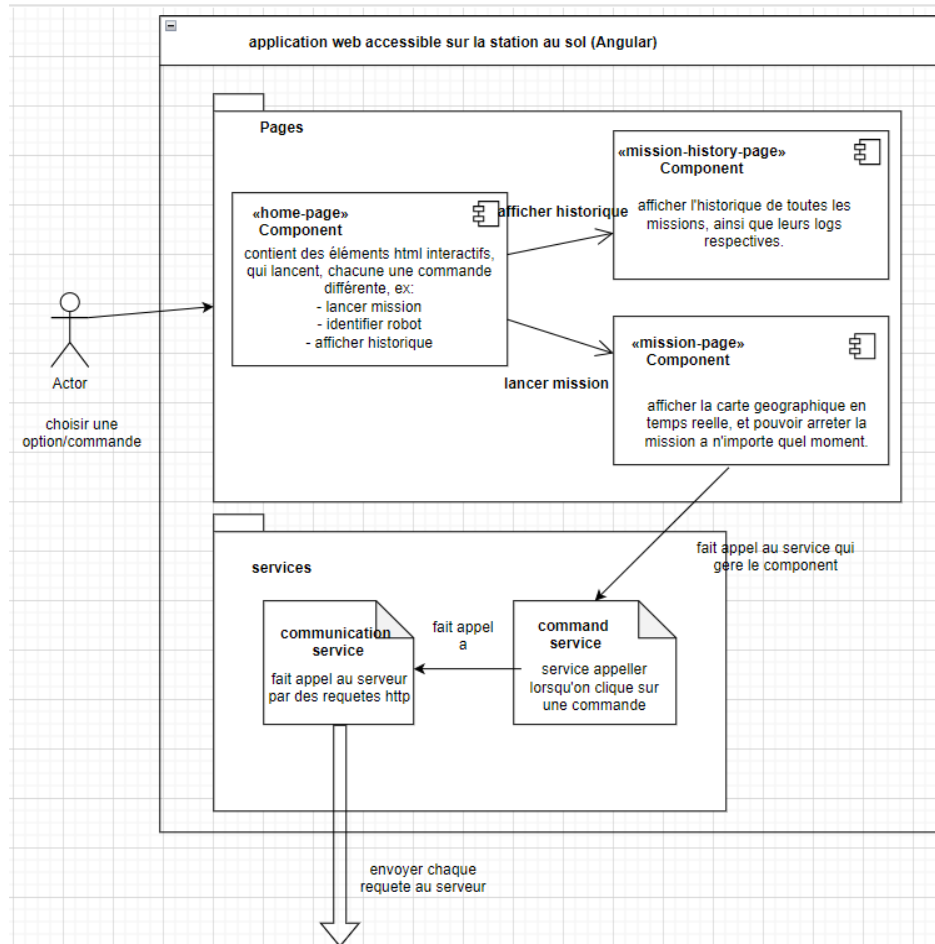


Figure 5 : Image décrivant l'architecture de l'interface utilisateur

### 3.6 Fonctionnement général (Q5.4)

Docker simplifie le processus d'empaquetage des dépendances dans un seul fichier Docker. Cela garantit une exécution fluide du code sur différentes machines et versions de code. Pour faire fonctionner l'application et exécuter les environnements du frontend et du backend il suffit de cloner [le répertoire GitLab](#). Ensuite, utilisez Docker Compose pour lancer les deux environnements dans des conteneurs différents en exécutant la commande `'docker compose up'` à partir du dossier  `'/webapp'`. Cela démarrera les deux conteneurs, rendant l'interface utilisateur accessible sur `'localhost:80'`, et le réseau local, tandis que le serveur backend sera disponible sur `'localhost:8000'`, disponible pour le frontend.

Pour la simulation, après avoir cloné le projet il suffit de source le fichier `setup.bash` situé dans `INF3995104/embedded/g_sim_ws/devel/` et par la suite lancé gazebo avec la commande suivante: `'roslaunch limo_gazebo_sim sim_rosbridge_ser. launch'`.

Du côté des robots, après avoir cloné le projet il suffit de lancer la commande “source” le fichier setup.bash situé dans *INF3995104/embedded/robot/devel/* et par la suite lancer “roscore” sur un terminal puis “roslaunch rosbridge\_server rosbridge\_websocket.launch” sur un autre terminale. Ensuite, lancer la commande sur un nouveau terminale “roslaunch environment\_setting launch.py”. Enfin, rouler le serveur Back-end.

## 4. Processus de gestion

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

Notre équipe chargée de fournir le produit final fonctionnel se compose de six membres, comprenant un coordinateur et cinq développeurs. Le matériel nécessaire au projet inclut seulement les deux robots, et l'environnement qui permet d'imiter une réelle mission avec les robots physiques, comprenant par exemple des obstacles en tout genre.

Main d'oeuvre	Heures	Coût (\$) / Heure	Total (\$)
1x Coordinateur	105	145	15 225
5x Développeur-analyste	525	130	68 250
Total main d'oeuvre (\$)			83 475
Matériel	Quantité	Coût (\$)	Total (\$)
Robot AgilX Limo	2	4 000	8 000
Environnement de test	1	150	150
Total matériel (\$)			8 150
Total (\$)			91 625 \$

*Tableau 1: Coûts du projet*

On estime à 91 625 \$ les coûts totaux du projet.

### 4.2 Planification des tâches (Q11.2)

Dans les tableaux suivants vous pourrez trouver un tableau par jalon principal du projet, indiquant chaque tâche à l'intérieur et donnant le nombre d'heures

estimées par les deux équipes, l'équipe application web (front-end et back-end) et l'équipe robot (limo et simulation).

Preliminary Design Review (PDR)		
Tâche	Temps équipe web (h)	Temps équipe robot (h)
Recherche de documentation des technologies	5	20
Réunions & Organisation	20	20
Création du projet Front end Angular	5	0
Création du projet Back-end Flask	10	0
Adaptation du Frontend à tout type de résolutions et d'appareil	10	0
Intégration de la base de données sur MongoDB	5	0
Implémentation de la fonctionnalité de description du robot et de sa batterie dans l'interface web	10	0
Intégration/communication du simulateur et configuration du robot au Backend	5	10
Implémentation de la fonctionnalité "Identifier" du robot	10	20
Implémentation des fonctionnalités "Initier mission" & "Terminer mission" sur Gazebo	5	10
Correction de qualité de code	5	5
Rédaction des rapports	20	20
Total des heures	110	105
	215	

tableau 2: 1<sup>er</sup> Jalon

Critical Design Review (CDR)		
Tâche	Temps équipe web (h)	Temps équipe robot (h)
Réunions & Organisation	10	10
Déplacements autonome et libre du robot dans une zone sécurisée	0	10
Reconnaissance et prise en compte d'obstacle sur le parcours du robot	0	20
Gestion de plusieurs connexions simultanées	15	0
Implémentation de logs dans l'interface web	10	15
Intégration et génération d'une Map 3D scanné durant la mission	20	15
Gestion de l'historique de missions	10	0
Génération d'obstacles en environnement de simulation	0	15
Correction de qualité de code	10	5
Intégration de tests unitaires	10	5
Création de configuration Docker	5	5
Rédaction des rapports	15	15
Total des heures	105	115
	220	

tableau 3: 2e Jalon3

Readiness Review (RR)		
Tâche	Temps équipe web (h)	Temps équipe robot (h)

Réunions & Organisation	10	10
Implémentation de la fonctionnalité "retour à la base"	5	10
Implémentation de la fonctionnalité de retour automatique à la base si batterie faible	10	5
Affichage de la position du robot en temps réel	15	20
Implémentation de la fonctionnalité de mise à jour du robot via l'application web	15	15
Implémentation de la fonctionnalité de périmètre de sécurité	5	15
Correction de qualité de code	10	10
Intégration de tests unitaires	5	5
Rédaction des rapports	15	15
Total des heures	90	105
	195	

*tableau 4: tâches du 3<sup>e</sup> Jalon*

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

Nous avons décidé de diviser le projet en trois principales étapes, les alignant sur les livrables attendus à la fin de chaque mois.

PDR	CDR	RR
29 septembre 2023	3 novembre 2023	5 décembre 2023
Élaborer une réponse à l'appel d'offres en détaillant l'architecture à suivre pour le reste du projet.	Concevoir et développer le système en visant un achèvement d'au moins 80%	Le produit finalisé, incluant tous les tests et la documentation fournie pour la solution.

*Tableau 5: Dates cibles du projet*

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

Dans cette section, nous présentons l'équipe de projet qui sera responsable de la réalisation de cette initiative. Notre équipe se compose de six développeurs, parmi lesquels l'un d'entre eux assume également le rôle de coordinateur de projet.

Tous nos développeurs sont hautement qualifiés et ont une solide expérience en développement web full-stack. Leur expertise approfondie des technologies et des langages de programmation associés à cette approche garantit une mise en œuvre efficace et de haute qualité. Le coordinateur de projet, en plus de ses compétences de développement, apporte une perspective de gestion essentielle à l'équipe. Son rôle sera de superviser et de coordonner les activités de développement, en s'assurant que le projet avance de manière harmonieuse et conforme aux objectifs définis.

Cette composition d'équipe équilibrée, combinée à l'expertise de nos développeurs en full-stack web development, nous permet de répondre de manière adéquate aux besoins du projet, en garantissant une mise en œuvre réussie et une livraison de qualité.

## **5. Suivi de projet et contrôle**

### **5.1 Contrôle de la qualité (Q4)**

En ce qui concerne le processus de révision, une procédure spécifique est établie pour chaque type de livrable.

- Rapport d'avancement hebdomadaire :

Le rapport se compose de trois sections : une section traitant de l'équipe dans son ensemble, et deux sections dédiées respectivement à l'équipe Robot et à l'équipe Web. Chaque section est rédigée par un membre de l'équipe correspondante, que ce soit l'équipe Robot ou Web. La section portant sur l'équipe dans son ensemble est rédigée en collaboration par ces deux mêmes membres. Ensuite, tous les autres membres de l'équipe consacrent du temps à la relecture du rapport et à la fourniture de commentaires si nécessaire.

- Rapport PDR, CDR, RR:

Pour ce genre de rapport, chacun de nous se charge de rédiger une partie qui lui correspond le mieux en fonction de ses préférences et de son rôle dans l'équipe. Par exemple, si l'on fait partie de l'équipe Robot, on se concentre davantage sur la rédaction des parties liées au robot. Ensuite, on passe en revue les autres sections du rapport, puis on se réunit tous



pour échanger des avis et des critiques avant de finaliser le tout pour la remise.

L'approche employée pour les livrables garantit que nous assumons tous la responsabilité de nos contributions, et les problèmes éventuels ne sont pas attribués à une seule personne.

Les points suivants illustrent notre engagement envers la qualité du code, depuis la conception de l'architecture du projet jusqu'à la revue minutieuse de chaque ligne de code:

- Conception de l'architecture du projet dès le début :
  - Fournit une base stable pour le développement ultérieur
  - Réduit la complexité inutile
  - Facilite la gestion des fonctionnalités
- Utilisation de patrons de conception reconnus :
  - Résolution élégante et efficace de problèmes courants
  - Structuration du code de manière réutilisable
  - Maintien de la cohérence dans tout le projet
- Convention de nommage :
  - CamelCase pour le développement web
  - Snake\_case en Python
  - Noms de variables choisis avec soin
- Découpage du code en fonctions distinctes
  - Facilitation de la maintenance et du débogage
  - Chaque fonction a une tâche spécifique et bien définie
- Lisibilité du code :
  - Réduction de la dépendance aux commentaires
- Utilisation de merge requests pour la revue de code :
  - Détection rapide de problèmes potentiels
  - Assure la cohérence avec les normes du projet
  - Maintient une qualité constante
- Évitement des "code smells" :
  - Identification et correction des signes de code mal structuré ou potentiellement problématique
  - Maintien d'un code propre, performant et facile à gérer

En résumé, nous mettons en œuvre ces bonnes pratiques pour garantir la qualité de notre code, en nous assurant qu'il soit bien conçu, bien nommé, bien structuré et facile à comprendre, tout en utilisant des revues de code pour maintenir un niveau élevé de qualité tout au long du processus de développement.

## 5.2 Gestion de risque (Q11.3)

	Risques	Solutions
Ressources financières	Les retards de financement ou les dépassements de coûts peuvent entraver le progrès du projet.	Assurer une gestion budgétaire stricte et envisager des sources de financement supplémentaires en cas de besoin.
Ressources humaines	Si un membre de l'équipe ne remplit pas ses responsabilités ou s'il y a un manque de coordination, cela peut entraîner des retards ou une baisse de qualité.	Établir des rôles et des responsabilités clairs, prévoir un plan de remplacement en cas de défaillance d'un membre.
Ressources matérielles	Les problèmes liés aux composants physiques, tels que les capteurs, peuvent causer des retards ou des perturbations au cours du développement du projet. De plus, ils ne sont pas faciles à régler.	Maintenir un contrôle de qualité pour les composants physiques.
Communication interne	La communication inadéquate au sein de l'équipe peut entraîner des malentendus et donc des retards ou un malfonctionnement.	Établir des canaux de communication clairs et résoudre rapidement les problèmes de communication au sein de l'équipe

Communication externe	Un manque de communication avec le client ou des retards dans la remise des livrables peuvent avoir un impact négatif sur la satisfaction finale du client.	Définir des points de contact avec le client, respecter les délais de réponse et clarifier les attentes mutuelles.
-----------------------	---	--

*Tableau 6: Risques potentiels et leurs solutions*

Gestion des changements :

- Mettre en place un processus formel de gestion des changements, y compris l'évaluation de l'impact sur le budget, le calendrier et les objectifs du projet.
- Communiquer de manière transparente avec toutes les parties prenantes pour garantir une compréhension et une acceptation mutuelles des changements.

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

Pour identifier et préciser les tests pour chaque sous-système, nous devons nous assurer que nos tests sont pertinents pour le matériel et le logiciel impliqués dans notre projet, d'où tester :

- Système de contrôle et de communication :

La communication bidirectionnelle entre l'interface utilisateur Web et les robots.

La stabilité de la communication sur différents types de réseaux et appareils (PC, tablette, téléphone).

- Interface utilisateur Web :

La convivialité de l'interface Web en collectant des feedbacks sur l'expérience utilisateur (UX).

La compatibilité sur divers navigateurs en vérifiant que l'interface s'adapte correctement à différentes résolutions d'écran.

- Intégration avec Gazebo :

La synchronisation entre le simulateur Gazebo et le Back-end.

- Navigation autonome et Cartographie :

La cartographie de l'environnement en temps réel en utilisant leurs capteurs (IMU, caméra 3D, caméra RGB).

La capacité des robots à éviter les obstacles en explorant un environnement simulé ou réel.

- Transmission de données en temps réel :

La transmission en temps réel des données des capteurs depuis les robots vers l'interface utilisateur, en vérifiant la qualité des données reçues.

La mise à jour en temps réel de la position des robots sur la carte de l'interface utilisateur.

- Stockage et gestion des données :

Le stockage efficace et sécurisé des données collectées par les robots, en s'assurant qu'elles sont accessibles pour la cartographie ultérieure.

Dans l'ensemble, nos tests devraient viser à valider la fonctionnalité, la stabilité et la sécurité du système dans son ensemble, en garantissant une expérience utilisateur fluide et fiable. Les tests devraient être effectués de manière exhaustive pour couvrir un large éventail de cas d'utilisation possibles.

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

Pour garantir la qualité et la stabilité de notre projet, nous avons mis en place un système de contrôle de version robuste ainsi qu'une organisation du code source efficace. Voici comment nous gérons ces aspects :

- Gestion de version et organisation du code source :  
Nous utilisons GitLab comme plateforme de gestion de versions pour notre code source. Notre répertoire principal se divise en deux sous-dossiers distincts, l'un étant dédié à l'application Web qui contient deux sous-dossiers pour le Front-end et le Back-end et l'autre au code du robot, pour lequel nous utilisons un espace de travail Catkin pour sa gestion. Cela nous permet de maintenir un contrôle précis sur les différentes versions du logiciel du robot.
- Gestion des branches :  
Chaque membre de l'équipe crée sa propre branche lorsqu'il commence à travailler sur une nouvelle fonctionnalité, une correction de bogue ou tout

autre changement. Nous avons une convention de nomenclature pour nos branches, par exemple "feat/identify-robot" pour une fonctionnalité visant à identifier le robot. Le développement se fait dans ces branches spécifiques, ce qui facilite la gestion des modifications et des tests.

- Merge Request et Peer Code Review :  
Une fois qu'une tâche est terminée, le développeur crée une merge request (MR) pour fusionner sa branche avec la branche principale (main). Avant de fusionner, nous avons mis en place un processus de revue de code par les pairs. Cela garantit que les erreurs sont identifiées et corrigées avant la fusion. Ce processus contribue à maintenir la stabilité du projet et à éviter la propagation de bogues.
- Commits fréquents :  
Nous encourageons les commits fréquents à chaque petit avancement fonctionnel. Cela nous permet de suivre précisément les progrès de chaque membre et de revenir facilement à un point antérieur en cas de besoin. En plus de faciliter la gestion des versions, cette approche favorise un code de meilleure qualité qui peut être maintenu plus facilement.
- Documentation :  
Nous accordons également de l'importance à la documentation. Chaque composant du projet doit être accompagné d'une documentation adéquate. Cela inclut la documentation du code source ainsi que la documentation de conception, notamment des fichiers tels que le PDR. Nous utilisons principalement des fichiers README dans nos référentiels pour documenter les informations importantes.

En résumé, notre processus de gestion de version, d'organisation du code source et de revue de code par les pairs contribue à garantir la qualité, la stabilité et la traçabilité de notre projet. Ces pratiques sont essentielles pour un projet aussi complexe que le nôtre, où la collaboration entre les membres de l'équipe est cruciale pour son succès.

## ***5.5 Déroulement du projet (Q2.5)***

### **❖ Application Web :**

L'architecture initiale de notre application web, établie lors du sprint de la Revue Préliminaire de Conception (Preliminary Design Review - PDR), a été fidèlement respectée et développée tout au long du sprint de la Revue Critique de Conception (Critical Design Review - CDR), avec des ajustements et des évolutions mineures. Ces évolutions ont été apportées de manière réfléchie pour accompagner la croissance progressive du

projet, comme l'intégration des informations de logs et l'implémentation d'une carte interactive durant chaque mission robotique.

Il est essentiel de souligner que l'avancement de l'application web s'est déroulé en parfaite adéquation avec les délais fixés par l'appel d'offres, illustrant notre capacité à gérer efficacement le temps et à répondre aux exigences de notre clientèle. Chaque ajout, chaque fonctionnalité implémentée, a été pensée pour renforcer l'utilité et la réactivité du système, assurant que nous ne livrons pas seulement selon les spécifications, mais que nous visons également l'excellence opérationnelle. Cela reflète notre engagement envers l'innovation et le service client.

❖ Robots:

Notre projet a accompli avec succès plusieurs aspects essentiels. Les fonctionnalités de lancement des missions des robots et le suivi des logs fonctionnent. De plus, la communication des états des robots, la gestion des logs de mission, le suivi de la durée de vie de la batterie, et la détection des obstacles sont tous bien en place.

Cependant, nous avons rencontré des défis tout au long du projet. Le principal défi a été la cartographie. Au début, la génération de cartes individuelles pour chaque robot et leur fusion dynamique ont présenté des difficultés considérables, mais nous avons finalement réussi à les surmonter. À noter que, hormis le retard de deux jours dans la cartographie, nous avons respecté tous les délais que notre équipe avait convenus. Le projet progresse de manière satisfaisante, et nous sommes déterminés à le mener à bien.

Malgré ces défis, notre équipe de développement logiciel embarqué a fait preuve de sa capacité à gérer le temps de manière efficace et à répondre aux exigences du projet, en veillant à ce que toutes les fonctionnalités essentielles soient opérationnelles, même dans des situations complexes. Notre engagement envers l'innovation et la satisfaction du client reste inchangé, et nous continuons à travailler pour relever ces défis tout en améliorant davantage notre projet.

Il convient de noter que les ajustements apportés à notre architecture permettent une meilleure maintenance et facilitent l'ajout de nouvelles fonctionnalités, ce qui renforce la robustesse de notre projet.

❖ **Simulation:**

- **Intégration des Packages ROS :** Notre équipe a efficacement intégré, `explore_lite` et `gmapping` + `move_base` (nav stack), ce qui a permis une cartographie autonome fiable ainsi que l'exploration des robots dans un environnement comportant des obstacles statiques et dynamiques aléatoires.
- **Collaboration Virtuelle :** L'utilisation du package `multirobot_map_merge` dans notre simulation a très bien fonctionné, permettant la fusion des différentes cartes générées par chaque robot dans un environnement partagé et contribuant à la réussite de la mission d'exploration collective.

Cependant, il reste encore des points que nous aimerions améliorer:

**Arrêt des Robots dans la Simulation :** Un des défis rencontrés a été l'arrêt immédiat et précis des robots. Même après l'arrêt des nœuds `explore_lite`, une persistance du mouvement est observée, les robots continuent leur trajet pendant une courte période avant de s'immobiliser totalement. Cette situation découle du fait que '`explore_lite`' envoie des objectifs (goals) successifs à l'API d'action `move_base`.

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

❖ **Application Web :**

- **Compatibilité multiplateforme**

Notre application Web offre une expérience utilisateur optimisée pour le contrôle et la gestion de robots, à la fois physiques et simulés, et ce, sur une variété de dispositifs. Elle est conçue pour s'adapter aussi bien aux ordinateurs de bureau qu'aux appareils mobiles tels que les tablettes et les téléphones, assurant une utilisation souple et pratique sur le terrain ou dans le confort d'un bureau.

- **Lancement des robots physiques et de la simulation**

En ce qui concerne les robots physiques, chaque robot peut être lancé individuellement via une page d'accueil intuitive. Cette interface est conçue pour permettre aux utilisateurs d'initier des missions d'exploration avec une facilité remarquable. Pendant ces missions, une carte interactive est dynamiquement mise à jour, reflétant en temps réel l'avancement de l'exploration du robot, offrant ainsi une visualisation claire et précise quel que soit le dispositif employé pour accéder à l'application.

Dans l'environnement simulé, notre système est capable de gérer le lancement des robots individuellement ou bien les deux à la fois. Les coordonnées de chaque robot sont transmises à chaque seconde et visualisées sur la carte par des marqueurs triangulaires verts, fournissant une représentation graphique claire de leur parcours d'exploration.

- **Arrêt des robots**

Que ce soit pour les robots physiques ou bien pour la simulation, nous avons incorporé une fonctionnalité permettant l'arrêt des robots via un bouton dédié sur l'interface de l'application web. Cette mesure de sécurité assure que les utilisateurs peuvent intervenir rapidement en cas de nécessité.

- **Choix entre le mode simulation et mode robots physiques**

Notre page d'accueil sert de tableau de bord centralisé, où les utilisateurs peuvent non seulement identifier chaque robot physique mais également basculer entre le mode simulation et l'activation des robots physiques. Une section historique est accessible via un bouton dédié, permettant aux utilisateurs de consulter les missions antérieures.

- **Les logs au cours et à la fin de la mission**

L'affichage en direct des logs des missions actives offre une transparence complète, tandis que la sauvegarde automatique de ces données dans notre base de données garantit que rien n'est perdu. Les logs des missions précédentes peuvent être facilement revues, grâce au bouton qui se trouve à la page d'accueil.

En résumé, notre application web fonctionne de manière fluide et efficace, attestant de notre engagement à fournir une solution complète offrant un contrôle sans faille et à optimiser l'interaction entre l'utilisateur et les robots.

- ❖ **Robots :**

- **Identification du Robot**

Nous avons implémenté des fonctionnalités qui permettent à chaque robot de répondre individuellement à la commande "Identifier". Lorsque cette commande est activée, le robot effectue l'une des actions suivantes : afficher une animation sur l'écran ou émettre un son. Les réponses des



robots sont synchronisées avec l'activation de la commande "Identifier" à partir de l'interface utilisateur.

- **Lancer et arrêter des missions**

Pour la gestion des missions, nous avons mis en place des commandes pour le lancement et l'arrêt des missions des robots. La commande "Lancer la mission" démarre une mission, tandis que la commande "Terminer la mission" arrête la mission en cours. La commande "Terminer la mission" interrompt toute action en cours et arrête tous les robots.

- **Affichage de l'état**

Les robots communiquent leur état à une fréquence de 1 Hz, assurant des informations à jour en temps réel. L'état de chaque robot est synchronisé avec ses actions réelles, garantissant que l'état communiqué reflète la réalité de la mission.

- **Cartographie**

En ce qui concerne la cartographie, les robots commencent à explorer l'environnement de manière autonome après le départ. Ils évitent les collisions avec des obstacles en temps réel et poursuivent l'exploration jusqu'à ce que la condition de fin soit satisfaite (Terminer mission). La carte de l'environnement est mise à jour en temps réel et est communiquée à la station au sol, permettant aux utilisateurs de suivre dynamiquement la progression de l'exploration.

- **Détection d'obstacles**

Les robots prennent des mesures pour éviter les obstacles, qu'ils soient statiques ou d'autres robots. Ces mesures sont efficaces pour éviter des collisions et permettent aux robots de poursuivre leur exploration. En cas de détection simultanée de plusieurs obstacles, les robots perçoivent l'évitement en fonction du danger immédiat.

- **Logging**

Pendant chaque mission, un système de logging enregistre en continu des informations cruciales, notamment la vie de la batterie de chaque robot, les coordonnées des obstacles détectés, ainsi que les données des capteurs, telles que la distance et la position, à une fréquence de 1 Hz. Ces logs de débogage sont accessibles via l'interface utilisateur et sont sauvegardés à la fin de chaque mission. Cette fonctionnalité permet de diagnostiquer les problèmes du système en fournissant des informations lisibles, datées et horodatées pour une analyse chronologique.

❖ **Simulation :**

- **Intégration du Nav Stack**

L'intégration de `explore_lite` et `Multirobot_map_merge` avec `gmapping` et `move_base` notre **Nav Stack** s'est déroulée avec succès, permettant une navigation et une cartographie autonomes efficaces au sein d'un environnement peuplé d'obstacles statiques et dynamiques.

- **Cartographie et Exploration Autonome**

Les robots sont capables de cartographier et d'explorer l'environnement de manière autonome, identifiant et naviguant avec succès autour des obstacles rencontrés.

Cependant, il reste encore des points que nous aimerions améliorer:

- **Configuration de `Multirobot_map_merge`**

Actuellement, notre configuration de `multirobot_map_merge` est fonctionnelle pour la gestion de deux robots. Nous devons étendre et optimiser cette configuration pour qu'elle puisse prendre en charge un nombre indéterminé de robots et qu'elle les détecte de manière dynamique.

- **Initialisation Dynamique des Positions des Robots**

Les positions initiales des robots sont actuellement statiques et doivent être prédéfinies avant l'exécution. Nous visons à rendre ce processus dynamique pour que les positions puissent être assignées et ajustées en temps réel au moment de l'exécution, ce qui augmentera la flexibilité de notre système dans des scénarios d'application variés.

## 6. Références (Q3.2)

- [1] AgileX (2023). [En ligne]. Disponible: <https://global.agilex.ai/products/limo>
- [2] Angular. What is Angular? (s.d). [En ligne]. Disponible: <https://angular.io/guide/what-is-angular>
- [3] Python Tutorial. What is Flask (s.d). [En ligne]. Disponible: <https://pythonbasics.org/what-is-flask-python/>
- [4] roslibpy: ROS Bridge library. [En ligne]. Disponible: <https://roslibpy.readthedocs.io/en/latest/>
- [5] rosbridge\_suite. [En ligne]. Disponible: [http://wiki.ros.org/rosbridge\\_suite#:~:text=The%20rosbridge%20protocol%20is%20a.%3A%20%22geometry\\_msgs%2FTwist%22%20%7D](http://wiki.ros.org/rosbridge_suite#:~:text=The%20rosbridge%20protocol%20is%20a.%3A%20%22geometry_msgs%2FTwist%22%20%7D)
- [6] Why do we use gazebo instead of unreal or unity ? [En ligne]. Disponible: <https://discourse.ros.org/t/why-do-we-use-gazebo-instead-of-unreal-or-unity/25890/6>