



**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. A2025-INF3995 du département GIGL.

Conception d'un système d'exploration robotique

Équipe No 106

Aksas, Sarah - 2198935
Grandisson, Arnaud - 2241426
Menouar, Anis - 2247873
Paillé Dowell, Félix - 2256243
Nzudom, Patrick - 2218623
Roux, Matis - 2148625

1er Octobre 2025

1. Vue d'ensemble du projet

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

Le projet vise à concevoir et démontrer une preuve de concept, basée sur ROS 2 Humble, d'un système d'exploration multi-robot capable d'opérer dans un environnement intérieur. Plutôt que de s'appuyer sur un seul robot complexe, la solution repose sur l'utilisation de deux robots AgileX Limo dotés de capteurs de base, notamment lidar, caméras et IMU. Ces robots doivent être en mesure d'explorer une pièce de manière autonome, de détecter et éviter les obstacles, et de transmettre leurs données vers une station au sol disposant d'une interface web.

La portée du projet couvre à la fois le développement logiciel et l'intégration matérielle. D'une part, il s'agit de mettre en place une interface web en Angular permettant à des utilisateurs sans connaissances techniques de piloter le système grâce à de simples actions, comme l'identification ou le lancement d'une mission. D'autre part, un backend en Python (FastAPI) servira de passerelle entre l'interface et le middleware ROS 2, tandis que des nœuds ROS 2 embarqués sur les robots exécuteront les commandes haut niveau et assureront l'exploration autonome. Une base de données sera également mise en place afin de conserver l'historique des missions, des cartes et des journaux d'exécution, et une simulation Gazebo viendra compléter l'ensemble pour tester les fonctionnalités avant déploiement réel.

Les objectifs spécifiques se déclinent en trois étapes alignées avec les livrables principaux de la session. Le premier objectif est de démontrer une preuve de concept fonctionnelle dès le PDR avec l'intégration minimale de la chaîne complète (interface, backend, ROS2 et robot). Le second objectif est de développer la majorité des fonctionnalités clés au CDR, notamment l'exploration, l'évitement des obstacles, le retour à la base et la cartographie collaborative. Enfin, le troisième objectif est de livrer un système complet, robuste et documenté pour le RR, incluant une interface multi-appareils intuitive et toutes les fonctionnalités requises.

1.2 *Hypothèse et contraintes (Q3.1)*

Le plan du projet repose sur plusieurs hypothèses. Tout d'abord, on suppose que les robots Limo fournis par l'Agence seront fonctionnels, configurés avec Ubuntu 22.04 et ROS 2 Humble, et accessibles pendant toute la durée de la session. Il est également supposé que les environnements de test, tels que les corridors et les salles du laboratoire, seront suffisamment spacieux et sécurisés pour réaliser des expérimentations à basse vitesse. Une autre hypothèse importante est que les membres de l'équipe disposent de compétences complémentaires en développement web, programmation Python et robotique.

ROS, et qu'ils sont en mesure de consacrer le temps requis chaque semaine pour respecter l'échéancier. Enfin, les outils de développement choisis, notamment GitLab, Docker, ROS 2 et Angular, devraient rester stables et supportés durant tout le trimestre.

Plusieurs contraintes encadrent aussi le projet. La contrainte de temps est majeure, puisque le projet doit être mené à bien en une seule session académique avec des dates fixes pour les livrables. Les ressources matérielles sont limitées à deux robots Limo, ce qui exige une planification rigoureuse pour partager leur utilisation. La communication réseau repose exclusivement sur le Wi-Fi fourni, ce qui impose de composer avec ses limites en matière de bande passante et de stabilité. Une contrainte technique additionnelle est la nécessité de conteneuriser tous les modules côté station (UI, backend, base de données, simulation) dans des environnements Docker. Enfin, les requis obligatoires définis dans le cahier des charges doivent absolument être respectés pour valider le projet, ce qui limite la marge de manœuvre dans les choix de conception.

1.3 Biens livrables du projet (Q4.1)

Le projet sera livré en trois étapes principales. Le Prototype de base (PDR), prévu pour le 29 septembre 2025, inclura une première version de l'interface web permettant d'exécuter les commandes "Identifier" et "Lancer/Arrêter mission", un backend fonctionnel relié au middleware ROS 2, des nœuds embarqués sur les robots pour l'exécution des commandes de base, ainsi qu'une documentation initiale décrivant l'architecture physique et logicielle.

Le livrable intermédiaire (CDR), prévu pour le 11 novembre 2025, livrera la majorité des fonctionnalités attendues. Il comprendra l'exploration autonome des robots, l'évitement des obstacles, le retour à la base manuel et automatique lorsque la batterie descend sous le seuil de 30 %, ainsi qu'une cartographie collaborative 2D/3D en temps réel. L'interface web sera enrichie pour afficher l'état des robots et visualiser la carte, et une documentation technique détaillée accompagnera ce livrable.

Enfin, le livrable final (RR), attendu pour le 4 décembre 2025, représentera le produit complet et stable. Il comprendra une interface web multi-appareils (PC, tablette, mobile), une base de données intégrée pour stocker et consulter l'historique des missions et des cartes, des vidéos de démonstration, des journaux de mission, une documentation finale complète ainsi qu'une présentation orale. Ce dernier jalon aura pour but de valider non seulement la complétude du système mais aussi sa robustesse et son utilisabilité.

2. Organisation du projet

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

Pour organiser le projet, nous nous sommes séparés en trois équipes de deux. Ceci permet une meilleure communication interne pour l'équipe. Chaque paire travaille de façon individuelle et donc les deux membres de la paire peuvent facilement communiquer et travailler ensemble. Les trois paires se rencontrent deux à trois fois par semaine et communiquent quotidiennement par discord afin de tous se mettre au courant du progrès des autres paires. Quand il faut intégrer le travail de deux paires ensemble, le fait que chaque membre connaît très bien le travail de sa paire fait qu'aucun ne se sent perdu ou inutile.

La première tâche que nous nous sommes donnés avec ces sous-groupes est de compléter chacun une question de la documentation du projet. Avec cette première tâche, nous avons pu constater que cette méthode de travail est efficace. Effectivement, on encapsule le problème en des sous-tâches (dans ce cas, des questions de la documentation) et on s'y met en paires. Nous trouvons que se mettre en paires et mieux que de travailler individuellement car on garde un esprit d'équipe et de communication.

Pour y aller plus en détail, chaque membre a un rôle défini dans l'équipe. Bien entendu, ces rôles sont au-delà des responsabilités de chaque membre d'être assidu dans ses démarches et de respecter ses engagements. Ce sont plutôt des rôles implicites qui se sont développés naturellement et que les membres ont accepté volontairement lors d'une rencontre.

Anis était loquace quant il s'agit d'aider à définir les règlements dans l'équipe. De ce fait, son rôle est d'assurer la cohésion et le maintien d'un cadre de travail clair. Ces règlements facilitent une meilleure productivité et favorisent un environnement de travail sain et positif. Ces règles ont été acceptées par tous et peuvent être ajustées au besoin.

Matis a un rôle de facilitateur dans l'équipe. En effet, en se portant volontaire pour commencer le site web pour le projet, il facilite l'inclusion des autres membres dans le développement de cette interface en emmenant une structure solide. Il a donc illustré un bel exemple de d'innovateur dès le début du projet. Cela a permis à l'équipe d'avancer sans perte de temps.

Dès notre première rencontre en équipe, Sarah s'est portée volontaire pour créer un document permettant la planification de nos rencontres et la prise en notes de nos idées. Grâce à cette initiative, nous avançons avec une vision claire de ce qui nous attend tout au long du projet. Son implication nous permet de ne pas perdre de vue nos priorités et de rester alignés vers nos objectifs.

Félix a un rôle d'analyste. Il travaille efficacement quand il maîtrise bien les outils nécessaires pour accomplir une tâche. Or, dès le début du projet quand il fallait faire les premiers script pour contrôler le robot, il a su prendre du recul, évaluer ses options et s'assurer de bien comprendre les outils comme ROS2, puis proposer des solutions pour compléter les requis.

Patrick a un rôle de coordinateur dans l'équipe. Il est orienté vers l'harmonie au sein des membres de l'équipe et s'assure que les compétences de chacun soient utilisées à leur plein potentiel tout en aidant au maintien d'une excellente atmosphère et d'une bonne cohésion dans l'équipe. Il s'assure que chacun a assez d'espace pour s'exprimer.

Arnaud est un propulseur qui fait avancer l'équipe. En effet, il est axé vers l'action et le partage de ses connaissances. Dès le début du projet, il était le premier à avoir pu Dual Boot son laptop ou à avoir pu diriger les robots. Lors de ces deux instances, il a aidé les autres membres de l'équipe à avancer avec efficacité et confiance.

2.2 Entente contractuelle (Q11.1)

Nous proposons un contrat clé en main - prix fixe. Nous connaissons à l'avance les spécifications exactes du projet et elles ne changeront pas. Nous cherchons à livrer un produit final avec un délai fixe et non de maintenir un système à long terme. Le promoteur cherche un suivi minimal, il veut juste recevoir le produit final. L'agence spatiale cherche une preuve de concept.

Un contrat à terme ne serait pas idéal pour ce projet, car le client cherche un produit final livré à une certaine échéance. Un contrat à terme ne demande que du contracteur qu'il mette de l'effort jusqu'à la fin du contrat, sans nécessairement livrer un produit final avant l'échéance.

Un contrat à partage d'économies ne serait pas idéal non plus, car le produit final n'est pas à fins économiques mais scientifiques. L'Agence Spatiale cherche simplement une preuve de concept afin de pouvoir approuver une mission spatiale future, alors le but du projet pour eux est plutôt l'avancement scientifique que le profit.

Pour conclure, on choisit un contrat clé en main - prix fixe car l'agence cherche à recevoir un produit fonctionnel à la fin du contrat.

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)

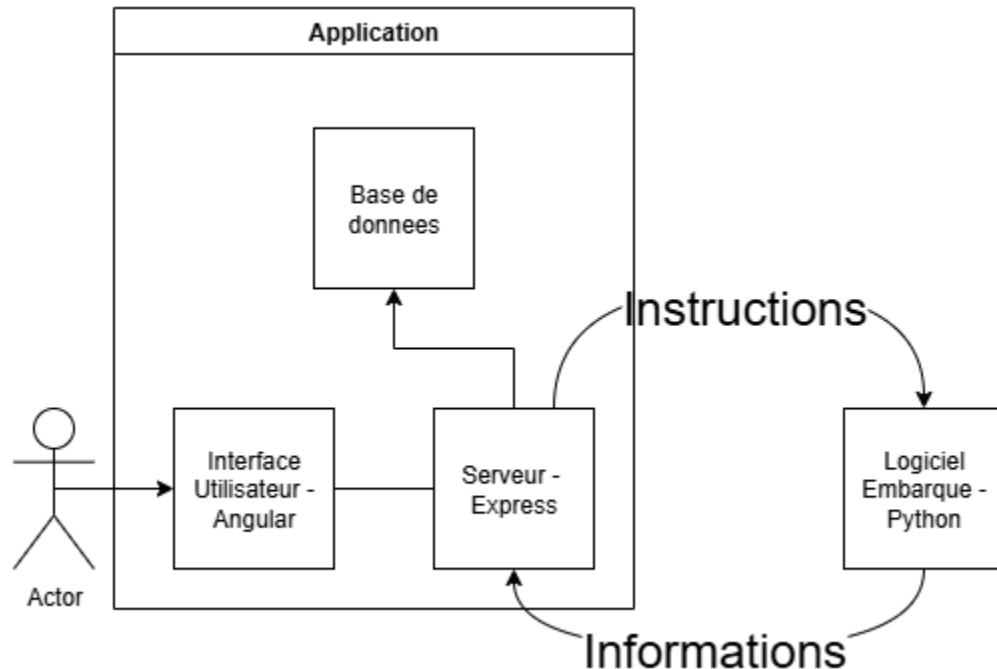


Figure 1: Architecture du projet

Nous avons choisi de développer une application web qui s'exécutera en local, connectée au même réseau que les robots. Cette interface web simplifie l'expérience utilisateur en permettant de lancer et de suivre les missions. Elle communique avec un serveur Node, qui joue le rôle de middleware : il assure l'affichage dynamique de l'application, la transmission des instructions vers le logiciel embarqué du robot et la réception des données envoyées par celui-ci. Le serveur gère également le stockage des informations de mission (cartes sauvegardées, données de retour, historiques, etc.) dans une base de données.

3.2 Station au sol (Q4.5)

Le système repose sur une station au sol qui assure à la fois le contrôle, la gestion des données et des communications. Cet élément n'est pas monolithique : il s'appuie sur plusieurs modules complémentaires fonctionnant de manière autonome mais interconnectée. Pour assurer une architecture claire et maintenable, nous avons séparé ce système en trois principaux blocs : La base de données, le serveur et l'interface utilisateur, comme on peut le voir dans le diagramme ci-dessous.

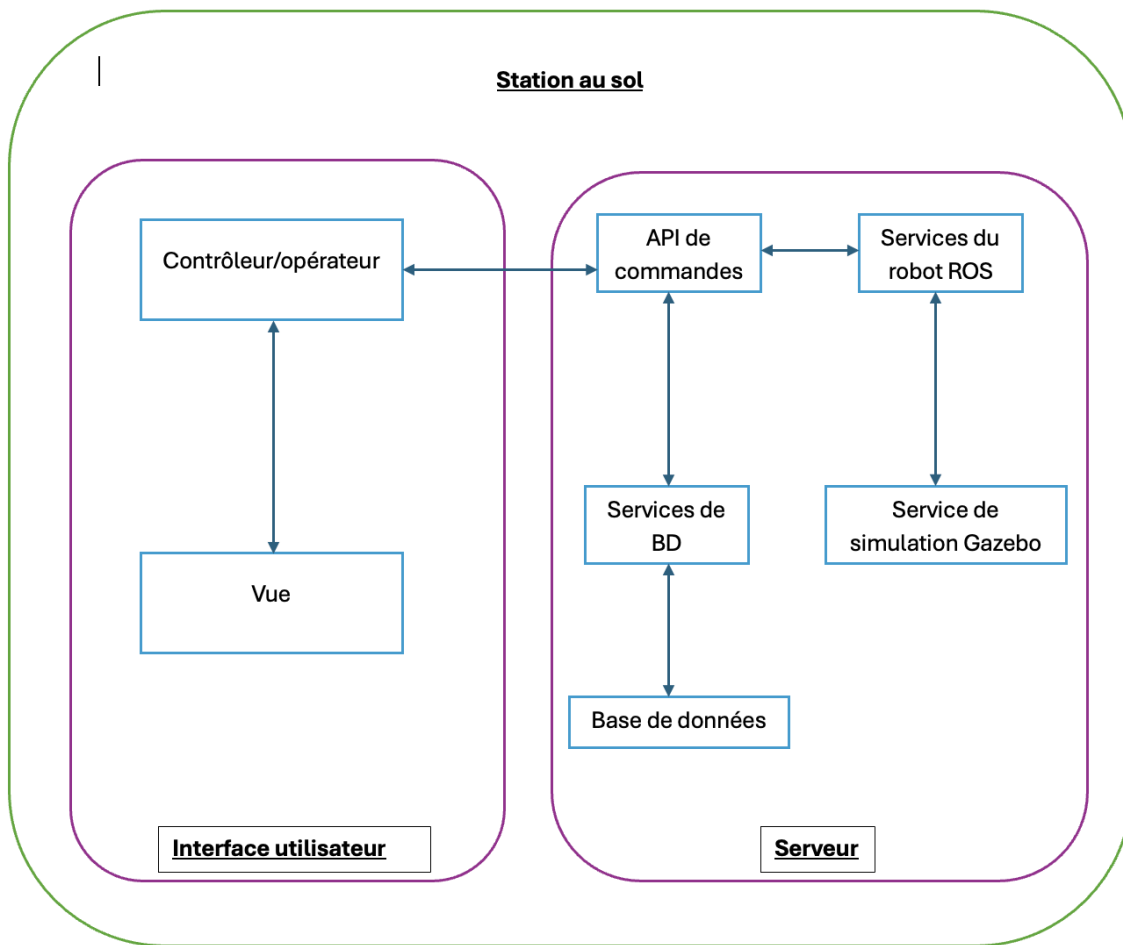


Figure 2: Architecture de station au sol

Analysons à présent ce diagramme pour avoir une meilleure compréhension de l'architecture de la station au sol.

Du côté de l'interface utilisateur, le contrôleur/opérateur correspond à la logique applicative qui gère les actions saisies par l'utilisateur. Il joue un rôle central en transmettant les commandes vers le serveur et en recevant les retours d'état des robots ou de la base de données. Directement lié à ce contrôleur, la **vue** représente l'affichage graphique : c'est ce que l'utilisateur voit à l'écran dans l'application Angular (cartes, boutons de commande, état des robots, etc.). La vue et le contrôleur interagissent en continu afin de refléter les changements de l'état du système en temps réel.

La partie serveur assure la logique de communication et de persistance. L'API de commandes constitue le point d'entrée principal : elle reçoit les requêtes envoyées par le contrôleur/opérateur (ex. : démarrer une mission, identifier un robot, arrêter un processus), par requête HTTPS ou par websockets selon la situation, et les redirige vers les services appropriés. Ces requêtes peuvent ensuite être transmises aux services, topics et actions du robot ROS, qui

exposent des fonctionnalités comme l'identification, la navigation ou la publication de commandes de mouvement.

Le serveur inclut également des services de base de données (BD) qui encapsulent la logique de persistance. Ils gèrent la sauvegarde et la consultation des cartes, l'historique des missions et la journalisation des événements. Ces services échangent directement avec la base de données MongoDB, qui stocke les données sous forme de documents JSON, facilitant ainsi les échanges rapides et l'adaptation à des données hétérogènes. Sachant que les données/informations qu'on veut sauvegarder ne sont pas énormes, l'utilisation de MongoDB est la meilleure. Avec cette base de données, la complexité des transferts est réduite et la bande passante est optimisée.

3.3 Logiciel embarqué (Q4.5)

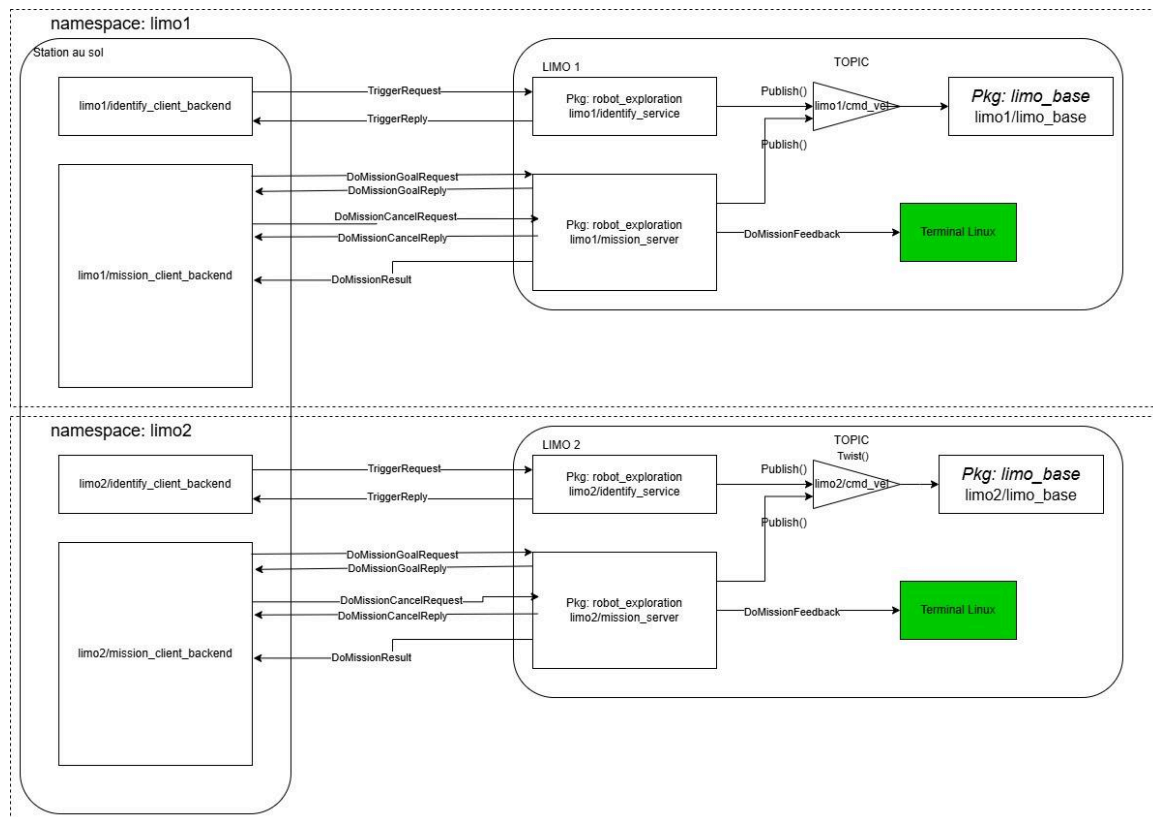


Figure 3: Diagramme du logiciel embarqué

Nous avons deux nodes principales par robot. Une est une node service qui s'occupe d'exécuter l'identification du robot. L'autre est une node action qui s'occupe d'initialiser et annuler la mission.

La node **mission_server** correspondant au R.F.2 est un serveur action qui utilise une interface DoMission pour la communication avec la station au sol. La node **mission_client_backend** est le client à la station au sol. DoMission est une interface qui décrit le type de message pour envoyer un Goal au serveur, retourner un résultat au client ainsi que retourner un feedback. Pour l'instant, DoMission a des valeurs qui ne servent qu'à tester la communication entre les nodes et la mission elle-même n'est qu'une simple manœuvre. La node **mission_server** reçoit un Goal à partir de la station au sol. Ceci l'active à publier des messages Twist sur le topic cmd_vel de façon infinie. **limo_base**, le subscriber de cmd_vel, transforme ces messages en un mouvement circulaire du robot. **mission_server** peut aussi recevoir une requête d'annulation du Goal à partir de la station au sol. Ceci cesse la loupe infinie de publication sur cmd_vel ce qui arrête le robot. La node est ensuite prête à recevoir une autre requête de démarrage de mission. Finalement, la node envoie aussi le nombre de publications sur cmd_vel qu'il a effectué jusqu'à présent au terminal du robot grâce à un feedback. Ceci est utile pour le débogage. Dans le futur, ce feedback contiendra des informations plus pertinentes et seront renvoyés à la station au sol.

DoMission se trouve dans le package limo_interfaces, sous le fichier DoMission.action. Pour l'instant, **DoMission** ne transmet aucune information importante d'un côté ou de l'autre. Il sert plus d'un signal programmable dans le futur. Le Goal envoie un temps de mission (int64 mission_length). Il décrit le nombre d'itérations d'envois de twist que fera la node serveur. La node serveur continuera à émettre de façon infinie si elle reçoit une longueur de mission 0. Le Result retourne un code (int64 result_code) et un message (string message) qui ne servent à rien pour l'instant. Le feedback émet un time_elapsed et percent_complete qui sont affichés au terminal du limo par la node serveur.

La node **identify_service** répondant au R.F.1 est un service qui est à l'écoute d'événements provenant de la node **identify_client_backend**. Ce service contient deux éléments principaux: un callback attendant une requête et retournant une réponse et une routine d'identification utilisant twist. De ce fait, lorsque **identify_client_backend** envoie un **TriggerRequest** (un objet vide), la node mission service publie sur le topic twist cmd_vel dont la node limo_base est un subscriber. Il envoie ensuite un **TriggerReply** à la node **identify_client_backend** afin de confirmer la réception de la requête. limo_base se sert des messages twist pour bouger le robot afin de l'identifier.

Nous donnons un namespace "limo1" ou "limo2" à toutes les nodes du robot correspondant. Les nodes de la station au sol sont aussi assignés un namespace soit "limo1" ou "limo2". Ceci permet à la station au sol d'envoyer des messages à l'un ou à l'autre. Nous avons une node par fonction et par robot sur la station au sol, par exemple une node qui permet de commencer/annuler une mission pour le limo1.

3.4 Simulation (Q4.5)

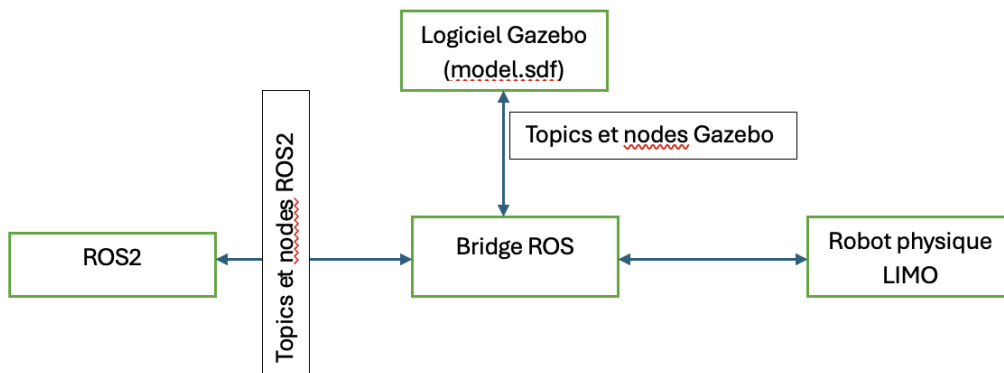


Figure 4: Architecture de la simulation

L'architecture de simulation que nous avons mise en place repose sur une interaction tripartite entre ROS2, le Bridge ROS et le logiciel Gazebo. Comme l'illustre la figure 4, ROS2 conserve son rôle central : il exécute les nœuds de haut niveau (contrôleurs, exploration, SLAM) qui communiquent via des topics standards. Ces topics sont ensuite relayés par le Bridge ROS, qui fait office de traducteur bidirectionnel entre l'écosystème ROS2 et le simulateur. Cette passerelle assure que les commandes et les données issues des capteurs virtuels de Gazebo sont accessibles sous le même format que celles provenant d'un robot physique.

Le logiciel Gazebo, décrit à travers des fichiers SDF (Simulation Description Format), prend en charge le réalisme de l'environnement et des modèles simulés. Dans notre cas, le modèle du robot AgileX Limo est représenté avec ses capteurs (LiDAR, caméra, IMU) et ses propriétés dynamiques (roues, inertie, collisions). Gazebo génère alors des données de capteurs virtuels et applique les lois physiques, ce qui permet de tester l'impact de l'environnement (obstacles, terrain, interactions entre robots) de manière réaliste.

Enfin, l'architecture intègre également le robot physique LIMO, connecté au Bridge ROS. Cela permet d'utiliser exactement le même schéma de communication que dans la simulation.

3.5 Interface utilisateur (Q4.6)

L'interface utilisateur commence par une page d'accueil où l'on choisit le mode simulation ou le mode réel. L'application sera faite pour être utilisée sur différents appareils (ordi, tablette par exemple) (R.F.10).

Après , cette étape, tout se passe dans un tableau de bord central. Cela rend l'application facile à utiliser (R.C.4) et permet de se concentrer sur l'objectif principal : lancer et arrêter des missions (R.F.2).

(Voici à quoi devrait ressembler le produit finale)

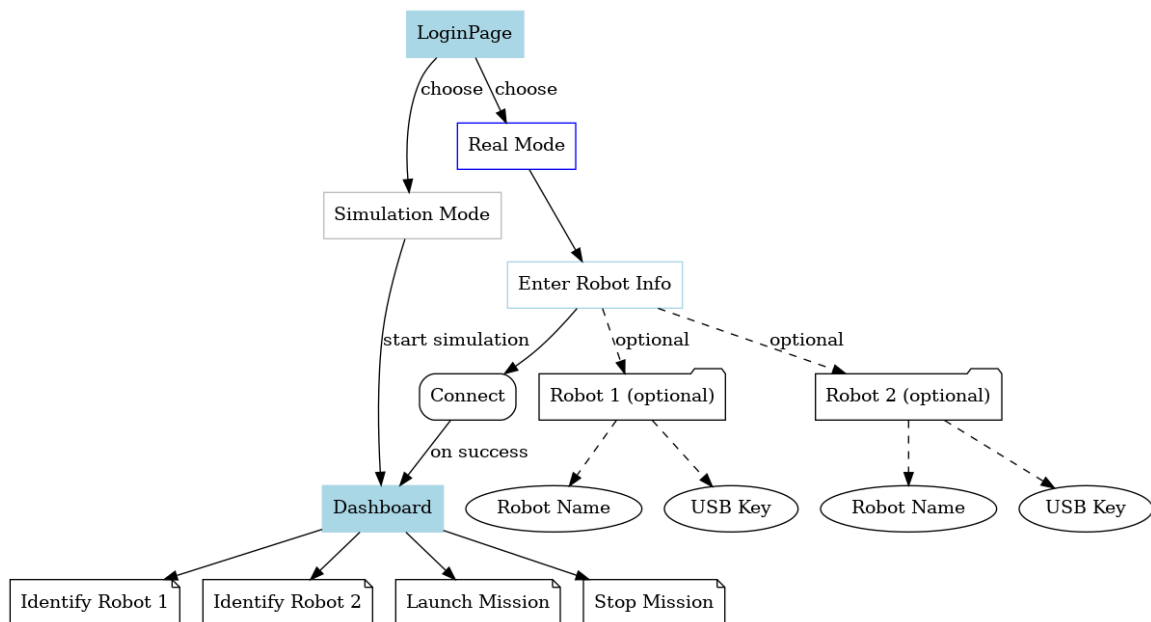
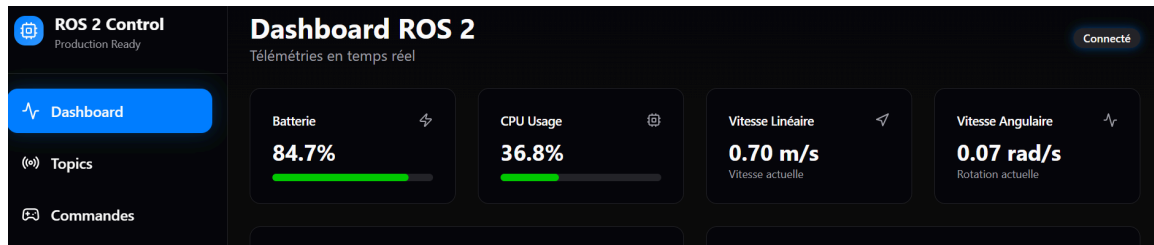


Figure 5: Diagramme de l'interface utilisateur prévisionnelle

Pour l'instant, le tableau de bord n'est pas encore complet. Il n'affiche pas encore les informations en temps réel sur les robots (R.F.3), mais il permet déjà d'utiliser les commandes essentielles : identifier un robot (R.F.1), lancer une mission et l'arrêter (R.F.2).

L'interface utilise Docker Compose pour simplifier l'installation et le lancement du système en une seule commande (R.C.2).

Pour la suite, on ajoutera plus de fonctions : voir l'état et la position des robots en direct (R.F.3, R.F.9), afficher une carte de l'environnement (R.F.8), consulter des journaux pour le suivi et le débogage (R.C.1) et assurer la qualité du code et des tests (R.Q.1, R.Q.2).

3.6 *Fonctionnement général (Q5.4)*

Étape 1: Station au sol

1. Cloner le repo git suivant:
<https://gitlab.com/polytechnique-montr-al/inf3995/20253/equipe-106/INF3995-106.git> (HTTP)
git@gitlab.com:polytechnique-montr-al/inf3995/20253/equipe-106/INF3995-106.git (SSH)
2. Ouvrir un terminal dans INF3995-106/project_ws
3. Faire les commandes:

```
colcon build  
source install/setup.sh  
cd ../server  
npm rebuild rclnodejs  
npm start
```

4. Ouvrir un terminal dans INF3995-106/client et faire la commande:

```
npm start
```

Étape 2: La simulation

Ceci est pour rouler en mode simulation. Ne pas faire pour rouler en mode physique. Voir étape 3 pour rouler en mode physique.

1. Ouvrir un terminal dans INF3995-106/project_ws
2. Faire les commandes suivantes:

```
colcon build  
source install/setup.sh  
ros2 launch robot_exploration robot_bringup.launch.py
```

3. Faire Ctrl-C pour terminer la simulation ou pour aller en mode physique

Les étapes vont être mises à jour une fois la fonctionnalité docker est implémentée.

Étape 3: Robot physique

Ceci est à faire pour faire fonctionner en mode physique.

1. Connecter par ssh sur les 2 robots
2. Cloner le repo / faire un git pull sur chaque robot
3. Mettre son terminal dans INF3995-106/project_ws sur chaque robot.
4. Pour le premier robot, faire les commandes suivantes:

```
sudo chmod 666 /dev/ttyTSH1
colcon build
source install/setup.sh
ros2 launch robot_exploration robot_bringup.launch.py use_limo:=true
```

5. Sur le second robot, faire les commandes suivantes:

```
sudo chmod 666 /dev/ttyTSH1
colcon build
source install/setup.sh
ros2 launch robot_exploration robot_bringup.launch.py use_limo:=true namespace:="limo2"
```

Les étapes vont être mises à jour une fois la fonctionnalité docker est implémentée.

Étape 4: Utiliser l'interface client

Aller sur le mode voulu et utiliser les boutons pour effectuer la commande voulue.

- Identifier le N^{ième} robot: fera avancer et reculer le robot sélectionné
- Commencer la mission: Fera les deux robots tourner en cercle jusqu'à indication d'arrêt
- Arrêter la mission: Fera arrêter de tourner les robots s'ils sont en mission.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

4.1.1 Hypothèses de calcul

L'estimation repose sur la limite imposée de **630 heures-personnes**, qui incluent les travaux pratiques hebdomadaires des étudiants (6 h/semaine × 11 semaines × 6 étudiants = 396 h). Ces heures de TP ne s'ajoutent pas au budget, mais font partie intégrante de la charge disponible pour mener à bien le projet.

Les taux horaires appliqués sont :

- **130 \$/h** pour les activités de développement et techniques,
- **145 \$/h** pour les activités de coordination assurées par le coordonnateur de projet (un des six membres de l'équipe).

4.1.2 Répartition des heures par lots de travail

La ventilation des 630 h tient compte des exigences de l'appel d'offres et des livrables attendus :

Tableau 1: Répartition d'heures de travail

Lot de travail	Heures totales (incluant TP)	Pourcentage
Développement embarqué & simulation	190 h	30 %
Développement application web	130 h	21 %
Intégration & conteneurisation Docker	80 h	13 %
Gestion de projet (coordonnateur)	65 h	10 %
Documentation & rapport technique	75 h	12 %
Réunions hebdomadaires & présentation finale	90 h	14 %
Total	630 h	100 %

Cette répartition démontre que les TP sont utilisés de manière productive pour soutenir les phases de tests, de validation et de préparation aux revues (PDR, CDR, RR).

4.1.3 Ventilation budgétaire

En appliquant les taux horaires différenciés, on obtient le budget suivant :

Tableau 2: Répartition budgétaire du projet

Lot de travail	Heures facturées	Taux appliqué	Coût
Développement embarqué & simulation	190 h	130 \$/h	24 700 \$
Développement application web	130 h	130 \$/h	16 900 \$
Intégration & conteneurisation Docker	80 h	130 \$/h	10 400 \$
Gestion de projet (coordonnateur)	65 h	145 \$/h	9 425 \$
Documentation & rapport technique	75 h	130 \$/h	9 750 \$
Réunions & présentation finale	90 h	130 \$/h	11 700 \$
Total global	630 h	—	82 875 \$

4.1.4 Analyse et justification

Le coût global du projet est estimé à **82 875 \$**. Cette estimation respecte strictement la contrainte des 630 heures-personnes et intègre l'ensemble des activités nécessaires : développement, intégration, gestion, documentation et communication avec l'Agence.

La présence de 396 h de travaux pratiques incluses dans les 630 h permet d'assurer une capacité de travail régulière et encadrée, garantissant ainsi la faisabilité du projet dans l'échéancier prévu. La ventilation équilibrée des efforts entre développement, gestion et documentation offre une visibilité claire sur la progression attendue et assure la qualité des livrables.

4.2 Planification des tâches (Q11.2)

La planification des tâches repose sur une répartition équilibrée du travail entre les membres de l'équipe et sur l'échéancier fixé par l'appel d'offres. Nous avons structuré le projet autour de trois jalons principaux : le PDR (29 septembre 2025), le CDR (11 novembre 2025) et le RR (4 décembre 2025).

Chaque jalon est associé à un ensemble de requis obligatoires, auxquels s'ajoutent certains requis facultatifs permettant de bonifier la note. Afin d'assurer une progression continue, les tâches ont été découpées en lots de travail (développement embarqué et simulation, développement web, intégration Docker, documentation et gestion de projet), chacun étant attribué à un ou plusieurs membres en fonction de leurs expertises et de la difficulté de la tâche. Pour l'instant cette répartition d'équipe n'a pas encore été décidée étant donnée la jeunesse de l'équipe ainsi que la versatilité de l'ensemble des membres. C'est un sujet que nous détaillerons par contre pour la suite du projet.

Pour chaque requis, nous avons défini une période de réalisation (début et fin), un nombre de personnes assignées, ainsi qu'un taux de progression estimé en pourcentage permettant de suivre l'avancement de manière objective. Les requis critiques sont pris en charge par au moins deux membres, tandis que les requis plus légers ou de support peuvent être confiés à une seule personne. Enfin, les requis de qualité (tests, conventions de code) concernent l'ensemble de l'équipe afin d'assurer la robustesse et la cohérence du système.

Le diagramme de Gantt ci-dessous illustre cette planification : il montre l'allocation du temps pour chaque tâche, la position des trois jalons principaux dans le calendrier, la répartition des ressources humaines mobilisées, ainsi que la progression estimée de chaque tâche.

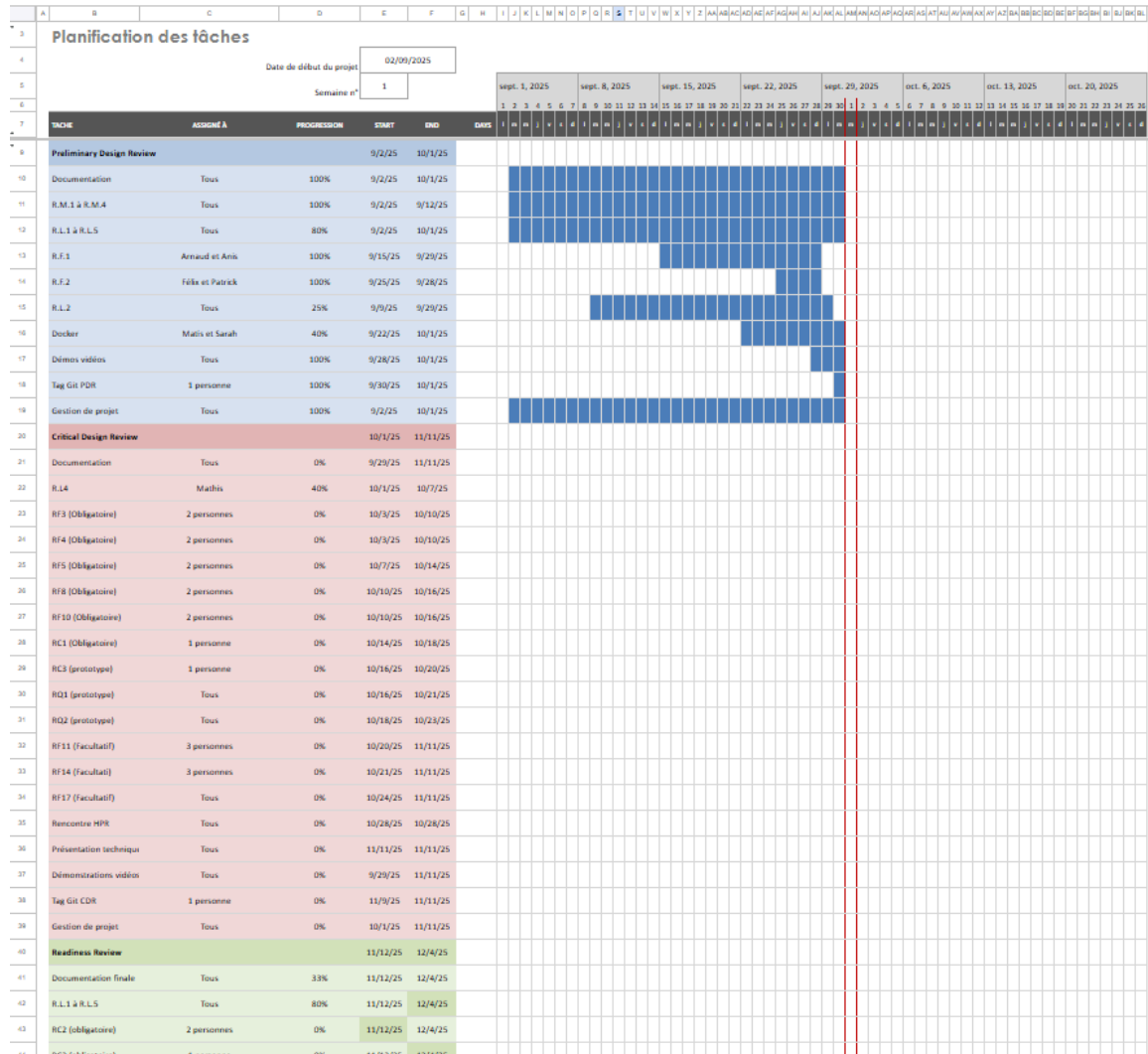


Figure 6 : Diagramme de Gantt du projet

4.3 Calendrier de projet (Q11.2)

Le calendrier du projet est structuré autour de trois jalons principaux : le PDR, le CDR et le RR. Chaque jalon est associé à des requis obligatoires à livrer et, dans certains cas, à des fonctionnalités prototypes ou facultatives permettant de bonifier la note. Le tableau ci-dessous résume les dates cibles et les requis associés.

Tableau 3: Calendrier de projet

Dates cible	Jalon
29 septembre 2025 (PDR)	<u>Obligatoire (7 points):</u> <ul style="list-style-type: none"> - R.F.1 - 3 points - R.F.2 - 4 points
11 novembre 2025 (CDR)	<u>Obligatoire (33 points):</u> <ul style="list-style-type: none"> - R.F.3 - 2 points - R.F.4 - 4 points - R.F.5 - 8 points - R.F.8 (prototype) - 10 points - R.F.10 - 4 points - R.C.1 - 5 points Prototype: <ul style="list-style-type: none"> - RC3 - RQ1 - RQ2 <u>Facultatif (15 points)</u> <ul style="list-style-type: none"> - RF11 - 5 points - RF14 - 5 points - RF17 - 5 points
4 décembre 2025 (RR)	<u>Obligatoire (35 points):</u> <ul style="list-style-type: none"> - RC2 - 4 points - RC3 - 1 point - RC4 - 5 points - RQ1 - 5 points - RQ2 - 5 points - R.F.6 - 10 points - R.F.7 - 2 points - R.F.9 - 3 points <u>Facultatif (11 points)</u> <ul style="list-style-type: none"> - RF16 - 5 points - RF19 - 6 points

Ce calendrier assure une progression incrémentale vers un système complet, en passant d'un prototype fonctionnel minimal (PDR), à une intégration des fonctionnalités principales (CDR), jusqu'à la version robuste et finale (RR). Nous définirons par la suite

4.4 Ressources humaines du projet (Q11.2)

L'équipe projet est composée de six membres aux compétences complémentaires, permettant de couvrir l'ensemble des volets techniques et organisationnels. La coordination générale est assurée par Sarah Askas, qui veille au suivi des tâches, à l'intégration sous GitLab et à la cohérence globale du projet. Les volets développement web et serveur sont pris en charge par Anis Menouar et Matis Roux, responsables respectivement du protocole de communication entre front-end et back-end ainsi que de la mise en place de l'interface utilisateur et de la conteneurisation. Le développement embarqué et simulation est assuré par Felix Paillé Dowell, Arnaud Grandisson et Patrick Nzudom, dont les expertises combinées en ROS2, Gazebo, MongoDB et protocoles de communication garantissent l'intégration, la validation et le bon fonctionnement du système sur robot réel et en simulation. Cette répartition claire des rôles assure une progression efficace et une couverture complète des requis du projet.

Tableau 4: Allocation de rôles des membres

Membre de l'équipe	Rôle	Qualifications
Sarah Askas	Coordination du projet	Supervision générale, ROS2 et gestion de projet sur gitlab
Anis Menouar	Développeur Full Stack	Serveur et protocole de communication entre front end et back end. Gazebo
Matis Roux	Développeur Web	Développement web, conteneurisation Docker, ROS2
Felix Paillé Dowell	Développeur Embarqué	Intégration du système, ROS2, Gazebo et protocole de communication
Arnaud Grandisson	Développeur Embaqué	Test de validation, MongoDB, ROS2, Gazebo et protocole de communication
Patrick Nzudom	Développeur Embarqué	MongoDB, ROS2, Gazebo et protocole de communication

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Tout au long du projet, nous voulons assurer la qualité de nos livrables grâce à un processus de révision systématique. Étant donné que ce projet contient de nombreux composants technologiques, il est crucial d'établir une base solide dès le départ et de maintenir des standards élevés à chaque étape.

- **Pipeline de développement/Intégration continue:** C'est un premier processus de contrôle notamment utilisé dans le cours de LOG2990 où chaque étape d'intégration et de développement. En fait, une configuration sur GitLab permet de lancer un pipeline de validation sur le projet complet en 3 étapes; install, lint et test. Ce pipeline est lancé lors d'une demande d'intégration (*Merge Request*) ou lors d'un commit sur la branche principale. Ainsi toutes les nouvelles fonctionnalités devront suivre un standard d'intégration en continu.
- **Revue de code:** Chaque ajout de fonctionnalité devra passer par une merge request avec une description complète à l'intérieur de la MR. Notamment la raison du MR, la description de la nouvelle fonctionnalité, les fichiers et dossiers ayant été modifiés et les tests faits pour cette nouvelle fonctionnalité. De plus, avant de faire accepter cette demande d'intégration dans la branche principale un minimum de deux autres personnes de l'équipe devront analyser cette demande pour ensuite qu'elle soit acceptée.
- **Standard de programmation et qualité de code:** Différents standards de programmation comme le nom des fonctions, la longueur des fichiers, le nom des commits, le nom des branches et la taille des MR sont des connaissances déjà acquises par tous les membres et qui devront être à nouveau utilisées dans ce cours projet.
- **Tests d'intégration:** Des tests automatisés et manuels permettront de vérifier que les changements dans un module (par ex. UI Angular) n'ont pas d'effet indésirable sur les autres (par ex. backend ou nœuds ROS2).
- **Tests de déploiement :** Évidemment il serait important de tester notre implémentation dans des conditions réelles, c'est-à-dire sur le robot physique et non pas seulement sur le logiciel de simulation gazebo par exemple.

Si on fait une analyse plus globale sur le contrôle de la qualité au niveau des trois livrables majeurs de la session — le PDR (développement du prototype), le CDR (développement de la majorité du fonctionnement du système) et le RR (développement des dernières caractéristiques du projet) — on remarque que chacun joue un rôle stratégique dans l'assurance qualité. Au PDR, l'accent est mis sur la mise en place d'une base technique solide et stable : la qualité se mesure surtout par la capacité de l'équipe à livrer un prototype fonctionnel minimal, démontrant la chaîne complète de communication (UI, backend, ROS2 et robot). Pour le CDR, l'objectif est de valider l'intégration des différents modules et la complétion de la majorité des requis fonctionnels ; la qualité se traduit alors par la robustesse des interactions entre les composantes (station au sol, robots, simulation) et par la couverture des tests unitaires et d'intégration. Enfin, pour le RR, le contrôle de la qualité vise à garantir un produit final robuste, complet et agréable à utiliser : cela passe par l'ajout des dernières fonctionnalités, l'optimisation de la performance, la réalisation de tests d'acceptation complets et l'exécution de scénarios de validation réalistes avec les robots physiques. En

somme, le contrôle de la qualité évolue d'un souci de fondation technique au PDR vers une approche d'intégration au CDR, pour culminer dans une validation finale de la robustesse et de l'expérience utilisateur au RR.

5.2 Gestion de risque (Q11.3)

- **Endommagement du robot Limo**

Comme dans tout projet impliquant du matériel physique, l'un des principaux risques concerne l'endommagement des robots en soi. Une utilisation inadéquate de celle-ci peut facilement endommager différentes composantes physiques. Pour réduire ce risque, les premières expérimentations se feront à basse vitesse avec des mouvements délicat. De plus, une grande partie des tests se feront avec le simulateur Gazebo. Ainsi, en théorie, on ne devrait pas avoir de comportements inconnues

- **Problème de communication réseau:**

La communication réseau avec les robots peut avoir certains risques dans le sens où les contraintes pour permettre la communication entre PC et robot peuvent ne pas être respectées et donc avoir une mauvaise communication. Au niveau des contraintes, on parle de l'utilisation du même réseau WIFI que le robot et d'une configuration réseau contrôlée avec un *Domain ID*. Ce "*Domain ID* permet à une équipe d'isoler le robot qu'ils utilisent dans un sous réseau ce qui évite les interférences entre équipes qui utilisent eux aussi ROS 2 sur le même réseau.

- **Problèmes d'intégration logicielle:**

D'autres risques concernent l'intégration logicielle. Le projet étant divisé en plusieurs modules (interface Angular, backend FastAPI, ROS 2 embarqué sur les Limo, simulation), un mauvais couplage ou des divergences entre ces modules pourraient ralentir le développement. Pour limiter ce risque, le travail sera découpé en tâches incrémentales avec des tests réguliers à chaque étape.

- **Manque de cohésion ou retards de livraison**

Comme dans tout projet d'équipe avec plusieurs membres, il existe des risques où la cohésion n'est pas présente et donc peut affecter le déroulement du projet et les livraisons des parties de celui-ci. Des réunions de suivi hebdomadaire, de la documentation sur GitLab et l'utilisation d'un canal de communication sont toutes des techniques qui peuvent venir renforcer la cohésion et faciliter le travail.

- **Manque d'expérience avec ROS 2/Docker**

Lorsque de nouvelles notions sont utilisés dans un projet comme ROS2 et Docker, il est important de faire les tutoriels nécessaires de ROS 2 et Docker, utiliser la documentation disponible sur le moodle et le Discord du cours pour changer ce manque d'expérience à un gain d'expérience sur ces sujets

5.3 Tests (Q4.4)

Conformément aux exigences du R.Q.2, chaque fonctionnalité doit être accompagnée de tests unitaires ou de procédures de validation. Des tests spécifiques seront définis pour chacun des sous-systèmes.

Pour la partie web, les fonctionnalités seront vérifiées à l'aide de tests unitaires. Par exemple, la station au sol sera évaluée en envoyant des messages prédéfinis tels que « lancer la mission », puis en observant les réponses du système à travers les logs générés. L'interface utilisateur fera également l'objet de tests en étant utilisée sur différents navigateurs, afin d'évaluer sa compatibilité et sa robustesse lorsqu'elle est sollicitée par plusieurs tâches simultanées.

Du côté du sous-système embarqué, l'approche retenue sera la mise en place de procédures de test complètes. Celles-ci permettront de valider le comportement du système de bout en bout. Elles seront décrites dans un document intitulé « Tests.pdf ». Un exemple de procédure consistera à exécuter une mission complète — du lancement au retour à la base — et à comparer les artefacts produits avec ceux d'une simulation équivalente réalisée sur Gazebo. Cette comparaison permettra de confirmer que l'ensemble des composants interagit conformément aux attentes.

5.4 Gestion de configuration (Q4)

La gestion de configuration du projet repose sur l'utilisation de Git pour le contrôle de version et la coordination entre les membres de l'équipe. Le dépôt est structuré en plusieurs répertoires spécialisés, facilitant l'organisation et la traçabilité du code source :

- **Application :**
 - *Client* : le code Angular est situé dans `application/client/src`, organisé par modules (`app`, `assets`, `environments`) afin d'assurer modularité et réutilisabilité.
 - *Serveur* : le code backend (FastAPI ou NestJS selon le module) est placé dans `application/server`, avec des sous-répertoires distincts pour les contrôleurs, services et ressources statiques.
 - *Commun* : contient les interfaces et énumérations partagées, assurant la cohérence des types entre client et serveur.
- **Docker** : ce dossier regroupe les fichiers de conteneurisation, incluant les différents `Dockerfile` et un fichier `docker-compose.yaml` qui orchestre le lancement coordonné des services (client, serveur, simulation Gazebo).

- **Embarqué** : comprend le code ROS2 exécuté sur les robots Limo, ainsi que les messages personnalisés nécessaires à la communication entre les nœuds.
- **Simulation (Gazebo)** : regroupe les fichiers model.sdf, les configurations de monde virtuel, ainsi que le workspace ROS2 (project_ws) utilisé pour tester les fonctionnalités en environnement simulé.
- **Documentation** : centralise les rapports hebdomadaires, les gabarits de communication et le fichier ReadMe.md décrivant les instructions de déploiement et de configuration.

Concernant les tests, ceux-ci sont regroupés au sein de chaque répertoire source (par exemple *.spec.ts pour Angular), afin que chaque module dispose de ses propres scénarios de validation.

Enfin, la conteneurisation par Docker garantit la portabilité et la reproductibilité des environnements, réduisant les problèmes liés aux différences de configuration entre développeurs ou machines.

5.5 Déroutement du projet (Q2.5)

[CDR et RR seulement]

[Dans votre équipe, qu'est-ce qui a été bien et moins bien réussi durant le déroulement du projet par rapport à ce qui était prévu dans l'appel d'offre initialement.]

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

[CDR et RR seulement]

[Qu'est-ce qui fonctionne et qu'est-ce qui ne fonctionne pas.]

7. Travaux futurs et recommandations (Q3.5)

[RR seulement]

[Qu'est-ce qui reste à compléter sur votre système? Recommendations et possibles extensions du système.]

8. Apprentissage continu (Q12)

[RR seulement]

[Un paragraphe par membre (identifié en début de paragraphe) de l'équipe qui doit aborder chacun de ces aspects de façon personnelle:

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

2. *Méthodes prises pour y remédier.*
3. *Identifier comment cet aspect aurait pu être amélioré.]*

9. Conclusion (Q3.6)

[RR seulement]

[Par rapport aux hypothèses et à la vision que vous aviez du système lors du dépôt de la réponse à l'appel d'offre, que concluez-vous de votre démarche de conception maintenant que le système est complété?]

10. Références (Q3.2)

Ros 2 documentation. ROS 2 Documentation - ROS 2 Documentation: Foxy documentation. (accessed 01/09/2025). <https://docs.ros.org/en/foxy/index.html>

Agilexrobotics. (accessed 02/09/2025). Agilexrobotics/limo_ros2: Limo ros2 packages. GitHub. https://github.com/agilexrobotics/limo_ros2

Creating ROS 2 actions. (accessed 02/09/2025). Foxglove - Visualization and observability for robotics developers. <https://foxglove.dev/blog/creating-ros2-actions>

Docker Documentation. (accessed 12/09/2025) <https://docs.docker.com/>

ChatGPT. (accessed 05/09/2025) <https://chatgpt.com/>

ANNEXES

10 usability heuristics for user interface design. (1994, April 24). Nielsen Norman Group. <https://www.nngroup.com/articles/ten-usability-heuristics/>

GNOME Developer. <https://developer.gnome.org/>

What is Agile project management? - Complete guide | Freshservice. (2025, May 1). Freshworks. <https://www.freshworks.com/agile-project-management/>

Design patterns. (accessed 20/09/2025). Refactoring and Design Patterns. <https://refactoring.guru/design-patterns>