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

11 Novembre 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)

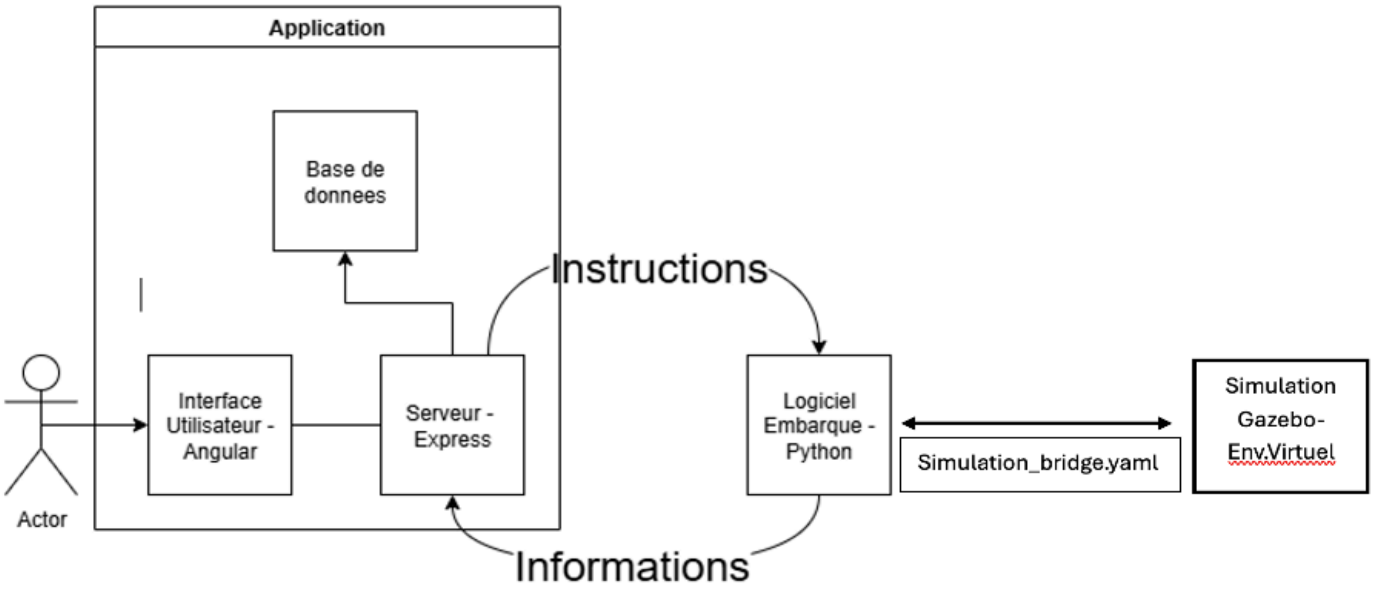


Figure 1: Architecture du projet

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.

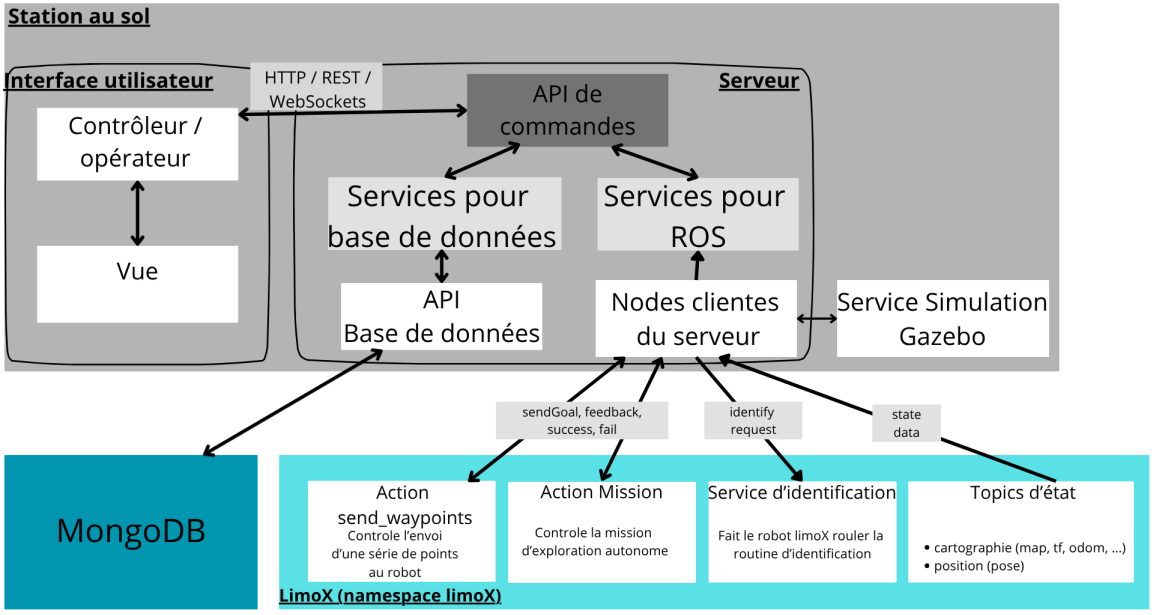


Figure 2: Architecture de station au sol

L'architecture de la station au sol a été conçue de manière à séparer clairement les responsabilités entre l'interface utilisateur, le serveur d'orchestration et le réseau robotique ROS2. Cette conception répond à deux exigences principales : assurer une communication fiable et bidirectionnelle avec les robots tout en optimisant la charge réseau et la modularité du système. Du côté de l'interface utilisateur, le *contrôleur/opérateur* représente la couche logique qui interprète les actions de l'utilisateur et les traduit en requêtes destinées au serveur. La vue correspond à la partie graphique de l'application Angular qui affiche en temps réel l'état des robots, les cartes explorées et les boutons de commande. Cette séparation entre la vue et la logique de contrôle suit une approche de type MVC (Model-View-Controller), garantissant la réactivité et la maintenabilité du système, deux requis essentiels pour une station de supervision robotique.

Le serveur constitue le cœur de la communication entre l'interface et les robots. Il reçoit les requêtes de l'utilisateur via une API de commandes exposée en HTTP ou WebSocket, selon la nature de la requête. Ce serveur ne fait pas tourner de nœuds ROS lourds — tels que la navigation ou le SLAM — afin de limiter la complexité, d'éviter les conflits de synchronisation et de réduire la charge de calcul. À la place, il exécute des **nodes clientes**, qui permettent d'interagir à distance avec le réseau ROS2 des robots à travers le protocole DDS (Data Distribution Service). Cette décision est justifiée par le besoin d'isoler les fonctions critiques de bas niveau (localisation, navigation, exploration) du traitement applicatif du serveur. Ainsi, le serveur conserve uniquement un rôle de coordination et d'orchestration, tandis que le robot reste responsable de l'exécution des calculs temps réel.

La communication entre la station au sol et le robot se fait par l'intermédiaire du réseau ROS2 en utilisant une combinaison de **topics**, **services** et **actions**, chacun étant choisi selon la nature de la donnée échangée. Les **topics** sont employés pour transmettre en continu les informations d'état telles que /map et /pose, car ils permettent un flux de données constant et asynchrone, parfaitement adapté à la surveillance temps réel. Le service d'identification est utilisé pour faire une requête d'un protocole court et immédiat. Ceci permet de garder la simplicité, car nous n'avons pas besoin de feedback ni d' option d'annuler une identification. Enfin, les **actions** (mission, send_waypoints) sont privilégiées pour les commandes longues ou supervisées, telles que le lancement d'une mission d'exploration ou la navigation par une série de points, car elles offrent une boucle complète de communication (envoi du goal, suivi du feedback, et réception du résultat final). Ce choix combiné permet de tirer parti des forces de chaque mécanisme de communication ROS2, assurant un équilibre entre rapidité, fiabilité et retour d'information.

L'optimisation de la communication constitue également un aspect central de la conception. La station au sol et le robot partagent un même ROS_DOMAIN_ID, ce qui permet la découverte automatique des nœuds sur le réseau DDS sans avoir à recourir à des configurations manuelles complexes. De plus, la qualité de service (QoS) est adaptée selon le type de donnée : les flux à haute fréquence comme les scans laser peuvent être limités ou filtrés lorsque le robot n'est pas en mission, afin de réduire la bande passante utilisée. Par ailleurs, les échanges entre le serveur et la base de données sont optimisés grâce à MongoDB, dont la structure orientée documents JSON permet de stocker efficacement des données hétérogènes telles que les cartes, les journaux de mission ou les états des robots. Ce choix est justifié par le faible volume de données à gérer, la nécessité de flexibilité et la rapidité d'accès en lecture/écriture. En encapsulant la logique de persistance dans des services dédiés, la base de données reste découplée du reste du système, ce qui favorise la modularité et simplifie la maintenance.

En somme, cette architecture répond à la fois aux exigences de modularité, de performance et de fiabilité. Le serveur agit comme un coordinateur léger et

distant, communiquant avec les robots via DDS à l'aide de mécanismes ROS2 appropriés selon la nature des échanges : les topics pour les données continues, les services pour les requêtes ponctuelles, et les actions pour les missions supervisées. Ce découplage des responsabilités, combiné à l'utilisation d'une base de données flexible et d'une configuration réseau optimisée, permet de réduire la charge de communication, d'améliorer la réactivité du système et d'assurer une supervision efficace et évolutive des robots.

3.3 Logiciel embarqué (Q4.5)

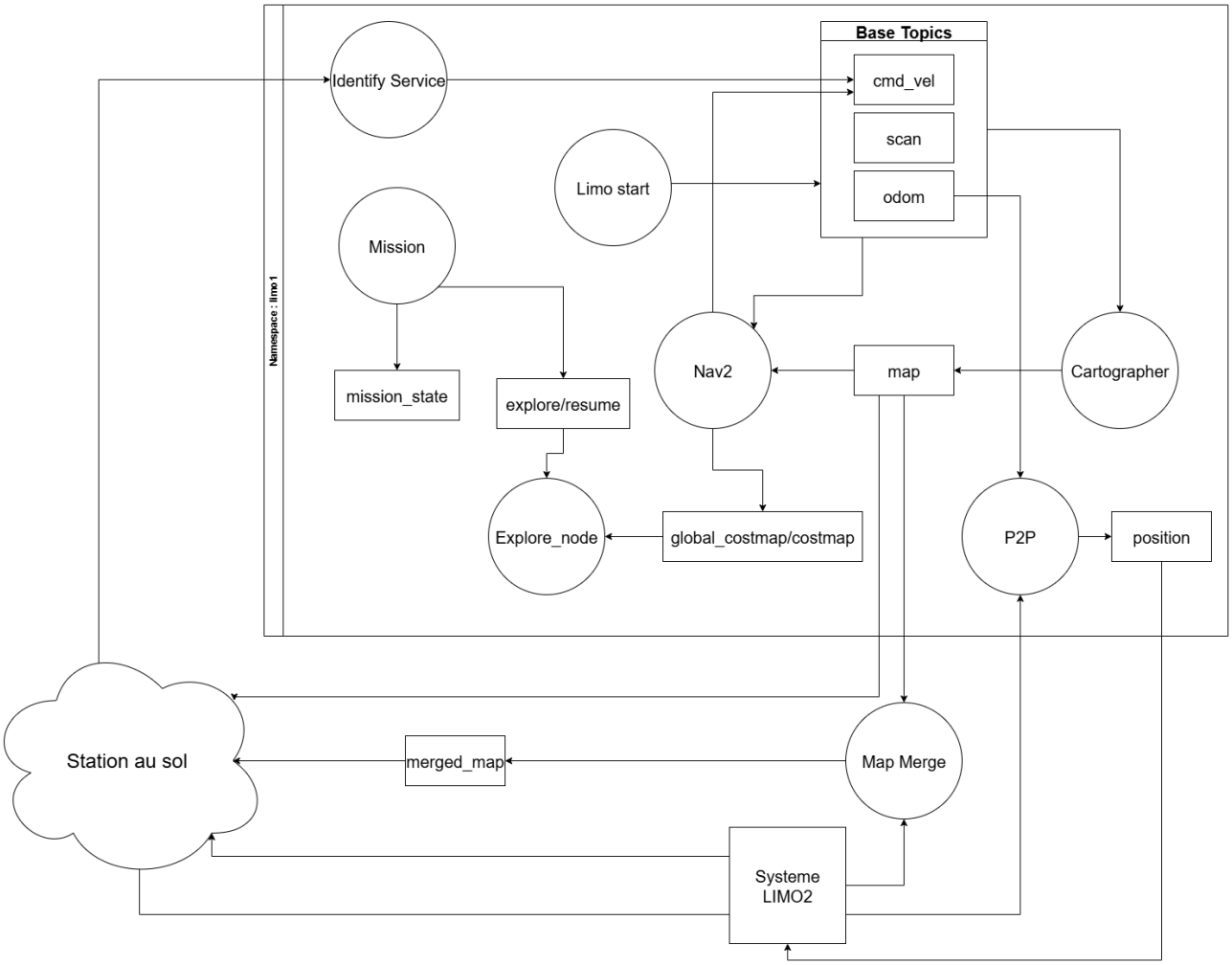


Figure 3: Diagramme du logiciel embarqué

Dans le cadre de notre projet d'exploration autonome multi-robots, nous avons conçu une architecture embarquée modulaire et hiérarchisée afin de rendre le système à la fois fiable, extensible et simple à maintenir.
Chaque robot est isolé dans un namespace spécifique (par exemple limo1 ou limo2), ce qui évite toute collision entre les topics, services et frames lorsqu'on fait fonctionner plusieurs robots en parallèle. Ce découpage nous permet aussi de dupliquer facilement la configuration et d'avoir une supervision claire depuis la station au sol.

Tout part du nœud Limo Start, qui initialise la base du robot et gère les topics essentiels : /cmd_vel pour les commandes de vitesse, /scan pour les mesures du LiDAR et /odom pour l'odométrie.
Cette couche est volontairement indépendante du reste du système afin de pouvoir changer de matériel ou de capteur sans avoir à modifier la logique logicielle. Les données produites par cette couche alimentent ensuite les modules de cartographie, de navigation et d'exploration. Ce choix nous a permis de garder une structure claire et adaptable tout au long du développement.

Pour la partie cartographie, nous avons choisi d'utiliser Cartographer. Ce choix s'explique principalement par le fait qu'il applique un filtre EKF nativement sur les données d'odométrie, ce qui améliore considérablement la précision de la carte et réduit le drift sur les longues distances. Dans la pratique, nous avons observé une meilleure stabilité de la localisation et une cartographie beaucoup plus cohérente, surtout dans des environnements partiellement fermés.

Les cartes locales générées par chaque robot sont ensuite fusionnées grâce au module Map Merge. Contrairement à d'autres approches plus complexes, ce module n'a pas besoin d'un système de communication P2P : il se base simplement sur les topics de carte namespacés (/limo1/map, /limo2/map) et sur les frames de chaque robot pour aligner et combiner les cartes. Nous avons choisi ce module avant tout pour sa simplicité d'utilisation et sa compatibilité directe avec ROS2.

Pour éviter les problèmes de synchronisation, le Map Merge est lancé uniquement sur le robot Limo 1, qui agit comme coordonnateur principal. Il reçoit les cartes des autres robots et publie une merged_map globale que nous utilisons ensuite sur la station au sol.

La navigation est assurée par le framework Nav2, qui prend en charge la planification globale et locale, la détection d'obstacles et la gestion des comportements de récupération.

Nous avons préféré Nav2 à une solution développée à la main, car il est déjà bien intégré à ROS2 et offre une architecture modulaire qui s'adapte à plusieurs types de robots. Il se base sur la carte de Cartographer et sur la costmap, ce qui permet de calculer des trajectoires précises et sécurisées.

Cette brique représente le cœur décisionnel du déplacement du robot : elle assure que chaque mission se déroule de manière fluide, même dans des environnements complexes ou partiellement inconnus.

L'exploration est réalisée par notre nœud Explore_node, qui sélectionne automatiquement les points à visiter dans l'environnement.

Une des décisions importantes que nous avons prises est de baser cette exploration non pas sur la carte statique /map, mais sur la global_costmap issue de Nav2. En effet, la costmap est mise à jour en temps réel avec les obstacles détectés par les capteurs, ce qui la rend plus fiable et plus représentative de la

réalité du terrain. Cela permet au robot d'éviter les zones bloquées ou déjà explorées, et de prioriser des régions accessibles et pertinentes pour la mission.

Cette approche a rendu le comportement du robot beaucoup plus cohérent pendant les tests physiques, notamment lors de transitions entre zones étroites et grands espaces.

L'ensemble du système est coordonné par le nœud Mission, qui gère le lancement, la pause et l'arrêt des explorations. Il publie également un état global (mission_state) permettant de suivre la progression des robots en temps réel.

Le topic de mission joue un rôle central dans l'équilibre du système : il évite que plusieurs commandes (par exemple une commande manuelle et une commande automatique) se chevauchent et provoquent des comportements incohérents. Grâce à ce mécanisme, la mission reste fluide et chaque robot agit de manière contrôlée et prévisible.

Nous avons aussi intégré un Identify Service, qui permet à chaque robot de s'annoncer dans le réseau au démarrage. Cette étape simplifie la découverte automatique et la coordination dans la flotte, surtout lorsque plusieurs robots rejoignent ou quittent la mission en cours.

La station au sol assure la supervision et la visualisation du système. Elle reçoit la carte fusionnée (merged_map), les positions des robots et leur état de mission. Cette interface nous permet de suivre en temps réel l'évolution de la mission et d'intervenir si nécessaire.

Nous avons choisi de centraliser la fusion et la supervision sur un seul robot (Limo 1) afin de limiter la charge sur les systèmes embarqués et d'assurer une synchronisation plus stable des données.

Même si notre architecture actuelle ne dépend pas d'un lien direct entre les robots, nous avons réfléchi à la manière dont un module P2P (peer-to-peer) pourrait être intégré dans une version plus avancée du système.

L'idée serait de permettre à chaque robot d'échanger directement, sans passer par la station au sol, des informations de position, de progression ou d'exploration. Par exemple, un robot pourrait envoyer périodiquement sa position aux autres via un canal réseau (DDS inter-domaines ou socket UDP). Les robots pourraient alors adapter leur comportement en conséquence, évitant les redondances et améliorant la couverture collective.

Ce type de communication rendrait la flotte plus autonome et collaborative, capable de continuer la mission même en cas de coupure de connexion avec la station. C'est une piste que nous souhaitons explorer à l'avenir pour rendre notre système encore plus intelligent et résilient.

3.4 Simulation (Q4.5)

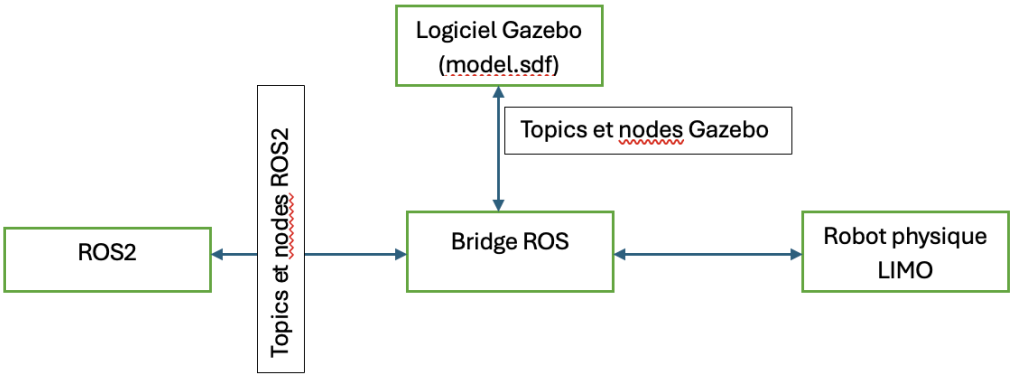


Figure 4: Architecture de la simulation

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 la même architecture de noeuds et de topic que celle du logiciel embarqué (cf. 3.3). Ce dernier est connecté au Bridge ROS, afin d'utiliser exactement le même schéma de communication en simulation que dans les conditions réelles.

3.5 Interface utilisateur (Q4.6)

Au démarrage de l'application, l'utilisateur arrive sur une page d'accueil où il peut saisir le nom de la mission et choisir le mode d'exécution, en simulation ou en mode réel. Une fois cette étape validée, la mission est créée et l'utilisateur est redirigé vers le tableau de bord principal.

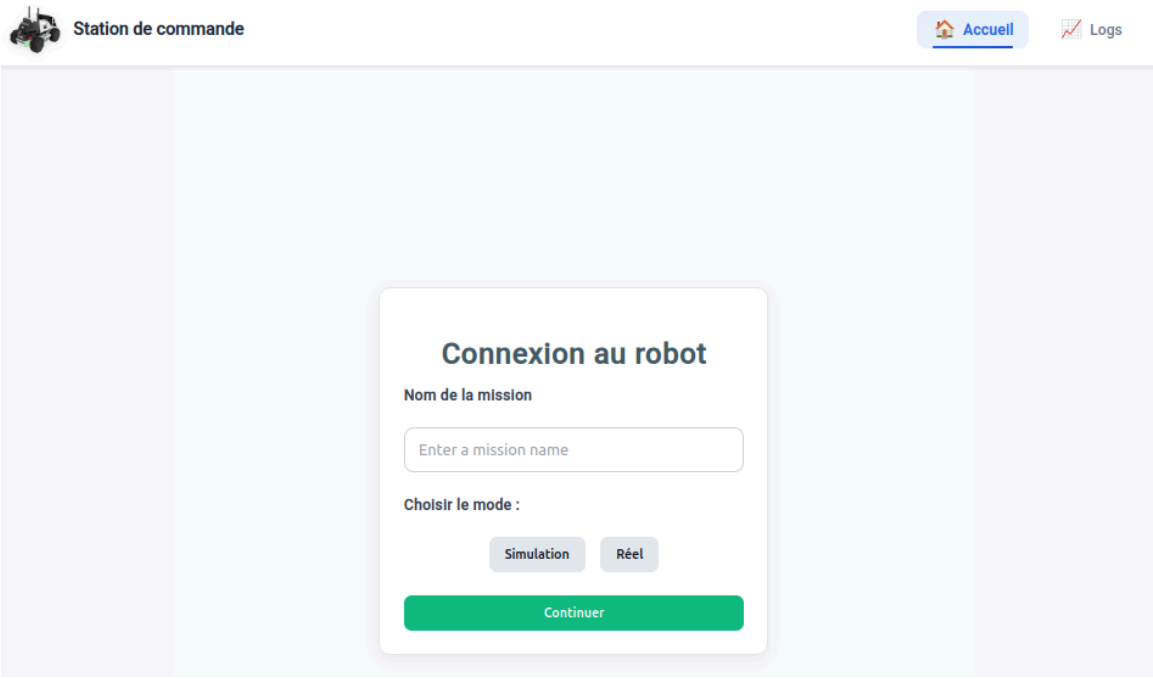


Figure 5.1 : Page de connection

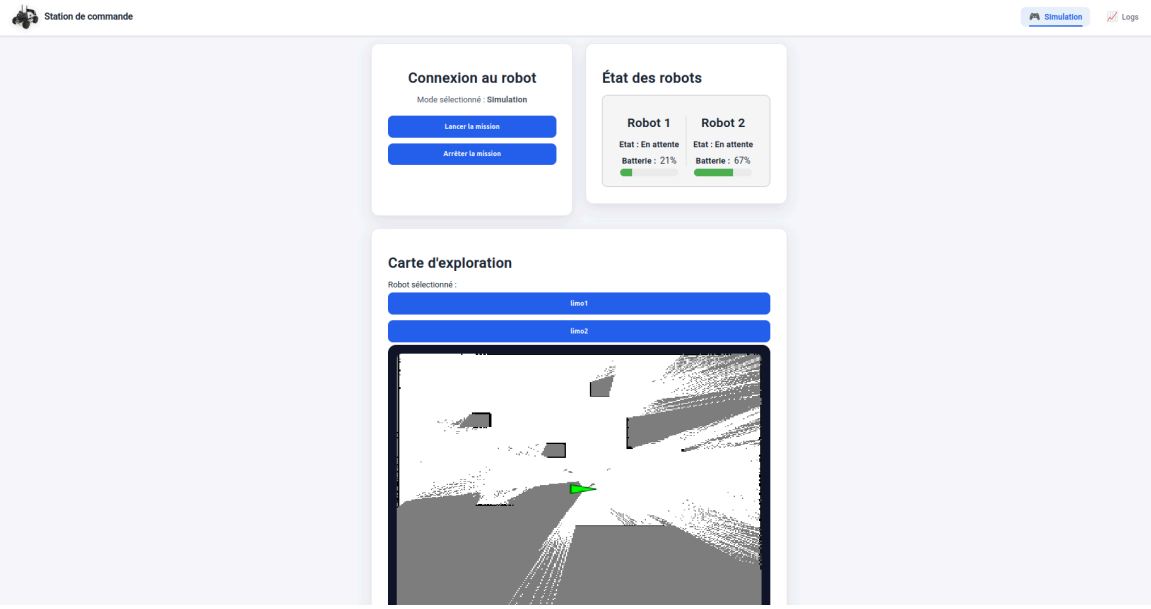
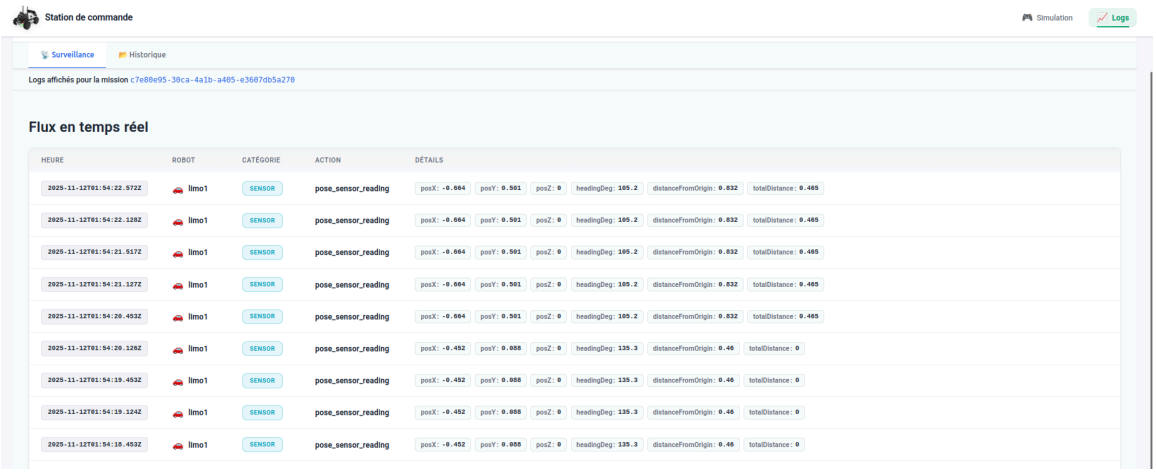


Figure 5.2 : Vue Tableau de Bord (mode sim)

Ce tableau de bord constitue le cœur de l'application. Il permet notamment de lancer et d'arrêter une mission, conformément au requis R.F.2, et d'afficher les informations propres au mode sélectionné (par exemple il a un bouton identifier robot 1 et identifier robot 2 Pour le mode réel), tel qu'exigé par R.L.2. Depuis ce tableau de bord, on peut consulter l'historique des missions ainsi que les logs en temps réel dans un onglet dédié, ce qui garantit le respect du requis R.C.1 portant sur la disponibilité et l'accessibilité des journaux de



débogage.

Figure 5.3 : Affichage des logs en temps réel

L'application est aussi conçue pour être utilisée sur différents types d'appareils (ordinateur, tablette, téléphone) et s'adapte automatiquement à leur format d'affichage, conformément au requis R.F.10. Ce design centré sur un tableau de bord unique assure une expérience simple, lisible et intuitive, en accord avec les attentes de convivialité définies dans R.C.4.

(Voici à quoi ressemble l'architecture de notre site web)

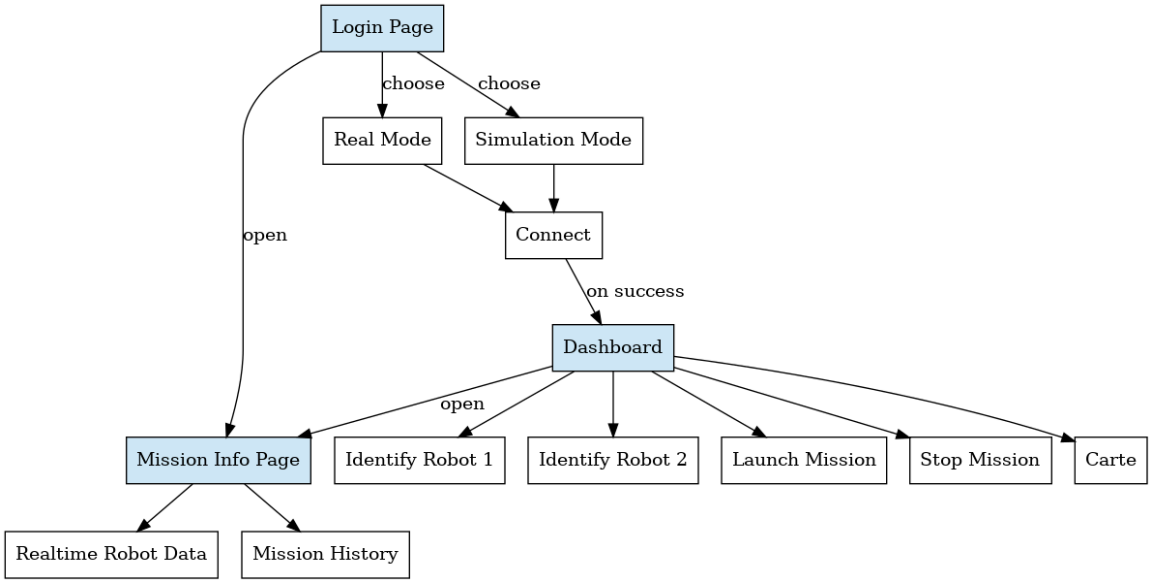


Figure 5.5 : Diagramme de l'interface utilisateur prévisionnelle

Nous avons conçu l'interface pour qu'elle reste utile même sans robot connecté. L'utilisateur peut toujours consulter l'historique des missions et les logs, ce qui est important pour l'analyse, le diagnostic ou la vérification d'anciennes missions. Ce choix rend l'outil plus flexible, car ces actions ne nécessitent pas la présence d'un robot.

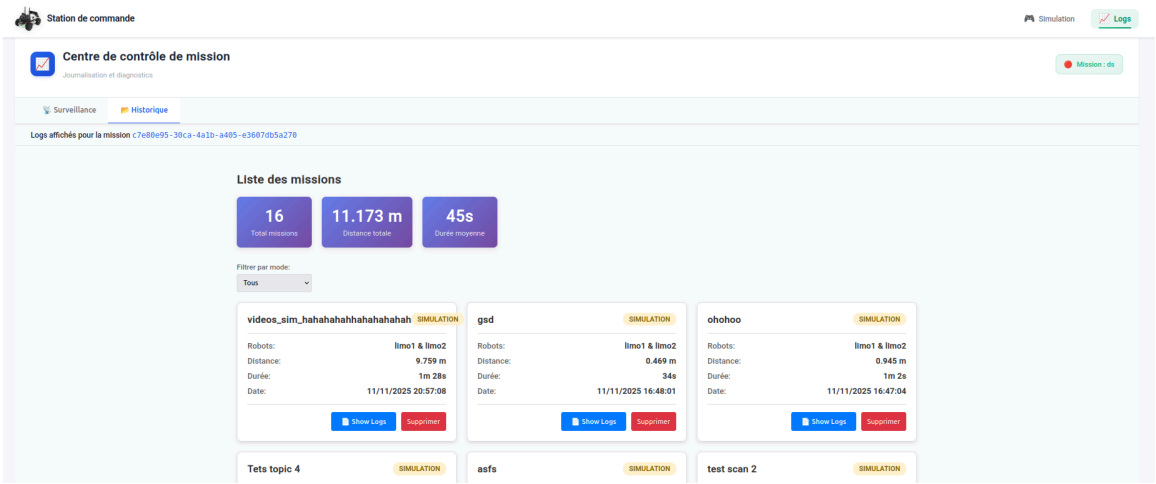


Figure 5.4 : Vue historique des missions

Lorsque qu’un robot est connecté, La page principale ’DashBoard’ ajoute automatiquement ses informations en temps réel (Carte, puis batterie des robots à l'avenir). Nous avons choisi de garder une seule page adaptable plutôt que plusieurs pages séparées, afin de simplifier l’utilisation et d’offrir une transition naturelle entre la consultation et la supervision.

Nous avons également décidé de verrouiller la vue pour tous les utilisateurs du même réseau. Par exemple, si l’application est en mode simulation, tout le monde voit ce mode. Cela évite les conflits, comme plusieurs missions lancées en même temps, et garantit que tout le système reste cohérent. Ce choix de verrouillage protège l’intégrité des missions et facilite la coordination entre les utilisateurs.

Depuis le PDR nous avons donc ajouter l’état et la position des robots en direct (R.F.3, R.F.9), permis la vue de la carte de l’environnement (R.F.8), rendu les logs faciles à consulter pour le suivi et le débogage (R.C.1) et améliorer à la fois la facilité d’utilisation du site et sa sécurité.

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 répo / 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/ttyTHS1
colcon build
source install/setup.sh
ros2 launch robot_exploration robot_bringup.launch.py
```

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

```
sudo chmod 666 /dev/ttyTHS1
colcon build
source install/setup.sh
ros2 launch robot_exploration robot_bringup.launch.py 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: Permettra de commencer l'exploration pour les deux robot
- Arrêter la mission: Fera arrêter l'exploration et enverra la carte dans la base de données puis nous redirigera vers la page de départ.

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 une 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 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. Voici le lien vers le diagramme: Diagramme de Gantt - CDR.xlsx

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)	Obligatoire (7 points): - R.F.1 - 3 points - R.F.2 - 4 points
11 novembre 2025 (CDR)	Obligatoire (44 points): - 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 - R.C.3 (prototype) - 1 point - RQ1 - 5 points - RQ2 - 5 points Facultatif (14 points) - RF12 - 1 points - RF9 - 3 points - RF14 - 5 points - RF17 - 5 points
4 décembre 2025 (RR)	Obligatoire (25 points): - RC2 - 4 points - RC3 - 1 point - RC4 - 5 points - R.F.6 - 10 points - R.F.7 - 2 points - R.F.9 - 3 points (CDR) Continuation - RQ1 - RQ2 Facultatif (20 points) - RF16 - 5 points - RF19 - 6 points - RF18 - 5 points - RF20 - 4 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 Déroulement du projet (Q2.5)

En ce qui concerne la remise du PDR, notre équipe estime que le processus s'est globalement déroulé de manière satisfaisante. La rédaction du rapport a été bien structurée et rigoureuse, même si certaines sections auraient pu être davantage approfondies. Malgré quelques imperfections, l'ensemble du document témoignait d'une compréhension solide des requis et d'une bonne répartition des responsabilités. Sur le plan technique, toutes les tâches liées au développement du code ont été réalisées conformément à l'échéancier. Mis à part une mauvaise interprétation concernant les démonstrations vidéo, l'ensemble des fonctionnalités prévues étaient opérationnelles et démontrées. D'ailleurs, même la fonctionnalité omise pour des raisons d'incompréhension était pleinement fonctionnelle, bien qu'elle n'ait pas été incluse dans la démonstration officielle.

Entre la remise du PDR et celle du CDR, certaines difficultés techniques et organisationnelles sont apparues. Alors que nous respections scrupuleusement notre plan de travail initial, plusieurs facteurs ont ralenti notre progression, notamment la gestion des logs et l'intégration des différentes composantes logicielles. Cette phase a également été marquée par une période de forte charge académique, ce qui a diminué le temps disponible pour le projet. De plus, nous avons rencontré des défis d'intégration entre la simulation et l'environnement physique, particulièrement au niveau de la cartographie et de l'exploration.

En simulation, l'exploration fonctionnait de manière fluide, mais en contexte réel, les différences liées à la qualité du capteur Lidar, aux imprécisions de la localisation et aux bruits environnementaux ont rendu l'expérience plus complexe. Afin d'obtenir une cartographie plus stable et précise, nous avons pris l'initiative d'utiliser le package *Cartographer* plutôt que *Slam-Toolbox*, ce qui a permis d'obtenir une carte visuellement plus fidèle à l'environnement physique et offre une meilleure précision de cartographie. Cette décision a exigé un effort d'adaptation considérable, mais elle témoigne de notre capacité à analyser nos limites techniques et à choisir des solutions adaptées.
Nous avons également remarqué que certaines tâches prenaient plus de temps que prévu en raison d'une dépendance entre les modules : par exemple, la fonctionnalité d'exploration nécessitait que la localisation et la communication inter-robots soient déjà stables. Cette interdépendance a parfois créé des goulots d'étranglement dans le développement et mis en évidence la nécessité

d'une meilleure planification incrémentale et d'une intégration continue plus fréquente.

Malgré ces difficultés, notre bonne entente, notre rigueur et la qualité de nos échanges nous ont permis de surmonter ces obstacles. Les discussions fréquentes, la volonté d'écouter les points de vue de chacun et la répartition claire des responsabilités ont favorisé un climat de collaboration efficace. Nous avons aussi su démontrer une capacité d'adaptation en réévaluant nos priorités lorsque certaines approches ne donnaient pas les résultats escomptés.
Par ailleurs, notre rencontre avec un conseiller HPR a été un moment charnière. Elle nous a permis d'identifier plusieurs aspects organisationnels à améliorer, notamment en ce qui concerne la coordination des tâches, la prise de décision collective et la gestion du temps en laboratoire. Même si cette rencontre a eu lieu peu avant la remise du CDR, elle a servi de déclencheur pour une réflexion en profondeur sur notre fonctionnement d'équipe. Nous avons ainsi convenu de mettre en place des scrums plus structurés, dans un environnement isolé (Un appel vocal sur Discord ou Teams), afin d'éviter les distractions fréquentes qu'on peut avoir dans les laboratoires de robots. Nous avons aussi décidé de prendre des notes systématiques lors de ces réunions, afin d'assurer un meilleur suivi des décisions et d'améliorer la continuité entre les séances.
Au-delà des aspects techniques, cette période nous a permis de développer une maturité collective importante. Nous avons appris à mieux anticiper les imprévus, à répartir les charges de travail selon les forces de chacun et à accepter que certaines erreurs fassent partie du processus d'apprentissage. Cette capacité d'introspection nous a permis de transformer nos difficultés en occasions d'amélioration, en renforçant à la fois notre cohésion d'équipe et nos compétences individuelles.

En somme, même si la progression entre le PDR et le CDR a été ponctuée de défis, elle a aussi été marquée par une évolution notable dans notre manière de collaborer, d'analyser nos erreurs et d'ajuster nos méthodes de travail. Ces apprentissages constituent une réussite en soi, puisqu'ils nous ont permis de consolider notre efficacité collective et d'accroître la qualité globale du projet.

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

- **R.F.1 (Fonctionnel) :** Lorsqu'un utilisateur appuie sur le bouton d'identification d'un robot via l'interface web, le robot sélectionné reçoit la commande "Identifier" et commence à faire des rotation en continu.

- **R.F.2 (Fonctionnel - À optimiser):** Dans l'interface web, la commande « Lancer la mission » démarre l'exploration aléatoire des robots, basée sur un algorithme de frontières qui leur permet de naviguer vers les zones libres tout en évitant les obstacles. La mission peut être interrompue avec la commande « Arrêter la mission ». Le problème actuel est qu'il faut envoyer la commande à partir de deux terminaux différents pour chacun des robots (chacun leur ROS_DOMAIN_ID). Ainsi il faudrait trouver une façon de permettre d'envoyer la commande au deux robots. Nous avions pensé à implémenter un bridge pour les ROS_DOMAIN_ID.

- **R.F.3 (Fonctionnel - À optimiser) :** Dans l'interface web, Pour commencer on remarque que les robots sont dans l'état "**En attente**". Lorsque la commande "Lancer la mission" est cliquée, on observe un changement d'état qui passe à : "**Exploration**", qui est l'état du robot au début de la mission. Par la suite il sera possible de changer à l'état : "**Navigation**", en envoyant un message au topic "change_state", commande pas encore implémentée sur l'interface graphique.

- **R.F.4 (Fonctionnel) :** À la réception de la commande « Lancer la mission », les robots utilisent l'algorithme d'exploration aléatoire du framework Nav2, exploitant leurs capteurs LIDAR pour explorer les zones accessibles tout en évitant les obstacles.

- **R.F.5 (Architecture de code non optimal - fonctionnel):** Avec les framework/package *Nav2* et *Slam-toolbox*/*Cartographer* les deux robots peuvent naviguer dans l'environnement tout en évitant les obstacles détectés par le LIDAR. Pas totalement optimal en raison de l'utilisation de deux framework différents pour la carte. Il faudrait adapter notre architecture pour n'utiliser qu'un seul framework. Notre choix penche vers *Cartographer*

- **R.F.8 (Fonctionnel) :** La fusion des deux cartes (une par robot) est fonctionnelle. Pour l'instant, on ne voit la carte de chaque robot individuellement sur l'interface utilisateur qui est mise à jour dynamiquement.

- **R.F.10 (Semi-Fonctionnel) :** L'interface utilisateur est déployée sous forme de service web et peut être ouverte depuis un navigateur sur la machine locale peu importe le format de la machine (iphone, ordinateur..). Tous les utilisateurs accédant depuis cette même machine voient la même vue en temps réel. Cependant, cette synchronisation des vues est pour l'instant limitée à l'appareil hébergeant le serveur. Si deux ordinateurs distincts essaient d'accéder au site, ils n'auront pas encore une vue partagée, car le serveur n'a pas été déployé sur le réseau (ex : AWS). Cette prochaine étape permettra d'assurer une vraie synchronisation multi-appareils, comme le demande le requis.

- **R.C.1 : (Fonctionnel) :** Les logs de débogage sont enregistrés en continu pendant chaque mission via webSockets à une fréquence de 1Hz puis sauvegardés automatiquement à la fin de la mission dans une base de données. L'utilisateur peut les consulter à tout moment depuis l'interface web, soit en temps réel, soit pour une mission précédente.

- **R.C.3 (Fonctionnel):** On a bien 3 murs, autre que les 4 murs extérieurs) qui s'ajoute à l'environnement aléatoirement. On a des obstacles additionnels qui s'ajoutent aléatoirement.

- **R.Q.1 (Fonctionnel):** Les conventions établies par l'équipe ont bien été respectées dans l'ensemble des cas. De plus cette convention a bien été spécifié dans le README.md comme demandé.

- **R.Q.2 (Fonctionnel):** Les tests unitaires ont été effectués pour le serveur et le front-end pour les fonctions moins complexes. On a atteint un test coverage 72% des fonctions sur le client et 81% des fonctions sur le serveur. Nous avons rédigé le tests.pdf pour les fonctionnalités plus complexes impliquant ROS2 et l'environnement physique.

- **R.F.12 (Semi-Fonctionnel):** Il n'y a pas de fonctionnalité pour entrer des positions relatives ou d'orientation relative. Cependant, lors de la mission le système peut déterminer leurs positions relatives en générant une carte combinée.

- **R.F.14 (Semi-Fonctionnel):** Il y a la fonctionnalité de définir une série de points et le robot sélectionné les suivra. Il y a possibilité d'annuler une requête. La navigation se termine lorsqu'un point est inaccessible, mais il n'y a aucun avertissement sur le site, ni de possibilité de choisir un nouveau point. Le UI/UX est à améliorer.

- **R.F.17 (Fonctionnel)**: La base de données est fonctionnelle, elle enregistre pour chaque mission le nom, la date, la durée, le mode (réel ou simulation) et les robots utilisés. Ces informations peuvent être consultées et triées depuis l'interface web, qui permet aussi d'afficher les détails et les logs associés.

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