



**POLYTECHNIQUE
MONTRÉAL**

UNIVERSITÉ
D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

Rapport final de projet
no. H2023 – INF3995 du département GIGL

Conception d'un système d'exploration

Équipe 103

Étudiant	Matricule	Signature
Mathilde Brosseau	2078684	Mathilde Brosseau
Vincent Haney	2108812	Vincent Haney
Manel Keddami	1989876	Keddami
Xavier L'Heureux	2078098	Xavier LH
Yun Ji Liao	2017113	Yun Ji Liao
Thomas Moorjani-Houle	2086852	Thomas Moorjani-Houle

21 Avril 2023

Table des matières

Table des matières	2
1. Vue d'ensemble du projet	3
1.1 But du projet, porté et objectifs	3
1.2 Hypothèse et contraintes	3
1.3 Biens livrables du projet	4
2. Organisation du projet	6
2.1 Structure d'organisation	6
2.2 Entente contractuelle	6
3. Description de la solution	7
3.1 Architecture logicielle générale	7
3.2 Station au sol	8
3.3 Logiciel embarqué	10
3.4 Simulation	11
3.5 Interface utilisateur	12
3.6 Fonctionnement général	16
4. Processus de gestion	17
4.1 Estimations des coûts du projet	17
4.2 Planification des tâches	17
4.3 Calendrier de projet	19
4.4 Ressources humaines du projet	19
5. Suivi de projet et contrôle	20
5.1 Contrôle de la qualité	20
5.2 Gestion de risque	20
5.3 Tests	21
5.4 Gestion de configuration	22
5.5 Déroulement du projet	23
6. Résultats des tests de fonctionnement du système complet	24
7. Travaux futurs et recommandations	24
8. Apprentissage continu	25
9. Conclusion	27
10. Références	28

1. Vue d'ensemble du projet

1.1 *But du projet, porté et objectifs*

Le but du projet est de concevoir un système informatique mettant en relation les concepts de réseautique, de système embarqués, de bases de données et de sécurité informatique. Ce système sera composé d'une station au sol munie d'une interface web qui permettra de lancer des missions d'exploration menées par une équipe de deux robots, soit deux rover AgileX Limo [1]. Ces robots, dotés de capteurs, devront explorer de manière autonome une pièce d'un bâtiment de moyenne dimension afin de la cartographier. De plus, l'interface web permettra de visualiser en continu la carte produite par les deux robots, la position de ceux-ci dans ladite carte ainsi que leur état. Le mandat nécessite également de concevoir un système de simulation, à l'aide du simulateur Gazebo, permettant de contrôler les robots simulés via la même interface que la station au sol. Ce système sera utilisé pour tester conceptuellement les algorithmes de contrôle des robots et les fonctionnalités de la station au sol avant de les déployer.

Par ailleurs, trois livrables seront fournis durant ce processus : la PDR, la CDR et la RR. Les buts et la portée de ces documents seront décrits plus en détails dans la section 1.3 – Biens livrables du projet.

1.2 *Hypothèse et contraintes*

La conception des systèmes demandés reposent sur plusieurs hypothèses.

Premièrement, on suppose que l'ordinateur de bord ainsi que les robots pourront être programmés avec le système d'exploitation Ubuntu 20.04; et que les composantes logicielles du système seront conteneurisées à l'aide de Docker pour encapsuler nos configurations et éviter les problèmes de compatibilité. Deuxièmement, on suppose que notre implémentation utilise deux robots de type rover AgileX Limo. On suppose également que ces robots nous sont fournis par l'agence spatiale et que les coûts reliés ne sont pas comptabilisés dans notre projet. Troisièmement, on suppose que les modes de communication sélectionnés pour lier les robots à la station de contrôle, et la station de contrôle à son interface sont assez efficaces pour permettre de mettre à jour les données de missions avec une fréquence de 1 Hz. Quatrièmement, on suppose que la salle dans laquelle les robots seront placés possèdera un routeur désigné configuré avec des adresses IP statiques. Cela fait en sorte que nous pouvons assumer que les robots ont toujours la même adresse IP lors de nos utilisations. Cinquièmement, on suppose que la librairie Cartographer de Google [2] est compatible avec nos robots et nous permettra de cartographier efficacement l'environnement. Sixièmement, on suppose que l'environnement virtuel procuré par le simulateur Gazebo [3] sera assez proche de la réalité pour nous permettre de valider adéquatement le comportement des robots et de la station au sol.

Les exigences du projet nous imposent aussi plusieurs contraintes.

Tout d'abord, une contrainte de temps réduit la quantité de travail pouvant être investie par notre équipe dans ce projet à 630 heures-personne, ce qui limite la qualité et la quantité de requis du système que nous pouvons implémenter dans le cadre de ce projet. Ensuite, une contrainte d'espace et de disponibilité est imposée par le fait que les

robots sont seulement accessibles à certaines périodes de la journée et dans une seule salle spécifique de l'école Polytechnique Montréal. De plus, une quantité limitée de robots est mise à notre disposition, ce qui fait en sorte que nous devons partager les robots et la salle avec les autres équipes et que les opportunités de tests sont limitées.

Pour finir, un nombre de requis à compléter est également imposé, nous contraignant à compléter des requis ayant un poids total minimal de 100. Nous avons choisi de prendre un poids de 101, avec le RF12 de poids 1 comme marge de manœuvre. Le tableau suivant montre les requis choisis ainsi que le poids qui y est associé :

Numéro du requis	Description	Poids
R.F.1	Commande identifier	3
R.F.2	Commandes lancer/terminer mission	4
R.F.3	Montrer type et état à une fréquence de 1Hz	2
R.F.4	Exploration autonome	4
R.F.5	Évitement d'obstacles	8
R.F.6	Commande retour à la base avec précision de 0.5 m	10
R.F.7	Retour à la base automatique lorsque la batterie est à moins de 30%	2
R.F.8	Générer la carte en continu	10
R.F.9	Position du robot dans carte affichée en continu	3
R.F.10	Interface web sur plusieurs appareils	4
R.F.12	Position/orientation initiale spécifiés	1
R.F.17	Avoir une base de données	5
R.F.18	Enregistrer la carte générée sur la station	5
R.F.20	Zone de sécurité	5
R.F.21	Contrôleur Ackermann	5
R.C.1	Log de débogage à fréquence de 1hz sauvegarder dans la station au sol	5
R.C.2	Logiciel lancé avec une seule commande	4
R.C.3	Environnement Gazebo généré aléatoirement	1
R.C.4	Interface utilisateur respecte les 10 heuristiques	5
R.C.5	Interface se connecte au robot disponible automatiquement (max 2)	5
R.Q.1	Format du code standardisé	5
R.Q.2	Tests unitaires pour les composants du logiciel	5
Total		101

Tableau 1 : requis du projet

1.3 Biens livrables du projet

Preliminary Design Review (PDR) :

Le but de ce livrable est de répondre à l'appel d'offres en présentant notre prototype et en démontrant notre maîtrise des composantes matérielles et logicielles requises pour réaliser le projet. Ce livrable est séparé en trois parties.

Premièrement, il faut remettre une première version du document de réponse à l'appel d'offres qui présente le prototype que nous désirons réaliser ainsi que les modes d'organisation du travail choisis par notre équipe.

Deuxièmement, une démonstration vidéo de l'environnement de simulation doit être présentée. Celle-ci doit montrer deux robots suivant un parcours quelconque. De plus, on doit pouvoir interagir avec la simulation via l'interface web de la station au sol et le requis fonctionnel 2 doit être implémenté. Les robots doivent donc répondre aux commandes "Lancer la mission" et "Terminer la mission".

Troisièmement, une démonstration réelle de la station au sol, de son interface web et des robots physiques doit être réalisée. À cet effet, l'interface web du serveur doit permettre de lancer la commande "Identifier", à laquelle les robots physiques doivent réagir en faisant clignoter une LED ou en émettant un son. Ce rapport doit être remis le 10 février 2023.

Critical Design Review (CDR) :

Le but de ce livrable est de présenter une version fonctionnelle du projet, puisque le développement sera réellement entamé à ce stade-ci. Il a aussi comme objectif de présenter la documentation complète du projet, incluant la majorité des requis fonctionnels et des requis de conception.

Il faut donc en premier lieu remettre une version complète, concise et révisée de la documentation de notre projet, ainsi qu'une présentation détaillant les changements faits par rapport à la PDR ainsi que la conception actuelle du projet suite à ces changements. En second lieu, le code source des requis fonctionnels implémentés devra être prêt pour la CDR.

Il faudra également présenter des vidéos de démonstration montrant le bon fonctionnement de tous les requis demandés à cette étape. Ce rapport doit être remis le 17 mars 2023.

Readiness Review (RR) :

Le but de ce livrable est de présenter un produit entièrement complété et fonctionnel, en plus d'avoir une documentation révisée et mise à jour. Tous les logiciels doivent alors avoir été conçus et testés et l'application web doit être capable de communiquer avec les robots, que ce soit en simulation ou directement sur les robots physiques.

Il faudra également effectuer une présentation du produit final à une audience non familière avec le projet, afin d'exposer et mettre en avant la solution que nous avons proposée.

Finalement, tout le code, tests et instructions nécessaires à la compilation et lancement du projet doivent être complétés et remis sur le répertoire GitLab. Ce rapport doit être remis le 21 avril 2023

2. Organisation du projet

2.1 Structure d'organisation

Nous avons décidé d'opter pour une structure d'organisation plutôt décentralisée, se basant sur le développement Agile [4]. Chaque membre de l'équipe se verra donc assigner des tâches selon leurs compétences, ainsi qu'un estimé de date à laquelle chacune devrait être complétée. Pour se tenir au courant de l'avancement du projet, deux rencontres Scrum auront lieu chaque semaine et prendront place les mardis et jeudis, aux heures de laboratoire du cours. L'animation des Scrums ainsi que la production des rapports d'avancement seront gérés par Thomas, qui a le rôle de "Scrum Master" dans notre équipe. Cette méthode nous permet en premier lieu de travailler et s'organiser individuellement en ayant un calendrier plus ou moins flexible, mais également de répondre rapidement aux imprévus et changements qui pourraient avoir lieu lors du développement du projet grâce aux réunions et rapports d'avancements faits chaque semaine.

Concernant les rôles plus techniques, Thomas, Yun Ji s'occupent de la conception et maintenance de l'application web (frontend et UI), Mathilde et Vincent s'occuperont de la conception du serveur de la station au sol (backend), tandis que Manel et Xavier s'occupent de la simulation et du bon fonctionnement des robots. Manel sera la responsable de DevOps de notre projet et Xavier aura également le rôle de responsable technique (tech lead) au sein de l'équipe.

Nous avons également décidé de nous laisser une période de tests d'intégration d'environ un à deux jours avant chaque date de livraison de produit afin de nous assurer que toutes les fonctionnalités ajoutées s'intègrent bien entre elles et ainsi fournir un produit qui répond aux exigences.

2.2 Entente contractuelle

L'entente contractuelle que nous proposons pour réaliser ce projet serait une entente de type clé en main, ou contrat à prix ferme. Ce type de contrat nous engage à fournir un produit final respectant les exigences spécifiées par l'Agence. Les requis fonctionnels, matériels et logiciels sont déjà énoncés et détaillés par l'Agence et ces derniers ne risquent pas de changer lors de la conception du produit.

De plus, un suivi et implication limitée sera demandé de la part de l'Agence, étant donné que la conception du produit est entièrement notre responsabilité. La durée et la date de livraison de l'engagement ont également clairement été communiquées, ce qui renforce le choix de proposer un contrat à prix ferme. Nous nous engageons donc à fournir un produit respectant toutes les exigences à la fin de l'échéancier.

L'avancement du projet sera fourni en 3 étapes, où le meilleur produit possible sera offert pour chacune. Nous allons également fournir un estimé de temps et de ressources détaillé pour chacune des tâches à effectuer pour chaque étape intermédiaire, en se laissant une période de test et d'intégration pour s'assurer du bon fonctionnement du produit à la fin de celles-ci.

Nous sommes également confiants d'être en mesure de gérer les éventuels imprévus qui puissent avoir lieu et de fournir un produit fonctionnel et fiable à la fin du projet.

3. Description de la solution

3.1 Architecture logicielle générale

Notre architecture se sépare en trois composantes principales: l'application web pour la station au sol, le serveur de notre station au sol et notre contrôleur chargé de gérer la partie logiciel embarqué. Ces derniers seront expliqués plus en détail dans les prochaines parties du rapport. Le schéma général de notre architecture est représenté dans la figure suivante:

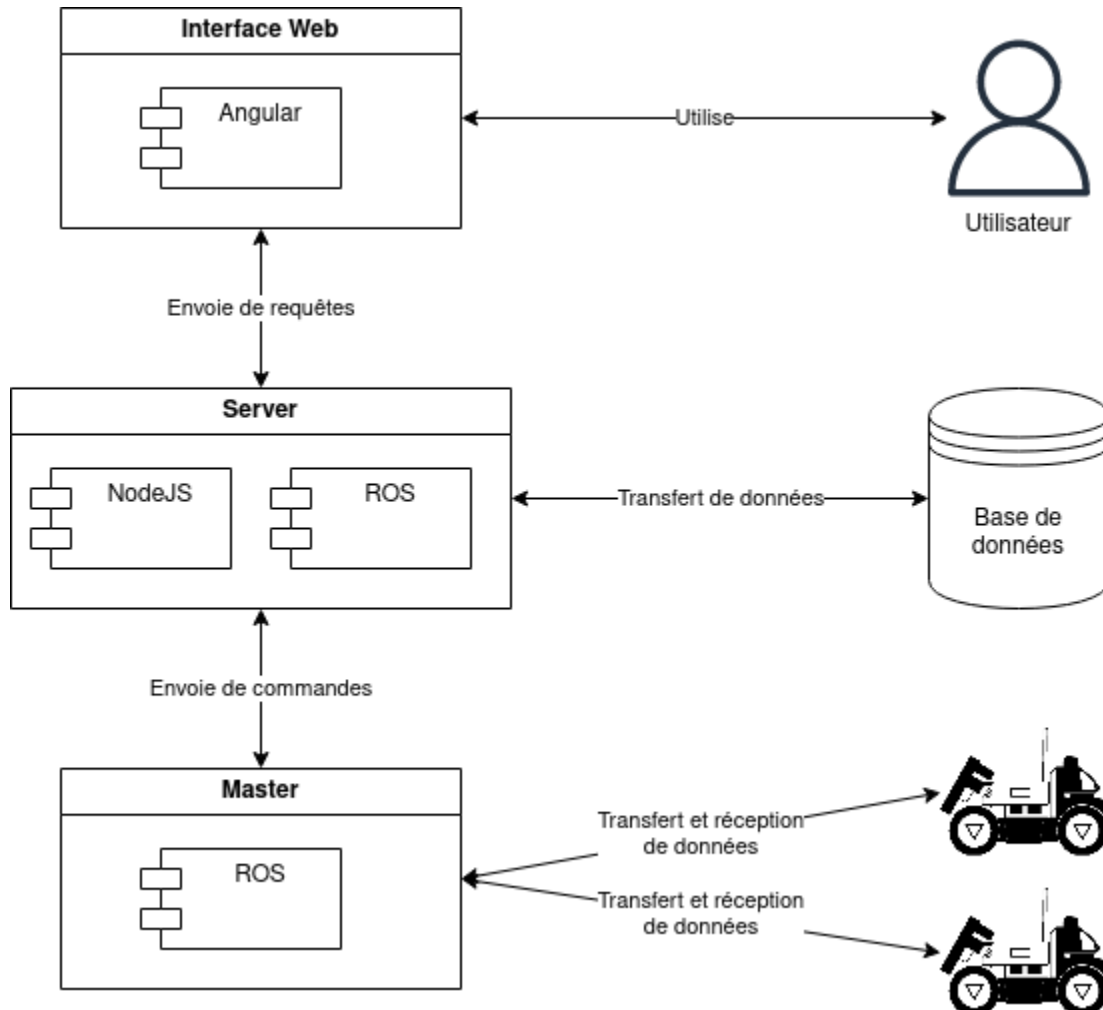


Figure 1 : Diagramme de l'architecture générale logicielle

Afin de concevoir notre interface web sur plusieurs types d'appareils (requis R.F.10) nous avons décidé d'implémenter celle-ci avec Angular [5], qui offre déjà plusieurs modules que l'on peut directement utiliser et adapter aux différents appareils. De plus, Angular permet d'afficher une carte d'environnement en continue en utilisant Canvas [6] (requis R.F.8). Pour le lancement des missions (R.F.2), l'interface communique avec un serveur NodeJS [7] via des requêtes HTTP. Le serveur communique pour sa part avec le contrôleur via ROS [8], qui utilise la même technologie pour interagir avec les robots.

Le serveur reçoit également des données telles que les positions des robots (R.F.9), le statut des robots (R.F.3) et la carte générée (R.F.8) via ROS, puis les transmet au client Angular à travers des websockets [9].

Les historiques des informations relatives aux missions (R.F.17), les cartes générées (R.F.18) ainsi que les logs (R.C.1) seront stockés dans une base de données SQLite [10] et seront accessibles à partir de l'interface utilisateur. Finalement, lors des missions, les robots vont générer les cartes en utilisant Cartographer].

3.2 Station au sol

La station au sol a le rôle de station de contrôles des robots. Elle est sous forme de serveur web. Ce dernier sera sous forme de «Reverse-Proxy» (voir la figure 2) afin d'obtenir une solution sécuritaire aux imprévus qui pourraient être rencontrés durant la mission. Cette architecture permet d'avoir plusieurs clients (les utilisateurs) (R.F.10) pouvant se connecter simultanément au serveur statique. Le serveur statique va communiquer les requêtes au serveur dynamique et vice-versa. Dans notre cas, le serveur statique est représenté par l'application Angular roulant sur l'ordinateur de la station au sol (composante "Interface Web") et le serveur dynamique correspond à la composante "Serveur" de notre architecture globale.

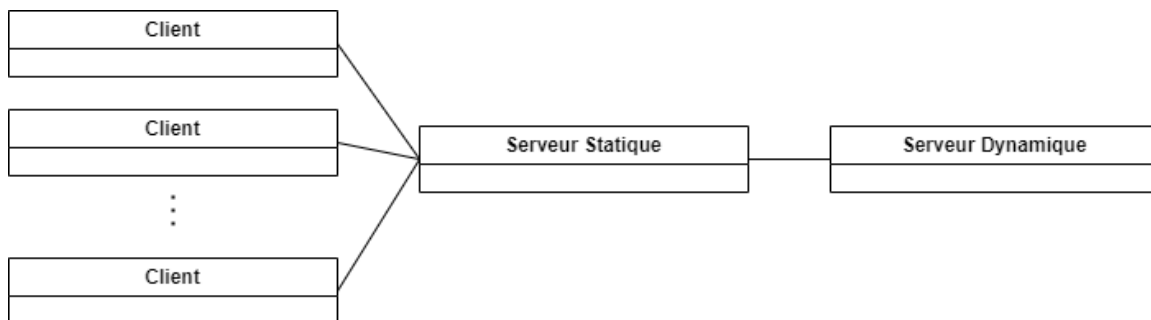


Figure 2 : Diagramme de l'architecture des communications client-serveur

Nous avons choisi un serveur sous forme d'application Node.JS, car nous savons déjà comment l'utiliser et qu'il y a vraiment beaucoup de bibliothèques disponibles, ce qui facilite le développement. De plus, il sera possible de l'exécuter à l'aide d'une commande à partir du terminal Linux (R.C.2) avec la commande « make », grâce à un fichier Makefile [11] qui. Cette application doit pouvoir :

- Contenir une base de donnée locale
- Se connecter aux robots
- Signaler aux robots de commencer une mission
- Collecter les informations recueillies par les robots
- Envoyer certaines informations à un ou plusieurs clients à propos des missions
- Conserver un historique des «logs» des missions
- Conserver les informations des cartes

Pour ce faire, nous avons décidé de procéder avec l'architecture haut niveau présentée à la figure 3.

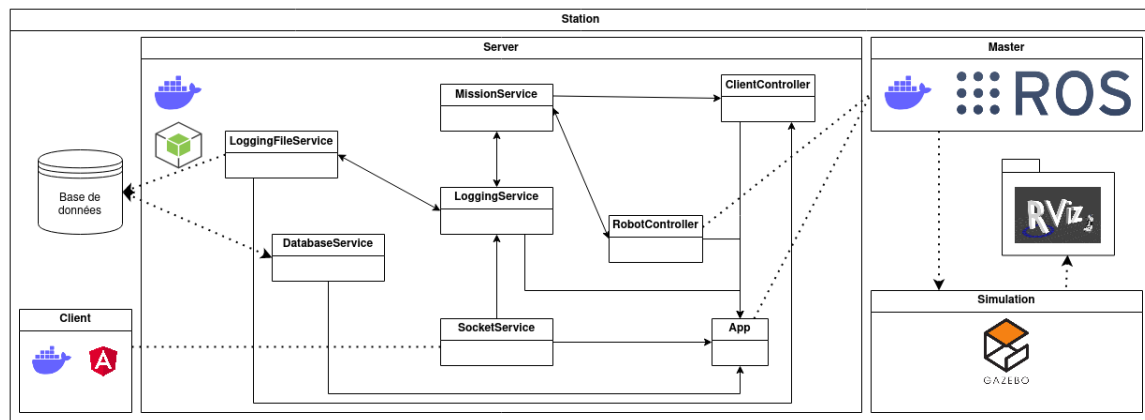


Figure 3 : Diagramme de l'architecture de la station au sol

La solution pour la station au sol est divisée en une base de données et deux conteneurs Docker: Server et Master.

D'abord, nous avons choisi une base de données accessible à l'aide du service «DatabaseService» pour les requêtes (R.F.17). Cette base de données va contenir les fichiers de logs et les données des cartes des missions passées. Les requêtes à cette base de données seront gérées par «DatabaseService» (R.F.18). La classe qui aura accès à ce contrôleur est la classe « App » qui a comme responsabilité de gérer les transactions d'information entre le client, les robots et la base de données.

Ensuite, la classe « App » aura comme responsabilité de créer le node ROS Server qui lui permettra de communiquer avec le node Master, de configurer le router de l'application pour recevoir les différentes requêtes du Client et de s'abonner aux différents topics ROS sur lesquels les informations provenant de l'exploration seront publiées. Les différents nodes auxquels App s'abonne sont: /mission_cmd (commandes envoyées au contrôleur), /map, /robot_position, /robot_state et /clock (pour avoir le temps écoulé dans la mission).

Les missions seront amorcées par un client à l'aide d'une requête HTTP envoyée vers le serveur et reçue par la classe « ClientController » qui gère les requêtes des clients. Au départ, l'utilisateur pourra entrer la position absolue ainsi que la présence ou non d'une zone de sécurité (R.F.20). Autrement, la position sera prédéterminée au départ (R.F. 12).

Lorsque le client saisit une commande (start, stop, identify, retour à la base) via l'interface web, le «ClientController» reçoit la demande et propage le signal approprié pour initialiser les services requis (notamment «LoggingService», «SocketService» et «MissionService») et transmettre les commandes au node Master (le contrôleur) via le topic /mission_cmd. à une mission.

Tout au long de la mission, le «LoggingService» enregistre simultanément les positions des robots et le temps écoulé dans la mission à une fréquence de 1 Hz (R.C.1) à l'aide de l'abonnement au topic /robot_position et /clock. La carte, de son côté, est propagée à l'aide du topic /map et l'état est noté à l'aide du topic /robot_state. Toutes ces informations sont redirigées à tous les clients à l'aide du «SocketService».

La carte reçue du topic /map sera représentée sous forme de données brutes (points) qui pourront être interprétées par le client. Ceci facilite l'enregistrement et le transfert des données ce qui permettra de répondre au requis R.F.8.

Dès qu'une mission sera terminée, le client le signale au serveur et un log sera enregistré à l'aide de « `LoggingFileService` ». Ce log, ainsi que l'information relative à la mission seront alors envoyés à la base de données.

Notons que le «SocketService» utilisé pour transiger certaines informations permet aussi de facilement gérer la gestion d'une seule mission avec un nombre arbitraire d'utilisateurs (R.F.10), puisque tous les utilisateurs qui se connectent au serveur sont automatiquement ajoutés à toutes les "rooms" utilisées pour communiquer les informations de mission : les logs, l'état de la mission, la carte et les positions.

Cette architecture est avantageuse pour plusieurs raisons:

- Séparation des responsabilités
- Simple de gérer un ou plusieurs utilisateurs en contrôle d'une mission
- Gestion simplifiée des robots

3.3 Logiciel embarqué

Notre architecture embarquée est avant tout composée de 3 containers Docker : le Serveur, le Master et le Robot (chaque robot s'exécute à travers un container). Le schéma de notre logiciel embarqué est représenté dans la figure suivante :

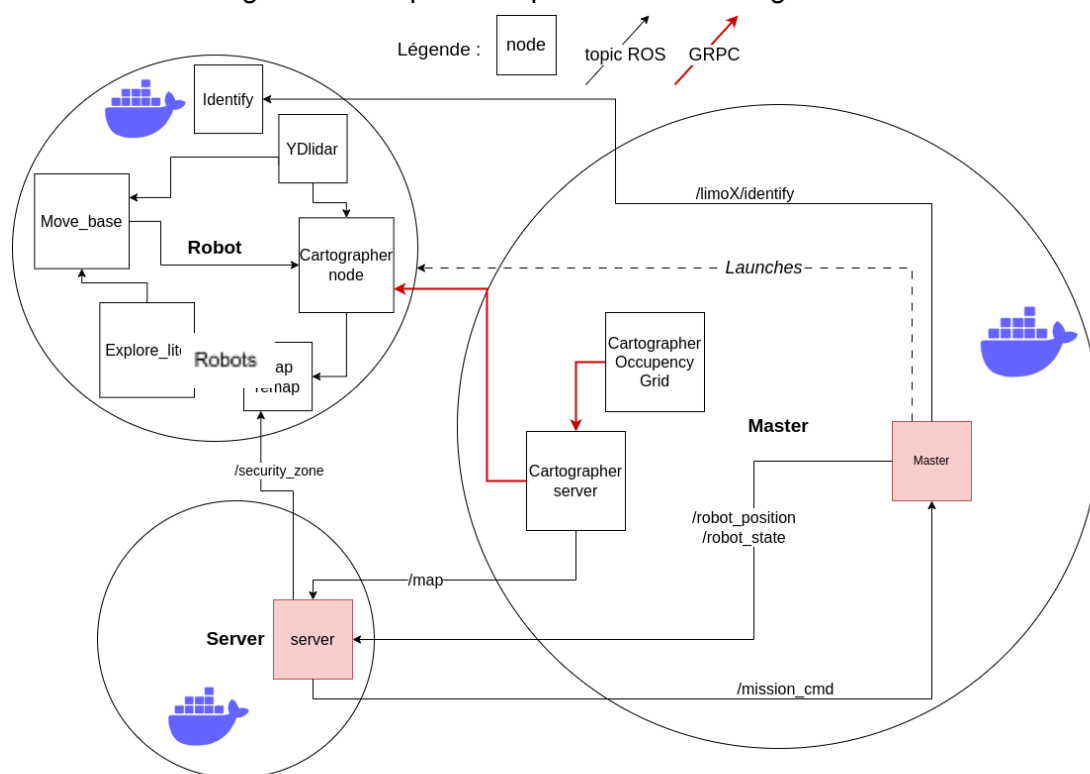


Figure 4 : Diagramme d'architecture du logiciel embarqué

Notre logiciel embarqué est basé sur ROS et son système de nodes. Il y a d'abord les nodes représentant le Master et le Serveur (en rose sur le schéma), qui servent uniquement à la communication et la synchronisation entre les différentes parties de l'architecture.

Le conteneur Server contient également le serveur web et est donc dans un conteneur distinct. Nous sommes ainsi assurés du fonctionnement du système même sans serveur web, par exemple lors de tests d'intégration. Ses responsabilités relèvent uniquement du pont entre serveur web et ROS. Le node server s'occupe donc d'envoyer les commandes et l'information sur la zone de sécurité, tout en recevant l'état des robots et la carte générée, qu'il transfère par la suite aux clients web.

Ensuite, les nodes Cartographer Server et OccupancyGrid s'exécutent directement sur le container Master, puisqu'ils servent à agréger les données des deux robots pour produire une carte globale.

Finalement, il y a les nodes spécifiques à chaque robot, qui roulent sur chacun des robots utilisés : *Identify*, *Move_base*, *YDlidar*, *Explorelite*, *Cartographer_node* et *map_remap* (ainsi que plusieurs autres générés automatiquement).

- Le node *map_remap* sert à faire l'interface entre *move_base* et *explorelite*.
- *Identify*, sert à la commande identifier et fait un pont entre *ros* et *alsa*[12] sur le système hôte.
- *move_base*[13] s'occupe de l'évitement d'obstacle et de la planification de mouvement basé sur le principe du *Timed Elastic Band*[14], plus adapté aux déplacements par contrôleur Ackerman.
- *explorelite*[15], quant à lui, s'occupe de l'exploration autonome selon le principe du *frontier based exploration*, c'est à dire qu'il cherche à se diriger vers les zones comptant le plus d'espace inexploré. Pour s'assurer que le robot reste dans la zone de sécurité, *map_remap* vient s'insérer entre *movebase* et *explorelite* pour s'assurer qu'aucune recherche ne s'effectue hors de la zone spécifiée.
- *YDlidar*[16] s'occupe de collecter les données du lidar et *CartographerNode* récolte les données publiées et les transmet par gRPC au serveur cartographer.

3.4 Simulation

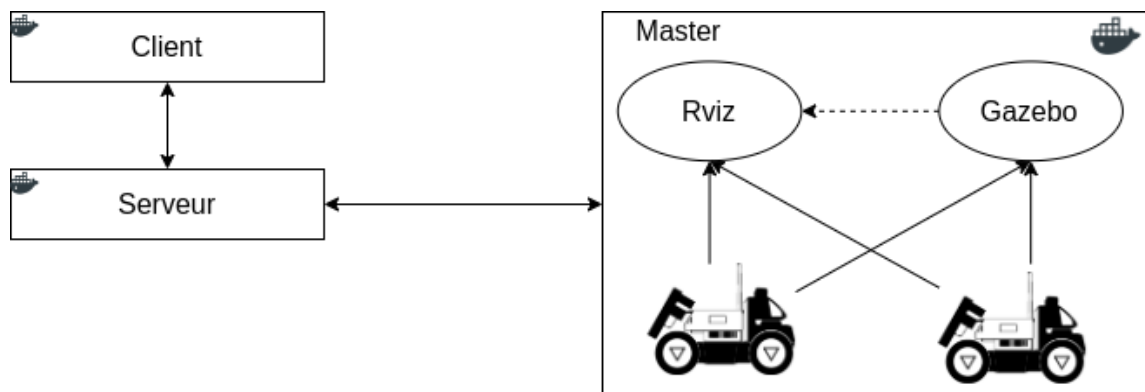


Figure 5 : Diagramme d'architecture de la simulation

La différence entre l'architecture réelle et celle de la simulation est que le contrôleur (node master) lance des nodes différents lorsqu'il reçoit une commande de démarrage de simulation. De plus, aucun container Docker supplémentaire n'est nécessaire, puisque la simulation s'exécute directement dans le container Master. Lors des simulations, on démarre gazebo, cartographer, deux robots simulés et Rviz [17] pour visualiser le tout. Ainsi, on peut visualiser la mission virtuelle avec Gazebo et percevoir les informations recueillies par les capteurs des robots avec RViz.

Du côté des nodes, Gazebo émule celles des robots. Ainsi, le contrôleur communique ses commandes aux robots à travers les topics (`cmd_vel`, etc.) comme il le ferait sans simulation. Il ne sait pas s'il collabore avec les vrais robots ou la simulation et l'interface reste donc de facto identique dans les deux cas (R.L.2). Évidemment, lors du déploiement, les nodes ne seront pas envoyés sur des machines tierces, mais resteront dans le conteneur docker du serveur, à l'aide d'une petite configuration dans le fichier `launch`.

En utilisant au maximum les mêmes outils et procédures pour démarrer la simulation, on s'assurera ainsi d'avoir une couverture de test la plus exhaustive possible, puisque le contrôleur et tout le reste des processus seront testés de la même façon que dans un système physique. En utilisant ROS et le système de *launch*, les tests seront de plus effectivement agnostiques à la plateforme.

Lors du lancement de la simulation, un node spécialement conçu s'occupera de générer les différents obstacles à ajouter à la simulation gazebo (R.C.3). 4 murs extérieurs seront générés pour contenir la simulation, en plus d'un nombre aléatoire d'obstacles (de 3 à 10). En s'intégrant avec le système de launch files de ROS, on pourra ainsi standardiser le système de déploiement. On pourra aussi utiliser le système de `spawn_model` pour générer les obstacles avec `gazebo_ros`, un système très convivial.

Il sera finalement possible, en passant en paramètre les robots à activer à la commande «make», de ne pas inclure les 2 robots planifiés à la simulation (R.C.5), de manière simple et efficace et donc de pouvoir fonctionner avec 1 ou 2 robots.

3.5 Interface utilisateur

L'interface utilisateur sera implémentée comme une page web interactive desservie par un serveur statique sur la station au sol (client).

Les pages tireront leurs données directement de la station au sol, en temps réel. Pour cela, elles établiront un lien de type WebSockets pour être notifiées directement par le serveur lors de mises à jour, ce qui permettra une faible latence avec une charge sur le serveur qui sera drastiquement moins grande.

La station au sol pourra ainsi être fonctionnelle sur un PC ou un laptop indépendant (R.M.4) tout en étant accessible par plusieurs types d'appareils simultanément (R.F.10). Elle sera également indépendante du système, s'utilisant tant dans la simulation que sur les robots physiques (R.L.2)

L'interface sera divisée en 3 pages, soit la page d'accueil, la page de mission et la page d'historique.

La page d'accueil sera composée d'un message d'accueil, d'un bouton de départ d'une mission et d'un bouton donnant accès à l'historique de mission. Si jamais un utilisateur tente de se connecter au site, il sera redirigé automatiquement vers la page de mission.

La page de mission de l'interface utilisateur contiendra les informations relatives aux robots, à la mission, ainsi que les commandes de haut niveau. En particulier, il sera possible d'envoyer des commandes pour débiter une mission et la terminer (R.F.2), ou faire un retour à la base (R.F.6).

Deux sections présentent les informations des robots: leur position, leur type, leur état (R.F.3) et leur niveau de batterie (R.F.7). Il sera finalement possible de cliquer sur un bouton « Identifier » pour émettre un son sur les robots connectés (R.F.1) Les boutons auront la couleur associée au robot auxquels ils sont liés. Les connexions et déconnexions de robots, si applicables, seront envoyées par la station au sol et l'interface sera automatiquement modifiée en conséquence (R.C.5).

Dans cette même page, lorsqu'approprié, les informations concernant la mission active seront disponibles. En particulier, on y trouvera une carte mise à jour en temps réel de la zone explorée (R.F.8) où seront visibles les robots actifs (R.F.9) selon deux couleurs distinctes. Il sera également possible d'y spécifier directement la position et l'orientation initiale du robot (R.F.12) avant le début de la mission. Lorsque souhaité, on pourra ensuite établir une zone de sécurité, qui sera affichée sur la carte, en spécifiant la dimension ainsi que le centre de la zone (R.F.20). Cette zone de sécurité s'appliquera à l'ensemble des robots d'une mission, tant en simulation qu'avec les robots physiques.

De plus, au bas de cette même page, les logs collectés à chaque seconde seront affichés dans une boîte dont l'affichage est optionnel. On y trouvera les lectures des senseurs de distance et position pour chaque robot ainsi que les commandes envoyées par la station au sol (R.C.1).

Dans une seconde page, il sera possible de visiter les missions précédentes (R.F.17) et de les classer par date et heure de la mission, temps, robots, physique/simulation et distance totale parcourue par les robots. Après sélection d'un vol précédent, dans une modale, les informations susmentionnées seront listées et la carte générée sera également affichée (R.F.18). Il sera aussi possible d'y voir, sur demande, les logs de la mission (R.C.1).


En centralisant toutes les informations pour la mission courante dans une seule page, l'opérateur aura une visibilité complète (Nielsen) sur le système, ce qui lui permettra de répondre rapidement et efficacement aux situations qui se présenteront. Les interactions seront aussi ainsi simplifiées et la charge cognitive en sera d'autant réduite. Toutefois, en séparant l'historique des missions et celle des logs dans des pages séparées, on sépare l'information selon l'utilisation nécessaire, ce qui réduira la quantité inutile de données lors de chaque type d'utilisation.

Les trois pages sont représentées aux figures suivantes :



Figure 6 : Représentation de la page d'accueil.

Missionpage



Zone de sécurité

Retour à la base

Lancer la mission

Terminer la mission

Historique des missions

Robot 1:

Identifier

Robot: OVNI

Type: Rover

État: Arrêté

Charge: 67%

☐ Position par défaut
Position en X

Position en Y

Robot 2:

Identifier

Robot: OVNI

Type: Rover

État: Arrêté

Charge: 67%

☐ Position par défaut
Position en X

Position en Y

< Mission >

1. Log text here 1
 2. Log text here 2
 3. Log text here 3
 4. Log text here 4

Figure 7 : Représentation de la page mission.


Historiquepage					
ID	Robot 1	Robot 2	Temps	Sim/Réel	Tot.Dist.(m)
1	OVNI	OVNI	03:20:14	Sim	12.31
2	OVNI	OVNI	05:10:27	Sim	21.31
3	OVNI	OVNI	04:21:14	Sim	8.31
4	OVNI	OVNI	07:36:46	Sim	32.31
ID	Robot 1	Robot 2	Temps	Sim/Réel	Tot.Dist.(m) ▾
1. Entrée de log 1 2. Entrée de log 2 3. Entrée de log 3 4. Entrée de log 4 5. Entrée de log 5 6. Entrée de log 6 7. Entrée de log 7					

Figure 8 : Représentation de la page d'historiques.

3.6 Fonctionnement général

Le projet est divisé en 3 dossiers séparés, soit master, client et server. Pour démarrer la base, il suffira de faire make. La commande compilera les trois conteneurs.

À l'aide d'une connexion SSH, les clients ROS seront démarrés automatiquement sur les rovers. Il ne sera donc pas nécessaire de manipuler les robots directement.

Après avoir démarré la base, il faudra se connecter avec le client (tablette, PC, téléphone intelligent, etc.) à l'adresse IP indiquée. L'utilisateur sera alors connecté à la page d'accueil où il pourra amorcer une nouvelle mission ou visiter les missions précédentes. Il pourra aussi décider s'il démarre une simulation ou une mission réelle.

En lançant la mission, réelle ou non, il sera redirigé vers la page de mission où cette dernière sera amorcée. Après quelques secondes, le système sera en marche et la carte sera mise à jour en conséquence. Si jamais l'utilisateur avait choisi une simulation, le logiciel Rviz sera exécuté afin de recevoir les informations à propos du robot dans la simulation.

Dans le cas où une mission a été lancée par un autre utilisateur et est déjà en cours, l'utilisateur sera redirigé vers la page de mission. Une fois sur cette page, il pourra terminer la mission en cours, rendre visible les logs, renvoyer les robots à la base ou afficher une zone de sécurité. Il pourra aussi visiter la page d'historique de mission.

Dans le cas où on souhaiterait faire les tests automatisés, il sera également possible d'exécuter « make tests ». On pourra aussi se déplacer dans les différents sous-répertoires et faire « make tests » pour les différentes composantes.

4. Processus de gestion

4.1 Estimations des coûts du projet

Pour le projet, les coûts reliés aux robots et à la station au sol ne sont pas à comptabiliser, puisque ce matériel sera fourni par l'Agence. De plus, tous les logiciels, outils et cadriciels utilisés sont gratuits. Il faut donc uniquement se préoccuper des coûts liés aux salaires des employés. L'équipe sera constituée de 6 employés qui seront rémunérés à l'heure selon l'entente contractuelle. L'équipe est constituée de cinq développeurs et d'un coordinateur de projet jouant aussi le rôle de « Scrum master ». Les développeurs travaillent à un taux horaire de 130\$ et le coordinateur de projet travaille à un taux horaire de 145\$.

Le projet sera étalé sur 14 semaines et nous avons un budget de 630 heures-personnes pour réaliser la totalité du projet. Cela représente donc 45 heures-personnes par semaine, ou plutôt 7.5 heures de travail pour chaque membre de l'équipe hebdomadairement. Le coût total du projet est donc :

$$\text{Coût} = 14 \text{ semaines} \times (1 \text{ coordo} \times 7.5h \times 145\$/h + 5 \text{ dev} \times 7.5h \times 130\$/h) = 83\,475\$$$

4.2 Planification des tâches

La figure suivante présente notre planification des tâches du projet. On peut y voir les dates de réalisation prévues pour chaque tâche ainsi que le nombre d'heures qui y sont allouées et à quelle équipe du projet elles sont attribuées. Les tâches en rouge seront complétées par l'équipe Backend (Mathilde et Vincent), les tâches en orange par l'équipe UI (Thomas et Yun Ji) et les tâches en vert seront réalisées par l'équipe Embarqué (Manel et Xavier). De plus, les tâches en mauve seront la responsabilité de tous et les tâches de plusieurs couleurs seront partagées entre les deux équipes.

Par ailleurs, nous avons gardé une marge de 39 heures non allouées pour se donner le temps d'adresser les différents bugs auxquels nous pourrions faire face, principalement vers la fin du projet.

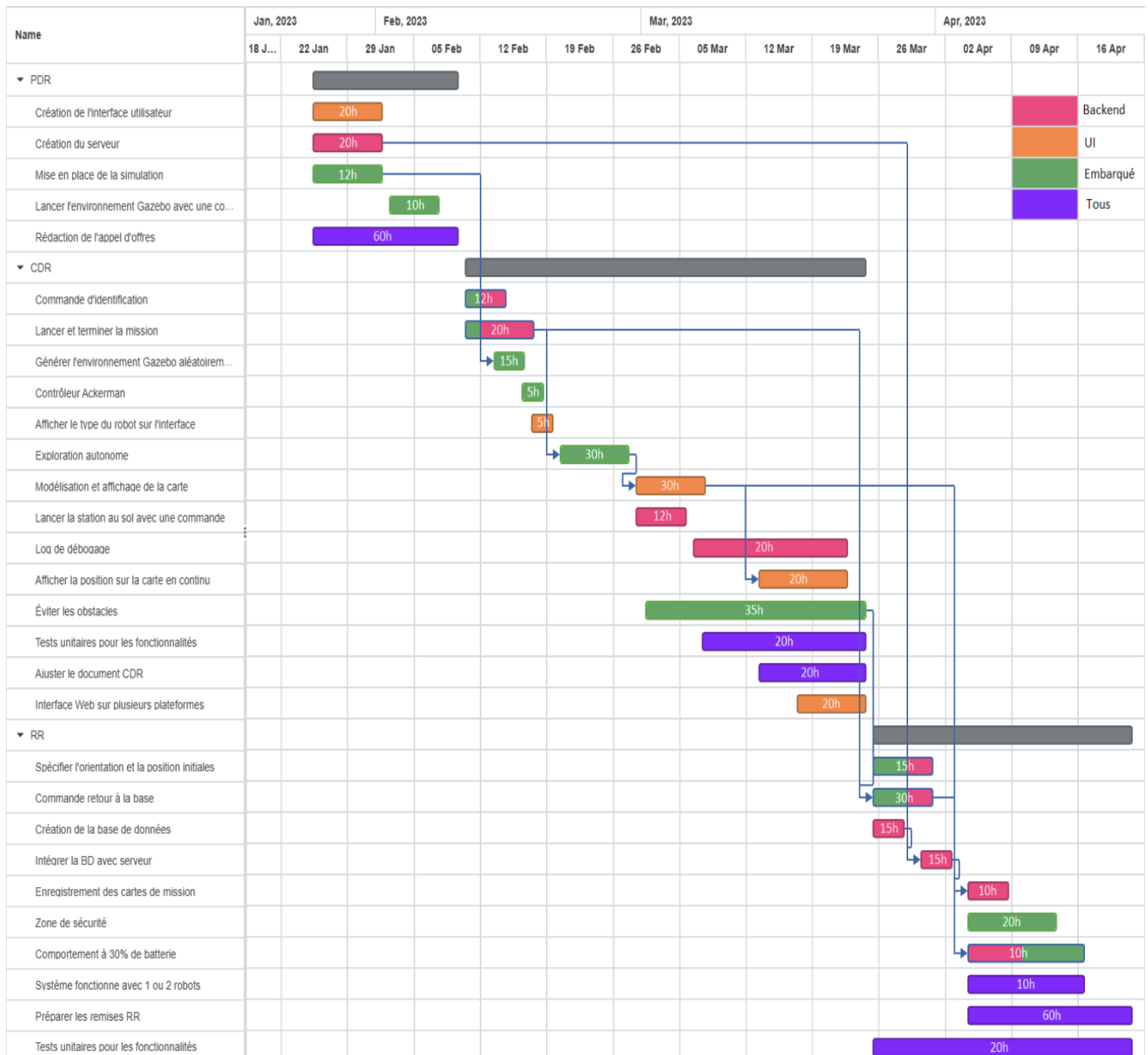


Figure 9 : Diagramme de Gantt de la planification des tâches et de leur allocation

4.3 Calendrier de projet

Jalon	Description	Date
Preliminary Design Review + Réponse à l'appel d'offres	Remise d'un prototype préliminaire minimal et du plan de projet complet. Document à remettre + démonstrations sur robots physiques et sur simulation	Vendredi 10 février 2023
Critical Design Review	Remise d'un système avec fonctionnement partiel. Document ajusté à remettre + démonstration des requis fonctionnels, de conception et de qualité.	Vendredi 17 mars 2023
Readiness Review	Livrable final complet avec toutes les fonctionnalités du système. Rapport final à remettre + évaluation du produit final + présentation orale	Vendredi 21 avril 2023

Tableau 2 : Date cible des jalons du projet

4.4 Ressources humaines du projet

Notre équipe est composée de 6 membres ayant chacun des expertises différentes :

- **Thomas** possède de l'expérience en développement Web tant en interface utilisateur que du côté serveur. Il possède une expérience comme étant « scrum master » ce qui lui permet de gérer les rapport d'avancement et les rencontres scrum.
- **Vincent** a de l'expérience en développement Back-end et a des connaissances de haut niveau pour l'ensemble des technologies utilisées. Il a aussi de l'expérience en équipe Agile. Il pourra facilement s'adapter aux exigences du projet et consolider l'équipe où il y aura des faiblesses.
- **Mathilde** possède de l'expérience en développement web avec Angular et back-end qui lui sera utile pour construire le serveur du système. Elle a également de l'expérience de travail en équipe selon la méthodologie Agile.
- **Yun Ji** a de l'expérience en assurance qualité. Il pourra implémenter des tests et fournir une assurance suffisante de la qualité logicielle de notre projet soit conforme aux exigences et aux attentes établies. D'ailleurs, il a de l'expérience de travail en équipe selon la méthodologie Agile.
- **Manel** possède de l'expérience en développement web, en gestion de contenu et bases de données ainsi que certaines connaissances en robotique qui pourraient s'avérer utiles tout le long du projet. Par ailleurs, elle possède également de l'expérience de travail en équipe suivant la méthode Agile.
- **Xavier** est expérimenté tant dans le développement web que dans les systèmes linux embarqués. Il a travaillé plusieurs années en entreprise et par travail autonome sur le déploiement de diverses applications web et sur la gestion de serveurs. Il a aussi une certaine expertise dans les systèmes linux embarqués après plusieurs années d'expérience professionnelle sur de tels systèmes.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité

Afin d'assurer la qualité de notre produit, nous avons établi des règles de contrôle de la qualité dès le début et durant toute la durée du projet. Une première règle que nous avons établie consiste à suivre les normes pep8 [18] pour notre style de code en python, et le guide Airbnb pour le code en JavaScript [19]. Cela a pour but de garantir une cohérence de style entre tous les membres et toutes les parties logicielles embarquées.

Ensuite, une étape de notre processus de contrôle de qualité consiste à mettre en place un pipeline de développement sur GitLab pour s'assurer que l'intégration continue des différentes parties de notre code se fasse de manière rigoureuse. Ce dernier sera composé d'une première étape qui vient compiler les différents fichiers ajoutés, suivi ensuite d'une seconde étape où un "linter" vérifiera que les standards de codage sont respectés, ensuite une étape de "build" qui vient s'assurer que le projet se construit et compile correctement, puis finalement une dernière étape aura pour but de vérifier que les différents tests unitaires passent pour toutes les composantes logicielles. Il a également été convenu que, avant d'ajouter une fonctionnalité logicielle au code principal, le fonctionnement de cette dernière doit être entièrement validé à l'aide de tests unitaires afin de garantir que tous les cas particuliers soient couverts.

Finalement, une fois les tests écrits et le pipeline passés, une dernière étape de révision de code est effectuée. Le code ajouté doit être révisé et approuvé par au moins deux personnes n'ayant pas participé à la conception de la fonctionnalité, qui se chargent de relire les modifications, s'assurer du respect des normes de codages établies, et donner une rétroaction sur les éléments à retravailler s'il y en a, avant d'être accepté.

Une fois toutes ces étapes passées, le code peut être jugé comme respectant les règles de contrôle de qualité et peut être ajouté au code source.

5.2 Gestion de risque

Un des risques principaux de ce projet est la qualité de nos robots. En effet, il est possible qu'il y a un bris accidentel ou une perte des matériels physiques au cours du développement embarqué. Effectivement, les drones et le rover fournis par l'agence (Polytechnique) ne sont pas assez résistants à toutes les chutes et toutes les collisions contre les murs. Il est difficile de garantir le bon fonctionnement des robots tout au long du développement surtout lorsqu'on manque d'expérience. Pour régler ce problème, tous les membres d'équipe sont responsables et prudents à l'utilisation. Par exemple, on doit tester le drone à une altitude raisonnable. Il ne faut pas sortir le robot de la salle. Au pire des cas, le robot dysfonctionnel sera remplacé par un nouveau robot, mais il y aura un coût supplémentaire pour notre agence, soit de 2900\$ environ.

Un autre risque potentiel est les vulnérabilités face aux risques de cybersécurité. En effet, il est possible qu'un pirate informatique fait une attaque sur notre serveur. S'il réussit à obtenir le droit admin, il peut falsifier ou retirer les données dans notre base de données, ce qui peut causer des problèmes lors de l'affichage des données dans le requis R.F.17. Une contre mesure possible est de ne pas utiliser des bibliothèques avec des failles de sécurité sur le serveur. D'ailleurs, il n'y a pas de risques dans les autres sous-systèmes parce qu'ils ne requièrent pas des données sensibles.

Dans un dernier temps, le manque de communication et l'organisation est évidemment un risque dans ce projet de grande complexité. L'environnement de travail de chaque membre de l'équipe est différent parce que les horaires des membres de l'équipe ne sont pas très compatibles, ce qui rend plus difficile d'organiser des rencontres. Pour éviter les retards, nous avons créé un diagramme de Gantt afin de planifier toutes les tâches assignées à chacun et les dates limites. Évidemment, les décisions ne sont pas coulées dans le béton. Il pourra y avoir des ajustements et des nouvelles planifications au fur et à mesure du développement. De plus, grâce à ce diagramme, chaque membre est au courant des tâches sur lesquelles les autres membres travaillent. De cette façon, on obtient une bonne estimation dans l'ensemble de projet. De plus, des scrums sont organisés chaque semaine pour discuter de l'avancement de chacun et des travaux à faire.

5.3 Tests

Comme mentionné précédemment à la section 5.1, le produit a été rigoureusement testé avant chaque date de livraison, en plus d'avoir une exécution de tests grâce à un pipeline sur GitLab avant de fusionner les branches.

Pour la partie front end de la station au sol, nous utilisons Jasmine [20] afin d'effectuer des tests unitaires, ce qui permet en premier lieu de nous assurer d'avoir une logique d'application robuste avec une interface fonctionnelle, mais aussi de tester le bon fonctionnement des requêtes envoyées au serveur ainsi que leur temps d'exécution.

Pour la partie backend, nous effectuons les tests unitaires avec Mocha, Sinon et Chai. Nous pouvons alors tester le bon fonctionnement des requêtes vers la base de données, la réception des requêtes provenant du frontend, ou encore la communication avec les robots et la simulation.

Les tests de chaque composante (front end ou backend) consistent en une suite de tests pour chaque méthode qui s'assurent du bon fonctionnement de ces dernières sous toutes les conditions, incluant par exemple le changement d'états, l'appel d'autres fonctions et méthodes, l'utilisation de bons paramètres, etc.

On teste également la communication entre les différentes entités du projet. Pour les communications client/serveur, nous utilisons la suite «Express» afin de simuler les requêtes HTTP. Pour les communications reliées à ROS, nous utilisons la suite «unittest» jumelée aux fonctionnalités de ROS. Ces communications impliquent l'ensemble des «subscribers» et «publishers» pour tous les «nodes» utilisés dans la solution.

Concernant la simulation, plusieurs tests usagers ont été effectués avec celle-ci, notamment grâce à la possibilité de générer des environnements aléatoires, ce qui nous permet de valider le bon fonctionnement de l'exploration avant de l'implémenter sur les vrais robots. Ces tests se concentreront principalement sur la détection d'obstacles, la gestion des collisions, le fonctionnement des capteurs ainsi que la capacité à générer des cartes de l'environnement.

Pour les tests de matériel, il faut tester d'abord que les rovers soient fonctionnels en arrivant. De plus, étant donné la complexité des tâches ainsi que la diversité des technologies impliquées, nous avons effectué des tests ayant comme objectifs de

vérifier le comportement de la solution. L'ensemble de ces tests seront sous forme de protocole à suivre et demanderont 2 exécutions indépendantes afin d'assurer la qualité des résultats. On teste alors que les rovers exécutent les commandes avec les bonnes instructions, et respectent les contraintes. Finalement, il faut aussi tester que les capteurs envoient les données au serveur.

5.4 Gestion de configuration

La gestion de configuration se divise en plusieurs aspects: le contrôle de version, l'organisation du code source, l'organisation des tests, la documentation du code source et la documentation de conception.

Le contrôle de version a été effectué à l'aide de Git avec l'outil Gitlab. Le projet est composé principalement de la branche «main» et de la branche « dev ». La première sera porter le développement final pour les différents sprints du projet tandis que la deuxième sera la branche principale de développement. Les branches de travail de chaque membre de l'équipe sont basées sur celle-ci.

L'outil Gitlab nous permet aussi d'organiser les tâches techniques à effectuer pendant chaque sprint selon la méthodologie «Kanban». Les tâches sont affichées dans un tableau divisé comme suit: « Open », « Ongoing », « Blocked », « In Review » et « Done ». Plus en particulier, voici ce que chaque statut veut dire:

- Open: La tâche est disponible.
- Ongoing: La tâche est assignée à quelqu'un.
- Blocked: La tâche est assignée, mais un élément bloquant empêche la complétion de la tâche.
- In Review: La tâche est terminée et une requête de tirage (MR) est en attente de révision.
- Done: La tâche est terminée et sa MR a été acceptée.

Au cours des rencontres hebdomadaires, les tâches sont attribuées. Celles qui ne sont pas attribuées peuvent être récupérées par les membres afin d'être complétées. Les membres peuvent aussi ajouter des tâches eux-mêmes afin de garder une trace des tâches effectuées qui ne faisaient pas partie des tâches initialement créées.

Afin d'apporter les modifications terminées à la branche de développement, le membre souhaitant migrer ses modifications doit ouvrir une requête de tirage afin que ses changements soient vérifiés par un collègue. Il n'est pas possible pour un individu d'apporter des modifications et d'approuver sa propre requête. Chacune des requêtes doit être accompagnée des tests appropriés. Les requêtes sans tests ne sont pas acceptées. L'utilisation de «tags» sera mise de l'avant afin de facilement retrouver les versions fonctionnelles du projet.

L'organisation du code source dans le répertoire Git a été effectuée comme suit: Le répertoire contient l'application web dans un dossier, et le code associé aux robots dans un autre.

Pour l'application web, deux dossiers contiennent d'un côté le serveur dynamique, de l'autre le serveur statique. Le serveur client, responsable de l'interface client, contient les composants Angular ainsi que l'ensemble des tests associés. Le côté serveur

dynamique contient le code ainsi que ses tests. Les fichiers de données sont aussi entreposés dans le serveur sous forme de base de données.

Afin de rendre la solution portable et accessible, nous avons décidé que les différents systèmes seront conteneurisés. Le client et la station ont leur propre conteneur. Le conteneur de la station est composé du serveur et du contrôleur ROS. Ces conteneurs peuvent être créés à l'aide d'un fichier Makefile

Pour la documentation de conception, celle-ci est disponible dans la racine de la solution sous format PDF, où les diagrammes de conception, les technologies utilisées ainsi que la documentation explicative sont entreposés et disponibles.

5.5 Déroulement du projet

Dès le début de la phase de conception de notre projet, notre équipe a su faire preuve de bonnes initiative et adaptation. En effet, l'entièreté des membres de l'équipe ont pu rapidement se familiariser avec les requis et les technologies demandées et se lancer dans la réalisation du projet, en exploitant au maximum les points forts de chacun. La bonne communication entre les membres de l'équipe a également permis de favoriser l'entraide et de ne pas perdre trop de temps lorsque des difficultés ont été rencontrées. Les Scrum réguliers ont été une bonne initiative et ont permis à tout le monde d'être au courant des avancements sur tous les volets du projet ainsi que d'apporter des suggestions ou de l'aide sur les problèmes rencontrés lorsque nécessaire.

Cependant, notre équipe a également rencontré certaines difficultés, notamment dans le respect de l'échéancier et la gestion de projet sur GitLab. Nous avons eu du retard sur une partie des tâches que nous avions prévu pour le CDR, dues à des problèmes imprévus et des estimations non réalistes que nous avons faites avec notre bas niveau de connaissances initiales. Par ailleurs, le retour quant à la provision des fichiers de support (simulation, etc.) pour le Cognify, nous avons décidé de concentrer nos efforts seulement sur les rovers. Nous avons donc dû faire des réestimations d'échéanciers et décaler certaines tâches initialement prévues pour le CDR vers le RR. De plus, malgré le fait que nous avions initialement prévu de compléter les requis RF11 et RF13, nous avons dû les mettre de côté étant donné qu'ils concernaient les Cognify.

La gestion de projet GitLab a également été légèrement négligée durant les deux premiers sprints lorsqu'il s'agissait d'assigner des tâches à des personnes spécifiques sur GitLab, mais nous avons rectifié le tir depuis le CDR. Les requêtes de tirage et la révision de code sur GitLab a cependant été parfaitement respectée et aucun code n'a été fusionné sans être révisé par deux membres de l'équipe. Ceci a permis de grandement minimiser les problèmes introduits dans la branche principale.

De façon générale, le projet s'est assez bien déroulé. Nous avons rencontré certains retards d'échéancier à cause du changement drastique de requis. Ceux-ci ont tous été rattrapés lors des dernières semaines de conception.

6. Résultats des tests de fonctionnement du système complet

Les composantes implémentées ont toutes été testées, que ce soit par test unitaire, test d'intégration ou tests usagers. Certaines composantes ont été plus compliquées à implémenter et à déboguer que d'autres. À titre d'exemple, le bon fonctionnement des robots physiques a été dur à valider, particulièrement lorsqu'il s'agissait de détecter et éviter des obstacles. Cela est dû à plusieurs raisons telles que le manque d'expérience dans la manipulation des senseurs disponibles, certains problèmes dans les fichiers fournis par Agilex, ou encore l'accès restreint à la volière.

Par ailleurs, grâce à plusieurs tests, nous pouvons affirmer que le reste des composantes fonctionne très bien. L'interface de la station au sol est fonctionnelle et permet, dans un premier temps, de correctement lancer une mission, identifier les limos participant à celle-ci, ainsi que produire et afficher la carte générée grâce aux capteurs des robots. Une fois la mission lancée, les logs de débogage de la mission sont affichés sur la page, afin de recevoir de l'information en temps réel sur la position et l'orientation des robots. Dans un second temps, l'interface permet également de bien sauvegarder les logs localement sur le serveur et d'y avoir accès sur la page d'historique des missions.

Une base de données a également été ajoutée lors du dernier sprint sous forme de fichier SQLite disponible sur le serveur, ce qui permet d'avoir des données persistantes sur les missions. Par exemple, la durée de la mission, la distance parcourue, la date, etc. peuvent être visualisées sur la page d'historique des missions sous forme de tableau d'extension.

L'affichage de la carte a également été amélioré et garde un bon ratio lors de l'affichage. Cette dernière est enregistrée localement sur le serveur à chaque fin de mission et on peut ainsi la visualiser avec les logs lorsque nécessaire sur la page d'historique des missions, une fois qu'une mission a été sélectionnée.

7. Travaux futurs et recommandations

Afin d'améliorer la performance de notre exploration autonome, il aurait été intéressant d'ajouter des obstacles dynamiques représentant la position des robots à chaque instant. En effet, comme l'exploration par frontière de chaque robot se fait de manière indépendante, les robots ne sont pas conscients de la position de l'autre robot lors de l'exploration et peuvent seulement le détecter lorsqu'il apparaît dans leur capteur. Ainsi, comme les robots ont souvent des trajectoires similaires et tentent de suivre des chemins qui entrent en conflit,

ajouter des obstacles dynamiques aurait fait en sorte que les robots auraient pu prévoir des trajectoires plus stables en s'assurant d'éviter toute collision entre les robots.

Il aurait également été avantageux et convivial d'avoir des retours visuels plus explicites sur l'interface utilisateur. Par exemple, on aurait pu avoir le contour de la zone de sécurité affichée sur la carte, recevoir des messages d'erreurs lorsqu'une requête échoue, ou encore avoir un avertissement lorsque le niveau de batterie est trop bas.

Il serait également intéressant d'ajouter des tests automatisés pour d'autres composantes du système, par exemple pour s'assurer que les robots se déplacent correctement aux positions indiquées, notamment pour la zone de sécurité et le retour à la base.

Pour les possibles extensions du système, avoir une communication directe entre les robots aurait pu être intéressante à ajouter, en plus d'utiliser des types de robots différents (drone et rover). Par ailleurs, si plus de robots sont disponibles, lancer plusieurs missions en parallèle pourrait être un ajout pertinent. Il serait également intéressant d'utiliser l'ensemble des fonctionnalités du matériel fourni dans nos différents algorithmes. Par exemple, on pourrait utiliser les caméras des Rovers pour la reconnaissance de l'environnement complémentirement aux lidars.

Par ailleurs, les robots ayant certaines spécificités (zéro pour le module inertiel, la correction pour `cmd_vel`, autres défauts dans les capteurs, etc.), il aurait également été pertinent de collecter des données de calibration pour chacun des robots, ce qui aurait permis une accélération de la recherche en réduisant l'erreur des mesures.

Concernant les recommandations, l'utilisation de matériel plus complet et en plus grande quantité, surtout pour les drones (avoir des caméras et des LED dès le début du projet par exemple) aurait grandement amélioré l'expérience de travail. Également, avoir tous les modèles de simulation et les ressources GitHub disponibles ainsi que des documents de référence mis à jour dès le départ. Ainsi, nous aurions eu plus de temps pour nous familiariser avec les outils avant d'entamer la conception, ce qui aurait grandement impacté l'efficacité de la conception du projet.

8. Apprentissage continu

Xavier L'Heureux

Xavier avait une propension à travailler en silo et à préférer résoudre les problèmes par lui-même plutôt que de demander de l'aide de ses partenaires. Cela rendait l'équipe plus à risque, puisqu'en cas de goulot d'étranglement, personne ne pouvait l'aider. Il l'a réalisé durant le projet et s'est assuré de déléguer des tâches à d'autres membres pour la suite et à ne servir que comme

référence pour les aspects techniques. Il s'est aussi assuré qu'il y avait une seconde personne assignée dans la quasi-totalité des « issues » restantes pour diviser le travail plus facilement.

Thomas Moorjani-Houle

Thomas n'avait aucune connaissance en robotique, peu d'expérience avec le système d'exploitation Linux et la conteneurisation. Pour acquérir de la connaissance en robotique, Thomas s'est servi des connaissances de ses coéquipiers pour se familiariser avec le sujet. Pour la conteneurisation Docker et le système d'exploitation Linux, Thomas a fait plusieurs lectures de la documentation sur ces sujets.

Mathilde Brosseau

Mathilde n'avait absolument aucune connaissance en systèmes robotiques et en conteneurisation par Docker avant le début du projet. Malgré son manque d'expertise, elle a tout de même décidé de prendre en charge des tâches reliées à ces aspects afin de développer ses compétences. Afin d'y arriver, elle a demandé de l'aide aux membres de l'équipe plus qualifiés et fait plusieurs lectures dans la documentation afin de se renseigner et comprendre le fonctionnement de ROS.

Manel Keddami

Manel avait peu de connaissances en robotique et aucune connaissance en conteneurisation Docker et en l'utilisation de ROS nodes et de Gazebo. Pour y remédier, elle a dû prendre le temps de se renseigner sur ces outils à l'aide de la documentation officielle, ou en demandant de l'aide à certains membres de l'équipe qui sont plus familiers. Elle a également essayé de partiellement prendre en charge des tâches utilisant ces technologies afin de renforcer son apprentissage.

Yun Ji Liao

Yun Ji n'avait travaillé que sur des projets très rapprochés du système embarqué ou sur des projets de développement web. Il n'avait jamais fait le pont entre les deux. Pour y arriver, il a demandé l'aide de ses collègues et a effectué ses recherches sur les pages de ROS. Bien qu'il possède des connaissances en frontend, il n'est pas expert dans ce domaine. Il a profité de l'opportunité du projet pour développer son expertise tout en consultant ses collègues afin de tirer avantage de leur expérience et de leurs connaissances sur ce sujet, tout en contribuant à l'élaboration du projet.

Vincent Haney

Vincent n'avait pas d'expérience en robotique sur un projet d'une telle envergure. Il s'est renseigné en lisant la documentation ROS et la documentation d'Agilex afin de pouvoir s'occuper d'une partie du backend. Il a aussi consulté ses coéquipiers afin d'en apprendre davantage sur les nouvelles connaissances qu'eux-mêmes ont acquises afin de profiter de l'expérience de projet et de faciliter le partage de connaissances dans l'équipe. Il a aussi choisi des tâches

dans lesquelles il n'avait aucune expérience afin d'en apprendre sur des nouveaux sujets et d'élargir ses champs d'expertise.

9. Conclusion

Lors du dépôt de la réponse à l'appel d'offre, la vision du projet était claire, mais la réalisation était vague. Ceci s'est traduit en une mauvaise planification des tâches, surtout pour le CDR, et en quelques changements de conception. Par exemple, nous avons initialement prévu de faire un plus grand nombre de requis, dont l'utilisation de drones que nous avons finalement abandonné à cause d'un manque de ressources.

Nous nous sommes donc décidés à changer pour l'utilisation de deux Limos, afin de nous concentrer sur le reste des tâches que nous avons choisies. Cependant, un temps considérable a quand même été accordé à ces tâches, ce qui a causé un certain retard. Néanmoins, notre conception s'est assez bien déroulée de façon générale et toutes les fonctionnalités clés du projet ont été complétées. Certains membres se sont trop spécialisés dans un type de tâches vers la fin du projet et il est devenu trop complexe d'effectuer des tâches différentes sans engendrer des délais excessifs.

Un point qui aurait pu être amélioré est la prise en charge de tâches un peu plus différentes pour chacun afin de faciliter l'entraide et varier les connaissances et apprentissage, et l'ajout d'une couverture de code plus complète dès le départ et au fur et à mesure de l'avancement du projet.

Malgré les difficultés, nous avons tout de même réussi à compléter l'ensemble des requis, et même plus. Nous avons mis de l'avant un produit de qualité dont le fonctionnement regroupe les fonctionnalités demandées de façon propre et organisée, tout en conservant les aspects techniques importants. Nous sommes allés plus loin que nécessaire sur certains aspects comme l'interface Web et le serveur ROS afin de présenter un produit au-delà des attentes.

10. Références

- [1] Agilex Robotics (2023) limo-doc (1.0.0) [En ligne]. Disponible : [https://github.com/agilexrobotics/limo-doc/blob/master/Limo%20user%20manual\(EN\).md](https://github.com/agilexrobotics/limo-doc/blob/master/Limo%20user%20manual(EN).md)
- [2] Cartographer ROS (2022). Cartographer ROS Integration. [En ligne]. Disponible : <https://google-cartographer-ros.readthedocs.io/en/latest/>
- [3] Gazebo (s.d.). Gazebo Documentation. [En ligne]. Disponible : <https://github.com/gazebo/gazebo-sim/docs>
- [4] Atlassian (s.d.). What is Agile ? [En ligne]. Disponible : <https://www.atlassian.com/agile>
- [5] Angular (2023). Introduction to the Angular docs. [En ligne]. Disponible <https://angular.io/docs>
- [6] Mozilla. (2023) CanvasRenderingContext2D. [En ligne]. Disponible <https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D>
- [7] Node.js (s.d.). NodeJS v19.8.1 Documentation. [En ligne]. Disponible <https://nodejs.org/api/>
- [8] ROS (2022) Documentation. [En ligne]. Disponible <http://wiki.ros.org/Documentation>
- [9] Mozilla (2023). The WebSocket API. [En ligne]. Disponible https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
- [10] SQLite (s.d.). SQLite Documentation. [En ligne]. Disponible <https://sqlite.org/docs.html>
- [11] GNU make (2023). GNU make. [En ligne]. Disponible <https://www.gnu.org/software/make/manual/make.html>
- [12] Linux (2023). ALSA [En ligne]. Disponible https://alsa-project.org/wiki/Main_Page
- [13] ROS (2018). Movebase [En ligne]. Disponible http://wiki.ros.org/move_base
- [14] ROS (2018). TEBLocalPlanner [En ligne]. Disponible http://wiki.ros.org/teb_local_planner
- [15] ROS (2018). Explorelite [En ligne]. Disponible http://wiki.ros.org/explore_lite
- [16] YDLIDAR (2023). YDLIDAR ROS driver [En ligne]. Disponible https://github.com/YDLIDAR/ydlidar_ros_driver
- [17] ROS (2018). Rviz [En ligne]. Disponible <http://wiki.ros.org/rviz>
- [18] Python enhancement proposals. (2023). PEP 8 - Style Guide for Python Code. [En ligne]. Disponible <https://peps.python.org/pep-0008/>

[19] Airbnb. (s.d.). Airbnb JavaScript Style Guide. [En ligne]. Disponible : <https://airbnb.io/javascript/react/>

[20] Santiago G. (2017) Angular: Unit Testing Jasmine, Karma (step by step). [En ligne] Disponible <https://medium.com/swlh/angular-unit-testing-jasmine-karma-step-by-step-e3376d110ab4>