



**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. H2023 – INF3995 du département GIGL

Conception d'un système aérien 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

10 Février 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	5
2.1 Structure d'organisation	5
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	15
4. Processus de gestion	15
4.1 Estimations des coûts du projet	15
4.2 Planification des tâches	15
4.3 Calendrier de projet	17
4.4 Ressources humaines du projet	18
5. Suivi de projet et contrôle	19
5.1 Contrôle de la qualité	19
5.2 Gestion de risque	19
5.3 Tests	20
5.4 Gestion de configuration	21
6. Références	23

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 un rover AgileX Limo et un drone CogniFly. 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 utilisera les deux types de robots disponibles, soit un rover AgileX Limo et un drone CogniFly. 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 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 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.

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. 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. 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 lundis et jeudis. Les Scrum ainsi que les notes 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). Yun Ji se concentre aussi sur l'assurance de la qualité des produits, tandis que Manel et Xavier s'occupent de la simulation et du bon fonctionnement des robots. Xavier a 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 temps plus frais, ou contrat à terme. Ce type de contrat offre plus de flexibilité lors de la conception et réduit les négociations préliminaires du projet. Il est également idéal lorsqu'on désire commencer un projet sans que toutes les spécifications soient déterminées. En effet, nous proposons ce contrat car bien que les requis soient pour la majorité spécifiés, le produit final n'est pas forcément définitif. Nous nous engageons donc à fournir un produit concurrentiel respectant l'appel d'offre, tout en offrant le plus de fonctionnalités additionnelles possibles. De plus, comme le projet est de courte durée, il serait idéal pour nous de commencer la conception et se mettre au travail au plus tôt possible, ce qui est possible étant donné que ce type de contrat réduit les négociations préliminaires du projet, comme mentionné préalablement.

Pour des projets de cette nature, il peut être assez compliqué de donner des estimations de temps exactes, car plusieurs imprévus peuvent survenir, autant en période de conception qu'en période de tests. Nous sommes cependant confiants d'être en mesure de gérer ces imprévus et de fournir un résultat fiable à la fin du projet. Ainsi, au lieu de donner des estimés de dates exactes pour chacune des fonctionnalités, l'avancement du projet sera fourni en 3 étapes, où nous nous engageons à fournir le meilleur produit possible pour chacune. Nous aurons également un estimé de 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.

3. Description de la solution

3.1 Architecture logicielle générale

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

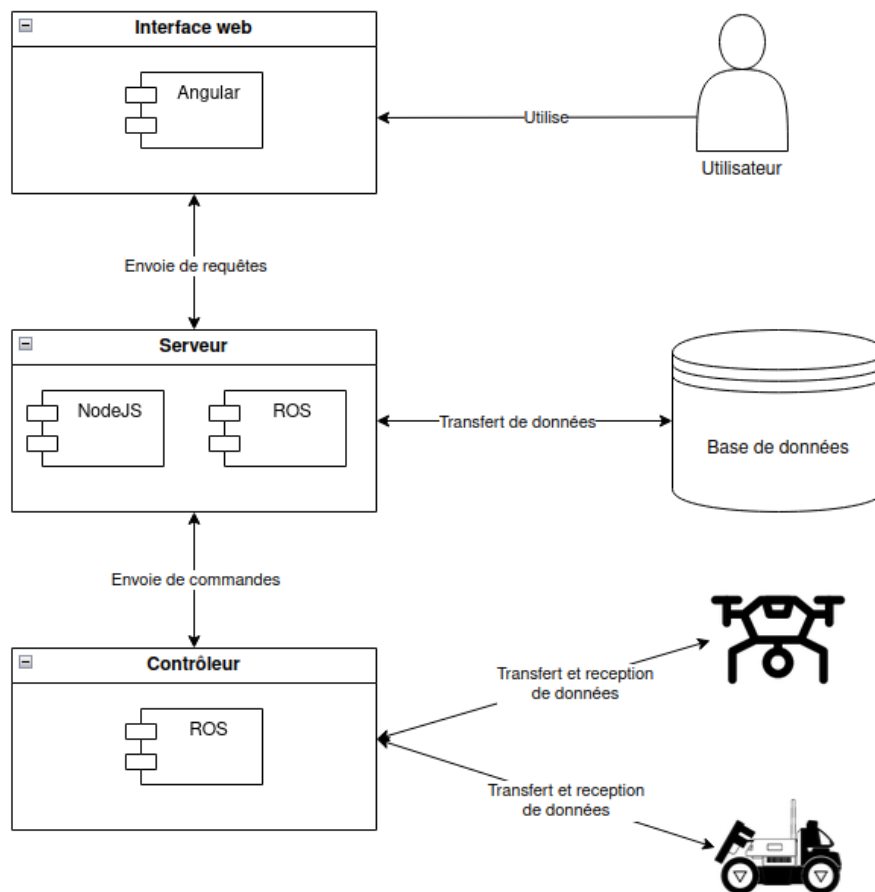


Figure 1 : Diagramme de l'architecture générale logicielle

Afin de concevoir notre interface web sur plusieurs types d'appareils (requis R.F.10) nous avons décidé d'implémenter celle-ci avec Angular, qui offre déjà plusieurs modules que l'on peut directement utiliser et adapter aux différents appareils. Pour le lancement

des missions (R.F.2), l'interface communique avec un serveur NodeJS via des requêtes HTTP, qui lui communique avec les robots en utilisant des ROS nodes.

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 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.

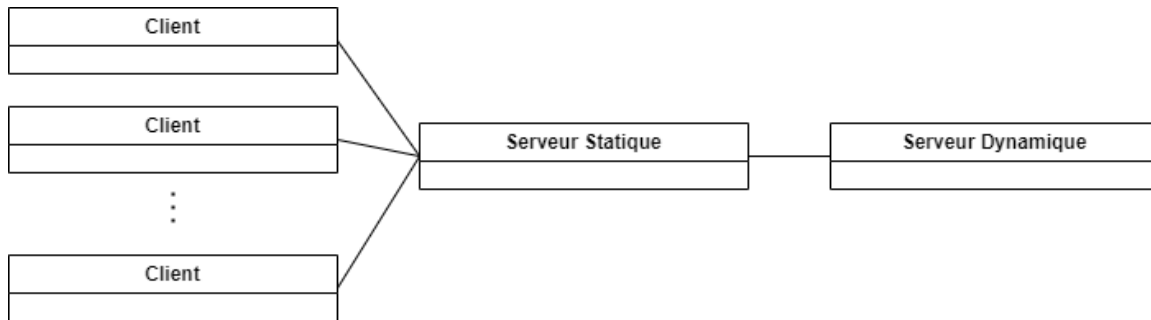


Figure 2 : Diagramme de l'architecture des communications client-serveur

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 que 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 « `npm start` ». 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.

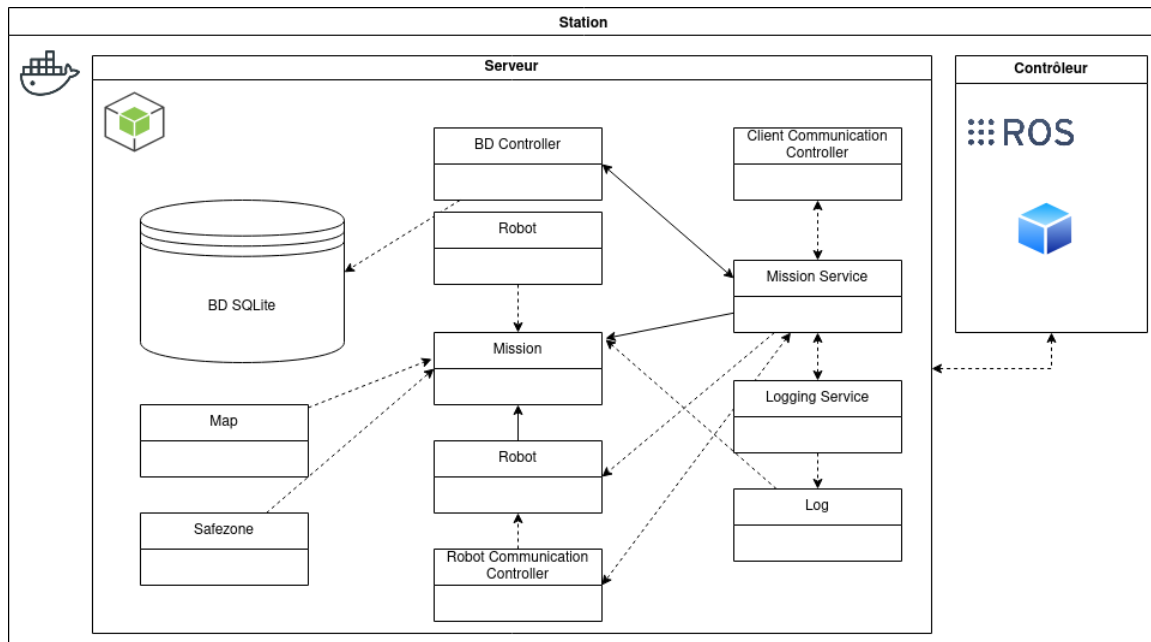


Figure 3 : Diagramme de l'architecture de la station au sol

D'abord, nous avons choisi une base de données accessible à l'aide d'une classe contrôleur accompagnée d'un service. La classe qui aura accès à ce contrôleur est la classe « Mission Service » qui a comme responsabilité de gérer les transactions d'information entre le client, les robots et la base de données.

Ensuite, la classe « Mission Service » aura comme responsabilité de rechercher les robots sur le réseau, établir la communication avec les robots et créer les instances de « Robot ». Ces « Robots » pourront ensuite être placés dans des « Mission ».

Les « Robot » existent alors indépendamment des missions. Ils sont ajoutés à une mission lorsqu'elle est amorcée et libérés lorsqu'elle se termine. Tout au cours de la mission, le serveur reçoit les informations des robots à propos de leur environnement, leur position ainsi que leur état. L'état reçu permet alors d'envoyer un message aux clients, satisfaisant le requis fonctionnel 13.

Les « Mission » seront amorcées par un client à l'aide d'une requête HTTP envoyée vers le serveur et reçue par la classe « Client Communication Controller » 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 déterminée au départ (R.F.12). Chacune des « Mission » sera accompagnée d'une carte sous la forme d'un objet « Map ». Ces « Mission » contiendront les données nécessaires aux différents requis nécessitant de l'information sur les missions ainsi qu'un log dans lequel les informations liées seront enregistrées au cours des missions à une fréquence de 1 Hz (R.C.1). Dès qu'une mission sera terminée, le client le signalera au serveur et un log sera enregistré à l'aide de « Logging Service ». Ce log ainsi que l'information relative à la mission seront alors envoyés à la base de données à travers « BD Controller ». Ceci permettra de répondre au requis R.F.8.

Les cartes « Map » sont reçues des robots et envoyées aux clients périodiquement sous forme de polygones provenant de l'algorithme SLAM. Cette forme brute de données permet aux communications de limiter les ressources qu'elles nécessitent.

Le système est muni de deux classes de communication: « Client Communication Controller » et « Robot Communication Controller ». Le premier sert à communiquer avec les interfaces clients et le deuxième avec les robots.

La base de données (R.F.17) va contenir une ou plusieurs tables et des répertoires contenant les fichiers de logs et les cartes. Ces artefacts seront associés aux tables à l'aide de liens (R.F.18).

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
- Possibilité d'avoir un nombre virtuellement illimité d'utilisateurs connectés (R.F.10)

3.3 Logiciel embarqué

Le schéma de notre logiciel embarqué est représenté dans la figure suivante :

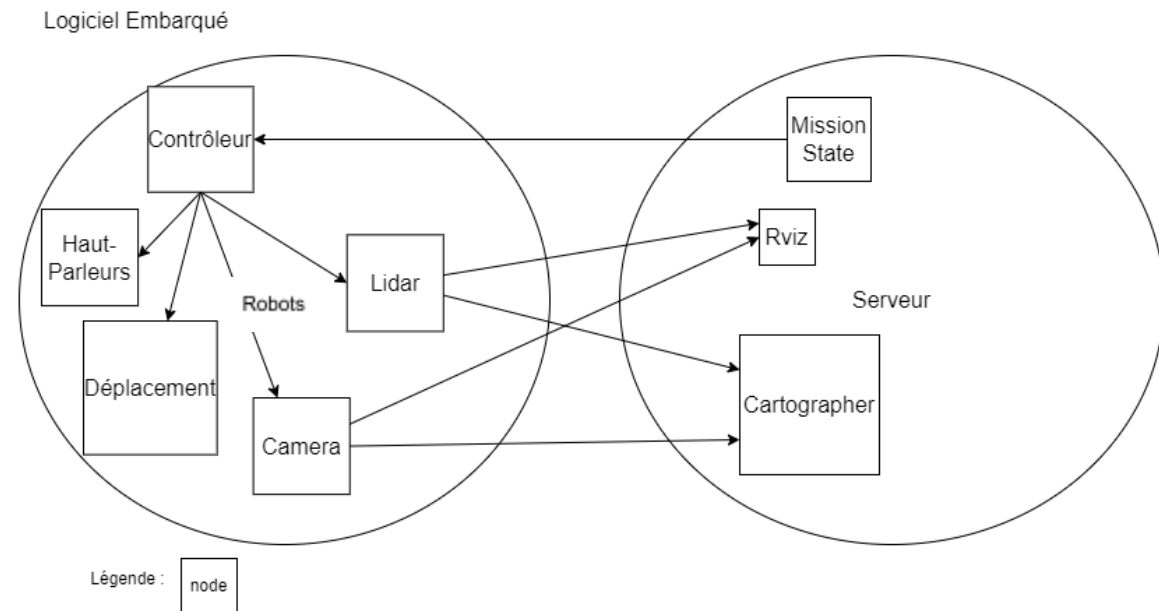


Figure 4 : Diagramme d'architecture du logiciel embarqué

Notre logiciel embarqué est basé sur ROS et son système de nodes. Il contient la node « MissionState » qui publie sur un topic pour dire à la node « Contrôleur » (R.L.3) si, par exemple, la mission est arrêtée ou en cours (R.F.2).

Ensuite, la node « Contrôleur » publie des topics aux différents capteurs et périphériques du robot qui vont effectuer des actions selon l'état de la mission (R.F.4). Lorsque la mission est en cours, la caméra et le lidar pour le Limo vont publier de l'information à la node Cartographe pour la modélisation de la carte (R.F.8) de l'environnement et à Rviz si on est en simulation [1]. Les nodes faisant partie du serveur interagissent avec notre station sol pour obtenir l'information envoyer aux nodes (R.C.1). Les connexions entre la station au sol et le logiciel embarqué est via un réseau sans-fil fourni par l'Agence spatiale (R.M.2).

3.4 Simulation

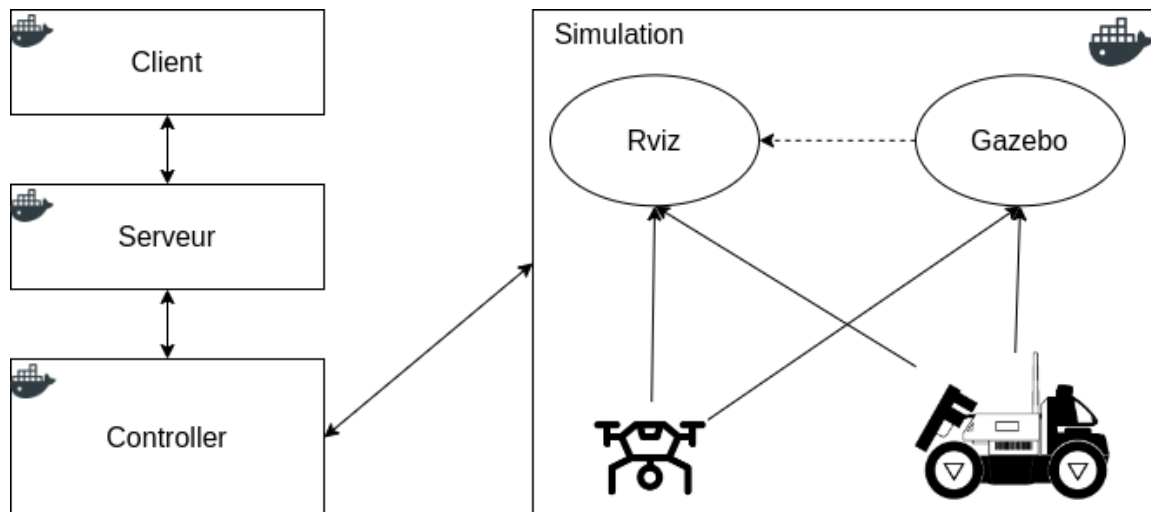


Figure 5 : Diagramme d'architecture de la simulation

La différence entre l'architecture réelle et celle de la simulation est que le contrôleur communique plutôt avec un conteneur Simulation. Ce conteneur émule un drone et un rover. Il contient les outils Gazebo et Rviz qui permettent respectivement de visualiser les robots dans un environnement et de percevoir les informations recueillies par les capteurs des robots.

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 réutilisant 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 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 make, de ne pas inclure les 2 robots planifiés à la simulation (R.C.5), de manière simple et efficace.

3.5 *Interface utilisateur*

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

Les pages tireront leurs données directement de la station au sol, en temps réel. Pour cela, elles établiront un lien de type Server-Sent Events 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 2 pages, soit la page principale et la page d'historique.

La page principale 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).

Comme les robots seront présents même lorsqu'une mission n'est pas en cours, il sera possible de voir, pour chacun de ceux connectés, leur type ainsi que leur état (R.F.3) et leur niveau de batterie (R.F.7). Dans le cas de drone, l'état pourra être remplacé par « Crashed » (R.F.13) si applicable. Il sera finalement possible de cliquer sur un bouton « Identifier » pour émettre un son sur les robots connectés (R.F.1). 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). 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. 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 de vol, 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 deux pages sont représentées aux figures suivantes :

Missionpage



Zone de sécurité

Historique des missions

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

Lancer la mission

Terminer la mission

État de la mission: Terminée

Robot 1

Identifier

Robot: OVNI
Type: Rover
État: Arrêté
Charge: 53%

☐ 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

Figure 6 : Représentation de la page principale des missions.


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 7 : Représentation de la page d'historiques.

3.6 *Fonctionnement général*

Le projet est divisé en 6 dossiers séparés, soit sim, client, server, common, drone et rover. Pour démarrer la simulation, il suffira de faire `make sim` dans le dossier racine. De même, une seule commande suffit pour démarrer la base, un `make base`.

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

Lorsqu'on démarre la simulation, il est toujours nécessaire de la démarrer en premier. Ensuite, la base peut ensuite être lancée. Il faudra alors se connecter avec le client (tablette, PC, téléphone intelligent, etc.) à l'adresse indiquée. Il faudra par la suite, dans l'interface utilisateur, demander de rafraîchir la liste des robots disponibles. Les systèmes seront alors fin prêts à débiter l'exploration.

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 matériels 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 nous sommes gardés une marge de 32 heures non allouées pour se donner le temps d'adresser les différents bugs auxquels nous pourrions faire face.

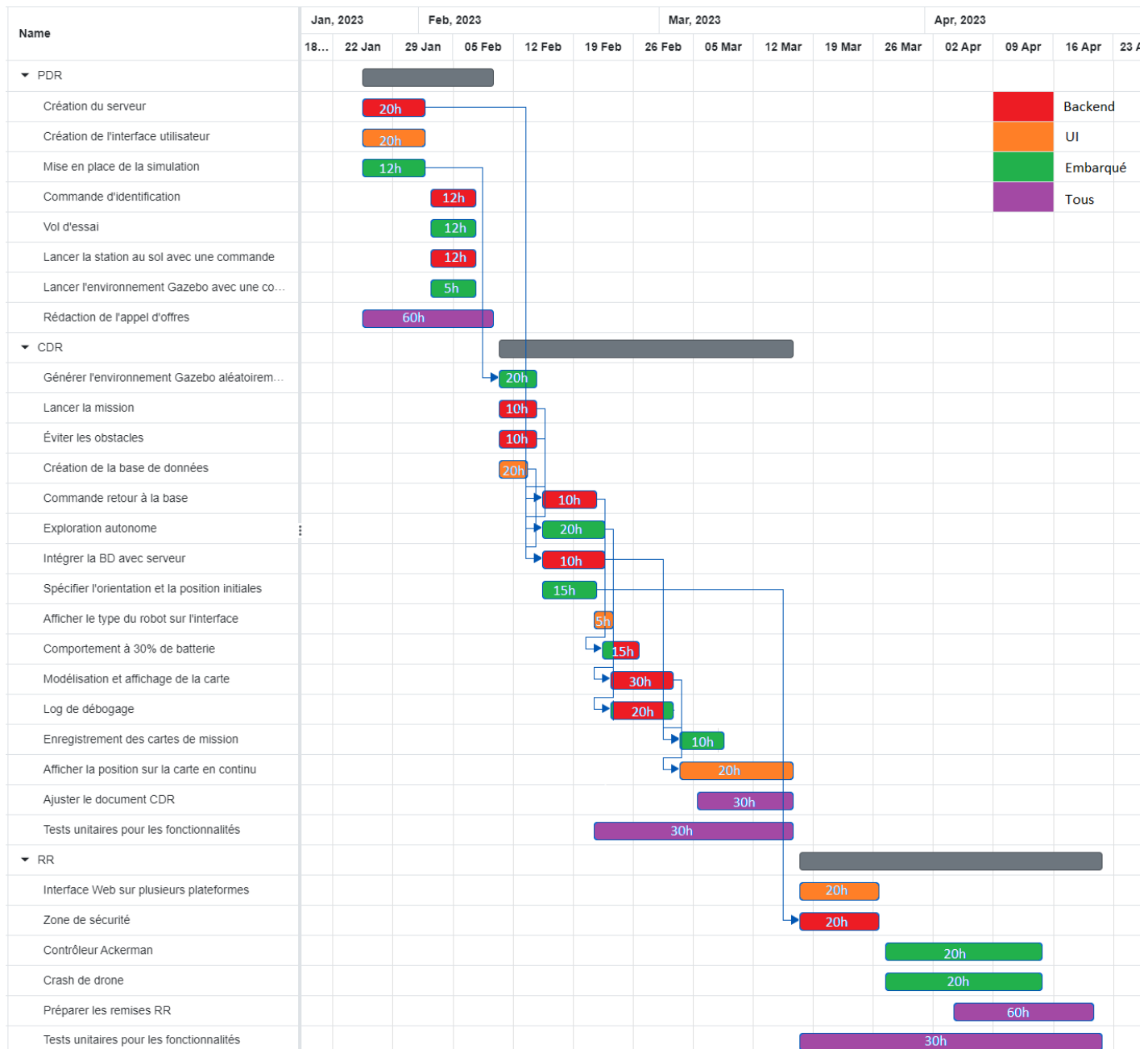


Figure 8 : Diagramme de Gantt de la planification des tâches et de leur allocation

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 1 : 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 comptons établir 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 le guide Google [2] pour notre style de code embarqué en C++, et le guide Airbnb pour le code en JavaScript [3]. 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 qui 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 devra être 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

Pour les tests de logiciel, comme mentionné précédemment dans le contrôle de qualité, les codes sont testés par une pipeline de GitLab avant le fusionnement des branches. Cela permet d'assurer qu'il n'y aucune erreur de compilation. Puisque nous utilisons le framework Angular, les tests Jasmine, Spy et Mock seront utilisés pour tester les fonctions et les composants de notre interface utilisateur pour l'affichage et le fonctionnement des boutons.

Pour la station au sol, il faut tester que les instructions envoyées par l'interface utilisateur et les données qui sont envoyées aux bases de données sont correctes et indexées. Par exemple, un test est implémenté pour assurer que la commande "Le retour à la base et l'atterrissage" sera appelée lorsque la batterie est à 30%. On peut implémenter aussi des tests qui assurent que les données seront bel et bien envoyées aux bases de données.

Pour les tests de matériel, il faut tester d'abord que le drone et le rover sont fonctionnels en arrivant. On teste également que ces derniers exécutent les commandes avec des bonnes instructions, et respectent les contraintes. Finalement, il faut aussi tester que les capteurs envoient les données au serveur. Au niveau de la simulation, le but est de pouvoir tester les fonctionnalités de la station au sol et des codes embarqués avant le déploiement sur le drone physique. Ces tests comprennent surtout les déplacements de robot et les collisions.

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 sera effectué à l'aide de Git avec l'outil Gitlab. Le projet sera 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 seront 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 devra ouvrir une requête de tirage afin que ses changements soient vérifiés par un collègue. Il ne sera pas possible pour un individu d'apporter des modifications et d'approuver sa propre requête. Chacune des requêtes devra être accompagnée des tests appropriés. Les requêtes sans tests ne seront 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 sera effectuée comme suit: Le répertoire va contenir l'application web dans un dossier, et le code associé aux robots dans un autre.

Pour l'application web, deux dossiers contiendront d'un côté le serveur dynamique, de l'autre le serveur statique. Le serveur client, responsable de l'interface client, va contenir les composants Angular ainsi que l'ensemble des tests associés. Le côté serveur dynamique va contenir le code ainsi que ses tests. Les fichiers de données seront 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 auront leur propre conteneur. Le conteneur de la station sera composé du serveur et du contrôleur ROS. Ces conteneurs pourront être créés à l'aide d'un fichier Makefile

Pour la documentation du code source, nous avons décidé de commenter le code dans les fichiers respectifs. Pour une classe donnée, la documentation nécessaire se retrouvera dans le même fichier au-dessus de la classe comme une en-tête.

Pour la documentation de conception, un répertoire sera disponible dans la solution dans lequel les diagrammes de conception, les technologies utilisées ainsi que la documentation explicative seront entreposés et disponibles sous format PDF.

6. 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] B. Weinberger, C. Silverstein, G. Eitzmann, M. Mentovai, and T. Landray. (s.d.). Google C++ Style Guide [En ligne]. Disponible : <http://home.ustc.edu.cn/~hqp/RootClass/AddFiles2/Google%20C++%20Style%20Guide.pdf>
- [3] Airbnb. (s.d.). Airbnb JavaScript Style Guide [En ligne]. Disponible : <https://airbnb.io/javascript/react/>