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. A2021-INF3995 du département GIGL.

Conception d'un système aérien d'exploration

Équipe No ***102***

Youssef Chourifi, Hannan Dissou, Ania Ben Abdesselam, Koceila Lakhdari, Mady Semega, Ismael Adam Soule

Février 2023

1. Vue d'ensemble du projet

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

L'Agence Spatiale a donné comme mandat de conceptualiser et d'implémenter un système d'exploration multirobots, plus précisément de développer un ensemble de logiciels qui permet à une équipe composée de deux robots d'explorer une pièce d'un bâtiment de dimension moyenne. Le système devra être développé sachant que les robots seront munis d'une caméra et d'un capteur laser et que ces derniers devront reporter diverses informations à une interface opérateur basée sur le Web. Un opérateur quelconque doit pouvoir démarrer le système, l'arrêter et faire le suivi des opérations depuis une station au sol. L'objectif de ce projet est ainsi d'arriver à être en contrôle des opérations des 2 robots à partir d'une station distante, au choix entre des rover et des drones, pendant que ces derniers effectuent leur mission d'exploration dans un espace déterminé, dont le but final de produire une carte de cet environnement.
Il y a plusieurs biens livrables attendus. Une documentation révisée et complète du projet, des vidéos démonstratives des requis complétés dans la simulation ainsi que des robots physiques, une remise du système complété, une présentation technique détaillant le produit final, le code nécessaire à l'exécution du projet et des tests et les instructions pour le lancement du système sont attendus.

1.2 Hypothèse et contraintes (Q3.1)

Pour réaliser ces plans, il faut poser plusieurs hypothèses et respecter certaines contraintes. Les robots d'explorations fournis par l'Agence Spatiale sont des AgileX Limo et CogniFly. La station au sol doit être un laptop ou PC et doit avoir un moyen de se connecter avec les robots physiques ainsi que la simulation, avec une communication Wi-Fi. Les robots doivent être programmés en utilisant l'OS Ubuntu 20.04. Le contrôle des robots doit se faire sur ceux-ci avec du code embarqué. Les robots ne possèdent que des capteurs et des caméras pour être opérés. La station ne doit qu'envoyer des directives assez générales. Tous les logiciels développés doivent être conteneurisés avec Docker. Les robots doivent avoir une batterie puissante pour avoir une autonomie suffisante. L'espace et la période de la mission d'exploration doivent être raisonnables selon les requis. L'espace d'exploration doit être explorable par un drone volant et par un rover roulant au sol. Les conditions d'explorations, même météo, doivent être souhaitables. Un réseau Wi-Fi doit être fourni par l'Agence pour la station au sol ainsi que les robots.

1.3 Biens livrables du projet (Q4.1)

Plusieurs artefacts seront créés durant le projet. Les fichiers sources du système embarqué et simulé utilisé sur les robots correspondant à une conception précise, ainsi que les fichiers produits par les cadriciels des langages utilisés seront à rendre. Le code source, les fichiers de configuration, les fichiers de test et quelques fichiers de documentation pour l'application web du système doivent aussi être remis. Il ne faudra pas oublier de remettre les fichiers de conteneurisation Docker de l'ensemble du projet, les fichiers contenant les scripts "Bash" et le rapport technique du projet qui documentera le système bâti.
Ces artefacts seront graduellement développés et publiés pour 3 évènements distincts, le PDR, le CDR et le RR, respectivement publiés le 10 février, 17 mars et le 21 avril 2023.

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Afin de gérer la charge de travail du projet, nous avons choisi de former des binômes pour s'occuper des modules principaux de l'architecture. Chaque membre de l'équipe aura aussi sous sa responsabilité des modules secondaires sur lesquels il/elle pourra travailler individuellement. Il s'agit d'une gestion décentralisée qui permettra à chaque membre de l'équipe de faire avancer une partie du projet tout en ayant en permanence connaissance des mises à jour apportées à l'ensemble du système. Cela a entrainé l'attribution des rôles suivants:

- **Ania Ben Abdesselam:** ***Coordinatrice de projet***
  Module principal: Côté ordinateur
  Module secondaire:
  - Interface Web
  - Gestion de projet (Répartition des issues et planification des réunions)

- **Mady Semega**: ***Analyste développeur***
  Module principal: Côté ordinateur
  Module secondaire: Serveur Node 1

- **Koceila Lakhdari**: ***Analyste développeur***
  Module principal: Côté Robot – Rover #1
  Module secondaire: Serveur Node 2

- **Hannan Dissou**: ***Analyse développeur***
  Module principal: Côté Robot – Rover #1
  Module Secondaire: Serveur Node 2

- **Youssef Chourifi**: ***Analyste développeur***
  Module principal: Côté Robot – Rover #2

Module secondaire: Serveur Node 3

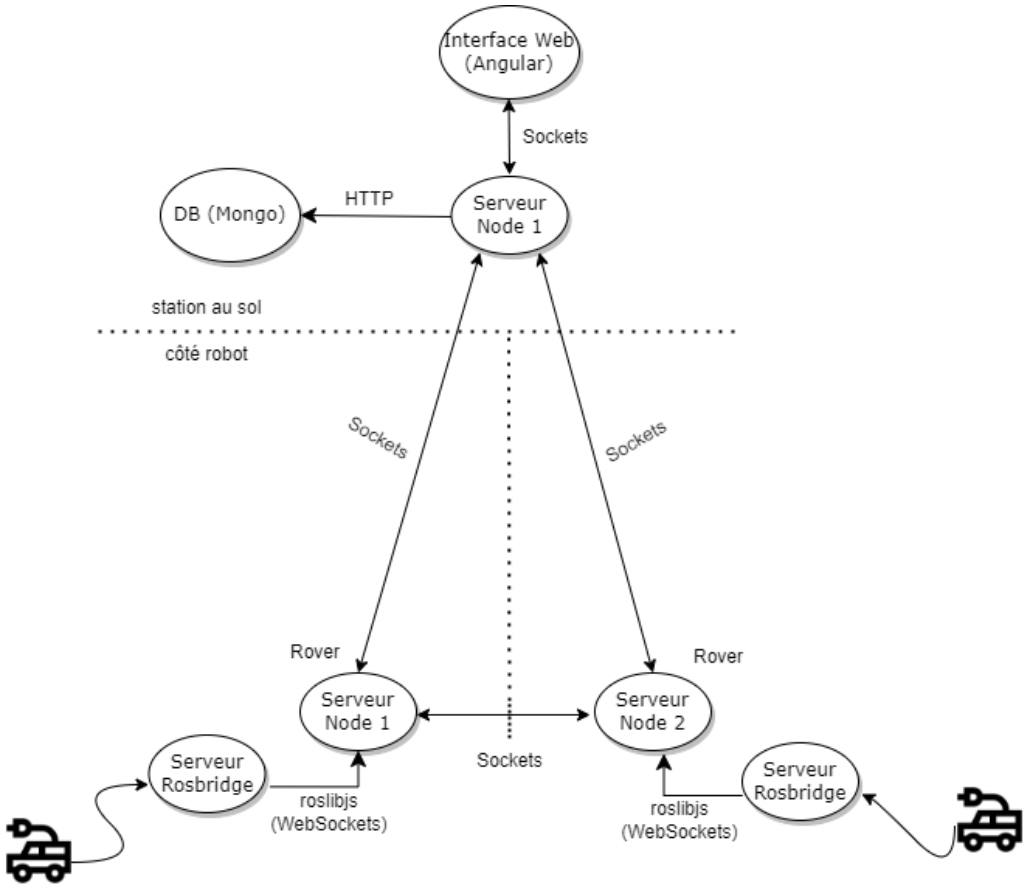
- **Ismaël Adam Soule**: ***Analyste développeur***
  Module principal: Côté Robot – Rover #2
  Module Secondaire: Serveur Node 3

2.2 Entente contractuelle (Q11.1)

Le type d'entente contractuelle proposé pour ce projet est un contrat fixe. Ce choix a été fait car il permet de fournir une stabilité à long terme pour les membres de l'équipe de projet en ce qui concerne les tâches, les délais et les attentes. Il assure également un engagement ferme envers les parties prenantes quant à la qualité et à la livraison des produits finaux puisque les attentes, contraintes et requis du projet sont déjà fixées et connus. De plus, un contrat fixe peut offrir une protection pour les investissements financiers des parties impliquées, car les coûts associés à ce projet sont clairement définis et les termes de paiement sont établis à l'avance. Enfin, ce type d'entente contractuelle peut également aider à minimiser les conflits potentiels, car les rôles, responsabilités et obligations des différentes parties sont déterminés de manière claire.

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)



Notre architecture globale consiste en une base de données MongoDB et trois serveurs Node.js distincts, c'est-à-dire 1 sur la station au sol et 1 sur chacun des robots.

Le premier serveur développé sous Node.js a pour but d'établir la communication entre l'interface et les robots. Il traite entre-autres les requêtes du client et lui fait parvenir des informations provenant des robots. Ces requêtes sont traitées via des sockets avec socket.io. Le serveur utilise le cadriciel Express pour instancier le serveur.

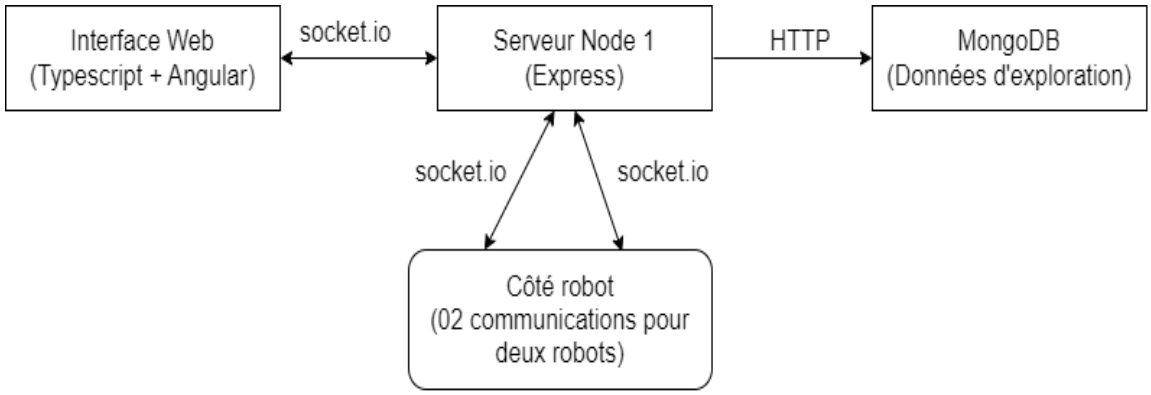
Les robots ont chacun un serveur sous Node.js qui communique via des sockets. Le serveur embarqué envoie les requêtes reçues de la station au sol via socket.io au serveur rosbridge en utilisant la librairie Roslibjs. Cette librairie permet d'envoyer des requêtes et commandes ROS sous format JSON, par web sockets, vers les systèmes d'exploitation des robots. Plus précisément, ces commandes sont reçues par un serveur web socket, lancé à l'aide du progiciel rosbridge qui s'occupe de les traduire en commande ROS pures pour manipuler les robots.

La communication « peer-to-peer » entre les 2 robots se fait avec les 2 serveurs Node.js embarqués sur ces derniers.

La base de données communique avec serveur de la station au sol et est localisée sur une instance MongoDB.

Nous préférons utiliser MongoDB, socket.io et des serveurs Node.js pour gérer les communications, car nous estimons être plus apte à gérer une base de données NoSQL et des serveurs Express vu l'expérience acquise durant le projet 2. Nous utilisons Roslibjs pour la communication avec le rover, car la librairie est plus performante que l'utilisation d'un serveur Python couplé à Rospy.

3.2 Station au sol (Q4.5)

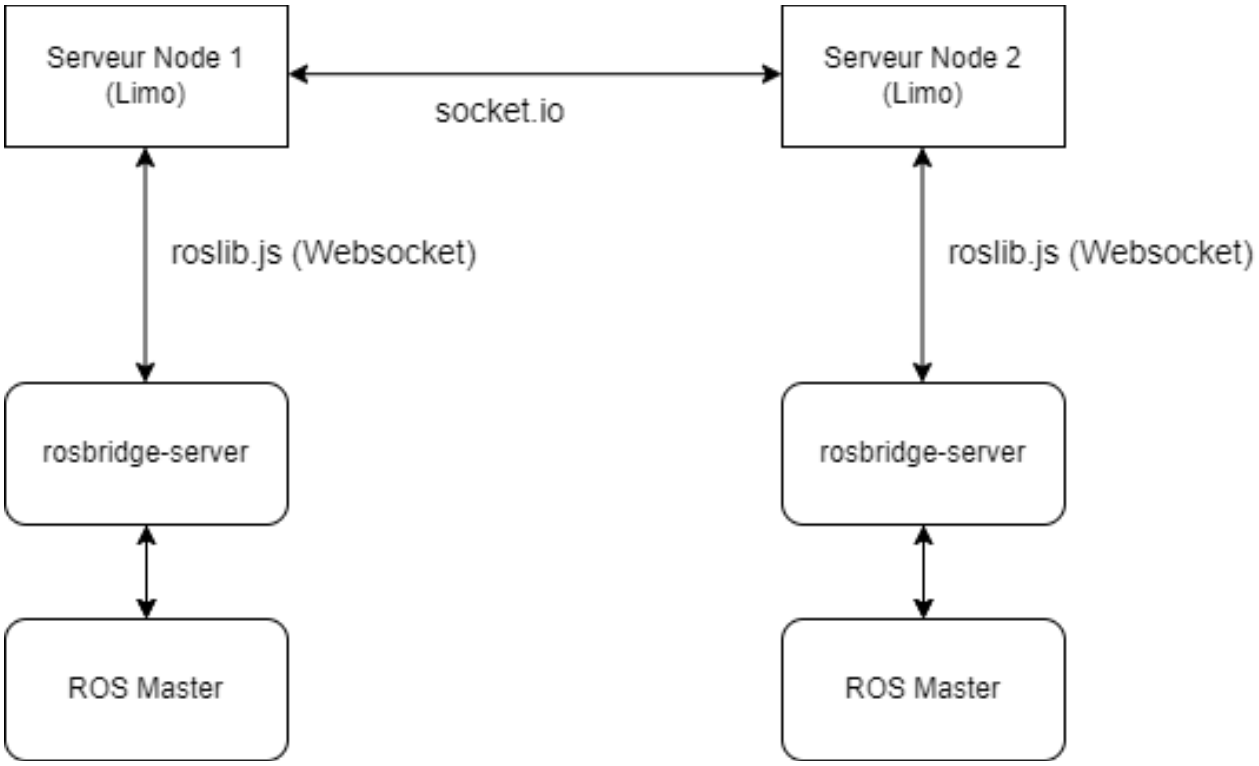


La station au sol se résume en une interface web faite avec Angular en TypeScript, un serveur Node.js avec Express et une communication avec la base de données sur MongoDB. Ces technologies ont été utilisées parce que nous y sommes habitués, vu nos expériences passées et la grande documentation qui existe sur ces derniers.

Plus précisément, la communication entre le client et le serveur se fera de façon bidirectionnelle à l'aide de socket.io comme montré dans le diagramme. Le protocole Web Socket sera aussi utilisé pour communiquer avec les serveurs embarqués dans la couche robot.

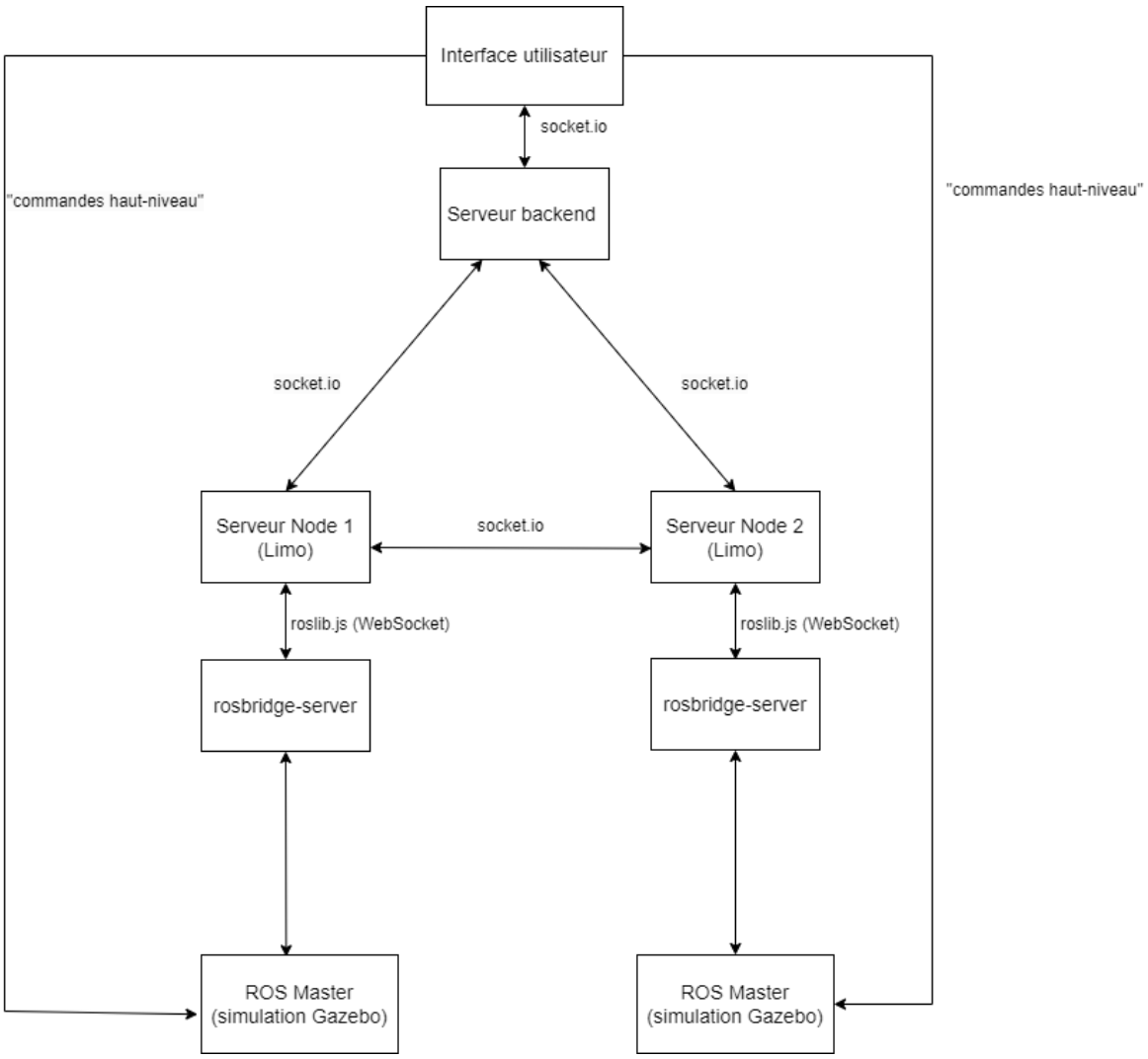
La base de données, quant à elle, communiquera avec le serveur via HTTP et se chargera de stocker les données d'exploration générées par les robots.

3.3 Logiciel embarqué (Q4.5)



Chacun des robots aura son serveur qui lui est dédié en Node.js.
Sur le rover, le serveur Node.js envoie des commandes ROS sous format JSON via des web sockets en utilisant la libraire Roslibjs. Ces commandes sont reçues, traitées et traduites en commandes ROS standard par le serveur web socket, lancé grâce au progiciel rosbridge. Les robots pourront ensuite effectuer le comportement désiré venant de la commande.
Le choix de Roslibjs s'explique par la facilité d'implémentation qu'elle offre, la rapidité de communication qu'elle confère contrairement à Rospy avec un serveur Python et la bonne documentation qui existe sur cette librairie.
De plus, on assurera la communication « P2P » entre les deux robots via une communication web socket entre les deux serveurs Node.js sur les robots.
Enfin, un système de machine à états sera développé pour l'exploration de la pièce. Les robots transitionneront entre les différents états afin de leur permettre de varier leurs actions et d'effectuer des manœuvres spécifiques dans la suite du projet.
Nous nous baserons sur les algorithmes de type « SLAM » afin de simultanément construire et améliorer la carte de l'environnement et de pouvoir s'y localiser.

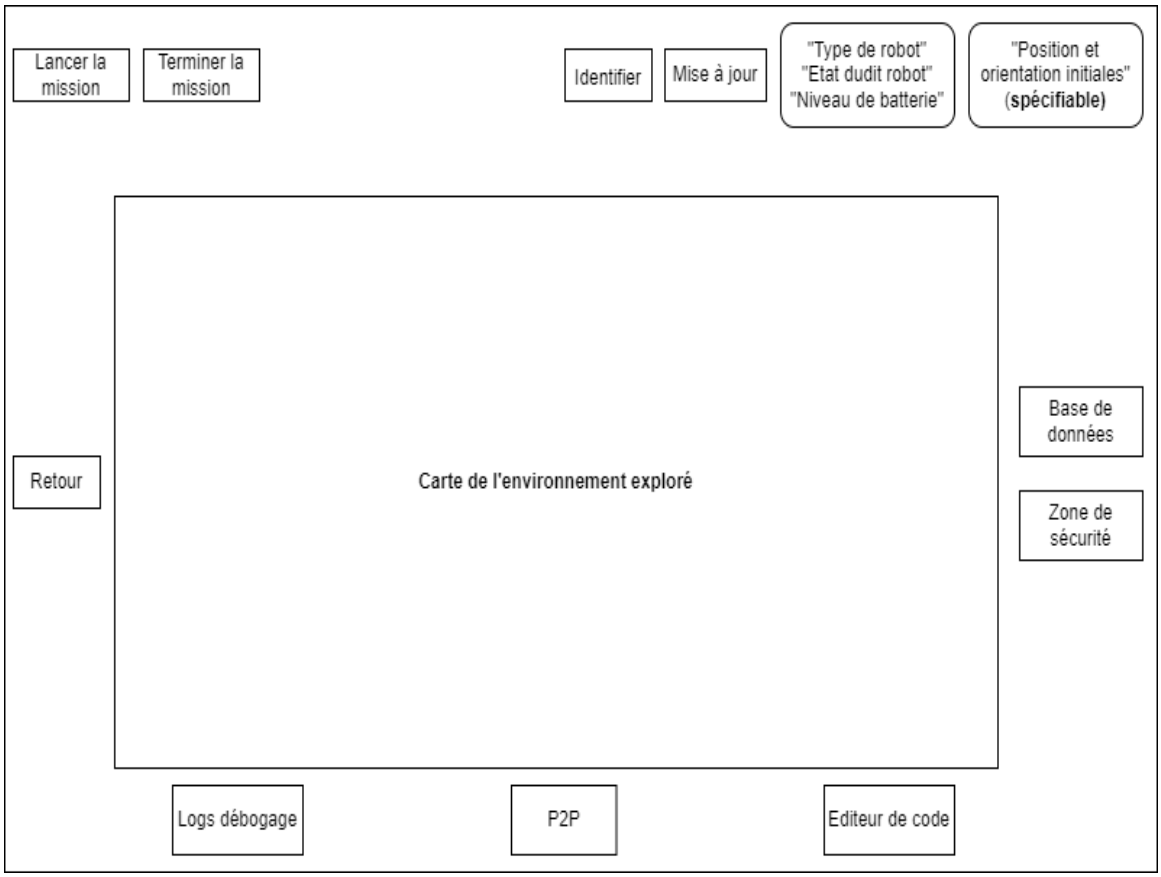
3.4 Simulation (Q4.5)



.

Notre architecture est conçue de manière à pouvoir se connecter à n'importe quel robot physique ou simulé possédant un système d'exploitation avec ROS, dont le ROS Master et le serveur rosbridge sont lancés. Ainsi, il n'y a pas à développer de système ou de logique distincts pour la simulation par rapport au système déjà développé pour les robots physiques. Il ne faudra que définir des variables d'environnements pour les adresses IP qui changeront selon la situation. En effet, comme le code de la simulation roule sur la station au sol, de façon locale, alors que le code embarqué sur les robots est sur une machine distante, les adresses IP spécifiées dans le code, pour établir les connexions, seront forcément différentes. Ces adresses sont utilisées dans le serveur de la station au sol pour se connecter aux robots et par les serveurs embarqués sur les robots pour pouvoir se connecter au rosbridge.

3.5 Interface utilisateur (Q4.6)



L'interface utilisateur est la partie visuelle de la station au sol qui permet à l'opérateur d'interagir avec le système.

Pour commencer, on y retrouve dans le coin supérieur droit, à travers deux boutons, les commandes "Identifier" (RF1) et "Mise à jour" (RF14). Quant aux deux boutons dans le coin supérieur gauche, ils représentent les commandes "Lancer la mission" et "Terminer la mission" (RF2).

Ensuite, pour le coin supérieur droit, on y affichera en premier le type de robot et son état présent (RF3). On y ajoutera, dans le même temps, le niveau de la batterie du robot (RF7). Ces infos seront rafraîchies continuellement. Il est à noter que lorsque le niveau de batterie est en dessous de 30%, la commande "Lancer la mission" deviendra grisée pour empêcher son fonctionnement.

À ceci sera juxtaposé un onglet pour afficher la position et l'orientation initiales du robot avec possibilité de les spécifier en début de mission (RF12). Le bouton "Retour" à gauche de la carte de l'environnement permettra d'instruire les robots de se rapprocher de leur position initiale (RF6). L'instruction sera automatique dans le cas où le niveau de batterie du robot passe en dessous de 30%.

Venons-en à la partie de l'interface la plus visuelle, c'est-à-dire la carte de l'environnement exploré par les robots. En effet, grâce aux données générées par les robots, la station au sol produira une carte qui sera affichée dans l'interface en

continu pendant la mission, de même que la position des robots sur celle-ci (RF7 et RF8). À la droite de ladite carte, on retrouve un bouton "Base de données" qui fera le sommaire des données sauvegardées. Il permettra d'afficher une mini-fenêtre où l'on retrouve toutes les données mentionnées dans le RF17 afin que l'opérateur puisse rechercher ce qui l'intéresse.

Dans le même temps, on y retrouvera la liste des cartes générées lors des missions précédentes à des fins d'inspection (RF18). Quant au bouton "Zone de sécurité", il permettra à l'opérateur, une fois le bouton pressé, de délimiter une aire de restriction de mouvement sur la carte (RF20).

Enfin, au bas de l'interface utilisateur, on retrouve à droite le bouton "Éditeur de code". Il permet de modifier le code des contrôleurs de robot et ainsi de changer leurs comportements avant ou entre des missions (RF16). Le bouton marqué "P2P" permet de déclencher une communication "peer-to-peer" entre les deux robots (RF19).

Pour finir, le bouton "Logs débogage" permet d'accéder soit aux logs de la mission en cours soit celles d'une mission précédente afin de diagnostiquer les problèmes du système (RC1).

3.6 Fonctionnement général (Q5.4)

Pour faire fonctionner notre code, nous utilisons majoritairement Docker. En effet, pour démarrer les différents conteneurs sur les robots et la station au sol, nous utilisons une commande dans chaque module pour lancer les différents sous-systèmes.

En premier lieu, pour démarrer les conteneurs sur les robots Limo, il faut se connecter en SSH au véhicule, puis télécharger le projet depuis le Gitlab et naviguer vers le dossier « Limo/ros-server ». Pour construire l'image Docker, nous pouvons utiliser la commande docker suivante :

« *docker build –t ros-server-102 .*»

Avant de pouvoir lancer les images, nous devons nous assurer que le progiciel rosbridge et le ROS Master du robot soient lancés.

Une fois l'image créée et les progiciels lancés, nous pouvons démarrer le serveur embarqué. De plus, il faudra passer deux variables d'environnement : une qui est l'adresse IP du Limo sur lequel nous déployons le container (LIMO_IP) et une qui représente l'identifiant de ce Limo (dans notre situation avec deux robots : 1 ou 2) pour pourvoir faire la distinction entre les deux Limo. La commande qui sert à démarrer les conteneurs est :

«*docker run -- e LIMO_IP:<IP du Limo>*
--e LIMO_ID:<Identifiant du Limo : 1 ou 2>
--p 9332:9332 ros-server-102»

En second lieu, pour démarrer les conteneurs sur la station au sol, il faut télécharger le projet et naviguer vers le dossier « Interface ». Il faut ensuite construire les différentes images avec le fichier docker-compose déjà disponible dans le dossier avec la commande :

« *docker-compose –f interface-docker-compose.yml build* ».

Une fois les images créées, nous pouvons directement démarrer les deux images où vont s'exécuter notre serveur pour l'interface graphique et le serveur pour la gestion des requêtes de l'interface. Nous devons deux variables d'environnement : les deux adresses IP des Limo pour pourvoir les retrouver et s'y connecter. Nous pouvons utiliser la commande :

« LIMO_IP_1=<adresse du premier robot>
LIMO_IP_2= <adresse du second robot>
docker-compose –f interface-docker-compose.yml up ».

Ici, LIMO_IP_1 représente l'adresse IP du robot auquel nous aurons passé l'identifiant 1 et LIMO_IP_2 représente celui avec l'identifiant 2.

En dernier lieu, pour le mode simulation, l'architecture fonctionne de la même manière. Il faudra seulement remplacer les adresses IP des robots par celui de la station de simulation. Un fichier « bash » est disponible pour lancer la simulation en une ligne de commande. Ce fichier assume que le dossier de la simulation « ugv_gazebo_sim » est téléchargé et que les progiciels ROS et « rosbridge » sont déjà installés.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

Afin de réaliser à bien ce projet, de nombreux coût sont à envisager.

Tout d'abord, pour le bon accomplissement, l'équipe doit être composé de 6 membres dont un qui coordonne l'équipe. Cela est nécessaire pour une séparation des tâches optimale. Les 5 développeurs-analystes sont payé 130$/h et le coordonnateur 145$/h. Ici, on estime le temps de travail total pour chaque membre à 630h. Ainsi, le coût à envisager pour les dépenses salariales sont de 500 350$.

Pour poursuivre, nous devons acheter deux robots Limo au prix individuel de 3851,49$ (1).

Ensuite, l'achat d'un ordinateur pour les membres de l'équipe est aussi à prendre en compte. Le Lenovo IdeaPad 3i 15,6 po (2) au coût de 1100 $ chaque est parfait pour les requis du projet. Cet ordinateur possède un processeur Intel i7 et 12 Go de RAM qui va nous permettre de pouvoir travailler sans problème de performance.

Il faut aussi considérer le local où la conception du projet va se passer pour les tests d'exploration. Le local choisit (3) se trouve au centre-ville de Montréal. Il va couter 1500$ par mois. Pour 4 mois de conception, ce local va nous couter 6000$.

En somme, le tableau 1 suivant résume les différentes dépenses nécessaires pour le projet :

Tableau 1: Tableau représentant les dépenses nécessaires du projet

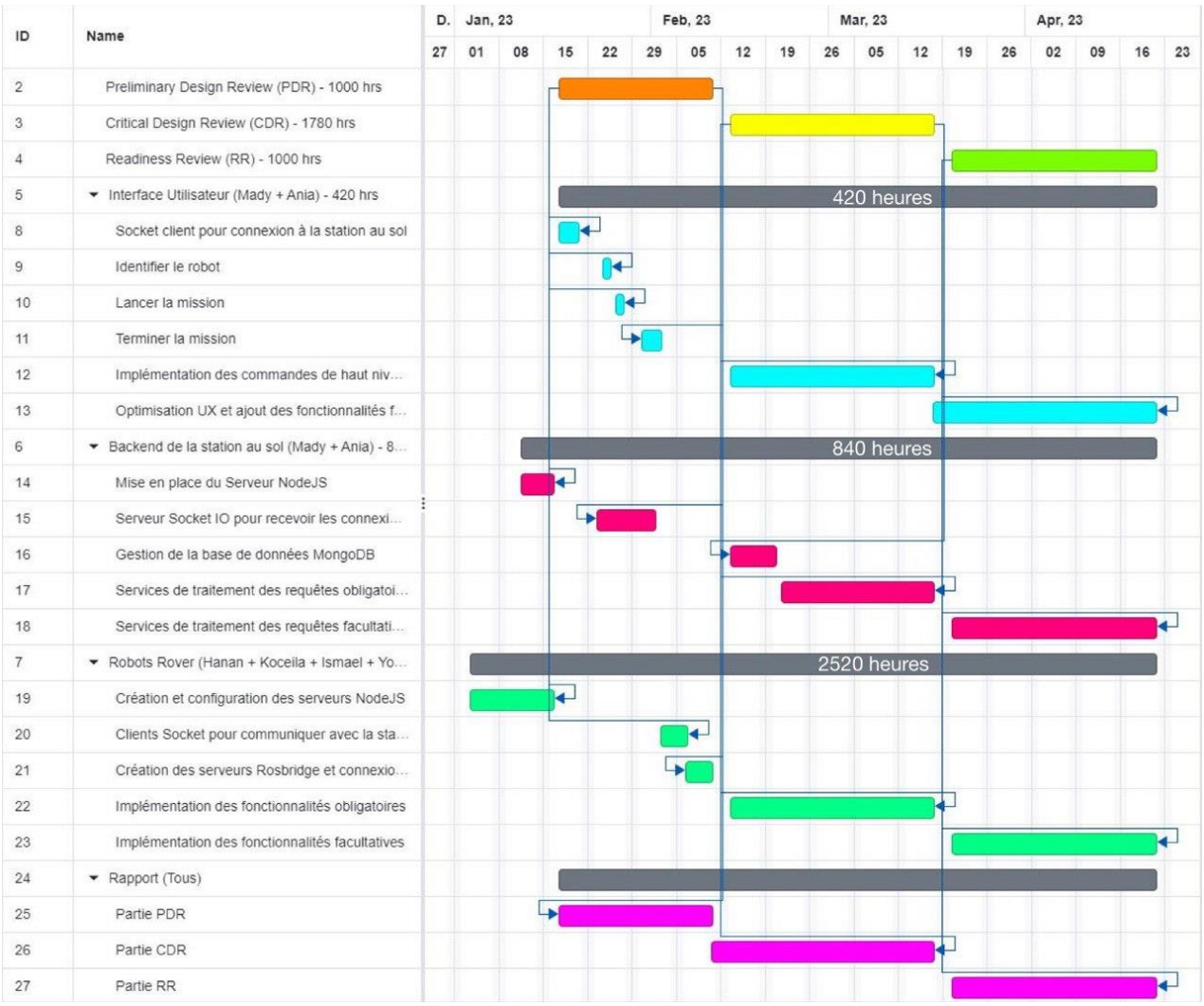
Investissement	**Coût ($)**
Membres de l'équipe	500 350.00 $
Deux robots Limo	7702.98 $
Ordinateurs Lenovo (x6)	6600.00 $
Local	6000.00 $
Total	*520 652.98 $*

4.2 Planification des tâches (Q11.2)

Pour réaliser ce projet, nous avons décidé de le diviser en 3 principaux jalons. Nous aurons donc le Preliminary Design Review (PDR), le Critical Design Review (CDR) ainsi que le Readiness Review (RR).

Le total des heures travaillées pour doit être égale à 3780 heures.
Le nombre d'heures pour la remise du jalon PDR doit être de 1000 heures.
Le nombre d'heures pour la remise du jalon CDR doit être 1780 heures.
Le nombre d'heures pour la remise du jalon RR doit être 1000 heures.

La figure suivante contient un diagramme résumant cette planification.



4.3 Calendrier de projet (Q11.2)

Tableau 2: Tableau représentant le calendrier du projet

Jalons	Date préliminaire	Date Final
PDR	05/02/2023	10/02/2023
CDR	10/03/2023	17/03/2023
RR	15/04/2023	21/04/2023

4.4 Ressources humaines du projet (Q11.2)

Pour la réalisation de ce projet, l'équipe sera composée de 6 membres : 5 développeurs-analystes et d'une coordonnatrice de projet.

Koceila Lakhdari a travaillé dans une société dans laquelle il avait pu explorer et approfondir ses compétences en système embarqué. Il a également appris à utiliser Docker lors de son stage.

Mady Semega possède de fortes aptitudes côté serveur. Il possède une grande expérience en gestion de système sur plusieurs serveuses acquises dans son travail en industrie.

Hannan Dissou a une excellente capacité de vision d'ensemble et possède une facilité pour voir les tâches manquantes ou qu'il y a faire. Cela est un plus pour l'organisation du projet. De plus, il a pu acquérir une bonne expérience en gestion de base de données lors de son expérience en industrie. Cela est très utile pour le projet.

Ismaël Adam Soule possède une forte affiliation avec l'informatique. Cela lui a même permis de gagner une bourse dans son pays natal, le Bénin. Il possède une vision des choses différentes qui nous permet de voir certaines tâches sous d'autres angles.

Youssef Chourifi a effectué un stage dans une société qui lui a permis de développer ces connaissances en tests et en intégration de système. De plus, lors de sa période de stage, il a appris beaucoup de technologie utile au projet tel que Docker.

Notre coordinatrice de projet, Ania Ben Abdesselam, possède une bonne capacité de gestion et d'organisation. Son expérience au sein de la société technique PolyHx et toutes les tâches de préparations d'événements lui apporte une expérience supplémentaire utile au projet.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Avant la remise de chaque livrable et régulièrement durant le processus de développement, nous allons nous assurer que le projet répond aux critères demandés par le livrable. Nous allons passer par chacun des requis et nous allons nous assurer par simulation avec le logiciel Gazebo et en testant directement sur les robots physiques que notre code satisfait chacun des requis demandés. En passant par la liste des requis, nous confirmons directement que nous répondons bien aux tâches demandées. En procédant de cette manière, nous nous assurons de bien réviser les tâches.

5.2 Gestion de risque (Q11.3)

Le premier principal risque de ce projet est le fait que l'un des robots tombe en panne et ne fonctionne plus. Dans cette situation, nous n'aurons que le choix de commander un autre robot et de le reconfigurer pour qu'il marche. Cette procédure peut prendre plusieurs semaines entre le moment où nous passons la commande et celui où il sera prêt à être utiliser. Cela pourrait nous retarder énormément dans le projet. C'est pour cela que ce risque est très important et que nous devons faire attention au matériel.

Le second risque principal est l'utilisation de la mauvaise plateforme de développement pour le projet (utilisation de Node.js au lieu de python). Dans ces conditions, il faudrait retranscrire tout le code déjà existant pour qu'il soit compatible avec la nouvelle plateforme choisie. Cela représente une charge de travail supplémentaire et son lot de problèmes, mais toutefois, ce n'est pas impossible. Il est difficile pour l'instant de déterminer si nous avons pris la bonne décision, car nous n'avons pas encore avancé suffisamment dans le projet. C'est pour cela que ce risque est moyennement important.

Le dernier risque principal est la mauvaise gestion du temps. Si certaines tâches prennent plus de temps que prévu et que cela empiète sur d'autres requis, cela pourrait devenir un problème pour la remise du livrable. Il faudrait alors faire des réunions d'équipe pour choisir les priorités sur lesquelles se concentrer et en abandonner d'autres. Pour éviter cette situation, nous allons mettre des échéances que tous devront respecter. Ce risque est très important, car il peut impacter le projet dans sa totalité.

5.3 Tests (Q4.4)

L'un des tests qui pourrait être fait pour le sous-système qui va être déployé sur les robots serait un test de connexion entre les serveurs embarqués et la station au sol.

Pour poursuivre, un autre test serait de s'assurer de recevoir les requêtes entre l'interface utilisateur et le serveur qui traite les commandes au sol.

Pour finir, il resterait à tester l'interaction entre le serveur et le client déployés sur les robots. La communication entre ces deux composantes doit être fait de manière rigoureuse pour faire le pont vers les systèmes d'exploitation des robots.

5.4 Gestion de configuration (Q4)

Le système de contrôle de version qui a été choisi est Gitlab. Via ce logiciel, nous allons mettre à jour les différentes versions et progression du code source.

Nous avons décidé de séparer les différents modules en deux grands dossiers : l'interface web et le véhicule automobile Limo. Ces deux grands dossiers vont contenir le code source de chaque sous-système du travail demandé. Le dossier de l'interface web va contenir le serveur statique qui distribue la vue de l'interface et le serveur au sol. Le dossier Limo, quant à lui, va contenir un serveur Node.js qui communiquera avec l'utilisateurs et le serveur Rosbridge qui contrôlera les robots.

Pour nous assurer d'utiliser les mêmes versions de développement pour tous les modules, nous utilisons des conteneurs Docker pour avoir des versions identiques de Node.js et ROS. Cela va isoler les différents serveurs pour un bon entretien du code et pour une meilleure intégration.

Chaque composante logicielle sera accompagnée d'un fichier de tests unitaires qui va assurer leur bon fonctionnement. Toutes les composantes logicielles qui ne peuvent pas être validées par des tests unitaires vont être régulièrement testés avec des tests d'intégrations. Ces tests seront faits avec la simulation des robots en utilisant le logiciel de simulation Gazebo.

Au fur et à mesure que nous allons avancer dans le développement du projet, la documentation sera tenue à jour. Il y aura un fichier d'information (README.md) qui va contenir toutes les informations sur le fonctionnement de l'application, les dépendances et les commandes de l'application. La documentation de conception sera aussi tenue à jour au fur et à mesure que nous développons le projet. Tous les changements dans la configuration et l'organisation sera reflété dans le document de conception dès que la décision aura été prise.

Chaque personne va développer chacune de ses tâches sur différentes branches. Lorsque nous seront satisfait du résultat, avec le logiciel Git, nous allons intégrer cette nouvelle fonctionnalité dans le code source stable en s'assurant de résoudre tous les problèmes que cela engendrera. Une approbation de minimum 2 membres de l'équipe devra être faite avant que la fonctionnalité ne soit intégrée au reste du code.

Toutes ces mesures vont nous permettre de bien organiser notre code et de mieux gérer les différentes tâches à venir.

5.5 Déroulement du projet (Q2.5)

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

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

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

7. Travaux futurs et recommandations (Q3.5)

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

8. Apprentissage continu (Q12)

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

1. *Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*
2. *Méthodes prises pour y remédier.*
3. *Identifier comment cet aspect aurait pu être amélioré.]*

9. Conclusion (Q3.6)

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

10. Références (Q3.2)

[Liste des références auxquelles réfère ce document. Un site web est une référence tout à fait valable.]

ANNEXES

[1] RobotLAB. (2023). *AGILEX LIMO RESEARCH*. *https://www.robotlab.com/higher-ed-robots/store/limo-agilex?srsltid=Ad5pg_FSbuR3Nz-sw_w9NP2SDeEz_M9OO8oUzL-wdikoVDNIbwDpTRoh2FM*

[2] Bestbuy. (2023). *Portable IdeaPad 3i 15,6 po de Lenovo. https://www.bestbuy. ca/fr-ca/produit/16204923*

[3] Utilisateur Kijiji. (10 février 2023). *LOCAL RÉNOVÉ de 600 PC -Centre Ville MTL* [Publication Kijiji]. Kijiji. https://www.kijiji.ca/v-local-commercial-bureau/ville-de-montreal/local-renove-de-600-pc-centre-ville-mtl/1649988844