

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 No 102

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

Mars 2023

1. Vue d'ensemble du projet

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

Un mandat de conceptualisation et d'implémentation d'un système d'exploration multirobots a été donné par l'Agence Spatiale. Ce système devra permettre à une équipe d'explorer une pièce d'un bâtiment d'une dimension moyenne à l'aide de 2 robots qui seront contrôlés à distance à travers des instructions accessibles sur une interface Web. L'objectif final sera de produire une carte d'environnement de la zone explorée par ces robots.

Des biens livrables sont également attendus. Une documentation révisée et complète du projet accompagnées de vidéos démonstratives des requis complétés lors de simulations virtuelles et physiques sera remise. Tout comme une remise du système global fini, ainsi qu'une présentation technique détaillant le produit final, le code nécessaire à l'exécution du projet ainsi que les tests ainsi que les instructions pour le lancement du système sont attendues.

1.2 *Hypothèse et contraintes (Q3.1)*

De nombreuses hypothèses doivent être posées afin de réaliser les plans du système d'exploration. Il faudra également prendre en compte les contraintes imposées par l'Agence Spatiale. Des robots spécifiés ont été fourni par l'Agence Spatiale tel AgileX Limo et CogniFly. La station au sol devra être un ordinateur portable ou un PC ayant un moyen de se connecter avec les robots physiques et ayant accès à la simulation virtuelle par communication Wi-Fi. Le système doit être programmés en utilisant l'OS Ubuntu 20.04. Du code embarqué est requis pour le contrôle des robots. Une contrainte s'applique également quant aux spécifications des outils fournis par le robot n'ayant que des capteurs et caméra. Uniquement des directives générales devront être envoyé par la station. Les logiciels développés devront être conteneurisés avec Docker. Les robots doivent avoir une batterie puissante afin d'avoir une autonomie suffisante pour chaque mission. L'exploration pourra être faite pas un drone et par un rover. Les conditions d'explorations ainsi que la météo devront être optimales ainsi que le réseau Wi-Fi fourni par l'Agence.

1.3 *Biens livrables du projet (Q4.1)*

De nombreux artefacts seront créés durant le projet. Les fichiers sources du système embarqué et simulé utilisés sur les robots correspondant à une conception précise, ainsi que les fichiers produits par les cadriels 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 également 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)*

1. Structure d'organisation

Pour mener à bien ce projet, une équipe de 6 ingénieurs a été formé. Pour améliorer la gestion du projet et le charge de travail globale, nous avons formé des binômes afin de s'occuper des modules principaux de l'architecture. Chaque membre de l'équipe aura sa propre responsabilité sur des modules secondes sur lesquels il/elle travaillera individuellement. Le choix d'une gestion de travail décentralisée a été adopté pour permettre à chaque membre de l'équipe d'avancer sur une partie du projet tout en ayant en permanence une connaissance des mises à jour apportées à l'ensemble du système. Ce qui a entraîné 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 Semeaga: 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

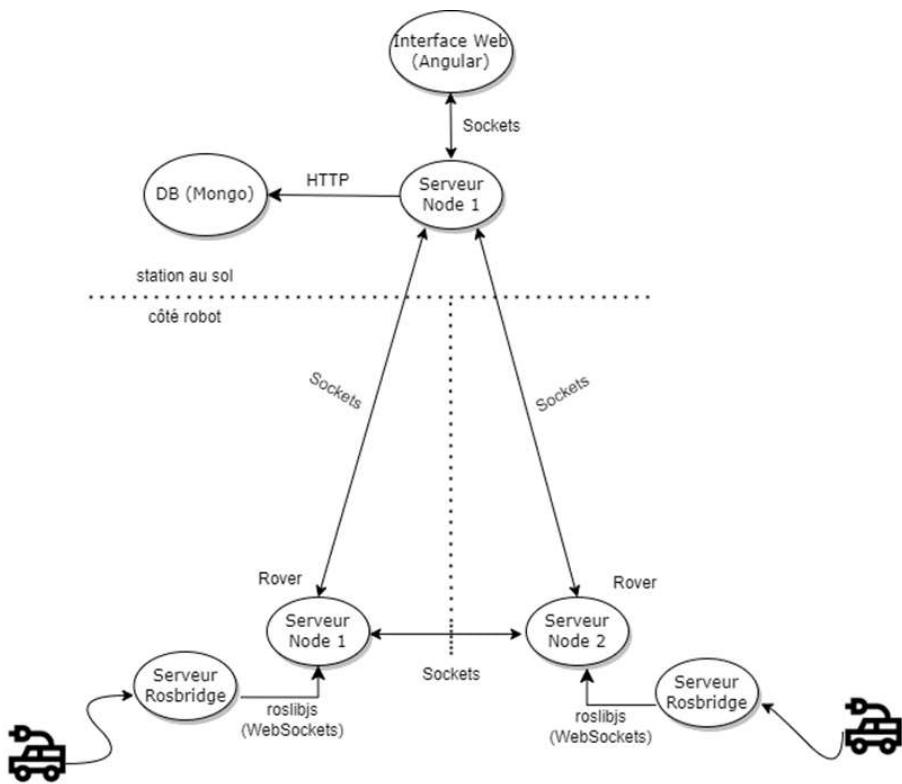
Module Secondaire: Serveur Node 3

2.2 *Entente contractuelle (Q11.1)*

L'entente contractuelle proposé pour ce projet est un contrat fixe. On explique ce choix par le fait qu'il permettra de fournir une stabilité à long terme pour les membres de l'équipe du 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.

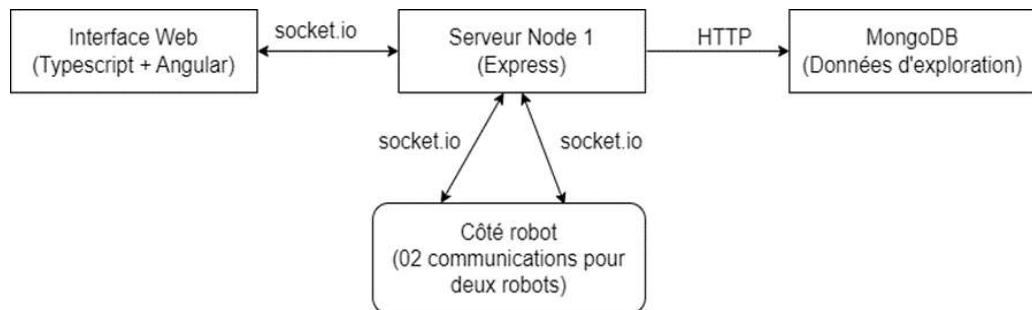
Les robots ont chacun un serveur sous Node.js qui communique via des sockets. Le serveur embarqué envoie les requêtes reçues de la station au sol via socket.io au serveur rosbridge en utilisant la librairie Roslibjs. Cette librairie permet d'envoyer des requêtes et commandes ROS sous format JSON, par 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)

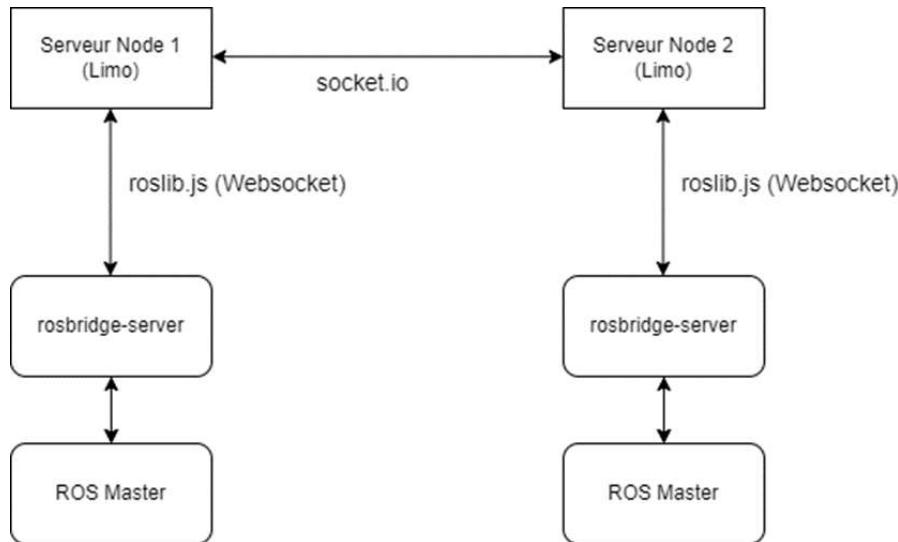


La station au sol se résume en une interface web faite avec Angular en TypeScript, un serveur Node.js avec Express et une communication avec la base de données sur MongoDB. Ces technologies ont été utilisées parce que nous y sommes habitués, vu nos expériences passées et la grande documentation qui existe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

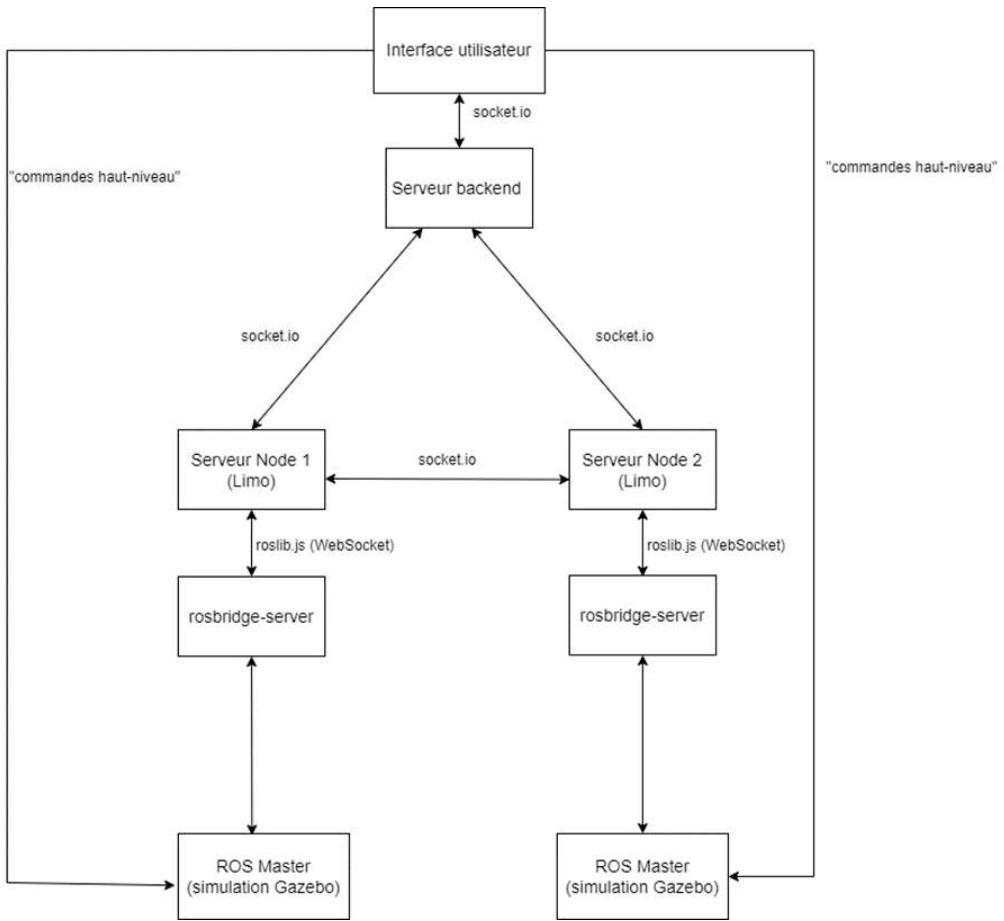
Le choix de Roslibjs s'explique par la facilité d'implémentation qu'elle offre, la rapidité de communication qu'elle confère contrairement à Rospy avec un serveur Python et la bonne documentation qui existe sur cette librairie.

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.

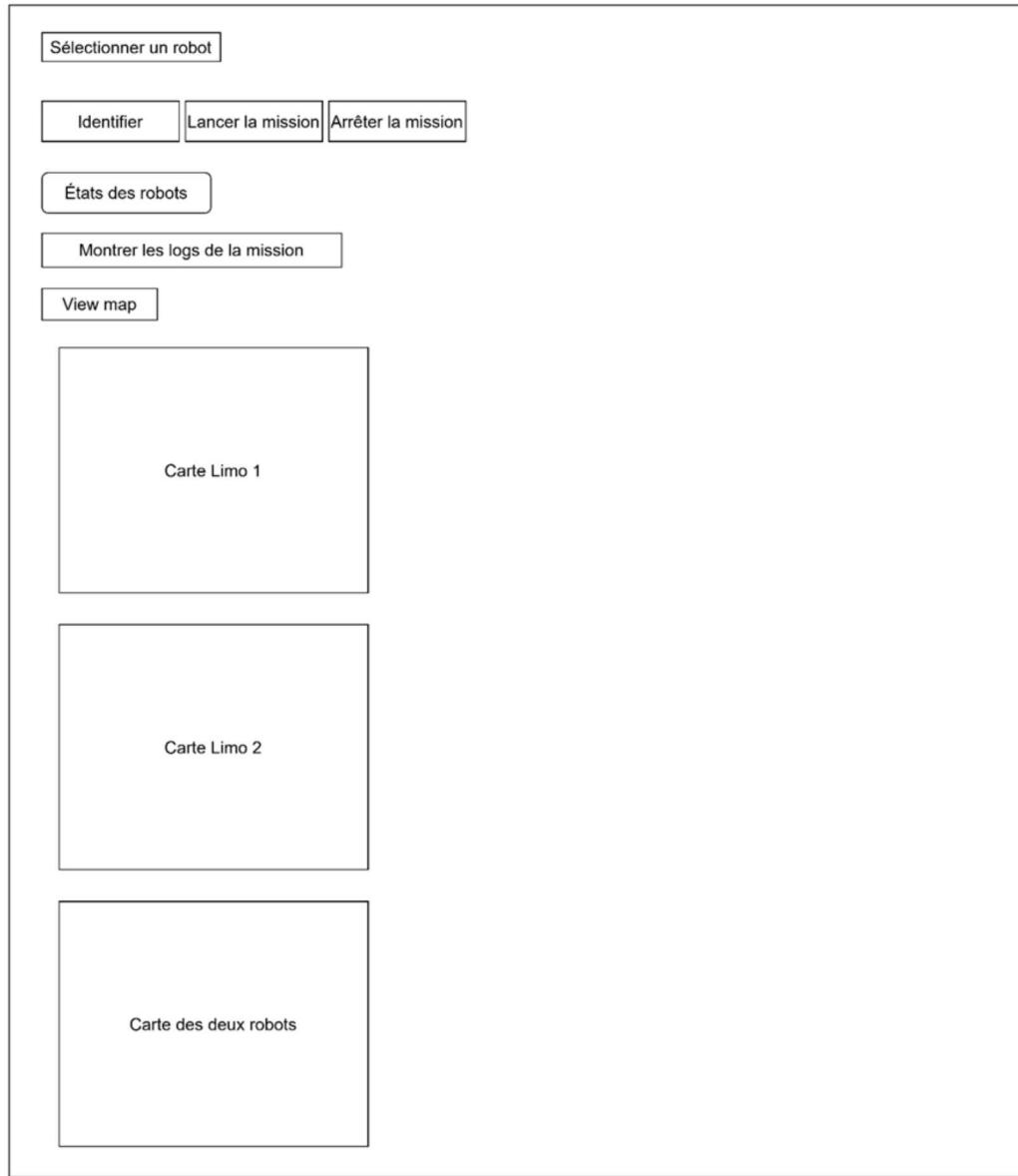
Nous nous baserons sur les algorithmes de type « SLAM » afin de simultanément construire et améliorer la carte de l'environnement et de pouvoir s'y localiser.

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 gauche, à travers deux boutons, les commandes “Identifier” (RF1), “Lancer la mission” et “Arrêter la mission” (RF2). Au-dessus de ces trois boutons, nous avons une section afin de sélectionner de quel robot d'agit-il.

En dessous de ces boutons, on affichera en premier le type de robot et son état présent (état Init, waiting, on mission, on stop,) (RF3). En dessous de ceci, il sera possible d'afficher les logs de la mission numéro 'x' en sélectionnant le numéro de la

mission souhaité et en appuyant sur le bouton “Montrer les logs de la mission numéro ‘x’ ” (RC1). Un peu plus bas, nous avons un bouton “view map” permettant d'afficher les 3 cartes suivantes : celle du premier robot, celle du second robot dans le cas où nous avons deux, et la carte ou les deux robots sont présent dessus.

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 grise 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 et traitées par le package slam_gmapping, 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.

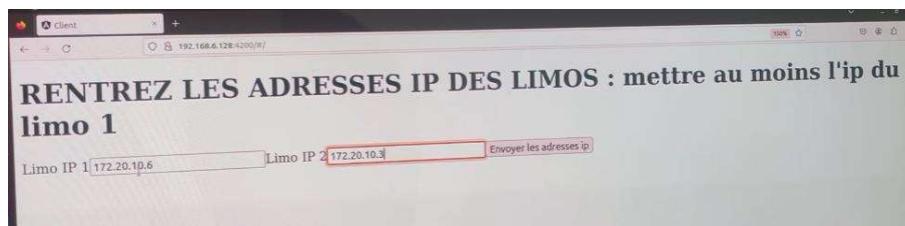
En premier lieu, pour démarrer les conteneurs sur les robots Limo, il faut se connecter en SSH au véhicule, puis télécharger le projet depuis le Gitlab et naviguer vers le dossier « Limo ». Pour lancer le script bash qui construit entre autres l'image Docker, nous pouvons utiliser la commande d'execution suivante :

```
« ./start-limo.sh »
```

En second lieu, pour démarrer les conteneurs de l'interface sur la station au sol, il faut télécharger le projet et naviguer vers le dossier « Interface ». Il faut ensuite lancer le script bash qui s'occupe de construire les différentes images docker avec la commande :

```
« ./start-interface.sh »
```

Une fois l'interface lancée, nous pouvons directement fournir dans l'interface graphique, c'est-à-dire à partir du client dans “localhost:4200”, les 2 adresses des robots Limos respectivement afin de permettre au serveur de gérer les requêtes de l'interface.



Ici, LIMO IP 1 représente l'adresse IP du conteneur 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. Sans oublier de rajouter le paramètre “1” dans les scripts de lancements. Donc, en générale, Les commandes pour les scripts de lancement se résument ainsi:

0. Ouvrir 2 terminaux dans sa VM ubuntu.
1. Dans chaque terminal, cd prj_ws puis source devel/setup.bash
2. Dans le premier terminal, cd /INF3995-102/Limo puis sudo ./start-limo.sh 1 (on met 1 si en simulation).
3. Dans le deuxième terminal, cd /INF3995-102/Interface puis sudo ./start-interface.sh 1

4. Processus de gestion

4.1 *Estimations des coûts du projet (Q11.1)*

De nombreux coûts sont à envisager afin de mener à bien ce projet. Le coût total prend en compte le salaire de chaque membre, le prix de deux robots Limo, ainsi que le prix des ordinateurs pour l'équipe.

Pour un temps de travail de 630 heures pour une équipe de 5 membres, soit 126 heures par personne, le salaire des 5 membres développeurs-analyste payé 130\$/heures et de la coordinatrice de projet payée 145\$/heure sera de 100'170.00 \$.

Le prix unitaire d'un robot Limo de la compagnie Agilex est fixé à 3'851.49 \$. Or, ce projet nécessite deux robots, ce qui coûtera alors 7'702.98\$.

L'ordinateur choisi pour chaque membre est le Lenovo IdeaPad 3i 15,6 pouces possédant un processeur Intel i7 et 12 Go de RAM fournissant de bonnes performances pour développer le projet. Le coût de cet ordinateur est de 1100\$ (2). Le total pour les 6 membres de l'équipe sera de 6600 \$ (6 * 1100\$).

Il faudra également prendre en considération le loyer du local dans lequel la conception du projet sera faite pour une durée de 4 mois. Un local optimal pour des réunions et le développement du projet va nous coûter environ 1500\$ / mois, soit 6000\$ pour la durée du projet. Le local (3) répond parfaitement à nos besoins. De plus, celui-ci est déjà meublé, ce qui va nous faire économiser des frais supplémentaires.

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

Investissement	Cout
Salaires des membres	100 170.00 \$
Robots limo	7 702.98 \$
Ordinateurs Lenovo (x6)	6 600.00 \$
Local	6 000.00 \$
Total	120 472.98 \$

4.2 *Planification des tâches (Q11.2)*

4.3 Calendrier de projet (Q11.2)

Livrable soumis	Date de remise	Requis complété
PDR	10 février 2023	R.F.1, R.F.2
CDR	24 mars 2023	R.F.3, R.F.4, R.F.5, R.F.10, R.C.1
RR	20 avril 2023	R.F.6, R.F.7, R.F.8, R.F.9, R.F.11, R.F.12, R.F.13, R.F.14, R.F.16, R.F.17, R.F.18, R.F.19, R.F.20, R.F.21, R.C.2, R.C.3, R.C.4, R.C.5, R.Q.1, R.Q.2

4.4 Ressources humaines du projet (Q11.2)

La réalisation de ce système d'exploration sera faite par une équipe composée de 6 membres : 5 développeurs-analystes et d'une coordinatrice 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 ce qui fournit un grand atout pour l'équipe.

Mady Semega possède de fortes aptitudes côté serveur. Il possède une grande expérience en gestion de système sur plusieurs serveurs, acquise 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 Poly Orbite 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)

L'accomplissement de ce projet est divisé en trois biens livrables représentant chronologiquement l'avancement du projet. Il est important qu'avant la remise de chaque livrable mais également durant le processus de développement de s'assurer que tout ce qui a été accompli jusqu'à présent est conforme aux attentes. Afin d'effectuer cela, chaque requis complété sera déclaré comme terminé. Ces requis seront également vérifiés à travers une simulation à l'aide du logiciel Gazebo mais également en les testant directement sur les robots physiques. Afin de vérifier le code de chacun, avant de pouvoir intégrer une nouvelle fonctionnalité à la branche principale, une approbation par les pairs est requise. Afin d'assurer une bonne compréhension du code, une convention a été adopté sur la langue pour code. Afin que chaque membre soit informé des avancées des autres, des mises à jour régulières sur l'avancement des tâches sont requises. Des périodes de co-coding seront faites pour favoriser l'entraide au sein de l'équipe ainsi qu'une avancée rapide et efficace.

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)

De nombreux risques peuvent survenir lors de l'avancement du projet ayant différentes échelles d'importances. Certains peuvent pourraient retarder l'avancement du projet et d'autres impactant et modifiant la façon de travailler de l'équipe. Un des principaux risques à prévoir est une panne de robot ou qu'il ne fonctionne plus. Dans ce cas, il faudra commander un nouveau robot et de le reconfigurer afin qu'il fonctionne. Cette procédure peut prendre plusieurs semaines entre le moment où l'on passe la commande et celui où il sera prêt à être utilisé. Cela pourrait retarder le projet en plus d'être coûteux dans le cas d'un nouvel achat de matériel.

Effectuer un échéancier de tâches qui ne s'avère pas réalisable constitue un risque auquel on peut être confronté. En effet, dans le cas où notre architecture vient à être modifier, ou que l'on se rend compte que certaines tâches doivent être en priorité pour permettre l'accomplissement d'une autre. Ce risque est mordoré, dans le cas où il survient, il peut être fixer en quelques jours en réattribuant les tâches prioritaires et en mettant à jour l'échéancier.

Un risque auquel nous risquons de faire face est celui lié à un changement d'avis du client. En effet, le client peut souhaiter modifier certaines choses au projet, cela risque alors de retarder le projet dans sa totalité. Il s'agit d'un risque important, auquel il n'est pas réellement possible d'estimer un délai pour le régler car le client peut demander de modifier quelque chose qui peut prendre quelques heures tout comme modifier toutes une fonctionnalité ou en ajoutant une ce qui pourrait prolonger le projet de quelques jours voire semaines.

Il est possible que le fonctionnement des robots de soit pas le même en simulation et sur robot physique. Ceci représente un risque auquel il serait difficile d'identifier le réel problème et donc ce qu'il faudrait apporter comme changement au code pour que tout soit adéquat. Pour éviter d'être confronté à ce risque, nous effectuerons le plus de tests possibles sur la simulation ainsi que sur les robots.

Une mauvaise gestion du temps représente un grand risque au sein de ce projet. En effet, si certaines tâches prennent plus de temps que prévu, cela risque d'empêter sur l'accomplissement d'autres requis et pourrait devenir un problème majeur lors de la remise d'un livrable. Il faudrait alors effectuer plus de réunions d'équipe pour clarifier les priorités sur lesquelles se concentrer et probablement en abandonner d'autres. Pour éviter cette situation, il serait important de bien évaluer le temps pour chaque tâche, mais également mettre des échéances qu'il faudrait respecter. Ce risque est important et peut impacter le projet dans sa globalité.

5.3 Tests (Q4.4)

Des tests peuvent être fait pour chaque sous-système pour s'assurer du bon fonctionnement de ces derniers. Chaque partie de code aura son banc de tests associé avec des tests unitaires et d'intégrations, en plus des essais sur le robot physique et de la simulation. Nous aurons des tests permettant de vérifier que tous les packages sont bien installés et que leur version est la bonne.

Le premier test qui peut être fait serait un test de connexion entre les serveurs embarqués et la station au sol. Ce test permet de vérifier que les sous-systèmes sont bien reliés entre eux. Ici, nous comptons afficher des données qui doivent être communiquées d'un serveur à un autre et vérifier que l'affichage obtenue correspond à celui souhaité.

Afin de tester nos multiples serveurs, des fichiers tests seront fournis dans lesquels des requêtes seront envoyées et nous vérifierons alors si la réponse reçue est bien adéquate.

Il serait également pertinent de s'assurer de recevoir les requêtes entre l'interface utilisateur et le serveur qui traite les commandes au sol. Ce test peut s'effectuer en lancer la page Web, initiant la connexion avec un robot, envoyer des commandes et vérifier que le robot les reçoit bien.

Pour le système embarqué, il sera possible de le tester grâce à la simulation. Nous pourrons ainsi voir si le robot effectue bien les comportements correspondant aux bonnes commandes reçues. On vérifiera également que le système embarqué fonctionne correctement en utilisant les robots physiques. Dans le cas de la simulation, on compte également modifier l'environnement en ajouter des obstacles par exemple pour vérifier que le déplacement du robot est conforme.

Notre site Web sera testé l'aide de la librairie Jasmine. Chaque page, composant et service de notre site aura ses tests unitaires et tests d'intégrations associés.

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. Ce logiciel permettra de mettre à jour les différentes versions et les progressions du code source.

Les différents modules seront divisés en deux parties. D'un côté, il y aura l'interface web et de l'autre le véhicule automobile Limo. Chaque partie contiendra le code source de chaque sous-système du travail demandé. L'interface Web contiendra le serveur statique qui distribuera la vue de l'interface et le serveur au sol. Le dossier Limo, quant à lui, possèdera un serveur Node.js qui communiquera avec l'utilisateur ainsi le serveur Rosbridge qui permettra de contrôler les robots.

Afin de s'assurer que tous utilisent les mêmes versions de développement pour chaque module, des conteneurs Dockers seront utilisés afin d'avoir des versions identiques de Node.js et Ros. Cela permettra également d'isoler les différents serveurs pour un bon entretien du code et fournir une meilleure intégration.

Un fichier de tests unitaires accompagnera chaque composante logicielle afin d'assurer son 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.

Durant tout le long de l'avancement du projet, la documentation sera tenue à jour. Il y aura un fichier d'information (README.md) qui contiendra toutes les informations sur le bon fonctionnement de l'application, les dépendances et les commandes utilisées de l'application. La documentation de conception sera également mise à jour régulièrement au fur et à mesure que le projet soit développé. Tous les changements dans la configuration et l'organisation sera reflété dans le document de conception dès que la décision aura été prise.

Chaque membre de l'équipe développera chacune de ses tâches sur différentes branches. Lorsque chaque tâche aura atteint à un niveau satisfaisant, avec le logiciel Git, la branche sera intégrée dans le code source stable en s'assurant de résoudre tous les problèmes éventuels qui peuvent être engendré. Une approbation d'au moins 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)*

Initialement, on avait prévu utiliser un rover et un drone. Or, en cours de développement, avec tous les problèmes rencontrés avec le drone, nous avons décidé de laisser tomber le drone (même s'il n'avait jamais décollé) et opté pour le deuxième rover à notre disposition. Cela nous a permis d'avancer plus rapidement dans la réalisation des tâches prévues pour le CDR et de ne pas rester bloqué sur une architecture incertaine. Ainsi, notre architecture logicielle a considérablement changé, et ceux qui avez prévu de travailler sur le drone ont finalement dû se renseigner davantage sur le rover afin d'assister ceux qui travaillait initialement dessus.

Aussi, pour la détection et l'évitement d'obstacle, nous avions décidé d'implémenter un algorithme d'exploration qui enverrait des commandes de vitesse aléatoires au robot et utiliserait le LiDar pour détecter les obstacles et ajuster sa trajectoire en conséquence. La première itération de cette algorithme était très limitée. Nous avons donc décidé d'utiliser un algorithme plus complet, nommé potential fields pour que le robot puisse se rendre à des objectifs défini aléatoirement sur la carte de manière autonome. Cet algorithme était meilleur que le précédent mais il n'était toujours pas assez consistant. Notre choix final s'est donc porté sur Explore Lite. Explore Lite est une librairie de navigation autonome qui permet de planifier et d'exécuter une trajectoire pour un robot mobile tout en évitant les obstacles sur son chemin. Cette librairie est basée sur le navigation stack, qui est un ensemble de packages ROS (Robot Operating System) qui permettent la planification de trajectoires et la navigation pour les robots mobiles.

Explore Lite utilise des capteurs tels que les scanners laser pour détecter les obstacles autour du robot et planifie une trajectoire pour éviter ces obstacles tout en essayant de couvrir la zone inconnue. La librairie utilise également des algorithmes pour déterminer la direction dans laquelle le robot doit se déplacer pour maximiser la couverture de la zone inconnue.

Le navigation stack, quant à lui, utilise des algorithmes pour résoudre le problème de planification de trajectoires pour les robots mobiles en prenant en compte les obstacles détectés par les capteurs du robot. Il utilise des données provenant de capteurs tels que les scanners laser et les caméras pour créer une carte de l'environnement et planifier une trajectoire optimale pour le robot. En cas de détection d'obstacles, le navigation stack ajuste la trajectoire du robot pour éviter ces obstacles.

En utilisant Explore Lite avec le navigation stack, la détection et l'évitement d'obstacles deviennent plus simples à implémenter. Cela permet d'éviter la complexité des algorithmes random walk et potential fields et assure une meilleure couverture des zones inconnues par le robot mobile.

La remise du second livrable ne s'est pas exactement passé comme prévu. Ainsi l'agence a fourni un délai d'une semaine supplémentaire pour la remise du CDR.

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

Un banc de test a été implémenté à tous les niveaux, c'est-à-dire aussi bien au niveau client qu'au niveau serveur du côté Interface sans oublier le serveur du robot. Il s'agit encore d'un prototype, mais la majorité des fichiers ont été testés dans le client de l'interface. Puis, du côté Frontend, les tests ont été réalisé selon le langage Jest, tandis que dans les serveurs, ce sont plutôt Mocha, Chai et Sinon qui ont été privilégiés.

Pour la navigation du robot, l'utilisation de la librairie explore lite nous a permis de remplir tous nos objectifs en ce qui concerne l'autonomie du robot pour définir sa trajectoire et l'évitement de tous les obstacles sur celle-ci.

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-wdikoVDNlbwDpTRoh2FM

[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>