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 No <***101***>

De Blas David
Deloumeau Nicolas
Desgroseilliers Philippe
Louesdon Théo
Ngay Munga Exaucé-Constantin

24 mars 2023

Table des Matières

1. Vue d'ensemble du projet

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

Le but du projet consiste à concevoir un système composé d'un ensemble de logiciels permettant à 2 robots de cartographier une pièce d'un bâtiment. Les données récoltées par les robots doivent être envoyées à l'opérateur via une interface web où il sera possible de visualiser la représentation de la pièce. L'opérateur pourra envoyer des instructions aux robots et surveiller leur état grâce à l'interface web, lui permettant ainsi d'interagir avec eux. Concernant le choix des deux robots, nous pouvons soit opter pour un drone CogniFly (The CogniFly Project, 2023) et un robot mobile AgileX Limo (Agilex Robotics, 2022) soit pour deux robots du même type. Tout au long du projet, l'équipe s'assurera que les robots respectent un certain nombre de critères assurant la qualité au niveau du matériel et du logiciel.

Concrètement, le système est composé de trois grandes sections : une station au sol, une partie embarquée pour les deux robots et une simulation logicielle afin de planifier le comportement des robots. On doit identifier des technologies adéquates pour chaque section du système et trouver un moyen de les relier afin de garantir son bon fonctionnement.

Le travail se divisera tout au long du trimestre en 3 parties :

- Le Preliminary Design Review (PDR)
- Le Critical Design Review (CDR)
- Le Readiness Review (RR)

Pour la première étape, il s'agit de construire un prototype minimal du projet, pour la seconde, il est nécessaire de développer une structure complète du système et pour la dernière, il faudra créer un système complet et opérationnel.

1.2 Hypothèse et contraintes (Q3.1)

1.2.1. Hypothèses

- Nous supposons que l'environnement Gazebo (Hedges, et al., s.d.) possède une physique et des composants des robots assez proches de la réalité. Ainsi, si nous arrivons à implémenter le comportement souhaité des robots dans gazebo (Open Source Robotics Foundation, 2023), on ne devrait pas trop avoir de problème à reproduire ce comportement dans la réalité.
- Nous présumons que l'ensemble du matériel fourni ne présente pas, ou peu, de problèmes qui pourraient influer sur le comportement des robots et sur le fonctionnement des différents capteurs.
- Nous faisons l'hypothèse que le temps de transmission des commandes entre la station au sol et les robots sera très rapide, car tous les tests se feront dans une pièce fermée et utilisée à cet effet, ce qui réduit fortement les interférences.

1.2.2 Contraintes à prendre en compte

1.2.2.1 Contraintes matérielles et logicielles

- Tous problèmes matériels et logiciels (endommagement du matériel et gestion de bugs)
- Tous les composants logiciels doivent être conteneurisés avec Docker (Docker Inc., 2023)
- Le logiciel complet de la station doit être lancé avec une seule commande impliquant Docker (Docker Inc., 2023) dans un terminal Linux roulant Ubuntu 20.04 (Canonical Ltd., 2018).
- L'interface utilisateur devrait rester identique lorsque la station au sol est reliée à la simulation ou aux robots réels.
- Le mouvement des robots doit se faire au niveau du code embarqué, la station au sol ne doit pas dicter ses mouvements

1.2.2.2 Contraintes techniques

Identifiant	Description
R.F.1	Chaque robot doit répondre individuellement à une commande "Identifier" qui clignote une DEL ou émet un son.
R.F.2	Les robots doivent répondre aux commandes "Lancer la Mission" et "Terminer la Mission".
R.F.3	L'interface utilisateur doit montrer le type et l'état de chaque robot.
R.F.4	Les robots doivent être en mesure de se déplacer de façon autonome.
R.F.5	Les robots doivent être en mesure d'éviter des obstacles détectés par leurs capteurs
R.F.6	Un bouton "Retour à la base" doit être disponible pour faire retourner les robots à moins de 0.5m de leur position initiale.
R.F.7	La fonction "Retour à la base" doit être automatiquement déclenchée si la batterie atteint un pourcentage de moins de 30%. Le pourcentage de la batterie doit être affiché sur l'interface et les robots ne devraient pas décoller si leur batterie est à moins de 30%.
R.F.8	La station au sol doit recueillir les données captées par les robots et produire une carte de l'environnement qui est visible sur l'interface.
R.F.9	Lors d'une mission, la position du robot doit être visible sur la carte.
R.F.10	L'interface doit être disponible comme service web et visible sur plusieurs appareils (mobile, PC, tablettes) via réseau. Au moins deux appareils doivent être en mesure de se connecter en même temps.
R.F.11	Les deux robots utilisés sont de types différents et explorent en même temps.
R.F.13	Le système doit pouvoir détecter un "crash" de drone et afficher l'état "crashed" sur l'interface utilisateur.
R.F.17	Une base de données présente sur la station au sol doit enregistrer des attributs comme la date et l'heure de la mission, le temps de vol, la distance totale parcourue, etc. L'utilisateur doit aussi être en mesure de fouiller les anciens logs des missions précédentes à partir de l'interface.

R.F.18	La carte générée durant une mission doit être sauvegardée sur la station au sol. L'interface doit aussi permettre d'afficher les cartes des missions précédentes sauvegardées dans la base de données.
R.F.19	Les robots doivent pouvoir communiquer entre eux leur distance par rapport aux points de départ, si le rover est plus loin, il doit montrer une icône sur son écran.
R.C.1	Des logs de débogage doivent être disponibles en continu pour s'assurer que le système fonctionne correctement. Ces logs doivent être sauvegardés sur la station et accessibles en tout temps.
R.C.2	Le logiciel au complet doit pouvoir être lancé à partir d'une commande Linux (docker-compose). La procédure doit être détaillée dans un README pour la simulation et le physique.
R.C.3	L'environnement virtuel de simulation doit être généré aléatoirement avec minimum 3 murs en plus des 4 murs externes.
R.C.4	L'interface utilisateur doit être facile d'utilisation, lisible, et suivre les 10 heuristiques d'interface usager (Nielsen Norman Group, 2020).
R.C.5	Le système doit s'adapter automatiquement pour marcher avec 1 ou 2 robots.
R.Q.1	Le format du code doit être standardisé à travers le projet.
R.Q.2	Des tests unitaires doivent être écrits pour chaque composante logicielle. Si aucun test ne peut être écrit, une procédure de tests doit être détaillée.

Tableau 1 : Requis obligatoires (beige) et optionnels (mauve) qui seront implémentés au courant de la durée du projet

1.2.2.3 Contrainte concernant l'équipe

- Toutes activités reliées aux autres cours (intra, travaux pratiques, etc.) risquent de retarder certaines tâches initialement prévues.
- Retard accumulé à la suite d'incompréhensions vis-à-vis d'une tâche.
- Fatigue accumulée selon les différentes périodes de la session.
- Un membre de l'équipe a quitté le projet, nous ne sommes plus que 5, cela risque d'avoir un impact sur le nombre de requis que nous serons capables de compléter et/ou sur la qualité des requis.

1.3 Biens livrables du projet (Q4.1)

Au fur et à mesure du projet, nous remettrons 3 livrables

Livrable	Date	Description
Preliminary Design Review (PDR)	10 février 2023 à 23H55	- Prototype minimal du projet
Criticical Design Review (CDR)	24 mars 2023 à 23H55	- Prototype avancé
Readiness Review (RR)	21 avril 2023 à 23H55	- Système complet et opérationnel

Tableau 2 : Livrables attendus au long de la session

Pour chacune des remises, des artéfacts relatifs à chaque livrable seront remis aux dates issues du tableau ci-dessus. Ces artéfacts seront organisés sous forme de dossier dans GitLab (GitLab B.V., 2023). Un rapport technique du projet, qui sera de plus en plus détaillé au fil des livrables, sera présent dans le répertoire principal et le code source sera divisé en sous-répertoires contenant les fichiers de conteneurisation Docker (Docker Inc., 2023) avec leurs fichiers de configuration pour lancer chaque module du système séparément. Enfin, des README fourniront les instructions pour exécuter les commandes dans l'invité de commande.

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Nous nous sommes entendus sur une méthode de travail décentralisée pour le développement du projet (Collin, Desmarais, & Gendreau, 2023). Ce type de structure nous permet d'identifier nos forces et nos faiblesses individuelles pour assigner les tâches aux membres les plus spécialisés ou à ceux qui y sont le plus intéressés. En travaillant individuellement ou au plus avec un coéquipier, nous

pouvons nous assurer que le projet se déroule sans trop de problèmes, si nous respectons bien les échéanciers prévus au calendrier.

Pour soutenir cette structure, nous avons décidé en équipe de déléguer le rôle de coordinateur de projet à Nicolas avec les cinq autres membres agissant comme analyste développeur. Le rôle de coordinateur est là pour s'assurer que la charge de travail sera distribuée équitablement entre les 6 membres de l'équipe en fonction de chacune de nos forces et faiblesses et les développeurs analystes, dont Nicolas fait aussi partie, sont là pour construire le projet. Nicolas sera au centre du réseau décentralisé, mais nous sommes aussi en mesure de communiquer entre nous pour résoudre toute défaillance qui pourrait survenir.

L'utilisation de rencontres hebdomadaires est un autre aspect que nous avons mis en place pour assurer le développement continu du projet et Philippe est celui qui s'occupera de les planifier et de dresser les ordres du jour. Ces réunions seront non seulement pour faire un bilan de la semaine précédente, mais aussi pour dresser une liste des tâches que nous planifions accomplir dans la semaine qui suit. Nous allons aussi utiliser ce temps pour soulever toutes inquiétudes ou problématiques que nous avons pour voir si nous pouvons en venir à une solution pour le long terme.

Pour ce qui est des choix à faire quant aux technologies utilisées, nous allons procéder d'une manière à chaque membre l'équipe ait une voix. Nous ne voulons pas mettre un coéquipier en situation d'apprentissage aigu juste parce qu'une ou deux personnes sont familières avec une technologie spécifique. Cependant, quand vient l'implémentation d'une fonctionnalité ou d'un requis, nous allons la laisser à la discrétion du membre qui l'effectue, car nous nous faisons assez confiance pour nous laisser choisir parmi les options possibles et d'implémenter la meilleure.

2.2 Entente contractuelle (Q11.1)

En raison de la nature du contexte de ce projet, nous optons ici pour un contrat de type prix ferme (Collin, Desmarais, & Gendreau, 2023). En effet, nous avons une liste détaillée des requis techniques et physiques pour le projet. Ce qui

nous donne une assez bonne certitude quant à l'estimation du temps et des coûts nécessaires pour la réalisation du projet. De plus, le temps est limité à 4 mois et il est nécessaire d'avoir une remise complète.

Les avantages de ce type de contrat sont que le promoteur n'a pas nécessairement besoin de suivre le développement du projet avec un œil attentif et qu'il sait à quoi s'attendre quant au coût avant même de commencer sa conception. Le tout est défini en avance avec le contractant. En revanche, ce type de contrat nécessite une expertise prédéterminée de la part des contractants, car le cahier des charges doit être mis au clair dès le début, ce qui est le cas dans notre contexte.

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)

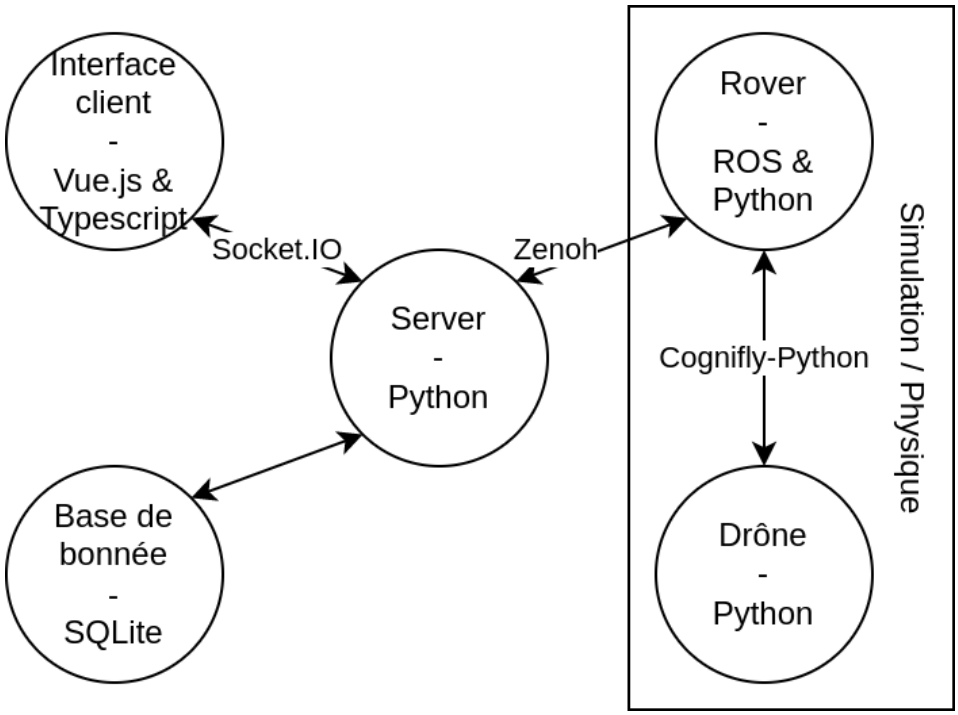

Figure 1 : Diagramme de l'architecture logicielle générale

L'utilisation de Zenoh (Eclipse Zenoh, 2023) pour la communication entre la station au sol et le(s) robot(s) permet aux utilisateurs de se connecter facilement aux robots et de les commander en utilisant des commandes simples. Les robots

peuvent également publier des données de capteurs et d'autres informations, qui peuvent être reçues par la station en s'abonnant à ceux-ci. De plus, l'utilisation de Zenoh (Eclipse Zenoh, 2023) permet une communication directe et efficace entre les robots sans avoir besoin de la station au sol (en P2P). Cela permet une latence faible et une communication efficace entre les robots. Zenoh (Eclipse Zenoh, 2023) offre donc une solution avantageuse au niveau de la facilité et de l'efficacité de la communication. Ce système de communication nous permettra de faire les requis R.F.1, R.F.2, R.F.3, R.F.6, R.F.8, R.F.9, R.F.10 et R.C.1. Tous ces requis utilisent principalement la communication entre la station au sol et les robots, qui sera faite avec Zenoh (Eclipse Zenoh, 2023).

La décision de prendre Python (Python Software Foundation, 2023) et Vue.js (VueJs, 2023) comme langages de programmation est réellement basée sur la préférence des membres de l'équipe. De plus, Vue.js (VueJs, 2023) ayant un cadriciel léger et Python (Python Software Foundation, 2023) étant un langage interprété, facile à utiliser permet d'uniformiser la compréhension de l'équipe sur ce qui a été fait par d'autres membres.

L'utilisation de Typescript (Microsoft, 2023) plutôt que JavaScript (ECMA International, 2022) est encore une fois, pour faciliter la compréhension du code pour les autres membres de l'équipe. Cela obligera également les membres de l'équipe à écrire explicitement les types de variables et les types de retours de fonction.

SQLite (SQLite Consortium, 2023) est encore une fois une préférence de l'équipe, car nous avons appris le langage SQL (W3Schools, 2023) et il sera assez performant et léger pour ce qui est demandé dans le cadre de ce projet.

Quant à ROS (ROS, 2023), nous avons pris la décision de le favoriser à ROS2 (Open Source Robotics Foundation, 2023) pour sa documentation très précise et abondante. De plus, l'environnement de simulation et les robots physiques sont déjà installés de façon à héberger ROS (ROS, 2023).

3.2 Station au sol (Q4.5)

La station au sol sera composée d'un contrôleur accessible sur le web, de services, de robots et d'une base de données. Ces modules communiquent ensemble par l'entremise de Zenoh (Eclipse Zenoh, 2023).

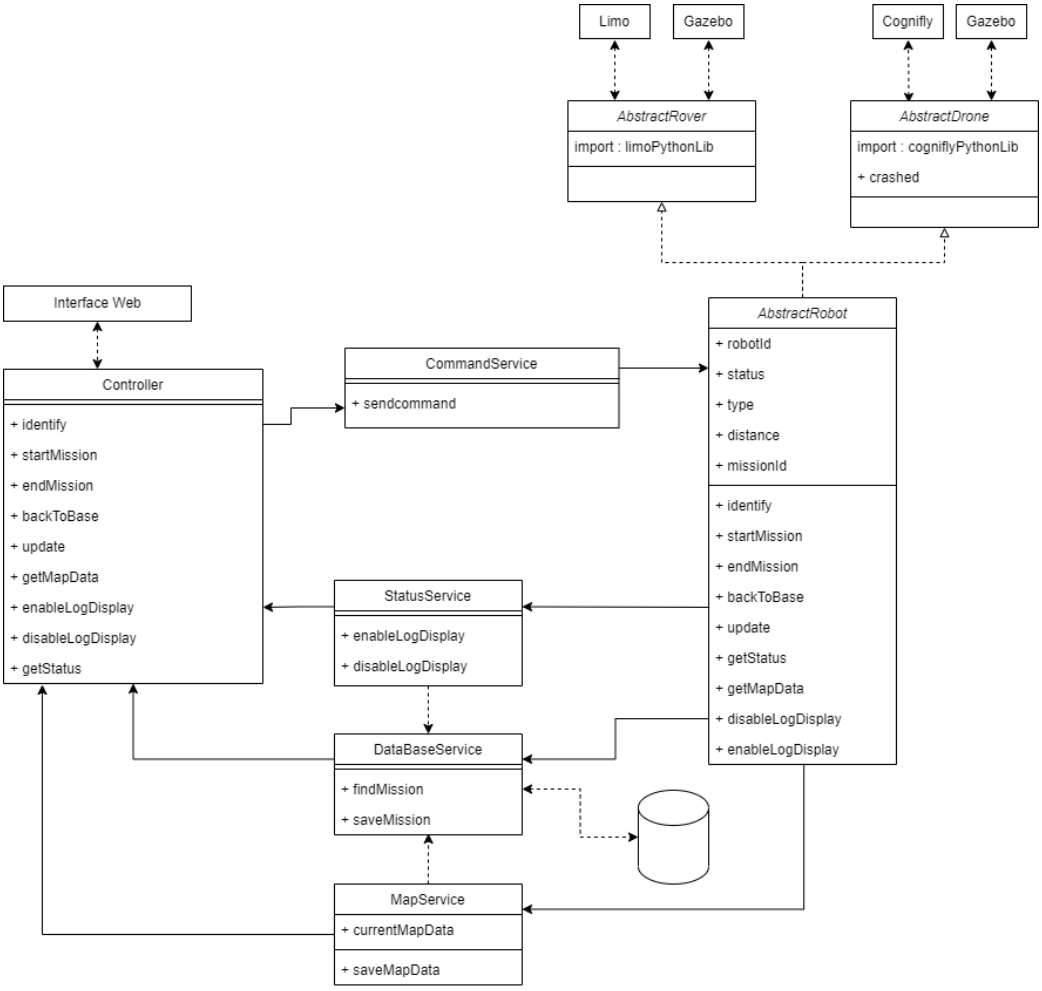

Figure 2 : Diagramme de classe de la station au sol

Le contrôleur accessible sur une interface web permet à l'utilisateur d'envoyer des commandes de base aux robots. La première commande "Identify" sera nécessaire afin d'identifier les robots, tel qu'il est demandé dans le requis R.F.1. Il y a aussi les commandes startMission et endMission qui seront utilisées pour lancer et mettre fin à une mission, tel qu'il a été énoncé dans le requis R.F.2. Il y aura une commande backToBase pour ramener le robot à son point de départ, tel qu'il a été demandé par le requis R.F.6. La fonction update sera aussi dans le

contrôleur afin de mettre à jour le logiciel de contrôle puisque nous avons choisi d'implémenter le requis R.F.14. La commande testSystem sera nécessaire afin de tester le système tel qu'il est indiqué par le requis R.C.1. Puis, la fonction getMapData sera utilisée afin de récupérer les informations de la carte graphique qui doit être générée.

Les services doivent s'occuper de toute la partie logique du système. Il y aura donc un mapService pour la gestion des cartes géographiques qui sont générées par les robots, un DataBaseService pour la gestion de la base de données et un CommandService pour gérer les commandes envoyées par le contrôleur.

Finalement, les robots auront beaucoup de méthodes identiques indépendamment de leur type et quelques méthodes qui varient selon leur type. Par exemple, l'état "crashed" ne doit être détecté que sur le CogniFly (The CogniFly Project, 2023).

3.3 Logiciel embarqué (Q4.5)

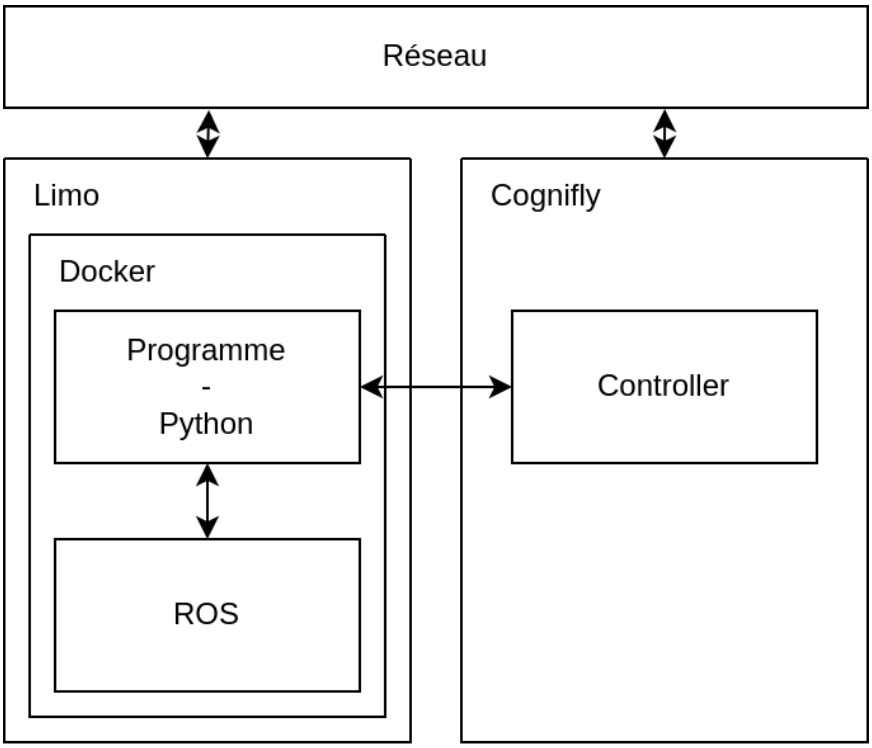

Figure 3 : Diagramme du logiciel embarqué

Le logiciel embarqué de notre système sera basé sur ROS (Open Source Robotics Foundation, 2014) pour faciliter l'interaction avec le robot et simplifier les processus de développement. Nous avons choisi d'utiliser Python (Python Software Foundation, 2023) comme langage de programmation pour le logiciel embarqué, car il dispose de nombreuses implémentations et librairies disponibles pour nous aider dans la réalisation de notre projet. La simplicité de Python (Python Software Foundation, 2023) permet également un développement plus rapide et facilite la maintenance et l'évolution du système. L'utilisation de ROS (ROS, 2023) nous permettra de compléter les requis R.F.6, R.F.7, R.F.13 et R.F.19.

En ce qui concerne la communication entre le rover, le drone et le serveur, nous utiliserons le protocole Zenoh (Eclipse Zenoh, 2023) pour simplifier les échanges de données. Cela permettra d'assurer une communication efficace et fiable entre les différents éléments du système, tout en simplifiant les processus de développement et de maintenance.

3.4 Simulation (Q4.5)

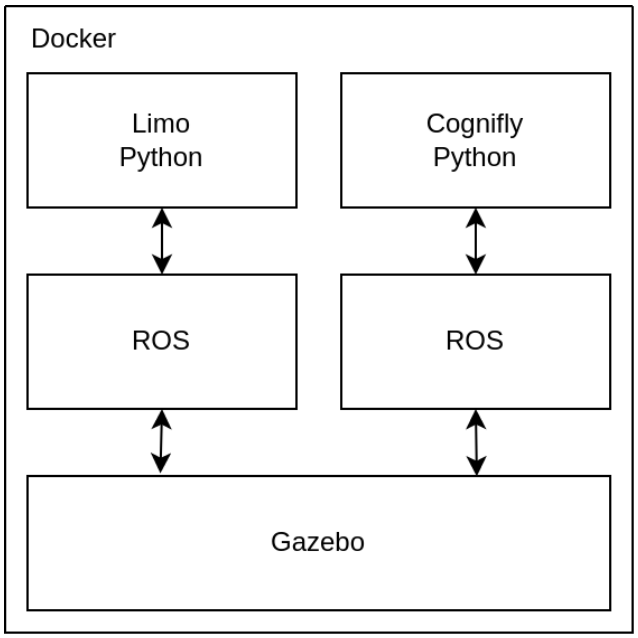

Figure 4 : Diagramme de la simulation

Pour améliorer la portabilité de notre programme, nous utiliserons un environnement Docker (Docker Inc., 2023). Cela nous permettra de garantir la

compatibilité de notre programme sur différents ordinateurs de l'équipe, en évitant les problèmes de dépendances matérielles et logicielles. En utilisant un simulateur Gazebo (Open Source Robotics Foundation, 2023) pour les simulations, nous pourrons également garantir un environnement physique réaliste, permettant ainsi au programme Python (Python Software Foundation, 2023) de ne pas faire la différence entre le robot physique et sa simulation. Cela permettra de faciliter les tests et les déploiements sur le robot réel.

3.5 Interface utilisateur (Q4.6)

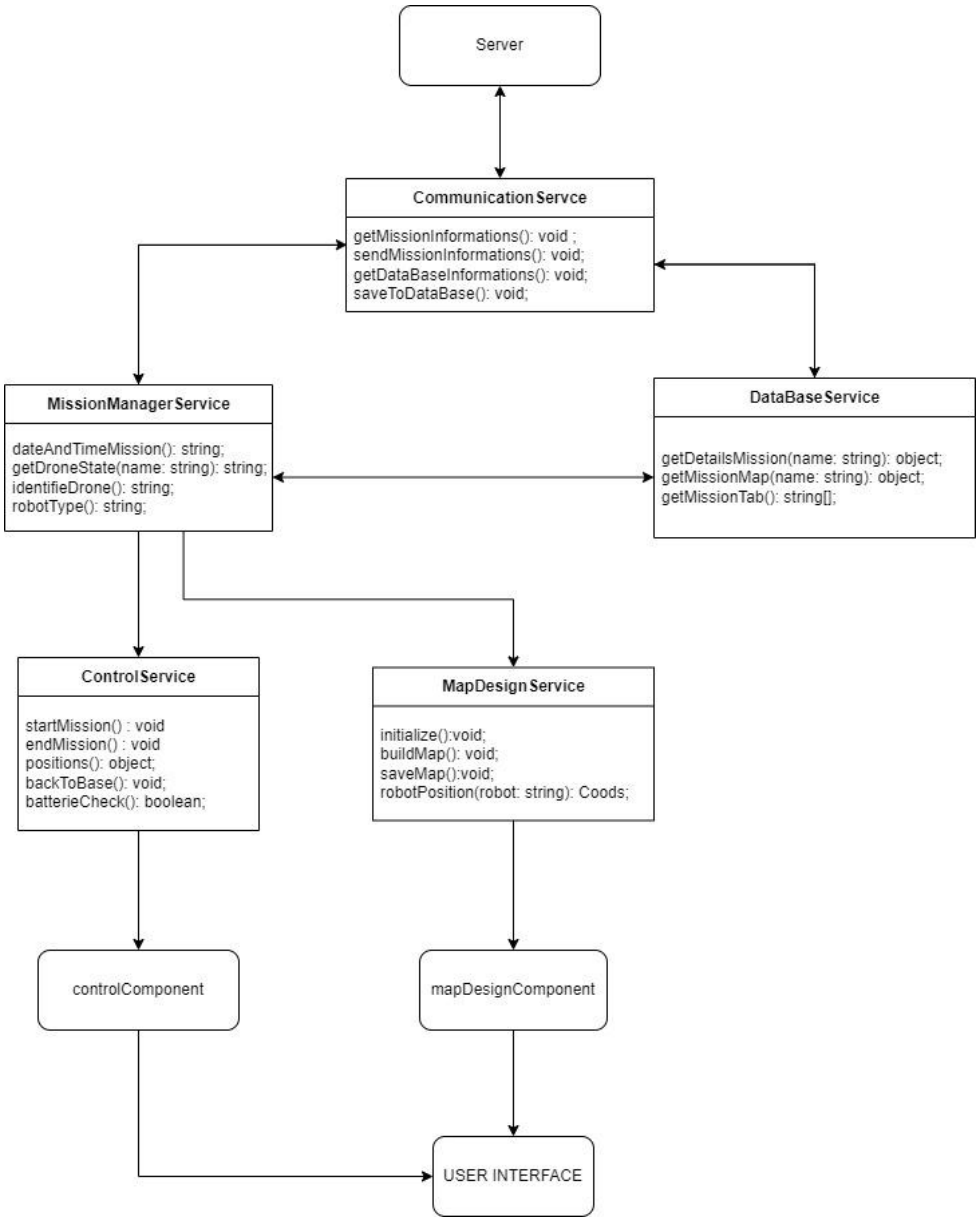

Figure 5 : Diagramme de l'interface utilisateur

L'interface utilisateur est un environnement web développé à l'aide du framework Vue.js (VueJs, 2023). La construction de cette structure est inspirée de celle que nous avions pour utiliser le framework Angular (Google, 2023) lors du deuxième projet intégrateur en sigle LOG2990. Nous utiliserons des services qui exécutent une logique, et des composants pour la gestion de l'affichage qui permettent l'interaction avec l'utilisateur. HTML (WHATWG, 2023), CSS (CSS Snapshot 2023, 2023), fichiers de test et fichiers TypeScript (Microsoft, 2023), tels sont les fichiers de code qui peuvent être trouvés dans notre interface utilisateur.

CommunicationService permet la communication de l'interface client avec notre serveur. La communication se fera via Socket.io (Socket.IO, 2023), ce qui permettra le transfert des informations nécessaires aux services MissionManagerService et DataBaseService pour le bon fonctionnement de l'interface.

Le service MissionManagerService est utilisé pour gérer les opérations de la mission. Celui-ci contient des sous-entités telles que MapDesign pour la carte et Control pour contrôler les robots effectuant les tâches. Ensemble, ils répondent aux exigences suivantes : R.L.3, R.F.1, R.F.2, R.F.3, R.F.6, R.F.7, R.F.8, R.F.9, R.F.11, R.F.13, R.F.14.

Le service DataBaseService fournit des informations provenant de la base de données, importantes pour les opérations de l'interface utilisateur. Le service MissionManager y récupère les données nécessaires à son fonctionnement et à celui de ses sous-systèmes. Cela répond aux exigences de R.F.17 et R.F.18.

3.6 Fonctionnement général (Q5.4)

Pour lancer notre système, il faut lancer la partie liée à la station de contrôle (client et serveur) en se plaçant dans le root du repository Gitlab (GitLab B.V., 2023) et en exécutant la commande ***docker compose up –build groundstation***. Le serveur et le client communiquent sur le port 8000, et le tout est lancé en mode production, ce qui signifie que le serveur sert les fichiers nécessaires au client. La station au sol devra être équipée de Docker (Docker Inc., 2023).

Pour faire fonctionner la simulation des robots, il faut suivre les étapes précisées dans le README du dossier sim. Ces étapes permettent de lancer une simulation avec deux robots et d'assigner un script d'exécution différent à chacun.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

En considérant que chaque membre du groupe consacre en moyenne six heures lors des séances de laboratoire et trois heures et demie en dehors de ces séances pour travailler sur le projet, nous pourrions arriver à une estimation des coûts pour une période de onze semaines de travail en utilisant les taux horaires suivant:

- Développeur-analyste : 130$/h
- Coordonnateur de projet : 145$/h

Nous obtenons:

Membre	**Salaire hebdomadaire**	**Total**
Coordonnateur de projet	145$/h * 9.5h = 1377.5$	15 152.5$
Développeur-analyste	130$/h * 9.5h = 1235$	13 585$

Tableau 3 : Prix total du coût de la main-d'oeuvre

Pour une équipe composée d'un coordinateur de projet et de cinq développeurs analyste sur une période de onze semaines soit un total de 627 heures de travail, nous obtenons un prix total de l'équipe de 15 152.5$ + 13 585$ * 5 = 83 077.5$

On ajoute 4300 dollars de coût de matériel, dont 1000 dollars pour la tour de contrôle, 400 dollars pour le CogniFly (The CogniFly Project, 2023) et 2900 dollars pour le LIMO (Agilex Robotics, 2022) (en sachant que la société nous fournit les 6 Laptops et le local), et nous obtenons un total de 87 377.5$.

4.2 Planification des tâches (Q11.2)

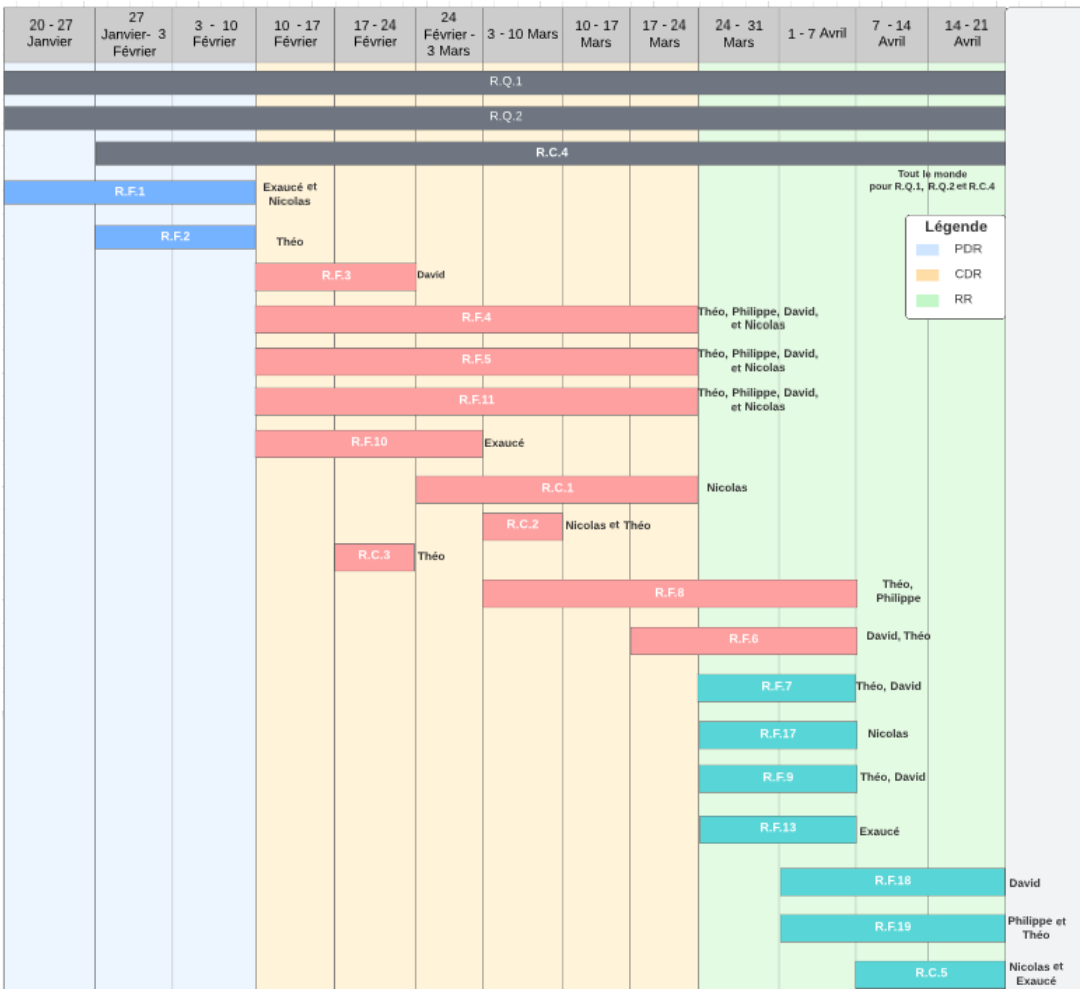

Figure 6 : Diagramme de Gantt de la planification des tâches pour le projet

4.3 Calendrier de projet (Q11.2)

Trois principaux livrables seront à remettre au cours de la session, correspondant aux trois principales versions de notre projet. Le calendrier ci-dessous donne les dates de remises et un aperçu de ce qui doit être complété pour chaque livrable.

Calendrier du projet		
Livrable	Date	Aperçu
Preliminary Design Review (PDR)	Vendredi 10 février 2023	- Plan de projet complet - Démo vidéo des 2 premiers requis
Critical Design Review (CDR)	Vendredi 17 mars 2023	- Plan de projet plus détaillé et révisé - Démo vidéo de 7 requis - Présentation technique des concepts du produit
Readiness Review (RR)	Vendredi 21 avril 2023	- Plan de projet final - Démo vidéo de tous les requis - Écriture des instructions liées à la compilation

Tableau 4 : Description des livrables à remettre tout le long de la session

Le premier livrable décrit la documentation initiale du projet. Il montre également le fonctionnement des 2 premiers requis par l'intermédiaire de démonstration vidéo.

Le deuxième livrable fournit une documentation plus détaillée et révisée du premier livrable. Il montre aussi les changements apportés depuis le premier livrable par l'intermédiaire d'une présentation technique. Aussi cette fois-ci 7 démonstrations vidéo sont requises pour illustrer la bonne fonctionnalité des requis suivants : R.F.1, R.F.2, R.F.3, R.F.4, R.F.5, R.F.10 et R.C.1.

Enfin le dernier livrable présente la documentation finale du produit. Tous les requis doivent avoir une démonstration vidéo représentant leur bon fonctionnement. Tous les tests doivent être finalisés et des instructions nécessaires à la compilation et au lancement du produit doivent être fournies.

4.4 Ressources humaines du projet (Q11.2)

Les ressources humaines du projet sont constituées de 1 coordinateur de projet et 5 développeurs-analystes. Nous avons aussi accès à l'équipe académique du cours INF3995 qui est constitué du Chargé de Cours Giovanni Beltrame et des deux chargés de laboratoires Guillaume Ricard et Antoine Robillard.

Les qualifications techniques et habiletés personnelles attendues du coordonnateur de projet sont de:

- Maîtriser des outils de gestion de projet comme GitLab (GitLab B.V., 2023)
- Pouvoir prendre des décisions rapidement, en ayant consulté les membres de l'équipe
- Déléguer les tâches de manière stratégique en fonction des compétences et des expériences de chacun des membres.
- Avoir une bonne capacité d'organisation, de planification et de leadership.

Du côté des développeurs-analystes, les qualifications techniques et habiletés personnelles attendues sont de:

- Maîtriser les technologies nécessaires pour effectuer le projet (Langages de programmation, écriture de tests, base de données, Docker (Docker Inc., 2023), Linux (Canonical Ltd., 2018))
- Maîtriser la programmation de systèmes embarqués
- Être ponctuel aux réunions comme aux remises
- Savoir travailler en méthode agile

Afin d'optimiser l'avancée de notre projet, le coordonnateur de projet devra probablement exécuter des tâches similaires à celles d'un développeur-analyste pour assurer un bon déroulement de celui-ci.

Le tableau ci-dessous répertorie les compétences et les expériences de chaque membre de l'équipe, qu'elles aient été acquises au cours de leur parcours professionnel ou personnel:

Qualifications	David	Philippe	Exaucé	Nicolas	Théo
Maîtrise du langage Python	X		X	X	X
Maîtrise de Vue.js et Typescript		X	X	X	
Maîtrise de programmation de système embarqué		X	X		
Écriture de bons tests	X	X		X	X
Maîtrise de Docker				X	
Utilisation d'une base de données SQLite	X		X	X	X
Connaissance du système d'exploitation Linux	X		X	X	X
Maîtrise des fonctionnalités de Gitlab	X	X	X	X	X
Développement d'une application Web	X	X	X	X	X
Développement d'un serveur Web en Python				X	X
Connaissance du protocole de communication Zenoh					
Connaissance des méthodes agiles	X	X		X	X

Tableau 5 : Expériences des membres de l'équipe

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Au cours de notre projet, nous veillerons à soumettre à tous nos livrables un processus de révision régulier, que ce soit pour les aspects techniques ou pour

les produits finaux. Pour assurer la qualité du code, nous adopterons les normes de style les plus couramment utilisées pour garantir la qualité de notre code. Ainsi, nous utiliserons PEP 8 pour le code Python (Python Software Foundation, 2023) et Prettier (Prettier, 2023) ainsi que ESLint (OpenJS Foundation, 2023) pour le code TypeScript (Microsoft, 2023) et Vue.js (You, Get Started, 2023). Cela nous permet de maintenir des standards élevés de programmation pour les noms de classes, de fonctions et de variables. De plus, tout ajout au code principal (merge) devra être revu par au minimum deux personnes de manière exhaustive pour s'assurer de la bonne qualité du code et du respect des conventions. Ceci répondra au requis R.Q.1.

5.2 Gestion de risque (Q11.3)

Au cours de la réalisation de notre projet, il est possible que des situations ou des problèmes techniques, matériels, ou de gestion surviennent, mettant ainsi en péril la réussite de celui-ci. Pour préparer ces éventualités, nous avons établi un tableau qui récapitule les différents types de problèmes potentiels, leurs niveaux d'importance (sur une échelle de 1 à 5) et des solutions envisageables. Cela nous permettra de réagir rapidement et efficacement en cas de besoin pour minimiser les retards et les coûts supplémentaires.

Risque	Niveau d'importance	Solutions envisageables
Manque de communication sur l'avancement des tâches	4	- Repensé la solution de gestion de tâche - Augmenter la fréquence des réunions
Panne d'équipement du drone ou du Rover	3	- Manipuler le drone avec plus de délicatesse - Utiliser le simulateur Gazebo tant que les robots sont en panne
Mauvais choix d'architecture ou de patrons de conceptions	3	- Choisir des patrons qui ne sont pas des anti-patrons - Faire du code réutilisable et facilement modifiable

Mauvaise gestion du temps	3	- Prendre le temps de bien estimer la durée d'implémentation d'une fonctionnalité - Avertir des avancements de chaque tâche pour qu'un retard puisse être prévisible
Capacité du serveur trop petite pour la quantité de requêtes	2	- Optimiser le nombre de requêtes à ce qui est strictement nécessaire - Instancié un second serveur pour prendre en charge des requêtes en plus
Mauvaise compréhension de la fonctionnalité à implémenter	3	- Parler avec les membres de l'équipe pour s'assurer d'une compréhension commune d'une tâche - Poser des questions de précision au client si une tâche est ambiguë

Tableau 6 : Niveau d'importance des risques envisageables

5.3 Tests (Q4.4)

Afin de garantir la qualité de chaque sous-système, nous effectuerons des tests unitaires et des tests d'intégration pour chacun d'entre eux. Ces tests nous permettront de vérifier que chaque partie du système fonctionne correctement et de détecter les erreurs potentielles avant la mise en production.

Nous utiliserons donc:

- Pour la partie client de l'application web, nous utiliserons les librairies de tests incluses dans Vue.js (Fu, et al., 2023) pour faire tous nos tests unitaires et pour viser une couverture de code et de branches de 100%.
- Pour le serveur et le code embarqué en Python (Python Software Foundation, 2023), nous nous servirons de la librairie de test unittest (Python Software Foundation, 2023) pour rédiger des tests unitaires pour chaque fonction du code.
- Pour les tests matériels tels que les déplacements d'un robot ou les itinéraires qu'il emprunte, nous procéderons d'abord à des tests dans le simulateur Gazebo (Open Source Robotics Foundation, 2014) pour garantir un fonctionnement optimal de la logique. Ensuite, nous validerons ces fonctionnalités sur les robots réels, dans l'espace dédié à cet effet.

5.4 Gestion de configuration (Q4)

En ce qui concerne la gestion de configuration, nous stockerons notre code sur notre dépôt GitLab (GitLab B.V., 2023), sur la branche principale (main). Cette gestion de configuration nous permet de gérer les différentes versions des fichiers et de pouvoir facilement retourner à des versions antérieures ou créer de nouvelles versions, sans impacter le projet en cours.

Notre projet sera séparé en quatre parties, un dossier pour la station au sol composé d'un client et d'un serveur, un dossier pour le code du CogniFly (The CogniFly Project, 2023) un dossier pour le Limo (Agilex Robotics, 2022) et un dernier dossier pour la simulation Gazebo (Agilex Robotics, 2021). Chaque dossier comprendra un Dockerfile (Docker Inc., 2023) pouvant lancer les parties séparément. Toute la documentation sera effectuée en français, mais le code source sera écrit en anglais.

Pour les tests, comme mentionnés dans la partie 5.3, nous effectuons des tests unitaires sur toutes les fonctions possibles, et pour les autres tests, nous les testerons de manière matérielle, dans le simulateur Gazebo (Open Source Robotics Foundation, 2023) puis avec les robots réels.

Toutes les données relatives à une exploration seront stockées sur SQLite (SQLite Consortium, 2023), elles seront disponibles sur le client par le biais du serveur web qui communiquera avec la base de données.

5.5 Déroulement du projet (Q2.5)

Comme décrit dans ce document, nous avons choisi d'utiliser un robot au sol et un drone pour répondre à l'appel d'offres. La simulation du rover (Agilex Robotics, 2022) nous a permis de progresser dans la réalisation des exigences de R.F.4 et R.F.5, c'est-à-dire d'être capable d'explorer ainsi que d'éviter les obstacles. Cependant, nous avions rencontré des ralentissements dans le processus pour faire fonctionner physiquement les robots. Nous avons dû changer de cartographer, vers gmapping (Gerkey, 2020) puisque celui choisi au départ ne nous a pas donné les résultats escomptés.

Concernant le drone, nous n'avions eu accès à la simulation qu'à la fin de la troisième semaine après avoir commencé le CDR. Contrairement au rover, nous n'avions jamais simulé un drone avant le CDR. Il nous a donc fallu un certain temps pour installer les données nécessaires pour faire fonctionner celui-ci. Après plusieurs tentatives, nous n'étions toujours pas capables de faire fonctionner la simulation (Agilex Robotics, 2021). Enfin nous avons changé les paramètres de la simulation (Agilex Robotics, 2021) utilisée pour résoudre ce problème.

Tester physiquement le fonctionnement du drone nous a posé de nombreux problèmes. Ces problèmes étant liés principalement au matériel, nous avions eu à tester plusieurs drones qui n'ont pas pu mener nos tests à un résultat concret. Ayant besoin de prendre nos vidéos pour la remise de CDR.

En ce qui concerne l'équipe, nous avons dû gérer le fait qu'un membre de notre équipe (Anne Jolette) a dû arrêter le projet pour des raisons de santé. Nous avons donc dû nous répartir les tâches dont elle devait s'occuper pour le CDR.

En raison de ces différents problèmes rencontrés, nous n'avons pu respecter le calendrier établi pour la planification des tâches ci-haut. Cependant, l'équipe a été suffisamment agile pour s'adapter ainsi que prendre l'initiative de trouver des solutions rapidement et efficacement.

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

Le processus suivant peut être observé: en premier lieu, les drones physiques répondent à la commande d'identification. Que ce soit les drones physiques ou ceux de la simulation (Agilex Robotics, 2021), ils peuvent lancer et terminer leur mission, qui consiste à explorer leur environnement tout en étant capables d'éviter les obstacles. De plus, l'état des drones peut être affiché sur une interface graphique adaptable à divers appareils tels que des ordinateurs, des tablettes ou des téléphones mobiles, pouvant être utilisée simultanément sur plusieurs appareils. Des logs de débogage sont également affichés dans un onglet distinct. À ce stade, une carte a été générée pour le Limo (Agilex Robotics, 2022), qui ne contient que la position actuelle du Limo (Agilex Robotics, 2022). Les deux

robots peuvent, dans la simulation seulement pour l’instant, faire un retour à la base. De plus, la génération d’un monde avec 3 murs généré aléatoirement est faite et la simulation comme la station au sol peuvent être lancées avec une seule commande. De multiples tests unitaires ont été effectués, tant côté client que serveur ou pour les fichiers python utilisés par les robots. Nous avons aussi fait attention à garder un format de code standardisé.

Par ailleurs, actuellement, certains aspects du projet ne fonctionnent pas encore pleinement. Tout d'abord, la carte du CogniFly (The CogniFly Project, 2023) n'a pas encore été générée ni combinée avec celle du Limo (Agilex Robotics, 2022). Aussi dans la simulation, nous avons dû toucher à des paramètres physiques concernant le drone pour permettre son mouvement avec le rover. Cependant, cela a eu un impact sur le visuel du drone et son accélération que nous avons réduite pour un rendu plus réaliste. Il reste donc à corriger les paramètres physiques du drone pour que son visuel retourne à la normale et qu’il puisse se déplacer avec le rover. De plus, le retour à la base pour le CogniFly (The CogniFly Project, 2023) en physique n'a pas encore été testé. Enfin, plusieurs autres exigences (R.F.7, R.F.13, R.F.14, R.F.17, R.F.18 et R.F.19) doivent être satisfaites au cours de la dernière phase du projet et n’ont pas encore été complétées.

7. Références (Q3.2)

Agilex Robotics. (2022). *UGV_Gazebo_Sim*. Retrieved from GitHub: https://github.com/agilexrobotics/ugv_gazebo_sim/tree/master/limo

Collin, J., Desmarais, M. C., & Gendreau, O. (2023). *INF3995 - Projet de conception d'un syst. informatique.* Retrieved from Moodle: https://moodle.polymtl.ca/course/view.php?id=1703

Docker Inc. (2023). *Home*. Retrieved from Docker Telepresence: https://www.docker.com/

Eclipse Foundation. (2023). *Home*. Retrieved from Zenoh: https://zenoh.io/

Eclipse Foundation. (2023). *Reference Manual - Abstractions*. Retrieved from Zenoh: https://zenoh.io/docs/manual/abstractions/

Eclipse Zenoh. (2023). *Zenoh*. Retrieved from GitHub: https://github.com/eclipse-zenoh

Fu, A., Vladmir, Patak, Bagher, M., Demchuk, I., Sánchez, J., . . . Perkkiö, A. (2023). *Get Started*. Retrieved from Vitest: https://vitest.dev/

Gerkey, B. (2020, Octobre 02). *gmapping*. Retrieved from ROS Wiki: http://wiki.ros.org/gmapping

Hedges, R., Støy, K., Bers, J., Douglas, J. K., Jinsuck, K., Martignoni, A. I., . . . Dandavate, G. (n.d.). *gazebo*. Retrieved from ROS Wiki: http://wiki.ros.org/gazebo

Meeussen, W. (2021, Mars 02). *robot_pose_ekf*. Retrieved from ROS Wiki: http://wiki.ros.org/robot_pose_ekf

Open Source Robotics Foundation. (2014). *Gazebo Tutorials*. Retrieved from Gazebo Sim: https://classic.gazebosim.org/tutorials

Open Source Robotics Foundation. (2014). *ROS overview*. Retrieved from Gazebo Sim: https://classic.gazebosim.org/tutorials?tut=ros_overview

Open Source Robotics Foundation. (2023, Mars 14). *Gazebo Classic*. Retrieved from GitHub: https://github.com/gazebosim/gazebo-classic

Open Source Robotics Foundation. (2023). *Get Started*. Retrieved from Gazebo Sim: https://gazebosim.org/home

Python Software Foundation. (2023). *Python*. Retrieved from GitHub: https://github.com/python

Python Software Foundation. (2023). *Python*. Retrieved from Home: https://www.python.org/

Python Software Foundation. (2023). *Unit testing framework*. Retrieved from Python Documentation: https://docs.python.org/3/library/unittest.html

Ricard, G. (2023). *Simple Quad Gazebo*. Retrieved from GitHub: https://github.com/willRicard/simple_quad_gazebo/tree/noetic-devel/src/simple_quad

ROS - Robot Operating System. (2023). *ROS Core Stacks*. Retrieved from GitHub: https://github.com/ros

ROS. (2023). *Home*. Retrieved from ROS - Robot Operating System: https://www.ros.org/

The CogniFly Project. (2023, Février 21). *Cognifly Python*. Retrieved from GitHub: https://github.com/thecognifly/cognifly-python

VueJs. (2023). *Vue*. Retrieved from GitHub: https://github.com/vuejs/vue

You, E. (2023). *Get Started*. Retrieved from Vue.js - The Progressive Javascript Framework: https://vuejs.org/

You, E. (2023). *Introduction*. Retrieved from Vue.js - The Progressive Javascript Framework: https://vuejs.org/guide/introduction.html