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
Jolette Anne
Louesdon Théo
Ngay Munga Exaucé-Constantin

10 février 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 à 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 et un robot mobile AgileX Limo, 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

- Les robots ne seront employés qu'à des fins académiques.
- On risque d'avoir des difficultés à bien utiliser la simulation Gazebo à cause du faible nombre de fps.
- Les robots doivent être dans une pièce fermée (au moins 4 murs)

● 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 gérer les bugs)
- Tous les composants logiciels doivent être conteneurisés avec Docker
- Le logiciel complet de la station doit être lancé avec une seule commande impliquant Docker dans un terminal Linux roulant Ubuntu 20.04.
- Au moins deux appareils doivent pouvoir se connecter sur l'interface utilisateur lors d'une mission
- 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

- Les robots ne doivent jamais quitter la pièce M-7703
- Les robots doivent être en mesure d'éviter des obstacles
- Le retour à la base doit rapprocher les robots à leur position de départ à moins de 0,5 m de cet emplacement.
- Le retour à la base et l'atterrissage doivent se faire automatiquement lorsque la batterie est en dessous de 30% et ils ne doivent plus être en mesure de décoller de leur point de départ.
- La station au sol doit collecter les données et générer une carte de l'environnement où la position des robots doit être affichée en continu

- L'environnement simulé doit avoir au minimum 3 murs
- L'interface utilisateur doit suivre les 10 heuristiques d'interfaces usager
- Chaque composant logiciel doit avoir ses propres tests unitaires ou une procédure de tests détaillés pour chaque fonctionnalité
- Des logs de débogage doivent être disponibles en continu pour s'assurer que le système fonctionne correctement
- L'interface utilisateur doit montrer le type et l'état de chaque robot

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 suite à des incompréhensions vis-à-vis d'une tâche.
- Fatigue accumulée selon les différentes périodes de la session.

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)	17 mars 2023 à 23H55	- Prototype avancé
Readiness Review (RR)	21 avril 2023 à 23H55	- Système complet et opérationnel

Tableau 1 : 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. 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 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. 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(euse). 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(euses) 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 ce que toute 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. Par contre, 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. 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. Par contre, 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)

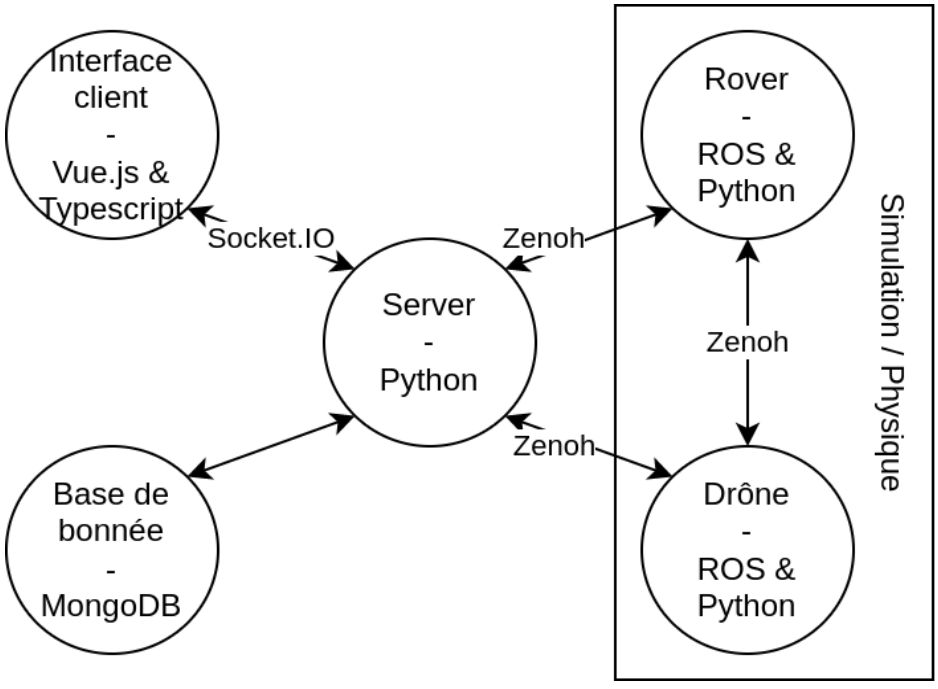

Figure 1 : Diagramme de l'architecture logicielle générale

L'utilisation de Zenoh 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 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 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.

La décision de prendre Python et Vue.js comme langages de programmation est réellement basée sur la préférence des membres de l'équipe. De plus, Vue.js ayant un cadriciel léger et Python 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 plutôt que JavaScript est encore une fois, pour faciliter la compréhension du code par les membres de l'équipe qui ne l'auront pas écrit. Cela obligera également les membres de l'équipe à écrire explicitement les types de variables et les types de retours de fonction.

MongoDB est encore une fois une préférence de l'équipe, car nous avons tous travaillé avec ce type de base de données par le passé, et il sera assez performant pour ce qui est demandé dans le cadre de ce projet.

Quant à ROS, nous avons pris la décision de le favoriser à ROS2 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.

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.

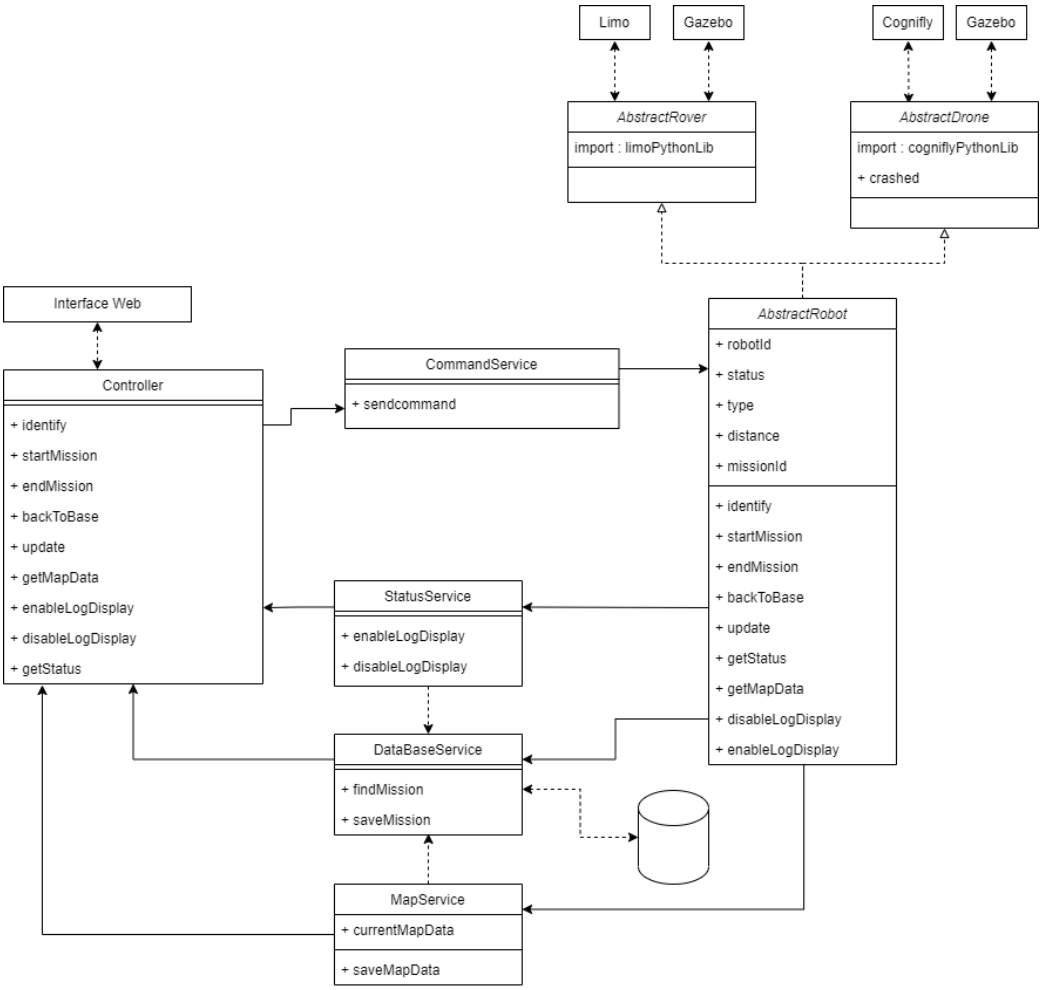

Figure 2 : Diagramme de classe de la station au sol

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 généré 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.

3.3 Logiciel embarqué (Q4.5)

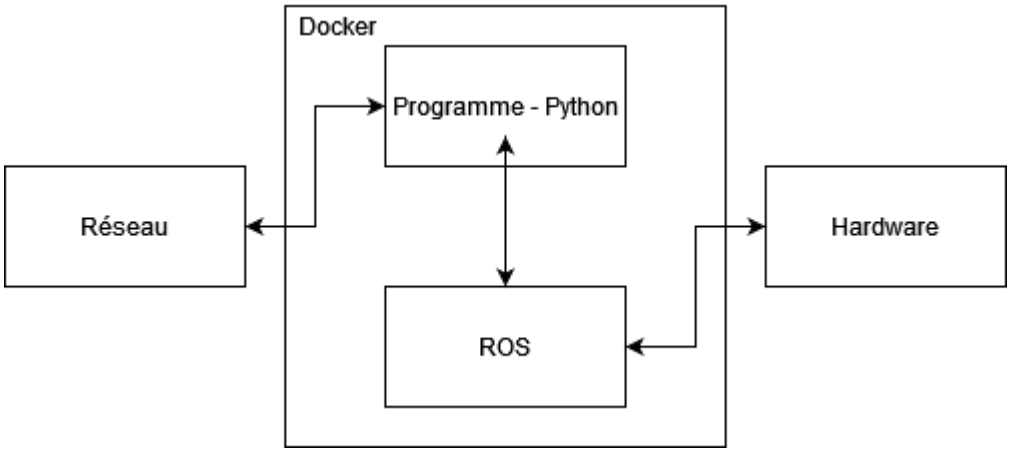

Figure 3 : Diagramme du logiciel embarqué

Le logiciel embarqué de notre système sera basé sur ROS (Robot Operating System) pour faciliter l'interaction avec le robot et simplifier les processus de développement. Nous avons choisi d'utiliser Python 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 permet également un développement plus rapide et facilite la maintenance et l'évolution du système. L'utilisation de ROS nous permettra de compléter les requis R.L.3, R.F.4, R.F.5, 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 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)

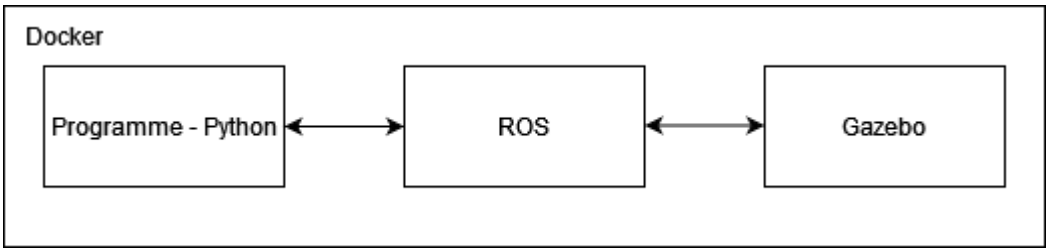

Figure 4 : Diagramme de la simulation

Pour améliorer la portabilité de notre programme, nous utiliserons un environnement Docker. 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 pour les simulations, nous pourrons également garantir un environnement physique réaliste, permettant ainsi au programme Python 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. L'utilisation du simulateur Gazebo permettra de compléter le requis R.C.3 et de facilement tester les requis R.F.2, R.F.4, R.F.5 et R.F.8.

3.5 Interface utilisateur (Q4.6)

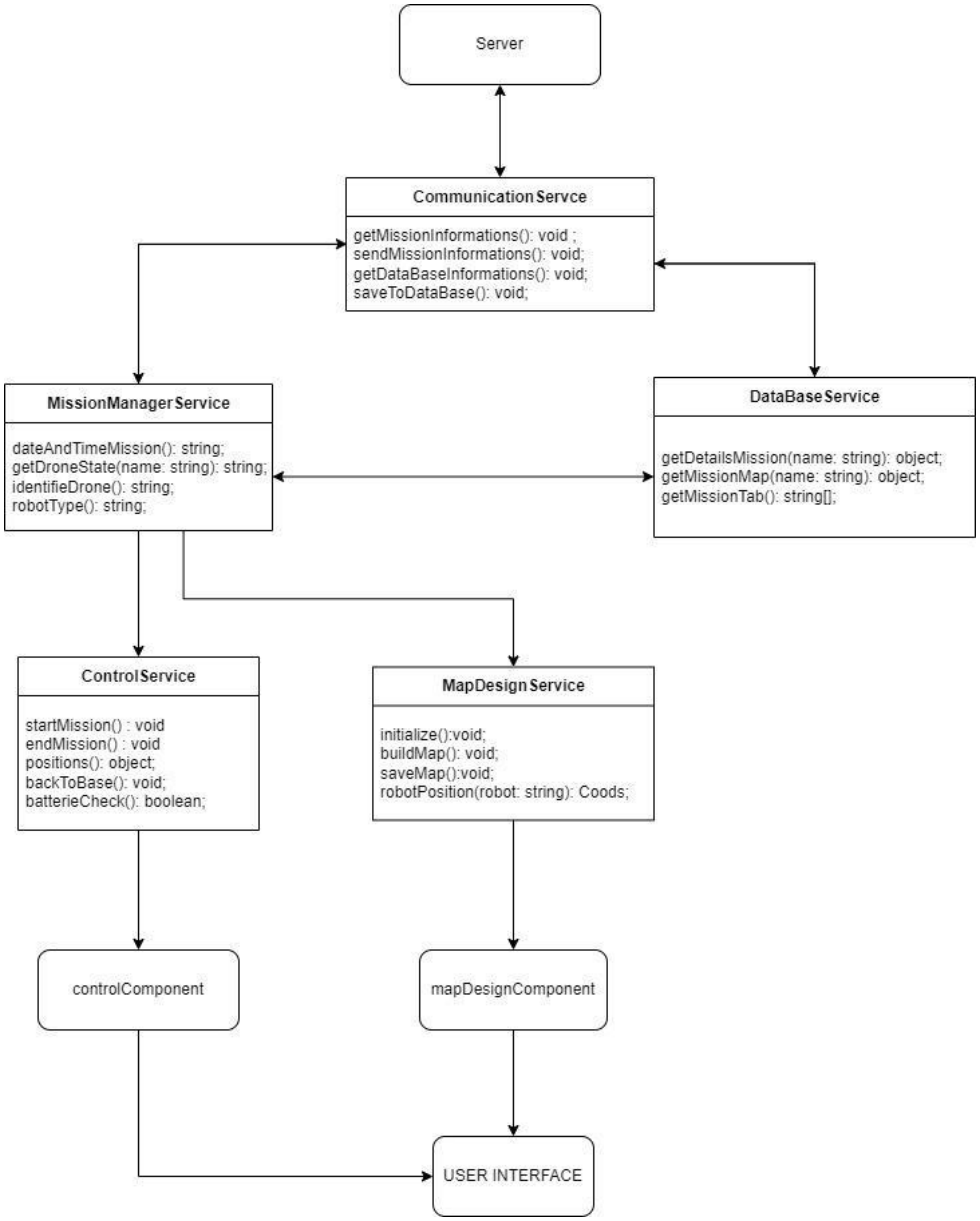

Figure 5 : Diagramme de l'interface utilisateur

L'interface utilisateur est un environnement web développé à l'aide du framework Vue.js . La construction de cette structure est inspirée de celle que nous avions pour utiliser le framework Angular lors du deuxième projet intégrateur en sigle LOG 2990. Nous utiliserons des services qui exécutent une logique et des components pour la gestion de l'affichage qui permet l'interaction avec l'utilisateur. HTML, CSS, fichiers de test et fichiers TypeScript tels sont les fichiers de codage 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, 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 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.

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 2 : Prix total du coût de la main-d'œuvre

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 et 2900 dollars pour le LIMO (en sachant que la société nous fournit les 6 Laptop et le local), et nous obtenons un total de 87 377.5$.

4.2 Planification des tâches (Q11.2)

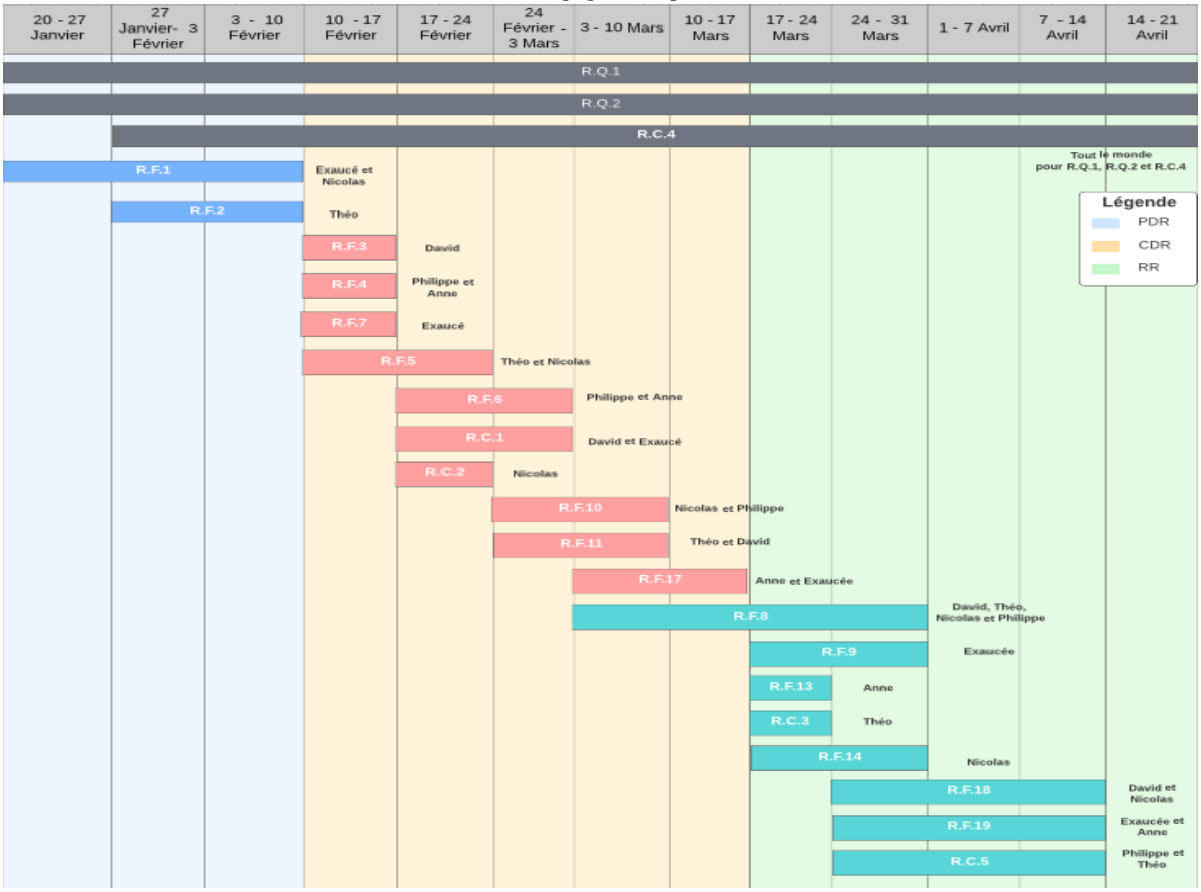

Figure 6 : Diagramme de Gantt de la planification des tâches pour le projet

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 3 : 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 Giovanni Beltrame et Antoine Robillard.

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

- Maitriser des outils de gestion de projet comme GitLab
- 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, Linux)
- 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	Anne	Exaucé	Nicolas	Théo
Maîtrise du langage Python	X			X	X	X
Maîtrise de Vue.js et Typescript		X	X	X	X	
Maîtrise de programmation de système embarqué		X		X		
Écriture de bons tests	X	X	X		X	X
Maîtrise de Docker					X	
Utilisation d'une base de données MongoDB	X	X		X	X	X
Connaissance du système d'exploitation Linux	X		X	X	X	X
Maîtrise des fonctionnalités de Gitlab	X	X	X	X		X
Développement d'une application Web	X	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	X

Tableau 4 : 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 et Prettier ainsi que ESLint pour le code TypeScript et Vue.js. 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	2	- 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 5 : 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 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, nous nous servirons de la librairie de test unittest 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 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, 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 (drone volant), un dossier pour le limo (rover) et un dernier dossier pour la simulation Gazebo (sim). Chaque dossier comprendra un Dockerfile pouvant lancer les parties séparément. Toute la documentation sera effectuée en français, mais le code source sera écrit en anglais.

Pour les tests, comme mentionné 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 puis avec les robots réels.

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

6. Références (Q3.2)

Collin J., Desmarais M.C., Gendreau O. (Janvier 2023). *Notes de cours INF3995, Cycles d'acquisition et ententes contractuelles.*

Vue.js. (s.d.). *Introduction*. Vue.js. https://vuejs.org/guide/introduction.html

Zenoh. (s.d.). *Abstraction.* Zenoh. https://zenoh.io/docs/manual/abstractions/