POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

Proposition répondant à l'appel d'offres no. H2021-INF3995 du département GIGL.

Conception d'un système aérien minimal pour exploration

Équipe No 204

Émilie Vaudrin
Alexandre Talens
Julio Dandjinou
Farid Elfakhry
Alexandre Morinvil

Janvier 2021

Table des matières

1 Vue d'ensemble du projet

1.1 But du projet, porté et objectifs

Par la présente soumission, l'équipe d'ingénieurs 204 offre un suivit de la progression de son avancement dans ses services afin de répondre à aux exigences pour la conception d'un système aérien minimal d'exploration.
La solution technique consiste notamment en un développement d'un programme informatique s'exécutant sur les robots fournis par l'Agence spatiale de Polytechnique. Comme exigé, le système sera composé d'un nombre arbitraire de drones qui pourront explorer de manière autonome leur environnement. Ceux-ci seront contrôlables à l'aide d'une interface utilisateur Web hébergée depuis une station au sol. Cette interface permettra alors d'afficher les informations envoyées par les drones et ainsi reconstruire une carte de l'environnement exploré. Le projet présenté a ainsi pour objectif de démontrer une preuve de concept pour accompagner l'agence dans ses futurs projets.
Notre réponse à l'appel d'offre est accompagnée d'une démonstration d'un prototype minimal fonctionnel. Les détails techniques de la solution proposée seront détaillés ci-après.

1.2 Hypothèse et contraintes

L'élaboration de ce projet comporte de nombreux défis et contraintes techniques. Certains de ces aspects sont soumis à des hypothèses que nous avons posées.
Ainsi, il n'est pas possible pour nous de tester et développer notre projet dans un environnement similaire aux conditions physiques rencontrées sur Mars. Cette contrainte apporte alors les hypothèses suivantes. Nous supposons que le fonctionnement du drone sur Mars sera similaire au fonctionnement du drone sur Terre. De plus, nous supposons que l'infrastructure de télécommunication sera fiable et donc que les robots seront toujours en communication avec la station au sol. Nous faisons abstraction des rayonnements cosmiques qui pourraient perturber le fonctionnement du robot. D'autres contraintes nous sont aussi imposées comme les règles sanitaires. Il devient donc impossible pour tous les membres de pouvoir développer et tester sur les robots et nous assumons que les environnements de tests que nous utilisons (Argos) sont fidèles à la réalité. Cependant, comme détaillé dans les sections suivantes, le code du robot sera toujours testé dans la réalité pour sa validation avant remise du projet. De plus, seuls 2 robots matériels nous sont fournis et on assume alors que le fonctionnement dans la réalité sera identique avec un nombre plus important de robots.

1.3 Bien livrables du projet

Il y a trois livrables à remettre. Le premier livrable est appelé Preliminary Design Review. Il nous permet de montrer un prototype minimal ainsi que la documentation de gestion de projet. Ce livrable est à remettre le 15 février. Les tâches à accomplir sont une simulation ARGoS avec deux drones, un serveur web sur la station au sol avec une page qui montre quelques fonctionnalités comme le niveau de la batterie du drone et un bouton pour allumer une de leurs DEL.

Le deuxième livrable est appelé Critical Design Review. Dans ce livrable, la conception du système doit être complétée avec la majorité des fonctionnalités. Les fonctionnalités qui y seront implémentée sont :

- Une simulation ARGoS avec 4 drones qui volent dans un environnement avec murs
- Les drones pouvant utiliser des capteurs de distance
- Un serveur web interfacé avec la simulation ARGoS
- Un code embarqué sur les drones qui envoie les mesures du ranging deck
- Un prototype visuel de la carte générée par les robots

Ce deuxième livrable aura pour échéance le 8 mars.

Le troisième livrable est appelé Readiness Review. Ce livrable est la présentation finale du projet. Ce livrable doit contenir un vidéo montrant le fonctionnement complet du système en simulation et tout les scripts et éléments

logiciels nécessaires pour lancer une simulation avec une seul commande Linux. L'échéance de ce livrable sera le 12 avril.

2 Organisation du projet

2.1 Structure d'organisation

Nous avons subdivisé le projet en deux : la partie logicielle embarquée et la partie logicielle de la station au sol. Trois personnes de l'équipe travailleront ensemble pour effectuer la partie logicielle embraquée et deux autres personnes pour la partie logicielle de la station au sol. Pour la partie embarquée, les personnes de l'équipe sont Farid Elfakhry, Alexandre Talens et Julio Dandjinou. Cette sous-équipe va se concentrera sur les tâches de la partie embarquée tout au long de l'avancement du projet. Ils devront se rencontrer pour parler de leur avancement et de la distribution du matériel. Du côté de la station au sol, l'équipe est constituée d'Alexandre Morinvil et d'Émilie Vaudrin. Ils devront eux aussi se rencontrer pour coordonner leurs tâches, parler de leur avancement et mettre leurs idées ensemble.

De plus, les cinq membres de l'équipe vont se rencontrer chaque dimanche lors d'une réunion sur Microsoft Teams pour mettre leurs idées ensemble avant de commencer leur travail. Ceci permettra à la totalité de l'équipe de savoir sur quoi travaillent les autres sous équipes et l'avancement de chacun. Nous commençons la semaine avec une rencontre d'une heure par équipe travaillant sur les différentes parties du projet. Cette rencontre permet de parler des objectifs de la semaine et de s'aider pour le commencement des tâches de chacun. Nous aurons aussi une réunion de type SCRUM pour discuter de l'avancement de chacun et des difficultés rencontrées. Pour finir la semaine, nous nous rencontrons le vendredi pendant 3 heures pour travailler tous ensemble.

Ensuite, nous nous sommes attribué des rôles pour un meilleur fonctionnement du projet. Premièrement, nous avons le rôle d'organisateur des rencontres qui est attribué à Alexandre Morinvil. Ce rôle constitue de faire un plan du déroulement de la rencontre avant chaque rencontre et d'animer celle-ci. Ce rôle permet d'avoir des rencontres organisées et donc plus courtes avec un but précis. Avant chaque rencontre, cette personne crée les documents de la semaine et prend en note ce qui est discuté dans les rencontres. Dans ce même rôle, nous avons le sous-rôle de secrétaire. Il s'assure que tout ce qui est dit de pertinent lors d'une rencontre est pris en note. Cette responsabilité a pour but de garder nos idées à l'écrit pour pouvoir revenir facilement si nous avons un doute ou si nous oublions ce que nous avions dit.
Deuxièmement, nous avons le rôle du Gitlab master attribué à Farid ElFakhry. Cette personne s'assure que le GitLab est utilisé de façon efficace et propre. Il s'assure que toute l'équipe utilise bien les précédés de gestions de versions (par exemple, les merge request) et qu'avant d'intégrer quelque chose au projet, le code soit bien structuré et fonctionnel.
Troisième, nous avons le rôle de responsable du rapport hebdomadaire attribué à Julio Dandjinou. Ce rôle crée le document pour le rapport hebdomadaire et s'assure que tout le monde rempli celui-ci avant la rencontre de Lundi.
Finalement, nous avons le rôle du coordonnateur de projet attribué à Alexandre Talens. Ce rôle a pour responsabilité de vérifier l'avancement des autres coéquipiers et de rappeler à tous les membres de l'équipe les dates d'échéances. Ce rôle s'assure que tout est fait dans les temps et que nous ne sommes pas en retard sur notre échéance. Enfin, tous les membres de l'équipe seront des développeurs logiciel.

2.1.1 Modification durant la CDR :

Organisation quinzomadaire (a chaque deux semaines) pour faire une retrospective sur notre avancement, ameliorees nos procedure d'operation et evaluer notre rythm.

2.2 Entente contractuelle

Nous proposons une entente contractuelle à terme avec un minimum de 573h travaillé et 57h tampons, donc un coût plafond de 630h et des économies pouvant aller jusqu'à 57h de travail.

Les motivations de ce choix d'entente contractuelle se base sur les critères suivants :

— Le degré de risque pour les coûts et les échéanciers : les clauses de ce contrat garanti un produit final livré à temps.

— La complexité des spécifications du système : Nous avons alloué un temps de plus pour régler les imprévues.

— Le niveau de compétition : Nous garantissons un service professionnel avec des qualités et un prix compétitif sur le marché.

Clause 1 : Un système d'exploration spatiale requiert une planification scrupuleuse, pour cela notre équipe s'engage à délivrer un produit final et fonctionnel en respect des échéances indiquées dans le document de demande d'offre rédigé par l'agence.

Clause 2 : Le projet est d'une grande complexité. Nous prévoyons donc ajouter 10% du temps estimé pour régler les problèmes imprévus et la correction des requis si l'agence le juge nécessaire.

Clause 3 : Si l'agence voulait continuer le travail avec notre équipe après la remise finale, une nouvelle négociation aurait lieu.

Clause 4 : Pour assurer un prix compétitif, nous demandons un paiement proportionnel au nombre d'heures et ne dépassant pas 630h.

3 Solution proposée

3.1 Architecture logicielle générale

Les exigences techniques du produit demandé requièrent le développement de deux infrastructures logicielles principales :

— Une architecture logicielle embarquée

— Une architecture logicielle de serveur de contrôle avec un avec un interface Web

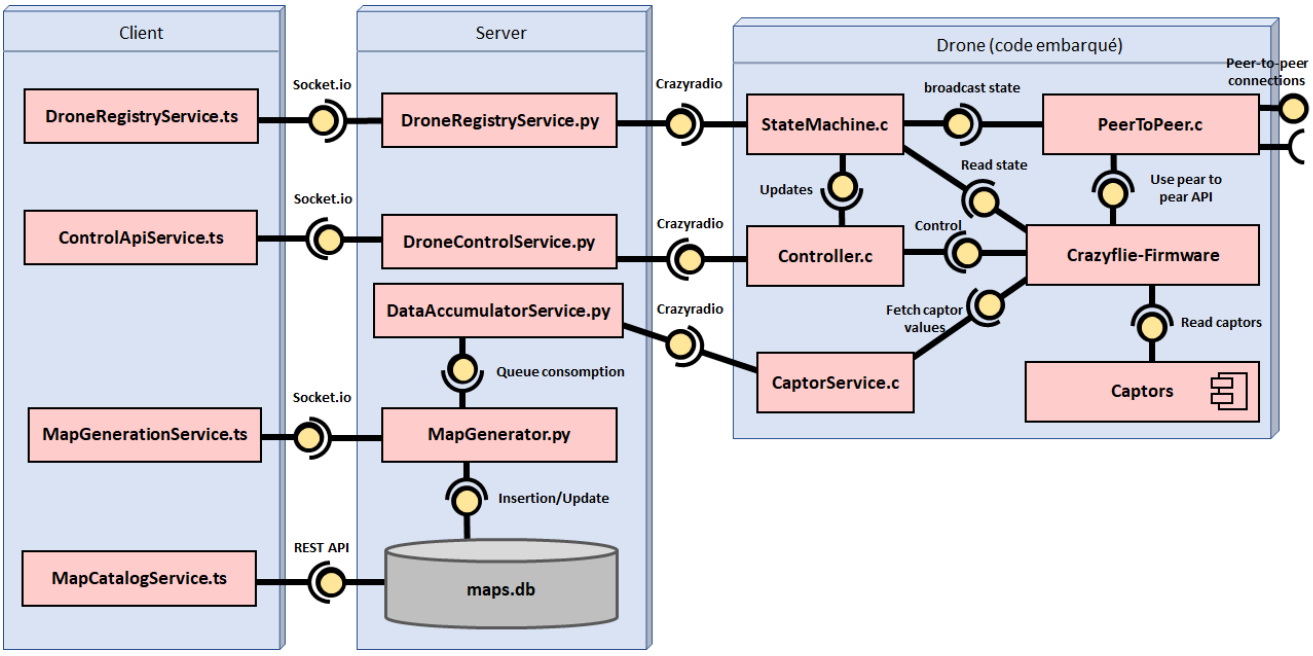
L'architecture logicielle embarquée se constitue de l'ensemble des éléments logiciels et matériels qui seront intégrés à l'essaim de drones. Ceci inclut les algorithmes de haut niveau permettant le contrôle décentralisé de l'essaim, le code embarqué, le microcode (firmware), ainsi que les composantes matérielles interfacées.

L'architecture logicielle du serveur de contrôle se constitue de l'ensemble des éléments permettant l'interfaçage entre l'utilisateur ainsi que l'essaim de drones à gérer. Ceci inclut l'interface graphique Web (ou le développement Web front end), l'infrastructure de l'ordinateur central de contrôle (ou le développement back end), la gestion de la permanence des données, ainsi que les protocoles de communication entre les différentes interfaces du serveur central.

La solution présentée propose une décomposition de l'architecture globale du système en trois composantes logicielles principales : une structure logicielle pour le code client, une structure logicielle pour le serveur central ainsi qu'une structure logicielle pour le code embarqué.

La figure suivante résume l'architecture globale du système.

FIGURE 1 – Diagramme de déploiement



L'aspect client sera implémentera une interface Web développée en avec le cadriciel Angular (et donc, en typescript). Le choix d'Angular permettra de faciliter le développement d'une infrastructure web par service, ce qui permettra d'efficacement découpler les trois principales fonctions du système : le monitorage de l'état de la flotte, l'envoi de commandes à la flotte, la cartographie en direct ainsi que la consultation de cartes précédemment générées. Ces quatre fonctions seront respectivement prises en charge par les services Drone Registry, Control API, Map Generation et Map Catalog. Une description plus détaillée de ces services sera abordée dans la section de l'architecture de la station au sol.

L'aspect serveur sera implémenté avec une approche par service en Python. Le choix de ce langage de développement est motivé par la simplification de l'interface de communication entre le serveur ainsi que la flotte de drones, qui se fera un utilisant un le module python de communication radio par le moyen de la « Crazyradio » communicant sur un canal de télécommunication à 2,4GHz. L'approche par service permettra également de réduire le couplage entre les différentes fonctionnalités du serveur, soit : diffuser l'état de la flotte à tous les clients connectés, relayer les commandes de contrôle du client vers la flotte de drone, recevoir en continu et de manière asynchrone les observations de la flotte de drones ainsi que de gérer la génération et l'entreposage des cartes. Ces fonctionnalités seront prisées en charge respectivement par les services : Drone Registry, Drone Control, Data Accumulator, Map Generator.

L'interfaçage entre le client et le serveur se fera en utilisant la librairie socket.io afin de permettre une connexion bidirectionnelle évènementielle entre le serveur et le client de manière à ce qu'il soit possible pour le client d'envoyer des requêtes vers le serveur et qu'il soit également possible pour le serveur de publier des informations sur l'état de la mission vers le client.

Enfin, l'aspect code embarqué sera implémenté avec une approche de machine à état implémentée en C et interfaçant le microcode (firmware) de BitCraze. L'approche par machine à état a été sélectionnée afin de pouvoir efficacement définir des comportements indépendants pour chaque drone en fonction des différents contextes dans lesquels ils seront appelés à naviguer. L'interfaçage avec le microcode Bitcraze permettra un contrôle de haut niveau

du comportement des drones, ce qui permettra d'implémenter des algorithmes de navigation décentralisés sur les drones sans préoccupation pour le fonctionnement matériel de chaque membre de l'essaim.

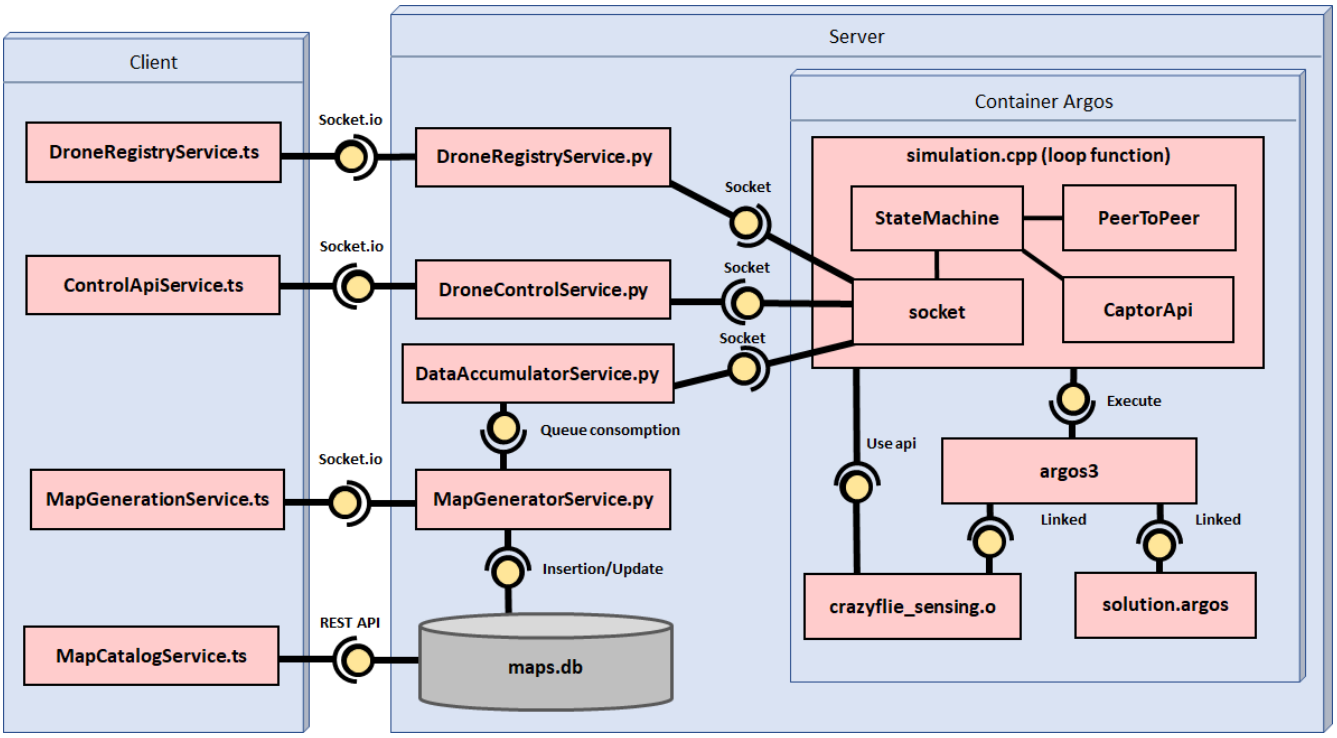
La communication entre les drones se fera en utilisant les fonctionalités intégrée du microcode de Bitcraze afin de de permettre une communication pair-à-pair sur un canal de 2,4GHz.

3.1.1 Architecture logicielle Intégration de l'environnement ARGOS

Additionnellement, le système développé devra pouvoir opérer dans un environnement de simulation. De manière à ce que le client utilisant l'application puisse fonctionner de manière analogue à un environnement réel. Le simulateur de physique d'essaim de drone utilisé sera le simulateur ARGoS. Ainsi, le simulateur

L'intégration du simulateur ARGoS à l'environnement de deploiement de notre application aura la responsabilité de pour pouvoir substituer tous les sous-systèmes du code embarqué de manière à permettre le prototypage efficace d'algorithmes de controle décentralisé d'essaim. La figure suivante affiche l'intégration du simulateur ARGoS de manière analogue à au code embarqué.

FIGURE 2 – Ajout pour la CDR : Diagramme de déploiement de l'application dans un environnement ARGoS



Ainsi, le code de simulation sera implémenté en C++ et interfacera un controleur de drone crazyfly (ou spiri) fournis par les libraries d'ARGoS. Ainsi, la même machine à état ainsi que la même logique des algorithmes des comportements associés à chaque état seront implémentés et testés dans cet environnement avant d'être traduits en C et en appel de fonction analogue de l'API bitcraze pour le code embarqué.

La communication entre les drones pair à pair, la lecture des observations des capteurs associés à chaque drone se fera en utilisant les fonctions de l'API d'ARGoS simulant ces fonctionalités.

La communication entre l'environnement de simulation et le serveur de la station au sol se fera par le moyen

N sockets TCP associés au N drones. Cette stratégie (par opposision à la stratégie de centraliser la communication par le moyen d'un seul socket pour l'ensemble de la flotte) permet de simuler plus fidèlement la communication de la flotte de drones matériels qui elle également se fera par le moyen de N différentes connections à la station au sol.

L'espace à explorer lors des simulations argos sera généré aléatoirement pour chaque mission d'exploration et les informations de l'environnement seront contenues dnas un fichier ".argos".

De plus, un mécanime permettant de passer d'un environnement à un autre sera implémenté sur le serveur afin de réinitialiser les connections entre les drones d'un environnement et initialiser le pairage avec les drones d'un autre environnement. Cette fonctionalité sera active sur le client seulement lorsque l'ensemble de la flotte sera dans un état sécuritaire (État "Ready") afin d'éviter de déconnecter les drones de la flotte dans un contexte potentiellement dangereux.

3.2 Architecture logicielle embarqué

Le programme du système embarqué sera écrit en C à l'aide de l'API fourni par Bitcraze. Afin de répondre aux exigences des requis, la structure du logiciel sera implémentée comme suit :
Tout d'abord, une grande machine à état avec les états suivants sera implémentée :

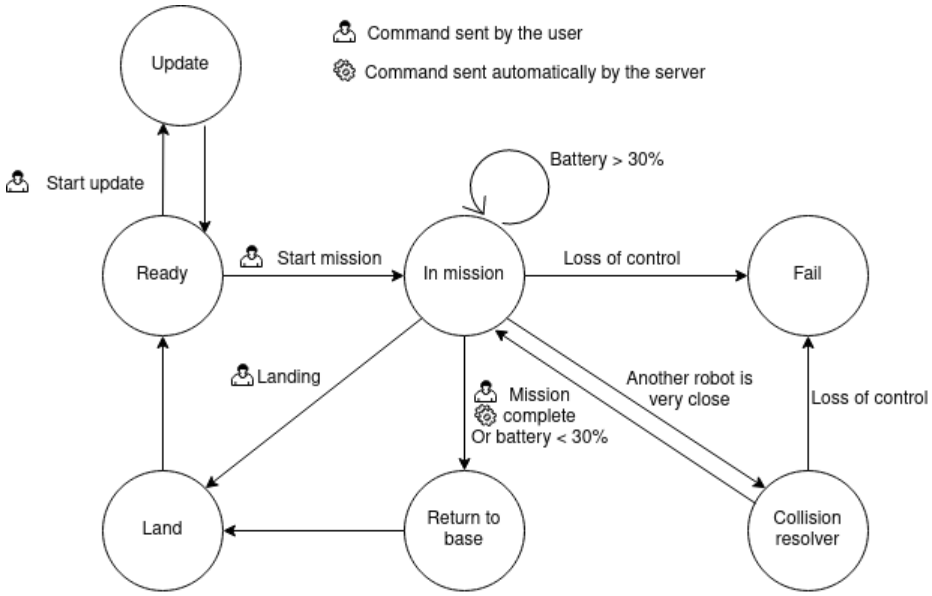
Table 1 – Définition des états

Etat	Description
Ready	Le robot est allumé et en attente d'inscruction.
In Mission	Le robot passe dans cet état lorsqu'un utilisateur en fait la requête à l'aide du client web. Le robot décolle alors et explore la zone.
Fail	Le robot arrive dans cet état lorsqu'un problème majeur est survenu (comme une collision non évitée). Il reste bloqué dans cet état jusqu'à son redémarrage manuel.
Update	Le robot passe dans cet état à l'aide d'une requête de l'utilisateur afin de même à jour son code source.
Return to base	Lorsque le robot n'a plus de batterie ou que l'utilisateur le demande, le robot passe dans cet état afin de retourner à la base.
Land	Le robot atterit de manière sécuritaire là où il est présentement positionné.
Collision resolver	Le robot se place dans cet état lorsqu'un autre robot est très proche. L'agorithme d'évitement de collision est alors activé.

Ensuite, un timer sera initialisé avec une période de 500ms. Il sera actif durant tous les modes de fonctionnement du robot. Ce timer appelera une méthode qui effectuera les actions suivantes :

- Vérification du niveau des batteries (qui peut impliquer un changement d'état du drone si il est dans l'état "In mission")
- Transmission en P2P (à tous les drones et en communication directe avec la station) des informations suivante :
  - Tension batterie
  - Position (x,y,z)
  - Vitesse au sol
  - Vitesse des moteurs
  - Distance par rapport aux murs (laser)

FIGURE 3 – Diagramme d'état



Les données de position et de distance par rapport aux murs envoyés contiendront un ensemble de données. L'échantillonnage exact reste à définir, mais si les mesures sont effectuées aux 50ms, alors le paquet envoyé contiendra un ensemble de 10 positions et distance. Les drones pourront recevoir en tout temps des paquets provenant des autres drones comme de la station au sol. La station au sol pourra alors demander aux robots de changer d'état en tout temps. Chaque robot recevra alors minimalement 2 paquets par seconde (provenant de l'autre robot en marche). À la réception du paquet (initialement envoyé par l'autre robot), le robot pourra déterminer sa distance avec l'autre robot et ainsi éviter une collision.

Afin d'éviter la collision, les deux robots vont se mettre d'accord sur une nouvelle altitude à acquérir. Par exemple, un des robots va diminuer son altitude tandis que l'autre l'augmentera. De cette manière l'impact sera évité.

3.2.1 Detail ajuter pour la CDR

Lors du demarage de la mission, les drone suive un chemin aleatoire. Dans le cas d'une collision, le drone pivote dans la direction oppose et ce deplace lateralement pour un bref moment. Cette approche nous permet d'eviter les obstacles tout en gardant le drone en mouvement.

Pareillement, l'evitement d'obstacle est le meme pour le retour a la base. La majeur difference est que le robot essayera toujour de s'orienter vers la base.

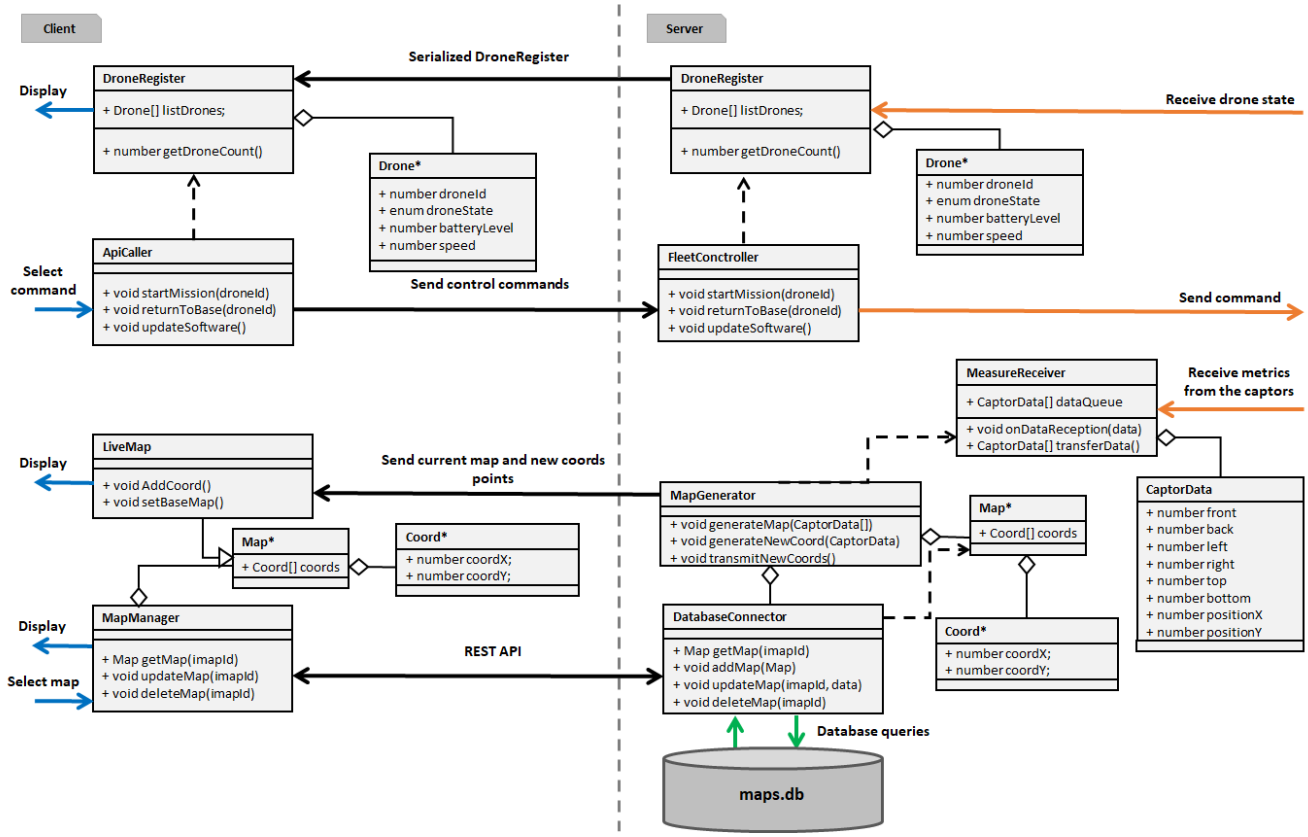
L'ensemble des comportements décrits dans cette section seront simulés en environnement de simulation ARGoS.

3.3 Architecture logicielle station au sol

L'architecture logicielle de la station au sol implique le développement front end (le client) via le cadriciel Angular ainsi que le développement back end (le serveur) en python. Le front end et le back end seront implémentés en utilisant une architecture par service. La communication entre les le front end et le back end se via la librairie socket.io, afin de permettre une communication duplex.

La figure suivante montre un résumé des principales classes qui seront nécessaires à l'implémentation des services composant la station au sol.

FIGURE 4 – Diagramme de classes de la station au sol



Le client implémentera quatre services de base :

— Registre de drones (Drone Registry)
— Contrôle de drone (Control API)
— Génération de cartes (Map Generation)
— Consultation de cartes (Map Catalog)

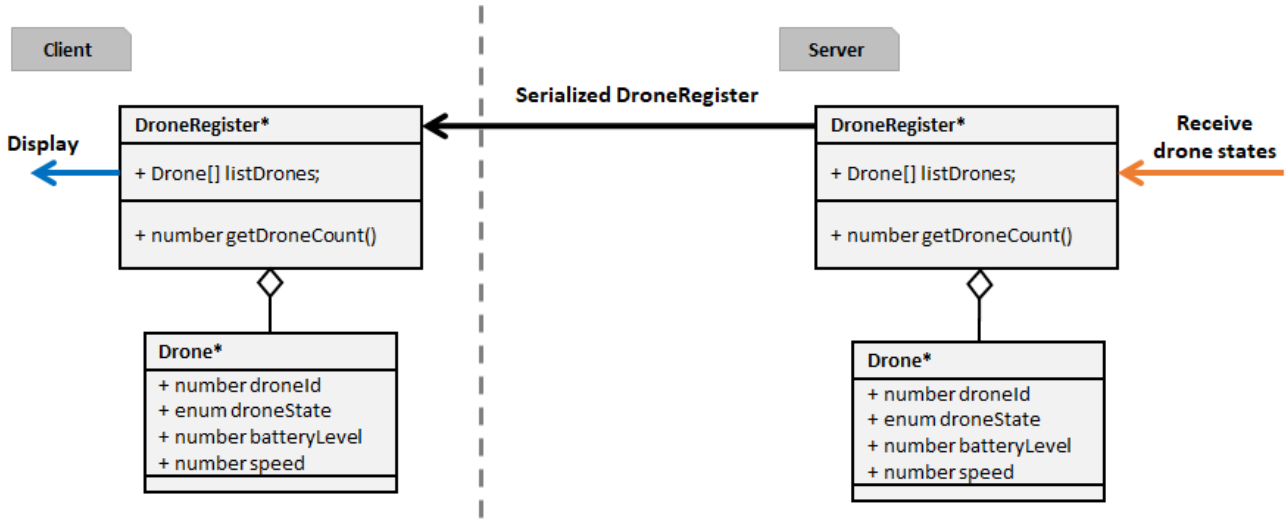
Chacun des services du client communiquera avec un service analogue du côté du serveur. Les responsabilités de chacun de ces services ainsi qu'une vue générale de l'implémentation proposée sera abordée.

3.3.1 Registre de drones (Drone Registry)

La responsabilité du service de registre de drones sera de connaître l'état de la flotte de drone et de le présenter à l'utilisateur. Ce service implémenté sur le client sera connecté avec le service homologue sur le serveur qui sera lui-même connecté à l'ensemble des drones de la flotte.

La figure suivante affiche l'implémentation proposée pour l'implémentation du service de registre de drones.

FIGURE 5 – Diagramme de classe résumant l'implémentation du service de registre de drones (Drone Registry)



L'implémentation de ce service se fera par le moyen d'une communication unidirectionnelle périodique partant de chaque drone de la flotte pour se rendre jusqu'à l'affichage à l'écran de chaque client connecté.

Chaque drone communiquera périodiquement et de manière asynchrone son état à la station au sol. Ainsi, le service de registre de drone implémenté sur le serveur aura la responsabilité de recevoir l'état de chacun des drones de la flotte et de le mettre à jour sur la mémoire le serveur. Ainsi, le serveur possédera en tout temps l'information la plus à jour possible quant à l'état des drones de la flotte.

Ainsi, à un intervalle de 500ms (2Hz) le serveur sérialisera et diffusera l'entièreté du registre des drones à l'ensemble des clients. Enfin, ceci permettra à l'ensemble des clients connectés d'afficher l'état de tous les drones, mis à jour à une fréquence supérieure à 1Hz.

Il sera nécessaire de définir une structure de donnée équivalente pour le registre des drones sur le client et sur le serveur.

3.3.2 Contrôle de drone (Drone Control)

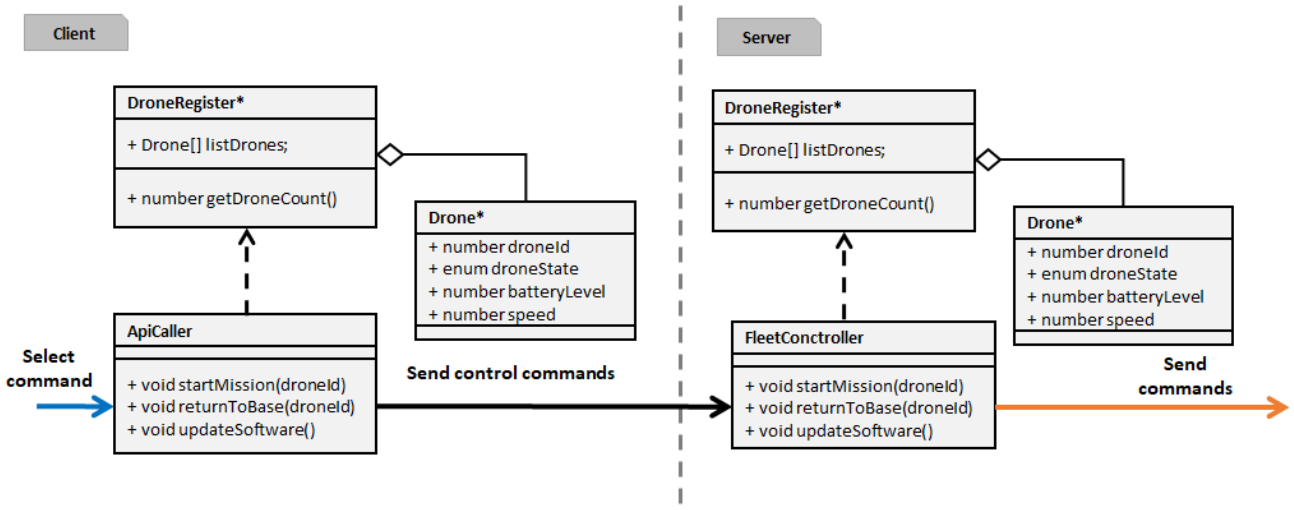
La responsabilité du service de contrôle de drones sera de valider et relayer les commandes envoyées par chacun des clients jusqu'aux drones concernés. Ce service implémenté sur le client sera connecté en communication unidirectionnelle avec le service homologue sur le serveur qui sera lui-même connecté à l'ensemble des drones de la flotte.

La figure suivante affiche l'implémentation proposée pour l'implémentation du service de registre de drones.

L'implémentation de ce service se fera par le moyen d'une communication unidirectionnelle périodique partant de l'interface utilisateur d'un client connecté jusqu'aux drones concernés dans la flotte.

Ce service aura accès au registre de drones afin de pouvoir valider l'accessibilité des drones auxquels une commande devrait être envoyée. Pour les drones accessibles, ce service implémentera les fonctions permettant de déclencher l'état de début de mission, de retour à la base et mise à jour logiciel sur les drones sélectionnés par une commande. Les commandes pourront être envoyées à un drone par le moyen de son identifiant unique et il

FIGURE 6 – Diagramme de classe résumant l'implémentation du service de contrôle de drones (Drone Control)



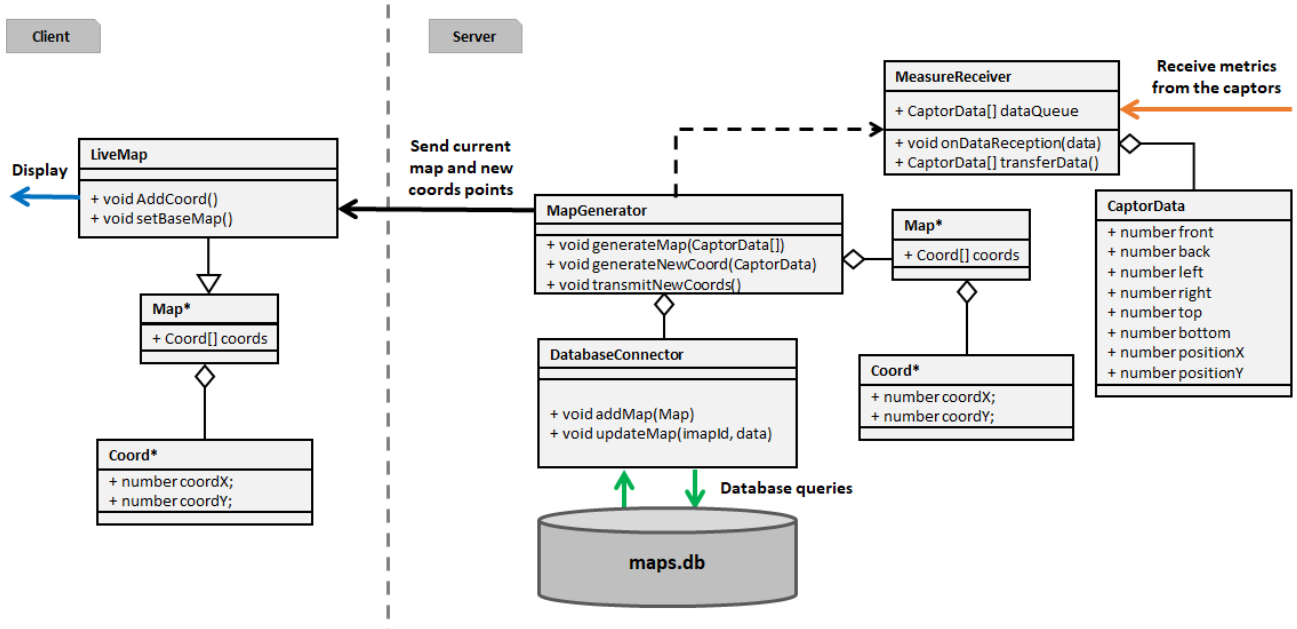
sera également possible de diffuser une commande à l'ensemble des drones lorsqu'une commande est envoyée sans identifiant spécifique.

3.3.3 Génération de cartes (Map Generation)

La responsabilité du service de génération de cartes sur le client sera de gérer la génération en direct de la carte résultant d'une mission d'exploration en cours. Ce service implémenté sur le client sera connecté en communication unidirectionnelle de manière à recevoir en continu des données provenant d'un service homologue sur le serveur.

La figure suivante affiche l'implémentation proposée pour l'implémentation du service de registre de drones.

FIGURE 7 – Diagramme de classe résumant l'implémentation du service de génération de cartes (Map Generation)



La génération de carte en direct se déroulera sur le serveur et impliquera le travail de deux services du côté du serveur : l'accumulateur d'observations ainsi que le générateur de carte.

L'accumulateur de données sera une interface entre le serveur et les drones qui aura la responsabilité de cumuler l'ensemble des observations provenant des capteurs de chacun des drones. Les drones de l'essaim communiqueront de manière asynchrone et périodique leur positionnement et les observations de leur capteur à la station centrale. Ainsi, ce service gèrera la réception de ces données et l'entreposage de celles-ci dans une file jusqu'à ce qu'elles soient utilisées par le service de génération de carte.

Le générateur de carte aura la responsabilité d'implémenter un algorithme capable d'utiliser les observations provenant de l'accumulateur de données afin de construire en continu une carte et de périodiquement mettre à jour l'état de la carte en cours de génération de manière locale dans une carte gardée en mémoire, dans la base de données ainsi que de diffuser la carte en cours de conception à l'ensemble des clients connectés.

La constitution en direct d'une carte pour un client qui se connecte se produira en deux étapes. Pour une nouvelle connexion, en première étape le serveur enverra l'état actuel de la carte ayant l'ensemble des coordonnées accumulées jusqu'à moment de la connexion, ce qui constituera la base de la carte pour une nouvelle connexion. En deuxième étape, le serveur enverra uniquement les nouveaux points à ajouter à la représentation de la carte de manière à ce que le client puisse continue avoir la carte la plus à jour simplement en intégrant les nouveaux points reçus du serveur.

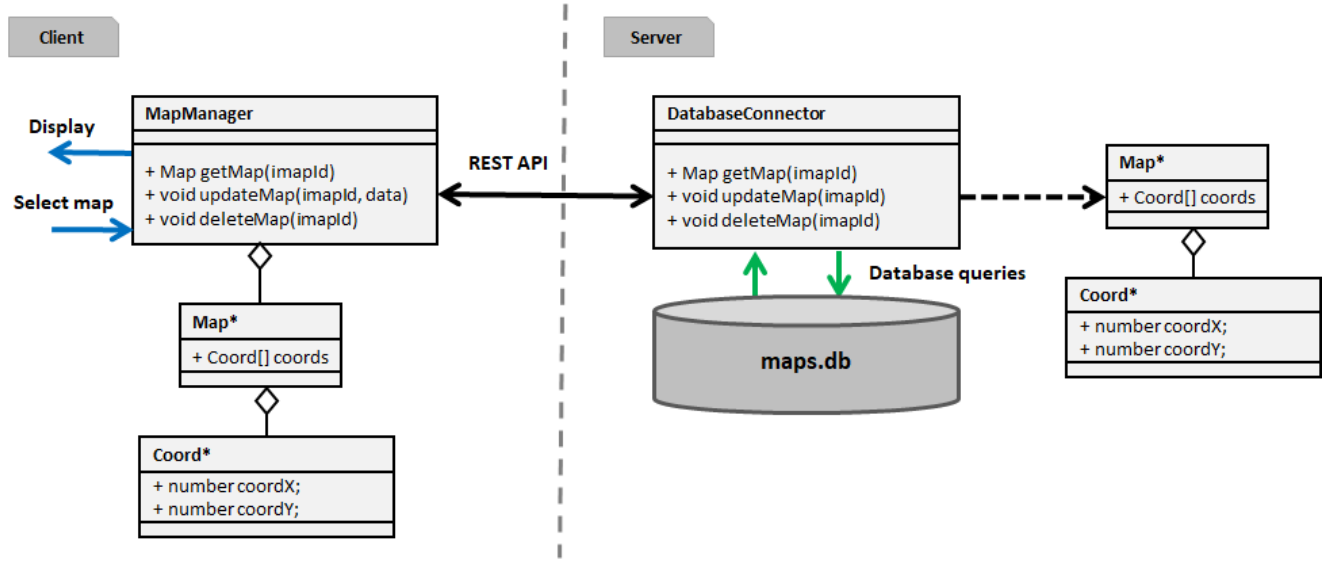
Il sera nécessaire de définir une structure de donnée équivalente pour la représentation des cartes sur le client et sur le serveur.

3.3.4 Consultation de cartes (Map Catalog)

La responsabilité du service de consultation de cartes sur le client sera de permettre la gestion des cartes précédemment générées.

La figure suivante affiche l'implémentation proposée pour l'implémentation du service de registre de drones.

FIGURE 8 – Diagramme de classe résumant l'implémentation du service de consultation de cartes (Map Catalog)



Ce service sera une implémentation d'un API REST permettant l'accès aux cartes (GET), la mise à jour des métadonnées et informations complémentaires des cartes (UPDATE) et la suppression de cartes (DELETE). L'ajout de cartes ne sera pas implémenté, car afin d'assurer l'intégrité des cartes conçues par mission d'exploration, il ne devra pas être possible de manipuler manuellement l'ajout de cartes dans la base de données.

3.4 Détails ajoutés pour la CDR

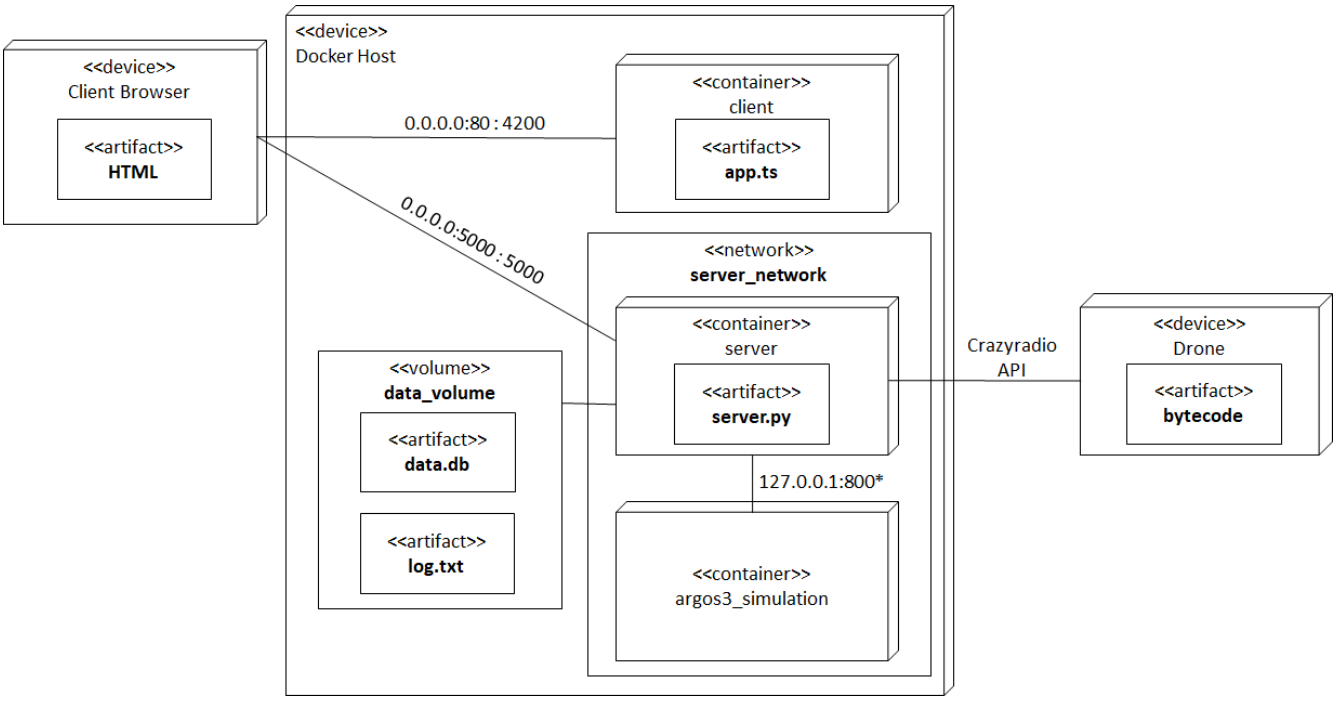
La base de donnée utilisée sera une base de donnée locale Sqlite3. Le module de base de données Sqlite3 permet d'entreposer les données localement sans l'utilisation d'un serveur de base de données dédié accessible par des connections exters. Cette décision a été prise en raison du fait que les données générées de seront pas sujettes à un grand volume de requêtes et de plus, il n'y a pas de nécessité à rendre la base de donnée directement addressable aux requêtes externes.

De plus, les communications entre les drones et la s

3.5 Stratégie de déploiement containeurisé

Enfin, le système sera containeurisé afin de simplifier dont déploiement dur le systèmes d'opération Linux. La figure suivante résume les principales caractéristiques de la structure de containerisation envisagée.

FIGURE 9 – Schéma de la containeurisation du système



Pour la containeurisation de l'application se fera par le moyen de 3 images dockers qui seront orchestrées par le moyen d'un fichier "docker-compose.yml". Les trois images dockers seront les suivates.

— Image du client
— Image du serveur

— Image de la simulation argos

Le conteneur ayant l'image du client aura la responsabilité de contenir le serveur Angular de l'application web. Son port 4200 sera mappé au paurt 80 de la machine hôtesse de l'application afin de permettre une connection au client par le port par défaut des connections http.

Le conteneur ayant l'image serveur aura la responsabilié de contenir le serveur python principal de l'application ausi que de contenir les library C nécessaire à la compilation de code embarqué pour les drones matériels. Cette image servira les clients via son port 5000 qui devra être exposée via le port 5000 de la machine hôte. Enfin, ce conteneur devra posséder un volume afin de permettre une permanance des données de la base de données ainsi que des logs de fonctionnement du serveur sur la machine hôte.

Finalement, l'image de la simulation ARGoS aura la responsabilité de contenir l'ensemble des fichiers nécessaires au lancement d'une simulation ARGoS. Ce conteneur devra posséder un accès à l'affichage de la machine du serveur, ceci se fera par le moyen d'un serveur X ou d'un VNC.

4 Processus de gestion

4.1 Estimation des coût du projet

Selon notre contrat contractuel, nous estimons faire un minimum de 573 heures travaillé et 57 heures tampons. Les cinq personnes de l'équipe travailleront toutes équitablement, donc ils risquent de travailler 114,6 heures chacun. Dans cette équipe, nous avons quatre développeurs-analystes qui sont payés 130$/h chacun. Ceci fait donc un salaire total de 74 490$ pour les quatre développeurs-analystes. Par la suite, nous avons un coordonnateur de projet qui risque de travailler, lui aussi, un minimum de 114,6 heures. Le salaire d'un coordonnateur de projet est de 145$/h. Le salaire total du coordonnateur de projet est de 16 617$. Les coûts salariaux minimaux du projet sont de 91 107$.

Pour les heures tampons, nous pensons les séparer équitablement à travers l'équipe. Donc, les développeurs-analystes vont faire 11,4 heures supplémentaires chacun ce qui donne un salaire supplémentaire de 1 482$. Pour le coordonnateur de projet, il risque de faire un salaire supplémentaire de 1 653$.

Le matériel est fourni par l'entreprise avec du matériel de rechange. Si nous brisons ou perdons du matériel, il y a toujours du matériel de rechange. Donc, nous ne pensons pas dépenser de l'argent sur le matériel.

4.2 Calendrier de projet

La réalisation du projet a été planifiée dans le temps en planifiant des jalons d'avancement basés sur les trois livrables prévus pour la réalisation du projet. Les trois livrables prévus étant les suivants :

— Livrable 1 : Preliminary Design Review (PDR) le 15 février 2021 avant 12h00 pm
— Livrable 2 : Critical Design Review (CDR) 8 mars 2021 avant 12h00 pm
— Livrable final : Readiness Review (RR) 12 avril avant 12h00 pm

De plus, des objectifs hebdomadaires et/ou bihebdomadaires ont été planifiés et constitueront les 7 phases de développement qui permettront de compléter la réalisation du système à l'intérieur de 10 semaines allouées. Les 10 étapes sont les suivantes :

— Phase A (1 semaine) : Mise en place l'environnement de travail
— Phase B (1 semaine) : Préparation à la PDR
— Phase C (1 semaine) : Conception d'éléments structurants du projet
— Phase D (2 semaines) : Préparation à la CDR
— Phase E (1 semaine) : Tâches finales d'intégration d'avant livrable
— Phase F (1 semaines) : Conception des dernières tâches du projet
— Phase G (3 semaines) : Vérification et intégrations finales du projet (ou temps tampon)

Les tableaux ci dessous affiche un calendrier détaillé avec les objectifs de chaque phase du projet. Il doit être pris en considération de que ce calendrier pourrait être sujet à des ajustements au cours des différentes phases de développement.

Certaines tâches avaient des estimations de temps qui n'ont pas été respectés, car nous avons passé soit moins ou soit plus de temps que prévu sur ces tâches. Nous avons dû ajuster les estimations de temps des tâches lorsque nous accomplissions nos tâches.

4.2.1 Revue durant la CDR :

De plus, nous avons réaliser que nous avions oublié quelques tâches lors de l'avancement du projet. Ces tâches ont été ajoutées à notre GitLab. Ces tâches sont :

— ARGOS/SERVEUR : Connecter dynamiquement les drones à la station au sol (estimation de 6h)
— DRONES/SERVEUR : Connecter dynamiquement les drones à la station au sol (estimation de 6h)
— CLIENT/SERVEUR/ARGOS : Avoir un bouton pour partir une nouvelle simulation ARGOS avec murs aléatoires (estimation de 2h)

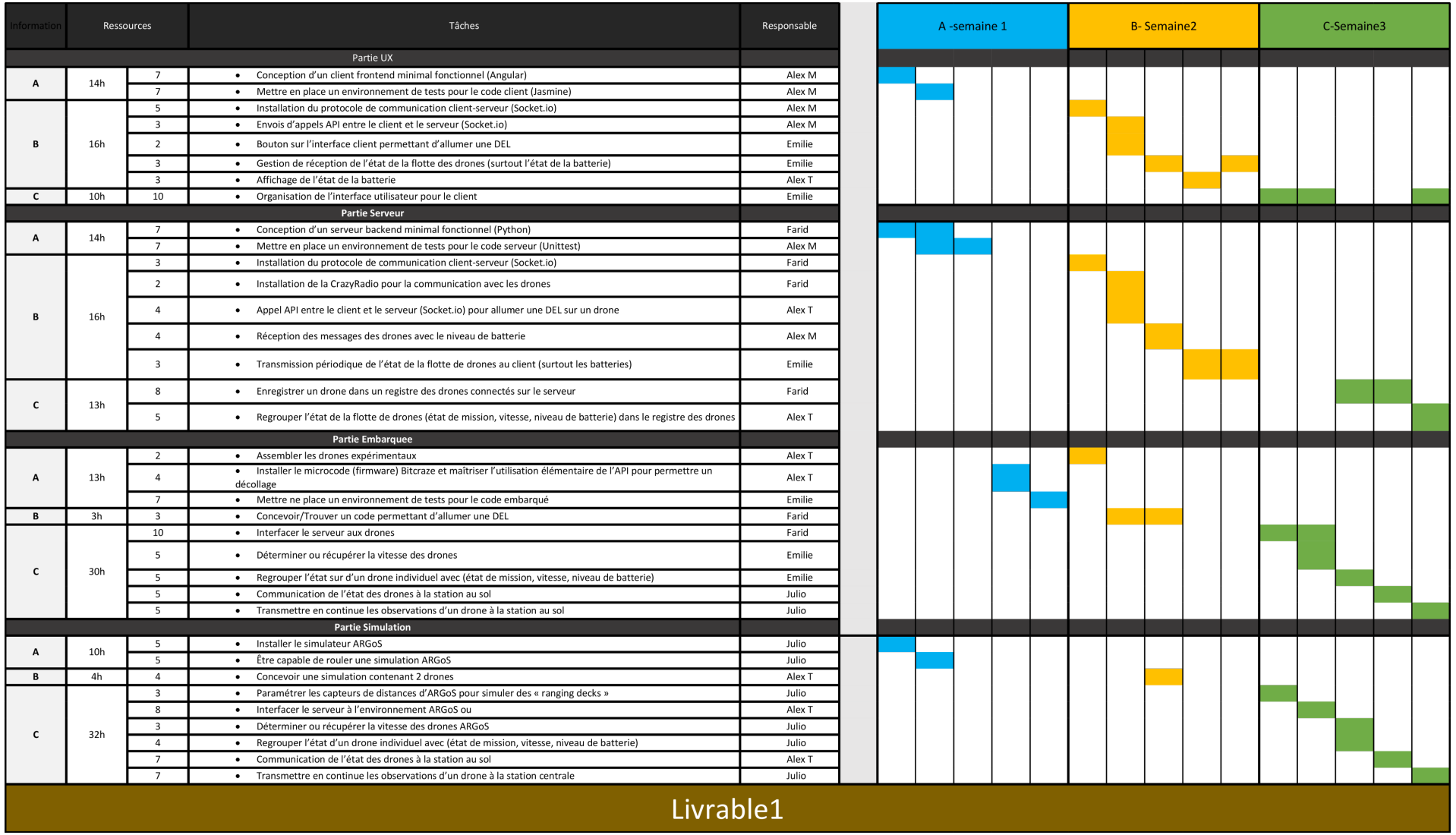
Malgres ces ajouts, les tableaux ci dessous sont toujours une reference utiles. Des changements mineurs du nombres des heures et des retards pour quelques taches durant des semaines intermediaires sont survennue. Ces changement sont visible sur notre GitLab.

TABLE 2 – Vue d'ensemble de la planification pour l'ensemble du projet

Phase	Temps	Objectifs	Ressources
A. Mise en place l'environnement de travail	1 semaine	**L'objectif est de mettre en place l'environnement de développement et de se familiariser avec les outils de travail du mandat :** — Maîtriser l'utilisation élémentaire de l'API Bitcraze (R.M.1 à R.M.4) — Choisir Standard de programmation (RQ1 à 3) — Préparer l'environnement de tests pour chaque partie du projet (R.C.2 et R.C.3)	50h
B. Préparation à la PDR	1 semaine	**L'objectif est de préparer tous les éléments de la Preliminary Desgin Review :** — Implémenter l'envois d'un message du client vers les drones — Réception d'un message du drone vers le client — Lancer un simulation Argos avec 2 drones	40h
C. Conception d'éléments structurants du projet	1 semaine	**L'objectif est de compléter en priorité des éléments structurels de l'infrastructure de notre projet afin de permettre un développement plus efficace du reste des fonctionnalités du mandat :** — Connecter le projet de l'interface client jusqu'à l'environnement de simulation ARGoS — Interfacer les drones matériaux	85h
LIVRABLE 1 : Preliminary Design Review (PDR)	15 février 2021 avant 12h00 pm	**Démontrer les éléments suivants :** — Simulation ARGoS avec deux drones qui suivent un parcours quelconque — Serveur web sur la station au sol avec une page qui montre le niveau de batterie des drones — Bouton qui allume ou éteint les DEL du drone	
D. Préparation à la CDR	2 semaines	**L'objectif est de préparer tous les éléments de la Critical Desgin Review :** — Simulation ARGoS avec 4 drones navigant dans un environnement avec murs générés aléatoirement — Serveur web interfacé avec la simulation ARGoS pouvant envoyer les commandes « Take off » et « Return to base » — Serveur web montrant l'état de la flotte de drones — Code embarqué sur les drones qui envoie les mesures du ranging deck (selon les requis R.F.5 et R.L.6) — Prototype de visualisation de la carte générée par les robots	105h

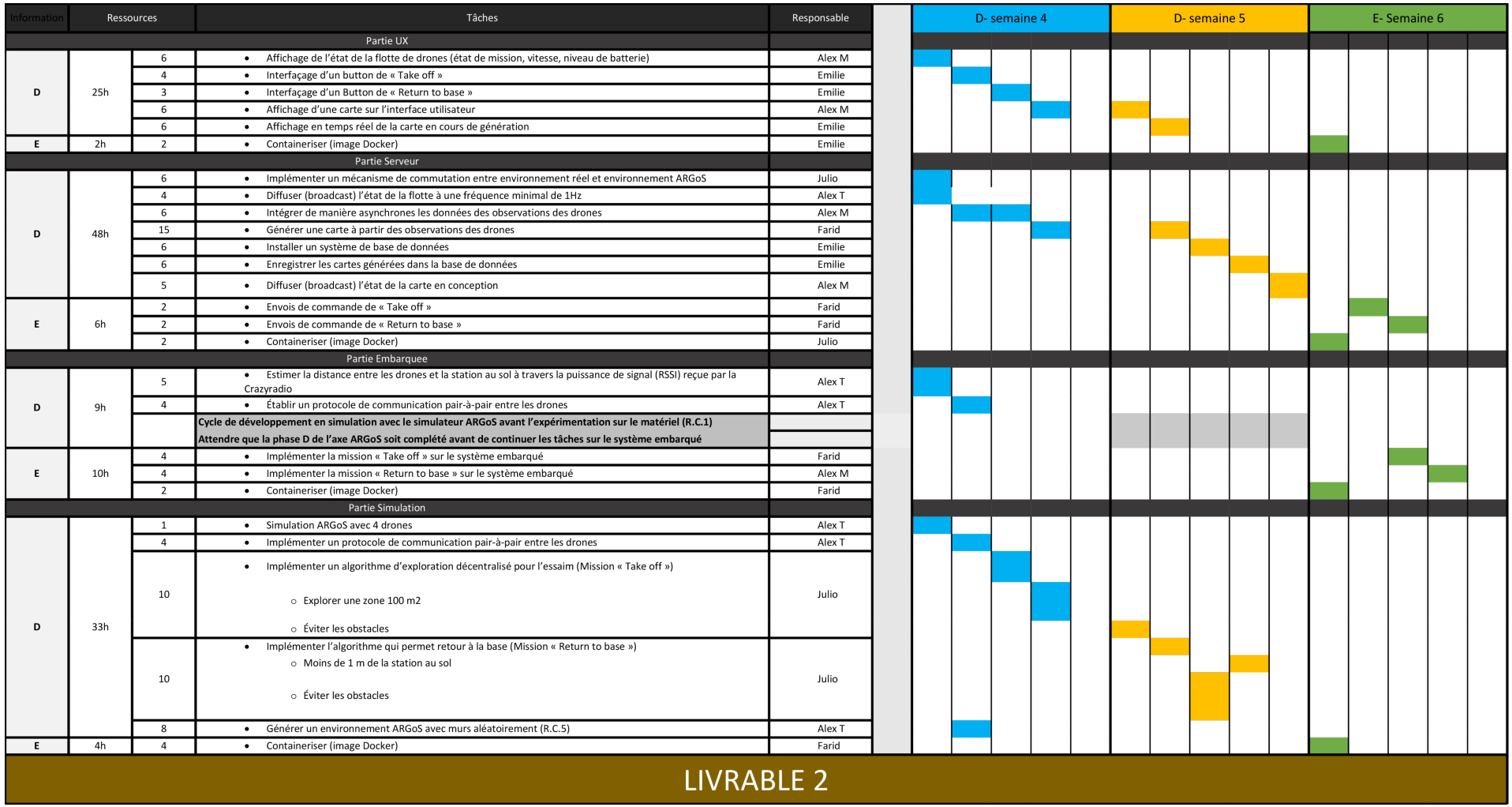
E. Tâches finales d'intégration d'avant livrable	1 semaine	**L'objectif est d'assurer que l'intégration des différents composants fonctionne et de prévoir une période de temps tampon.** — Containerisation du projet (docker-compose) — Intégration et vérification du fonctionnement de tous les composants	60h (22h prévues + 38h tampon)
LIVRABLE 2 : Critical Design Review (CDR)	8 mars 2021 avant 12h00 pm	**Démontrer les éléments suivants :** — Simulation ARGoS avec 4 drones qui volent dans un environnement avec murs générés aléatoirement — Les drones dans la simulation utilisent des capteurs de distance (comme le ranging deck) — Les drones évitent les obstacles et décident de leur parcours — Serveur web interfacé avec la simulation ARGoS — Commande « Take off » implémentée — Commande « Return to base » implémentée — Interface du système implémenté selon le requis R.F.5 — Code embarqué sur les drones qui envoie les mesures du ranging deck (selon les requis R.F.5 et R.L.6) — Prototype de visualisation de la carte générée par les robots	
F. Conception des dernière tâches du projet	1 semaine	**L'objectif est d'implémenter les dernières fonctionnalités requises du projet :** — Système de logs — Commande de mise à jour — Système de persistance des cartes — Consolidation du système de génération de carte — Consolidation de l'implémentation des algorithmes d'exploration (matériel et simulé)	75h
G. Vérification et intégrations finales du projet (ou temps tampon)	3 semaines	**L'objectif est d'assurer que l'intégration de tout le projet fonctionne et de prévoir une période de temps tampon.** — Vérification du système — Période de temps tampons pour la gestion des imprévus **Tâche globale :** — Containerisation complète du système — Concevoir une vidéo explicative	200h
LIVRABLE FINAL Readiness Review (RR)	12 avril avant 12h00 pm	**Démontrer les éléments suivants :** — Un logiciel informatique respectant l'ensemble des critères de l'appel d'offre — Une vidéo montrant le fonctionnement du système en simulation et avec des vrais robots — Tout script nécessaire pour lancer une simulation du système avec une seule commande sur Linux	

FIGURE 10 – Diagramme resumant les taches pour le premier livrable



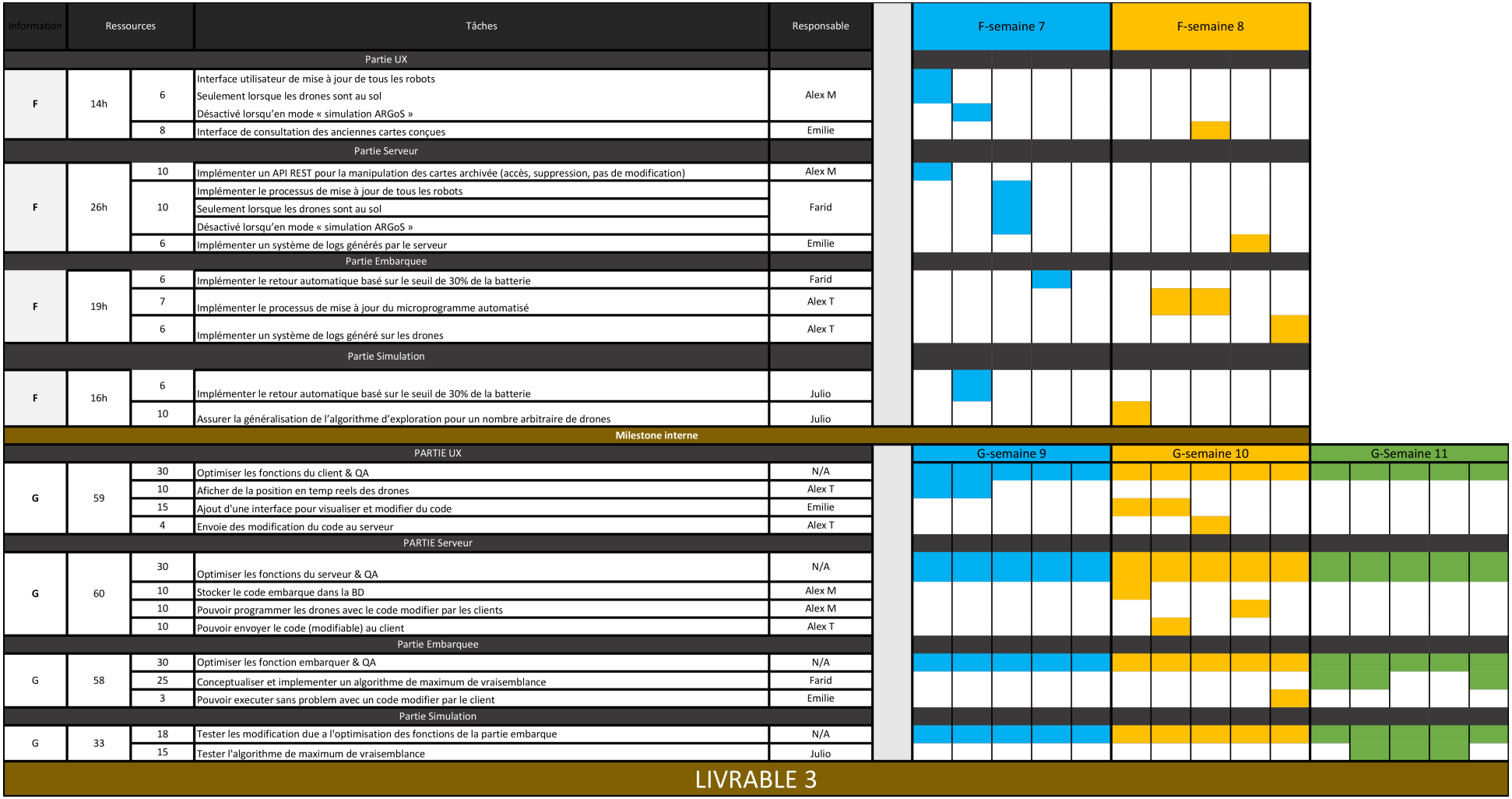
Information	Ressources		Tâches	Responsable
			Partie UX	
A	14h	7	• Conception d'un client frontend minimal fonctionnel (Angular)	Alex M
		7	• Mettre en place un environnement de tests pour le code client (Jasmine)	Alex M
B	16h	5	• Installation du protocole de communication client-serveur (Socket.io)	Alex M
		3	• Envois d'appels API entre le client et le serveur (Socket.io)	Alex M
		2	• Bouton sur l'interface client permettant d'allumer une DEL	Emilie
		3	• Gestion de réception de l'état de la flotte des drones (surtout l'état de la batterie)	Emilie
		3	• Affichage de l'état de la batterie	Alex T
C	10h	10	• Organisation de l'interface utilisateur pour le client	Emilie
			Partie Serveur	
A	14h	7	• Conception d'un serveur backend minimal fonctionnel (Python)	Farid
		7	• Mettre en place un environnement de tests pour le code serveur (Unittest)	Alex M
B	16h	3	• Installation du protocole de communication client-serveur (Socket.io)	Farid
		2	• Installation de la CrazyRadio pour la communication avec les drones	Farid
		4	• Appel API entre le client et le serveur (Socket.io) pour allumer une DEL sur un drone	Alex T
		4	• Réception des messages des drones avec le niveau de batterie	Alex M
		3	• Transmission périodique de l'état de la flotte de drones au client (surtout les batteries)	Emilie
C	13h	8	• Enregistrer un drone dans un registre des drones connectés sur le serveur	Farid
		5	• Regrouper l'état de la flotte de drones (état de mission, vitesse, niveau de batterie) dans le registre des drones	Alex T
			Partie Embarquee	
A	13h	2	• Assembler les drones expérimentaux	Alex T
		4	• Installer le microcode (firmware) Bitcraze et maîtriser l'utilisation élémentaire de l'API pour permettre un décollage	Alex T
		7	• Mettre ne place un environnement de tests pour le code embarqué	Emilie
B	3h	3	• Concevoir/Trouver un code permettant d'allumer une DEL	Farid
C	30h	10	• Interfacer le serveur aux drones	Farid
		5	• Déterminer ou récupérer la vitesse des drones	Emilie
		5	• Regrouper l'état sur d'un drone individuel avec (état de mission, vitesse, niveau de batterie)	Emilie
		5	• Communication de l'état des drones à la station au sol	Julio
		5	• Transmettre en continue les observations d'un drone à la station au sol	Julio
			Partie Simulation	
A	10h	5	• Installer le simulateur ARGoS	Julio
		5	• Être capable de rouler une simulation ARGoS	Julio
B	4h	4	• Concevoir une simulation contenant 2 drones	Alex T
C	32h	3	• Paramétrer les capteurs de distances d'ARGoS pour simuler des « ranging decks »	Julio
		8	• Interfacer le serveur à l'environnement ARGoS ou	Alex T
		3	• Déterminer ou récupérer la vitesse des drones ARGoS	Julio
		4	• Regrouper l'état d'un drone individuel avec (état de mission, vitesse, niveau de batterie)	Julio
		7	• Communication de l'état des drones à la station au sol	Alex T
		7	• Transmettre en continue les observations d'un drone à la station centrale	Julio

FIGURE 11 – Diagramme resumant les taches pour le deuxieme livrable



Information	Ressources		Tâches	Responsable
			Partie UX	
D	25h	6	• Affichage de l'état de la flotte de drones (état de mission, vitesse, niveau de batterie)	Alex M
		4	• Interfaçage d'un button de « Take off »	Emilie
		3	• Interfaçage d'un Button de « Return to base »	Emilie
		6	• Affichage d'une carte sur l'interface utilisateur	Alex M
		6	• Affichage en temps réel de la carte en cours de génération	Emilie
E	2h	2	• Containeriser (image Docker)	Emilie
			Partie Serveur	
D	48h	6	• Implémenter un mécanisme de commutation entre environnement réel et environnement ARGoS	Julio
		4	• Diffuser (broadcast) l'état de la flotte à une fréquence minimal de 1Hz	Alex T
		6	• Intégrer de manière asynchrones les données des observations des drones	Alex M
		15	• Générer une carte à partir des observations des drones	Farid
		6	• Installer un système de base de données	Emilie
		6	• Enregistrer les cartes générées dans la base de données	Emilie
		5	• Diffuser (broadcast) l'état de la carte en conception	Alex M
E	6h	2	• Envois de commande de « Take off »	Farid
		2	• Envois de commande de « Return to base »	Farid
		2	• Containeriser (image Docker)	Julio
			Partie Embarquee	
D	9h	5	• Estimer la distance entre les drones et la station au sol à travers la puissance de signal (RSSI) reçue par la Crazyradio	Alex T
		4	• Établir un protocole de communication pair-à-pair entre les drones	Alex T
			Cycle de développement en simulation avec le simulateur ARGoS avant l'expérimentation sur le matériel (R.C.1)	
			Attendre que la phase D de l'axe ARGoS soit complété avant de continuer les tâches sur le système embarqué	
E	10h	4	• Implémenter la mission « Take off » sur le système embarqué	Farid
		4	• Implémenter la mission « Return to base » sur le système embarqué	Alex M
		2	• Containeriser (image Docker)	Farid
			Partie Simulation	
D	33h	1	• Simulation ARGoS avec 4 drones	Alex T
		4	• Implémenter un protocole de communication pair-à-pair entre les drones	Alex T
		10	• Implémenter un algorithme d'exploration décentralisé pour l'essaim (Mission « Take off ») ○ Explorer une zone 100 m2 ○ Éviter les obstacles	Julio
		10	• Implémenter l'algorithme qui permet retour à la base (Mission « Return to base ») ○ Moins de 1 m de la station au sol ○ Éviter les obstacles	Julio
		8	• Générer un environnement ARGoS avec murs aléatoirement (R.C.5)	Alex T
E	4h	4	• Containeriser (image Docker)	Farid

FIGURE 12 – Diagramme resumant les taches pour le troisieme livrable



Le travail sera réévalué sur une base hebdomadaire afin de réaliser des sprints d'une semaine. Le projet sera réalisé par des étapes de sprints hebdomadaires. Ainsi, pour les phases ayant une durée de plus d'une semaines seront subdivisées en sprints d'une durée d'une semaine.

4.3 Planification des tâches

Afin de réaliser le système requis par le mandat, l'ensemble des tâches à accomplir ont été réparties selon l'aspect du projet qu'il concernait. La mise en place de la solution proposée nécessitera un développement en parallèle sur quatre fronts différents :

— Tâches liées à l'aspect client (le frontend)
— Tâches liées à l'aspect serveur (le backend)
— Tâches liées au code embarqué sur les drones
— Tâches liées à l'aspect simulation dans l'environnement de simulation ARGoS

Une planification détaillée de l'ensemble des tâches constituant le travail à accomplir est fournie dans la table 2, 3, 4 et 5 de l'annexe. De plus, des diagrammes de Gants affichant visuellement les dépendances des tâches dans le temps sont également affichés en annexe.

La planification des tâches est basée sur la dépendance des tâches, ainsi que les principaux jalons, qui sont les trois livrables et les objectifs hebdomadaire ou bihebdomadaires tels que décrits dans la vue d'ensemble de la planification pour l'ensemble du projet. De plus, il est à noter que la planification des tâches suit une approche orientée par le développement en simulation dans l'environnement ARGoS avant l'implémentation sur les drones matériels. Par conséquent, les tâches en simulation préemptent systématiquement l'implémentation sur le code embarqué.

Il est également à noter que la planification proposée représente une estimation conservatrice du temps nécessaire pour concevoir un système respectant les requis minimums du projet. Ainsi, les spécifications dites « Optionnelles » ont été planifiée uniquement à la dernière phase de développement et leur faisabilité pourraient être sujettes à une réévaluation. Ces tâches incluent l'affichage en direct des drones dans la carte en cours de conception, l'implémentation d'un éditeur de code pour le drone ainsi que la détermination de la position des drones réalisée par la résolution d'un problème d'optimisation.

Les temps nécessaires aux activités de tests n'ont pas été affichés dans les tableaux détaillés des tâches à accomplir. Il est calculé que le temps des tâches inclue le temps nécessaire à la complétion des activités de tests associées aux tâches.

Les intervalles de temps tampon sont planifiés dans les différentes phases de développement afin de prendre en considération les possibles imprévus ou délais supplémentaires pouvant survenir lors de la réalisation des tâches. Cette mesure devrait permettre d'éviter dépassements de temps imprévus dans la réalisation du projet.

Il doit être pris en considération de que la planification des tâches suggérée dans l'appelle d'offre pourrait être sujette à des ajustements dans la situation où les besoins du client évolueraient ou dans la situation où des défis techniques supplémentaires seraient découverts dans la réalisation du projet. Ces modifications dans la planification des tâches n'engendreront pas de coups supplémentaires imprévus en raison des temps tampons déjà prévus dans la planification suggérée.

4.4 Ressources humaines du projet

Notre équipe est constituée de cinq personnes qui sont des développeurs chevronnés avec plusieurs années d'expérience.
Avec le projet intégrateur 1, les membres de l'équipe ont développé les compétences nécessaires pour la manipulation et la programmation de petits robots mobiles :

- Assembler des composants
- Faire la soudure de composants électroniques
- Programmer un microprocesseur en C++
- Recueillir des données à partir des capteurs posés sur un robot.

Le projet intégrateur de la deuxième année a permis aux membres de l'équipe de renforcer leurs connaissances sur le développement web (frontend et backend), les bases de données, typescript, méthodologie agile scrum et bien d'autres. À tout cela, nous pouvons ajouter les connaissances particulières de chaque membre. Pour **Julio Dandjinou**, ces connaissances sont : les tests d'interface utilisateur d'un logiciel, certaines connaissances en méthodologies de test et en C#. Du côté d'**Alexandre Morinvil**, il a une bonne connaissance du langage python et il maîtrise bien l'outil Docker. Alexandre Morinvil est un atout, car il est un très bon planificateur. **Émilie Vaudrin** a une bonne maîtrise du langage python et elle est une bonne rédactrice. Pour **Farid El Fakhry**, il maîtrise bien l'outil de gestion de version Git et l'environnement Linux. Finalement, **Alexandre Talen** maîtrise lui aussi très bien l'outil de gestion de version Git et l'environnement Linux. Celui-ci a aussi de l'expérience professionnelle en développement.

5 Suivi de projet et contrôle

5.1 Contrôle de la qualité

Notre équipe s'engage à livrer une solution maintenable et évolutive. Nous suivons des standards établis par Google. Ce standard a un effet sur la totalité du code : nom de variable, fonction, espaces, indentation ...etc.

Des outils permettent la détection automatique des odeurs dans le code des "linter" comme pylint pour le code en python, tslint pour la partie web et cpplint pour le code embarqué. Nous effectuons aussi une vérification de chaque modification du code membre d'équipe par un autre membre d'équipe choisit arbitrairement grâce à un merge request. Ce processus consiste à notifier l'équipe qu'une personne est prête a soumettre une partie du code, une autre personne de l'équipe se chargera de revoir le code et d'y laisser des commentaires (dans les cas où des modifications seraient requises).

Une autre fonctionnalité utile de Gitlab est le pipeline pour requête merge. En effet, gitlab nous permet d'exécuter des scripts avant d'ajouter du code dans notre branche master. Nous utilisons des scripts simples permettant d'executer des tests d'intégrations pour vérifier le bon fonctionnement des multiples modules de la solution proposée.

5.2 Gestion de risque

Plusieurs risques pourraient survenir lors de notre projet ce qui entraine à nous faire changer l'organisation de notre échéancier et de notre équipe. Les risques possibles peuvent être des risques concernant le respect du calendrier, les risques humains et les risques techniques. Dans les risques humains, nous avons les risques d'une mauvaise communication et le risque d'une mauvaise répartition des tâches en fonction des compétences de chacun. Dans les risques techniques, nous avons le risque de ne pas avoir accès aux drones lorsque nous en avons besoin parce que le virus COVID-19 nous empêche de nous rencontrer et donc une seule personne peut accéder au matériel en même temps. Le risque le plus important serait le respect du calendrier et, par la suite, les risques techniques et humains auraient les mêmes priorités.

Pour le risque du respect du calendrier, notre solution serait de se mettre des dates d'échéance deux jours plus tôt. Cette solution nous laisserait deux jours supplémentaires si nous rencontrons des problèmes techniques avant la remise. C'est deux jours ne doivent techniquement pas être utilisé et nous devons être prêt à remettre le travail deux jours avance la date d'échéance.

Pour le risque d'une mauvaise communication, nous avons déjà évalué que ce risque pourrait se produire assez facilement, donc nous sommes immédiatement mis pour solution de se rencontrer couramment durant la semaine pour se garder informer sur nos avancements. Nous avons aussi décidé d'utiliser notre première semaine du projet pour faire une activité de cohésion pour se connaître d'avantage et que la communication soit plus facile.

Pour le risque d'une mauvaise répartition des tâches, si nous réalisons que ce risque se produit nous allons ajuster notre équipe pour qu'une personne de l'équipe aide les personnes en difficulté. De plus, nos rencontres sont justement faites pour parler de nos difficultés et pour qu'on s'entraide.

Pour le risque technique, ce risque est assez difficile à traiter, car nous n'avons aucun contrôle là-dessus. Notre solution serait de se passer le matériel régulièrement pour que ça ne soit pas juste une personne qui le manipule et pour que lorsque nous en avons besoin pour tester, il soit à notre disposition. Sinon, une personne serait responsable de tester le drone physiquement.

5.3 Tests

La réalisation rigoureuse de tests est indispensable car cela nous permettra de maitriser la qualité du produit livré et de minimiser les risques de retard de livraison.

Les tests seront réalisés de la manière suivante :

- Tout d'abord, des tests automatisés seront implémentés :
  - Chaque composante des systèmes sera testée par des tests unitaires exhaustifs
  - Chaque système et sous-système seront testés par au moins un test d'intégration.
- Ensuite, certains tests seront réalisés manuellement :
  - Le bon fonctionnement du programme (réponse correcte aux appels envoyés au robot, cartographie réalisée, détection de collision) sera testé au cours du développement à l'aide d'une simulation Agros.
  - Le bon fonctionnement des robots (contrôle des moteurs, led, déplacement) sera vérifié manuellement sur les robots matériels.

L'objectif recherché est de maximiser la couverture des tests automatisés afin d'effectuer le moins possible de tests manuels et d'assurer une présence minimale de bugs. Dans un contexte de pandémie, nous essayons d'effectuer le maximum de tests sur robot dans l'environnement Argos. Cependant nous garantissons que tout code remis sera testé rigoureusement sur un robot matériel.

5.4 Gestion de configuration

Pour développer continuellement notre solution, nous utilisons git, plus spécifiquement, la plateforme Gitlab. Nous suivons "workflow gitflow" qui définit un modèle de création de branches conçu autour de la livraison de projet. Ce modèle utilise deux branches principales. La branche "master" est utilisée pour stocker l'historique officiel des versions. La deuxième branche "dev" est utilisée pour le développement. D'autres branches sont créées à partir de

la branche de développement pour chaque fonctionnalité. À la fin du développement de chaque fonctionnalité, elle

sera mergée dans la branche 'dev' qui à son tour quand toutes les fonctionnalités prévues pour le sprint s'achèvent se verra attribuer un numéro de version et mergé dans la branche principale (master). Toute fonctionnalité mergée dans 'dev' sera considérée comme fonctionnelle et testée. La tâche correspondante sur Gitlab sera alors fermée. Ceci pourra nous permettre de visualiser le niveau d'avancement du projet. Les commentaires font partie de notre code. Chaque fichier source contiendra des commentaires expliquant le fonctionnement de la classe et des méthodes qu'il contient. De plus, chaque livrable sera accompagné d'une documentation décrivant le fonctionnement du code. En

addition à cela, des diagrammes UML expliqueront la relation entre les différentes parties de la solution. Enfin, une vidéo sera réalisée pour expliquer notre interface utilisateur. La documentation est séparée en trois parties principales : la documentation du système embarqué, la documentation du client et la documentation du serveur. Toute la documentation sera aussi disponible via un lien disponible sur l'application web.

6 Références

1. Beltrame, G. (2021). Projet de conception d'un système informatique, Appel d'offres Polytechnique Montreal.
2. Beltrame, G. (2021). Note de cours : Exigences techniques
3. Beltrame, G. (2021). Projet de conception d'un système informatique, Demande de proposition
4. Google (N/A). Google style guides https ://google.github.io/styleguide/
5. Atlassian (N/A). Workflow gitflow. https ://www.atlassian.com/fr/git/tutorials/comparing-workflows/gitflow-workflow

ANNEXES