

INF3995 - Projet de conception d'un système informatique



Système aérien d'exploration: « Preliminary Design Review »

Équipe 0 (GoExplore)

Hugo Lachieze-Rey
Hugo Lachieze-Rey

Pier-Luc Tanguay
Pier-Luc Tanguay

Sami Sadfa
Sami Sadfa

Wail Ayad
Wail Ayad

William Trépanier
William Trépanier

Remis à:

Pierre-Yves Lajoie, CPI, M.Sc.A. - Représentant, Agence Spatiale
Professeur Giovanni Beltrame, Ing., PhD

1^{er} Octobre 2021
Polytechnique Montréal

Table des matières

Introduction	4
1. Vue d'ensemble du projet	4
1.1 But du projet, portée et objectifs	4
1.2 Hypothèse et contraintes	5
1.3 Biens livrables du projet	7
2. Organisation du projet	9
2.1 Structure d'organisation	9
2.2 Entente contractuelle	9
3. Description de la solution	10
3.1 Architecture logicielle générale	10
3.2 Station au sol	12
3.3 Logiciel embarqué	15
3.4 Simulation	17
3.5 Interface Utilisateur	18
3.6 Fonctionnement général	19
4. Processus de gestion	19
4.1 Planification des tâches	19
4.2 Estimations des coûts du projet	21
4.3 Calendrier de projet	22
4.4 Ressources humaines du projet	22
5. Suivi de projet et contrôle	23
5.1 Contrôle de la qualité	23
5.2 Gestion de risque	24
5.3 Tests	25
5.4 Gestion de configuration	25
6. Références	27
7. Annexes	29

Introduction

Le cours INF3995: Projet de conception d'un système informatique est le troisième projet intégrateur de la formation en génie informatique à Polytechnique Montréal. Ce dernier consiste à concevoir et réaliser un système informatique qui met en pratique les concepts et méthodologies acquis depuis le début du curriculum. Étant considéré comme le projet le plus important du parcours académique, ce projet permet à notre groupe d'étudiants d'acquérir de l'expérience en gestion de projet et également d'appliquer nos connaissances en réseautique, systèmes embarqués, sécurité informatique, base de données, multimédia et systèmes d'exploitation. Il nous permettra aussi de raffiner notre esprit analytique et d'aiguiser nos compétences techniques face à un projet de grosseur moyenne. Dans le cadre de ce projet, l'Agence Spatiale sollicite notre expertise afin de développer une preuve de concept d'exploration de terrain inconnu avec des robots munis d'un minimum de capteurs. Le groupe GoExplore a donc été fondé et est heureux de présenter une réponse à l'appel d'offres de l'Agence Spatiale.

1. Vue d'ensemble du projet

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

GoExplore est constitué de cinq étudiants motivés et passionnés qui ont pour objectif de présenter à l'Agence Spatiale une preuve de concept qui leur permettra d'explorer un terrain inconnu avec des robots aériens équipés d'un minimum de capteurs. Nous proposons un système composé d'un ensemble de logiciels qui sera en mesure d'effectuer l'exploration, la communication, le contrôle à distance de deux robots Bitcraze Crazyflie 2.1 et d'effectuer une simulation de l'exploration.

Notre organisation cherche à démontrer qu'elle possède la solution la plus optimale pour répondre aux besoins de l'Agence Spatiale en s'assurant de maximiser la performance et la fiabilité du système, d'utiliser le moins de ressources possibles et ce, avec un coût minimal.

Au cours de ce projet, trois différents livrables sont attendus par le client:

- Le preliminary design review (PDR): 1^{er} octobre 2021
- Le critical design review (CDR): 29 octobre 2021
- Le readiness review (RR): 7 décembre 2021

1.2 Hypothèse et contraintes (Q3.1)

La solution de l'ensemble du système informatique proposé par GoExplore consiste en différents composants qui ont des spécifications et contraintes distinctes. Au-delà des aspects techniques, nous décrivons également les éléments externes à l'équipe.

1.2.1 Composition générale du système

Selon R.L.4, L'Agence Spatiale exige l'utilisation d'une seule commande pour exécuter l'ensemble du système et demande l'intégration par Docker.

1.2.2 Réseau de communication

Afin de communiquer avec les robots aériens, l'Agence Spatiale fournit à l'équipe, pour chaque robot, un transmetteur radio Crazyradio PA à longue portée (jusqu'à 1km). Ce dernier est contraint à un signal d'une fréquence de 2.4 GHz et propose un taux de transmission de 250 kbps, 1 Mbps ou 2 Mbps [3]. Nous faisons l'hypothèse d'utiliser le mode 2 Mbps pour obtenir le meilleur taux de transfert.

1.2.3 Station au sol

Pour assurer la communication entre les robots aériens et le serveur, nous sommes contraints à utiliser un ordinateur en tant que station au sol. Elle constitue également l'interface utilisateur de l'opérateur.

1.2.4 Logiciel embarqué

Selon R.L. 3 des requis, "le contrôle des drones doit se faire à bord de ceux-ci sur le code embarqué." Également, l'API de Bitcraze doit être utilisée pour programmer les robots aériens. Le langage C sera utilisé pour concevoir le logiciel embarqué.

1.2.5 Robots aériens

Deux robots aériens BitCraze CrazyFlie 2.1 sont fournis par l'Agence Spatiale. Nous tenons pour acquis que les composants matériels des robots aériens sont fonctionnels. Ils ne sont équipés que de deux types de capteurs :

- *Flow Deck* : mouvement par rapport au sol et distance par rapport au sol.
- *Multiranger Deck* : mesure de distance des objets autour du robot jusqu'à 4 mètres.

1.2.6 Simulation

La simulation du système informatique utilise l'environnement ARGoS. L'équipe est contrainte d'utiliser cet environnement et au langage de programmation C++ pour l'implémentation des fonctions du contrôleur. Ensuite, nous faisons l'hypothèse de reproduire l'algorithme développé pour le logiciel embarqué puisque nous voulons reproduire le plus possible le comportement physique des robots aériens. À propos du requis R.C.5, nous prévoyons développer un script python qui sera en mesure de générer une arène aléatoire.

1.2.7 Service web

Nous croyons qu'il sera plus efficace pour l'équipe de développer le service web par Angular puisque l'équipe a développé une expertise à son utilisation dans des projets académiques précédents. De plus, nous faisons l'hypothèse qu'une architecture de communication REST sera efficace pour uniformiser l'interface. Le client est en mesure d'envoyer certaines commandes aux robots aériens, dont :

1. Lancer la mission : décollage et début de la mission
2. Retour à la base : rapprocher le robots à moins de 1 mètre du départ
3. Terminer la mission : atterrissage immédiat
4. Mise à jour des robots aériens

1.2.8 Base de données

Nous faisons l'hypothèse que l'utilisation d'une base de données nous permettra de centraliser l'information de tout le système plus facilement. Notre choix d'utiliser une base de données de type SQL nous permet de mieux structurer l'information stockée, pour les obtenir de façon plus efficace.

1.2.9 Algorithme d'exploration

L'algorithme envisagé est le 'swarm gradient bug algorithm (SGBA)' qui a été proposé par une équipe de chercheurs de 'Delft University of Technology', dont Mme Kimberly McGuire, PhD [2]. Notre groupe est sûr que cet algorithme sera en mesure de satisfaire les besoins du projet.

1.2.10 Éléments externes

La version finale du projet est prévue pour le 7 décembre 2021 et la charge requise pour la livraison du projet requiert maximum 630 heures-personnes. La section 4.3 fournit plus d'informations par rapport au calendrier du projet. De plus, les membres de l'équipe, étant tous étudiants à Polytechnique Montréal, devront composer avec des contraintes de temps pour réussir les examens et travaux à remettre d'autres cours. De plus, la pandémie COVID-19 est un

facteur de risque pouvant affecter la productivité de notre équipe. Bien que GoExplore préconise les rencontres en présentiel, nous devons tout de même effectuer du travail à distance.

1.3 Biens livrables du projet (Q4.1)

Le projet s'étale sur un total de 4 mois, pendant le trimestre automne 2021 qui est prévu par le calendrier de Polytechnique Montréal. L'Agence Spatiale exige trois livrables qui seront utilisés en tant que documentation pour l'avancement du projet: le preliminary design review (PDR), le critical design review (CDR) et le readiness review (RR). De plus, notre équipe a ajouté le system design review (SDR) avant le livrable PDR.

Le system design review (SDR) [5]

Le SDR est le premier livrable que l'équipe s'est fixé comme objectif à atteindre le 16 septembre 2021. Durant cette première étape, l'équipe a également pris connaissance du projet réquisitionné par l'Agence Spatiale et élaboré le calendrier de la planification du projet. Le SDR est une révision technique qui permet d'assurer que les choix des composants du système seront suffisants pour accomplir les requis proposés par l'Agence Spatiale. À la lumière de cette étape, l'équipe a défini les sous-systèmes requis pour bien réaliser le projet. Ce livrable n'a pas été remis à l'Agence Spatiale puisqu'il n'est pas un livrable exigé.

Le preliminary design review (PDR)

Le PDR est le premier livrable exigé par l'Agence Spatiale en date du 1er octobre 2021 à 17h. GoExplore répond à l'appel d'offres par l'entremise de ce document. Ce dernier sera accompagné d'un prototype de notre système qui répond aux requis R.F. 1, R.F. 2 (seulement simulation ARGoS). De plus, deux démonstrations vidéo sont fournies.

Le critical design review (CDR)

Le CDR est le deuxième livrable exigé par l'Agence Spatiale en date du 29 octobre 2021 à 17h. Il dresse la conception détaillée système en y incluant toutes ses fonctionnalités. De plus, la documentation répondra aux modifications demandées de la part du client suite au PDR. La documentation sera accompagnée des éléments suivant:

- Implémentation prototype: R.F.8, R.C.3, R.Q.1 et R.Q.2;
- Implémentation et démos: R.F.1, R.F.2, R.F.3, R.F.4, R.F.5, R.F.10 et R.C.1;
- Présentation technique du produit et des changements apportés depuis le PDR.

Le readiness review (RR)

Le RR est le troisième livrable exigé par l'Agence Spatiale en date du 7 décembre 2021 à 12h30. Ce dernier inclut une remise du prototype final et fonctionnel du projet, ainsi qu'une documentation qui servira de réponse aux rétroactions du client suite au CDR. De plus, le RR comporte une présentation finale du projet qui expliquera en détail la solution. Finalement, il inclura également ces éléments:

- Démonstrations de tous les requis complétés;
- Code nécessaire à l'exécution des tests automatiques, accompagné du document de tests;
- Instructions pour compilation et lancement du produit, et tests automatisés (README).

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Le projet sera réalisé par une approche agile, itérative et décentralisée. Chacun des membres de l'équipe de GoExplore deviendra spécialiste d'un composant du système à développer et se chargera de supporter les autres membres de l'équipe par rapport à cette spécialité. Trois rencontres de 30 minutes sont planifiées par semaine pour mettre à jour l'avancement planifié préalablement. L'objectif est de maximiser le partage d'information à travers l'équipe pour un obtenir un rendement optimal. Ces rencontres fréquentes nous permettent d'ajuster dynamiquement la répartition des tâches si nécessaire. De plus, suite aux rencontres, des séances parking lot sont planifiées pour poser des questions, assister un collègue ou pour démontrer de nouvelles fonctions. Nous choisissons d'effectuer un sprint par livrable, donc trois sprints au total. Par contre, nous nous fixons des objectifs hebdomadaires par l'entremise des rapports d'avancements. Le rôle des membres de l'équipe se retrouve dans le tableau 1.

Tableau 1. Rôles des membres de l'équipe

Membre de l'équipe	Rôles
Sami Sadfa	Développeur - Logiciel embarqué
William Trépanier	Développeur - Station au sol
Wail Ayad	Développeur - Simulation Argos
Hugo Lachieze-Rey	Développeur - Interface Web
Pier-Luc Tanguay	Responsable de la gestion de projet

2.2 Entente contractuelle (Q11.1)

Le type d'entente contractuelle proposé est le contrat à terme. C'est ce type de contrat qui est le plus couramment utilisé dans les projets informatiques [7]. Selon nous, ce type de contrat offre à l'Agence Spatiale une meilleure flexibilité, puisque les requis et les spécifications pourraient être sujets à des changements durant le développement. De plus, il permet une mise en chantier plus rapide et permet au client de choisir le contractant le plus qualifié. Sachant que l'Agence Spatiale dispose d'un coût plafond incluant 630 heures-personnes, elle pourra potentiellement offrir le projet à une équipe qui n'est pas nécessairement la moins chère, mais celle qui sera la mieux qualifiée, tout en respectant leur coût plafond. Finalement, le contrat à terme est particulièrement intéressant dans la situation de ce projet puisqu'il réduit les négociations et les coûts préliminaires de spécifications. L'expertise de l'équipe GoExplore permet d'assurer à l'Agence Spatiale un produit final fonctionnel avec un suivi minimal de la part de cette dernière.

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)

Nous privilégions une architecture client-serveur où l'ensemble des communications entre les différentes composantes de la solution est centralisé via un serveur commun. Ceux-ci seront déployés dans des conteneurs Docker afin de favoriser la compatibilité et la reproductibilité de notre solution pour qu'elle soit déterministe. Ces images seront préalablement assemblées et hébergées dans des répertoires d'images GitLab afin de faciliter le déploiement. Dans la mesure où l'utilisateur est correctement authentifié auprès de GitLab, il sera possible en une seule commande de lancer l'ensemble des conteneurs Docker.

Le diagramme de la figure 1 résume l'architecture générale de la solution proposée. D'abord, il sera possible de contrôler le statut de l'exploration par l'entremise d'une interface Web. Cette page web permettra la gestion des drones physiques et de la simulation ainsi que l'affichage d'informations pertinentes relatives à l'état de ceux-ci. Celle-ci sera implémentée à l'aide du cadre Angular et sera servie par un serveur de contenu statique. L'ensemble des communications entre la page web et les drones passeront par l'entremise d'un serveur utilisant le cadre FastAPI [8]. Ce serveur sera responsable du maintien d'un lien de communication entre chaque drone (physique et virtuel) et le serveur. Quant au stockage persistant, il sera assuré par une base de données PostgreSQL exposée uniquement à l'intérieur du réseau Docker.

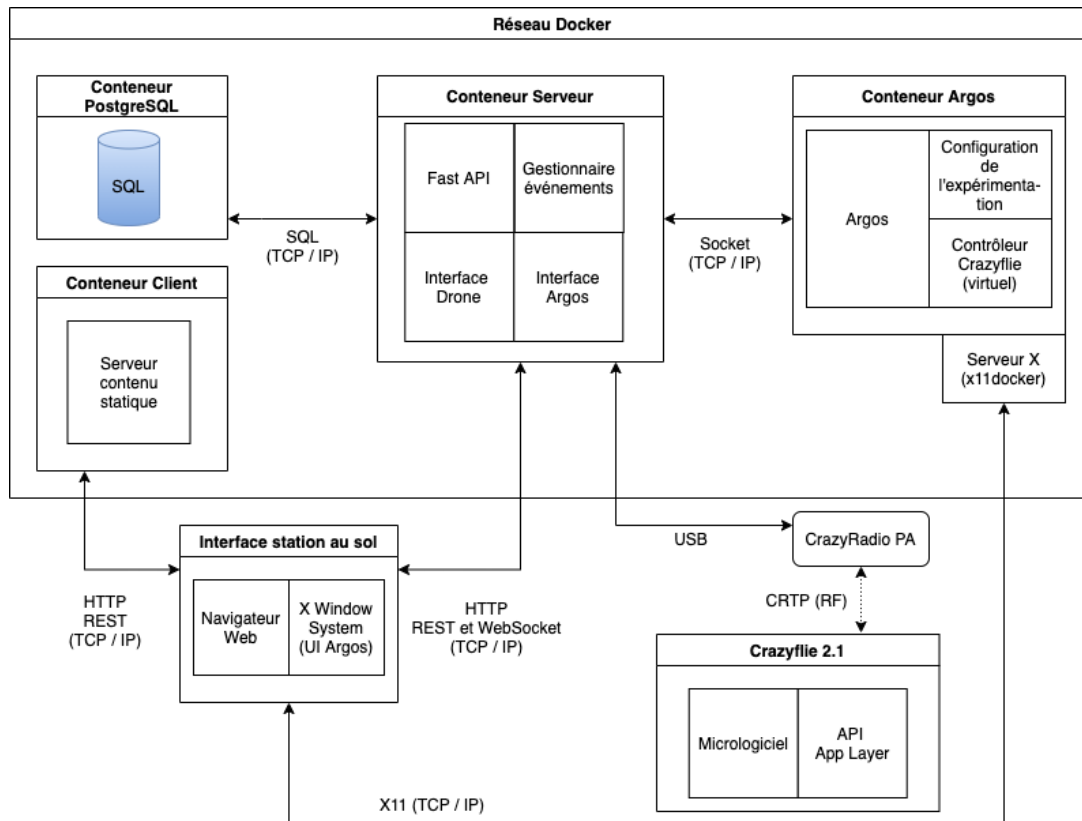


Figure 1. Architecture logicielle générale

La communication entre l'interface de la station au sol et les différents composants s'effectuera à l'aide de trois protocoles tous bâtis sur TCP/IP:

1. L'architecture REST permet au navigateur web de récupérer le contenu statique constituant l'interface web, mais également d'envoyer des instructions au serveur. Il s'agit du protocole de prédilection, car il est bien supporté par les cadriciels choisis.
2. Le protocole WebSocket permettant une communication bidirectionnelle entre le serveur et l'interface web de la station au sol. Il s'agit d'un protocole d'un protocole dont les cadriciels fournissent globalement moins de fonctionnalités, mais il permet l'envoi de données directement du serveur vers le client. Cela permet ainsi d'éviter au client d'avoir à périodiquement faire une requête au serveur pour obtenir l'état à

jour des drones notamment. Seuls les appels nécessitant des mises à jour périodiques bénéficient d'un tel protocole, alors celui-ci ne sera utilisé que dans ce contexte.

3. Le protocole X11 permet finalement à l'interface de la station au sol de présenter l'interface utilisateur d'Argos à même la station au sol. Comme le simulateur Argos est également conteneurisé, il est nécessaire d'utiliser ce protocole pour accéder aux contrôles haut niveau du simulateur (pour notamment démarrer la simulation).

La communication entre le serveur et les drones se fera par l'entremise de protocoles compatibles selon le type du drone. Pour les drones Crazyflie, le protocole utilisé sera le CRTP (Crazy RealTime Protocol), soit le protocole nativement supporté par Bitcraze. Il serait inutilement complexe de ne pas utiliser ce dernier, car il faudrait implémenter de nouveau plusieurs couches de logique applicatives fournies par Bitcraze. Quant aux drones virtuels se retrouvant dans le simulateur Argos, un socket TCP/IP sera plutôt utilisé. Puisqu'il s'agit de deux protocoles permettant la lecture et l'écriture de données arbitraires, il sera possible d'uniformiser les communications entre les drones physiques et virtuels en limitant les différences quant à la structure des données envoyées et reçues. De cette façon, il sera possible de simplifier la gestion des événements. Finalement, le serveur communiquera également avec la base de données PostgreSQL par son protocole usuel.

Comme les différents conteneurs partagent le même réseau Docker, il sera possible de communiquer librement entre eux, car Docker supporte la résolution DNS à même le réseau. Quant à l'interface de la station au sol, elle pourra communiquer avec les conteneurs Client, Serveur et Argos à leur port respectif qui sera exposé à la communication externe.

Requis: R.L.4

3.2 Station au sol (Q4.5)

Comme indiqué dans la section précédente, la station au sol peut être contrôlée par l'entremise d'une interface utilisateur dont l'architecture est détaillée à la section 3.5. Cette section s'intéresse ainsi davantage au fonctionnement interne de la station au sol. Comme la logique relative à la navigation du drone doit se retrouver à même le logiciel embarqué ou au sein du simulateur, les responsabilités de la station au sol sont essentiellement de permettre la communication avec les drones et le traitement des informations pertinentes relatives aux drones.

Pour la portion backend de la station au sol, nous privilégions l'utilisation du langage Python. Puisque la librairie fournie par Bitcraze permettant la communication avec les drones Crazyfly est implémentée dans ce langage, il est préférable de concevoir une solution qui fait usage des ressources fournies par le fabricant des drones plutôt que de réimplémenter des fonctionnalités existantes. Nous avons considéré les cadres suivants: Django Rest Framework, Flask et FastAPI. Le meilleur choix considérant les requis de l'Agence demeure FastAPI pour les raisons suivantes:

- Il est significativement plus rapide que les deux autres; [13]
- Il supporte native les coroutines asynchrones (ASGI);
- Il gère la sérialisation et la désérialisation des données incluant sa validation;
- Il permet la génération d'une documentation conforme à la spécification OpenAPI;
- Il est très facile d'apprentissage et supporte nativement les annotations de type.

La solution proposée basée sur FastAPI fera également une grande utilisation de la librairie native `asyncio`. Il s'agit de la solution supportée par le langage Python pour la programmation asynchrone. De façon similaire à NodeJS, elle permet d'utiliser la syntaxe `async/await` et utilise une `event loop`. L'avantage d'une telle approche est qu'elle est davantage performante lorsqu'il n'y a pas de traitement lourd à faire. En effet, une coroutine est beaucoup plus légère qu'un fil d'exécution qui doit être ordonné par le système d'exploitation. Comme le rôle principal de la portion backend est la gestion d'événements, cette approche est plus performante.

Comme Python supporte les fils d'exécutions (`threads`), `asyncio` supporte l'exécution de code bloquant à même l'`event loop` en déléguant les tâches potentiellement bloquantes à un second fil d'exécution. Cela permet ainsi d'utiliser la librairie fournie par Bitcraze même si elle ne supporte pas nativement la syntaxe `async/await`. Les autres composantes de notre solution sont toutefois compatibles avec cette syntaxe et ces fonctionnalités en bénéficient également de la programmation asynchrone.

Pour communiquer avec les drones, la station au sol crée un lien de communication avec chaque drone qu'il soit physique ou virtuel au démarrage. Le traitement des requêtes par la station au sol implémentera le patron de conception Chaîne de responsabilité. Il sera de la responsabilité des drones, une fois ce lien de communication établi, d'envoyer leurs informations pertinentes à la station au sol (télémétrie, position, état de la batterie, etc.). La station au sol en assurera ensuite le traitement et le maintien de la persistance. Comme le client crée également un lien

bidirectionnel avec le backend, il obtient également des mises à jour périodiques de la part du serveur. Le diagramme de la figure 2 illustre ce flot de données périodiques:

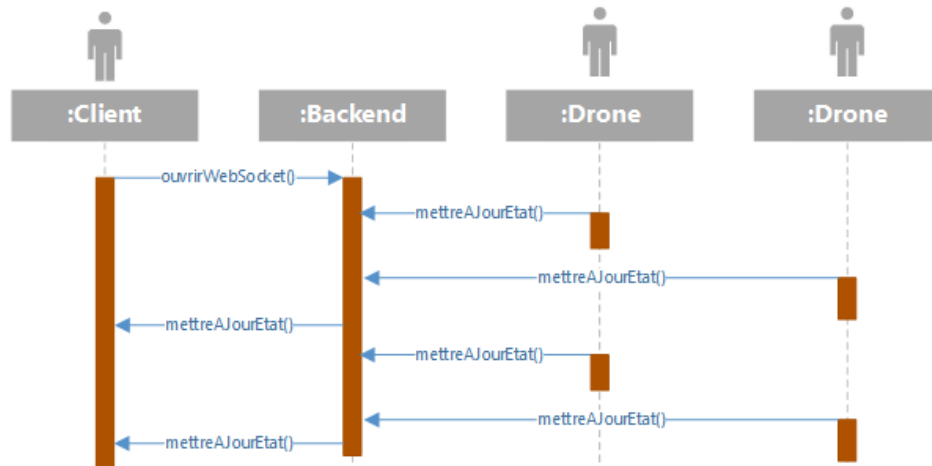


Figure 2. Diagramme de séquence de télémétrie des drones

Comme les drones envoient périodiquement leur état respectif à la station au sol, celle-ci n'a qu'à attendre de façon asynchrone cette mise à jour lorsqu'elle souhaite modifier l'état de l'essaim de drones. Le diagramme de séquence suivante démontre la façon dont l'utilisateur peut effectuer une modification de l'état des drones en démarrant ici l'exploration:

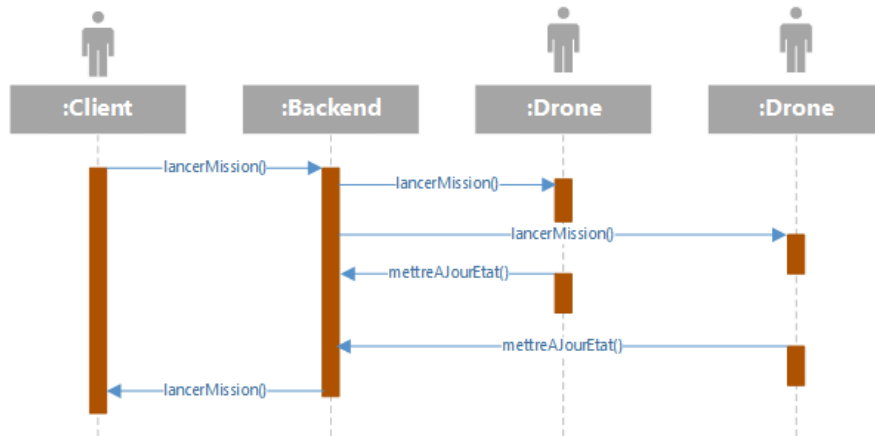


Figure 3. Diagramme de séquence du lancement d'une mission

Pour s'assurer de la persistance, des données traitées par le backend, les données relatives à l'exploration ainsi que celles traitées par la station au sol seront stockées dans une base de

données PostgreSQL. L'avantage d'utiliser une base de données relationnelle est qu'il est plus facile d'exploiter le lien entre les données comparativement à une base de données NoSQL. L'utilisation d'un ORM comme SQLAlchemy permet également de simplifier l'accès aux données.

Requis: R.M.2, R.F.8, R.F.11, R.F.12, R.F.17, R.F.18, R.C.1, R.C.2

3.3 Logiciel embarqué (Q4.5)

Dans le but de gérer efficacement le mouvement des robots aériens, nous allons utiliser l'API app-layer du micrologiciel Crazyflie [6], qui est une simple tâche FreeRTOS responsable du contrôle des états du drones. Une description architecturale est donnée à l'annexe 1. Dans ce qui suit, nous décrirons chaque composante de notre app layer :

Commander Unit

Unité chargée de produire les différents setpoints nécessaires au module commander. Ce module sera responsable de traduire des commandes de haut niveau (hover, takeoff, land...etc.) en setpoints précis (mode absolu ou vitesse avec les valeurs associées pour x, y, z et yaw), avant de les envoyer au *commander* du Crazyflie. [14]

Sensors Reader

Cette unité devra lire les différentes données provenant des senseurs du crazyflie. Pour les données de position (x, y, z), vitesse (vx, vy, vz) et angles (roll, pitch, yaw), elles seront extraites du state estimator EKF (extended kalman filter) [15]. En effet, avec notre *flow deck* et notre multiranger, c'est le choix du state estimator à privilégier. Pour les données de la batterie, on utilisera les voltages, et on traduira les données selon une table de conversion [16], car nous avons remarqué que le *battery level* n'est pas précis quand le drone est en vol. Effectivement, la relation entre le voltage des batteries et la capacité est non-linéaire. Pour obtenir ces données, on utilisera le Logging Framework qui est déjà dans le micrologiciel pour lire différentes données de l'état du drone [17]. Enfin, ces données seront acheminées au Controller Unit afin de les interpréter.

Communication Unit

Pour la communication avec la station au sol ainsi que les autres drones, on aura essentiellement besoin de trois composantes/scénarios que la Communication Unit devra implémenter :

Réception de données de la station : On utilisera le App Channel API directement. Pour chaque étape du cycle de contrôle, on écoutera sur le channel radio approprié si un paquet a été envoyé par la Crazy Radio Dongle. On ne bloquera pas le reste du traitement si un paquet n'a pas été reçu. [18]

Envoi de données à la station : On n'enverra pas de paquets nous-mêmes, car ce travail a déjà été implémenté dans le micrologiciel. Il existe déjà une tâche qui envoie le contenu de toute la table des variables du drone (TOC) par radio (port d'allocation 5 du CRTP) [18]. Du côté du drone on aura juste à rajouter d'autres logging groups (ex : état du drone) dans ce TOC afin qu'ils soient envoyés par la tâche crtp. [16]

Communication avec les autres drones : On utilisera la Peer 2 Peer API du micrologiciel. On pourra envoyer des messages en broadcast aux autres drones ainsi que les recevoir. [19]

Controller Unit

Il s'agit du cerveau de notre robot aérien. C'est dans ce module que toutes les données de l'environnement sont interprétées pour faire avancer les états du drone.

Une machine à états finis a été conçue pour mieux visualiser le comportement d'un robot aérien en fonction des événements et des commandes envoyées par l'opérateur. La conception de la machine à état fini se trouve à l'annexe 2. Lors du lancement du programme, l'état du robot est 'en attente'. Il passe vers l'état 'démarrage' lorsque la commande 'lancer la mission' est envoyée par l'opérateur. Le robot prend alors son envol et se stabilise. Les prochaines étapes correspondent aux états relatifs à l'algorithme d'exploration swarm gradient bug algorithm (SGBA) [4].

Dès que le robot est stabilisé dans l'air, le robot passe ensuite à l'état 'E3: rotation vers objectif', mais passe rapidement au prochain état. Si le robot détecte un obstacle à l'intérieur de son rayon de détection d'obstacle, celui-ci passe à l'état 'E4: suivre un mur' et se déplace le long de ce dernier, jusqu'à ce qu'un autre obstacle soit détecté, que sa cible soit atteignable et qu'il rencontre un coin de mur. Il retourne alors à l'état E3 dans ces conditions. À partir de E3, considérant que le robot reconnaît une cible atteignable, mais qu'aucun obstacle se situe à proximité, le robot passe à l'état 'E5: avancer vers l'avant' et ensuite à E4 lorsqu'un obstacle est détecté. Si à partir de E4 le robot détecte un autre robot et que celui-ci a une priorité une importante, l'état courant passe à 'E6: laisser passer' pour effectuer les manipulations pour éviter

une collision. Dans ce cas, le robot retourne à l'état E3 pour continuer l'exploration. L'algorithme d'exploration est représenté par l'annexe 2B. Lors de l'exploration, le robot peut à tout moment passer à l'état 'crashed' si un accident est survenu. De plus, si les batteries sont plus faibles que 30% ou que l'exploration est complétée, le robot transitionne à l'état 'E7: retour à la station'. Pour revenir à l'emplacement d'origine, les robots utilisent un mélange d'odométrie et un gradient, exprimé comme l'intensité de la force du signal (RSSI) reçu par la Crazyflie Radio PA observable vers la station de base. Lorsqu'il est à proximité de la station au sol, l'état passe à 'E8: atterrissage immédiat' et retourne à E1 lorsque le niveau du sol est atteint. Si la commande 'Terminer mission' est envoyée par l'opérateur, le robot passe immédiatement à l'état E8 pour ensuite atterrir.

Requis: R.L.1, R.L.3, R.F.1, R.F.2, R.F.4, R.F.5, R.F.6, R.F.7, R.F.13, R.F.19

3.4 Simulation (Q4.5)

La simulation se fera sur le logiciel ARGoS. Cette plateforme de simulation est extensible par l'entremise de plugins. La version d'ARGoS du MIST contient un plug-in du drone crazyflie. Ce sera ce robot que nous utiliserons dans la simulation. ARGoS doit attacher un contrôleur pour un robot. Le contrôleur est le cerveau du robot et est défini par une classe en C++. Le contrôleur sera composé de sensors et d'actuateurs. Un sensor est un dispositif en mode lecture qui retourne des données captées sur l'environnement, par exemple un capteur de lumière. Un actuator est un dispositif en mode écriture qui sert à modifier l'état du robot, par exemple un moteur. Puisque le micrologiciel des robots est écrit en C et que le logiciel ARGoS ne supporte que le C++, nous comptons porter l'algorithme d'exploration en deux versions, une en C et une autre en C++ pour ARGoS.

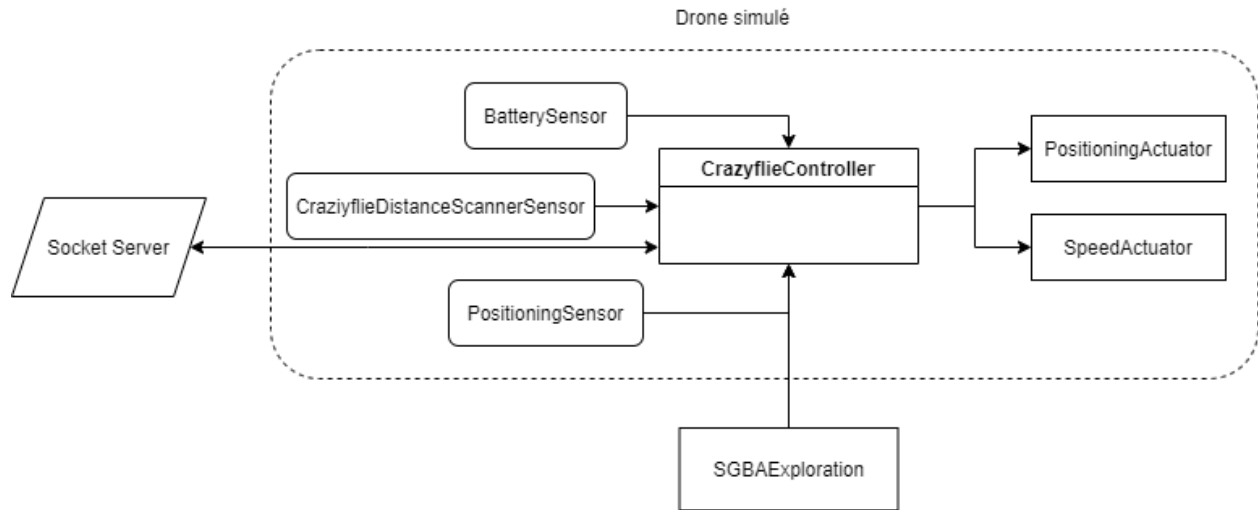


Figure 4: Diagramme général de la simulation ARGoS.

Nous avons considéré chaque drone simulé comme étant un serveur socket TCP. La station au sol agit comme client aux drones. Chaque socket est instancié dans son propre thread pour ne pas bloquer le thread principal qui roule la simulation.

Requis: R.F.2, R.F.4, R.F.5, R.F.6, R.C.3

3.5 Interface Utilisateur (Q4.5)

L'interface utilisateur représente la partie visible du projet, axé sur une expérience agréable, mais également facilitée par une bonne pratique du design. Nous voulons proposer une interface intuitive, logique, utilisable, et adaptée aux cibles définies. Elle représente une partie importante du projet, car c'est cette partie que le client va principalement utiliser.

L'interface utilisateur concerne beaucoup de requis que ce soit pour la partie physique des robots aériens ou pour la simulation. Les robots aériens physiques doivent pouvoir répondre à différentes commandes comme "Identifier", "Lancer la mission", "Terminer la mission". Il est aussi utile d'afficher le niveau de batterie des robots aériens. Il va aussi falloir afficher une carte globale de l'environnement affichant la position des drones et intégrer les données recueillies par les deux drones. Une mise à jour du logiciel de contrôle sur les drones doit être effectuée depuis l'interface utilisateur. Un prototype de l'interface utilisateur se trouve à l'annexe 3.

Tableau 2. Comparaison de React et Angular

	Angular	React
Modularité	Angular utilise le concept de composants	React utilise concept de composants
Niveau d'expérience dans l'équipe	Tout le monde a fait du angular en projet 2	Quelques membres de l'équipe possèdent une expérience minimum de react.
Documentation	Documentation très étoffée	Très peu de documentation officielle
Usage de librairies	Angular offre une solution complète. Vient avec un système de dépendances d'injection, une	React n'est qu'une bibliothèque, il faut choisir des third-party libraries pour avoir la
Apprentissage	Lourdeur et complexité du langage	Légèreté et flexibilité du langage
Language	Par défaut avec typescript	Écrit en Javascript, configuration du Typescript plus difficile que Angular
Modèle d'architecture logicielle	Implémente le Modèle-Vue-Contrôleur (MVC)	Seulement la vue. Requiert un effort additionnel pour implémenter le modèle et le contrôleur.

Finalement, à la vue des éléments du tableau 2, nous avons choisi Angular, car :

- Chaque membre de l'équipe GoExplore a déjà eu une expérience passée lors du projet intégrateur 2 avec Angular. Nous gagnons donc du temps puisque nous sommes déjà familiers avec l'outil ;
- L'équipe GoExplore préfère l'utilisation de typescript. Angular offre typescript par défaut, ce qui nous sauve du temps pour la configuration ;
- Angular offre une solution complète. Il permet de gérer les requêtes HTTP asynchrones facilement. Le modèle d'injection de dépendances d'Angular simplifie la modularité du code en services et le développement des fonctionnalités ;
- Angular possède une documentation très étoffée ce qui peut s'avérer très utile pour nos besoins.

Requis: R.L.2, R.F.3, R.F.6, R.F.7, R.F.9, R.F.10, R.F.12, R.F.13, R.F.14, R.F.18, R.C.4

3.6 Fonctionnement général (Q5.4)

Chaque image Docker sera poussée dans le *Container Registry* de *GitLab* pour réduire le temps d'attente d'un build. Pour faciliter l'exécution de notre système, notre équipe utilise l'outil *Docker Compose*, qui permet de lancer les conteneurs nécessaires. Nous prévoyons donc d'utiliser Docker Compose pour lancer l'ensemble du système qui inclura une image docker pour le client, le serveur, la base de données et la simulation ARGoS.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

L'équipe GoExplore est composée d'un coordonnateur de projet et de quatre développeurs. Chaque membre de l'équipe est employé à temps partiel avec des salaires de 145\$/h pour le coordonnateur de projet et de 130\$/h pour les développeurs / analystes. Afin que la solution proposée par GoExplore soit acceptable et réaliste, celle-ci ne doit pas dépasser 630 heures-personnes. Ces heures sont réparties parmi 105 points de requis et également d'autres tâches pour les remises de livrables.

Tableau 3. Estimation des coûts du projet

Équipe GoExplore	Taux horaire (\$/h)	Charge de travail (h)	Coût (\$)
Coordonnateur de projet	145	138.8	\$20,126
Développeur - Analyste	130	477.2	\$62,036
Estimation coût total projet:			\$82,162

On retrouve au tableau 3 le coût total estimé du projet. Le total de la charge de travail a été estimé à partir du tableau 4, représentant l'allocation de temps estimé individuellement.

4.2 Planification des tâches (Q11.2)

Tableau 4. Allocation du temps et répartition des tâches

Sprint	Tâches	Heures	Sous-total heures	Hugo	Pier-Luc	Samir	Wail	William
SDR	Planification du projet	4	12	✓	✓			
SDR	Décision de conception de l'architecture	8		✓	✓	✓	✓	✓
PDR	Mise en route du projet	14	115	✓	✓	✓	✓	✓
PDR	Documentation projet	20		✓	✓	✓	✓	✓
PDR	Installation Argos / Docker	3		✓	✓	✓	✓	✓
PDR	Description Exigences systèmes	20		✓	✓	✓	✓	✓
PDR	Création serveur	17						✓
PDR	Création interface Web	17		✓			✓	
PDR	Requis R.F.1	10				✓		
PDR	Requis R.F.2	12				✓	✓	
PDR	Démo vidéo des requis	2			✓			
CDR	Documentation projet	20	210		✓			
CDR	Préparation présentation	12			✓			
CDR	Requis R.F.3	8		✓				
CDR	Requis R.F.4	15				✓	✓	
CDR	Requis R.F.5	26				✓	✓	
CDR	Requis R.F.10	12		✓				✓
CDR	Requis R.C.1	20		✓				✓
CDR	Prototype requis R.F.8	30		✓			✓	
CDR	Prototype requis R.C.3	6					✓	
CDR	Prototype requis R.Q.1	20			✓			✓
CDR	Prototype requis R.Q.2	20			✓			✓
CDR	Tests automatiques	21		✓	✓	✓	✓	✓
RR	Version finale documentation	16	279	✓	✓	✓	✓	✓
RR	Préparation présentation finale	10		✓	✓	✓	✓	✓
RR	Présentation finale	2		✓	✓	✓	✓	✓
RR	Finalisation Gitlab	6						✓
RR	Finalisation README	6			✓			
RR	Rédaction documentation test	20			✓			
RR	Requis R.F.6	30				✓	✓	
RR	Requis R.F.7	9			✓			
RR	Requis R.F.9	10		✓		✓	✓	
RR	Requis R.F.11	12						✓
RR	Requis R.F.12	4						✓
RR	Requis R.F.13	10			✓	✓		
RR	Requis R.F.14	16				✓		
RR	Requis R.F.17	16			✓			✓
RR	Requis R.F.18	20		✓				✓
RR	Requis R.F.19	20				✓		
RR	Requis R.C.2	16					✓	✓
RR	Requis R.C.4	20		✓				
RR	Finalisation requis R.F.8	16		✓			✓	
RR	Finalisation requis R.Q.1	10			✓			
RR	Finalisation requis R.Q.2	10		✓	✓	✓	✓	✓
Total:		616	Total tâches:	20	22	19	20	21
			Total heures:	115.6	138.8	120.6	115.1	125.8
			Total % heures:	19%	23%	20%	19%	20%

Le tableau 4 représente l'allocation du temps pour chaque tâche, ainsi que la répartition de l'effort entre les membres de l'équipe. Lorsqu'une tâche est accomplie par plus d'un membre de l'équipe, le nombre d'heures de la tâche est divisé par le nombre de ressources. Un total de 616 heures est estimé pour le projet. La répartition des heures est homogène, ne variant que de 4% à travers l'équipe.

4.3 Calendrier de projet (Q11.2)

Pour que l'équipe GoExplore soit la plus productive et efficace possible, il est nécessaire de dresser un calendrier de projet. Un diagramme de Gantt a été conçu pour planifier la charge de travail en fonction des tâches à réaliser et des livrables. Le diagramme se retrouve à l'annexe 4. Les pourcentages reflètent la proportion par rapport à la totalité du projet.

4.4 Ressources humaines du projet (Q11.2)

Comme nous l'avons mentionné précédemment, l'équipe GoExplore est composée d'un gestionnaire de projet et de quatre développeurs-analystes. Le tableau ci-dessous représente le rôle de chaque individu de l'équipe, ainsi que leur qualificatif à effectuer leur rôle. Les différentes parties du système aérien d'exploration étant étroitement liées, il est fortement probable que les membres de l'équipe GoExplore soient amenés à travailler sur des tâches qui sortent de leur affection première.

Tableau 5. Rôle des membres de l'équipe

Membre de l'équipe	Rôles	Qualifications / Expériences
Sami Sadfa	Développeur - Logiciel embarqué	Stage en développement de systèmes embarqués
William Trépanier	Développeur - Station au sol	Expérience d'utilisation de Docker et implémentation de serveur
Wail Ayad	Développeur - Simulation Argos	Compétence de développement en C++
Hugo Lachize-Rey	Développeur - Interface Web	Développement Web notamment avec la framework Angular
Pier-Luc Tanguay	Responsable de la gestion de projet	Expérience de gestion pour des projets antérieurs de grosseur moyenne

Le gestionnaire de projet devra faire preuve d'habiletés techniques, mais aussi d'habiletés personnelles. Par exemple, il devra faire preuve d'intégrité, d'assurance, de compréhension organisationnelle ou encore de leadership. Il devra aussi être capable de maîtriser les outils techniques mis à dispositions pour livrer le meilleur produit possible comme une compréhension des différents langages de programmation ou protocoles utilisés.

Les développeurs-analystes devront démontrer leurs habiletés techniques lors de la résolution des tâches qui leur sont confiées. En effet, ils devront démontrer leurs connaissances dans

l'utilisation de Docker, dans la maîtrise des différents langages de programmation utilisés ou encore l'utilisation des différents protocoles comme HTTP ou TCP/IP. En termes d'habiletés personnelles, les développeurs-analystes devront faire preuve d'autonomie, de prise d'initiative et d'esprit d'équipe.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Tout au long de notre projet, tous les biens livrables doivent être soumis à un processus de révision afin de s'assurer que notre solution soit conforme aux attentes. L'équipe GoExplore a mis en place, dès le début du projet, des lignes directrices et des mécanismes pour remettre une solution (livrable et code) optimale :

- Le respect de la planification des tâches et du calendrier de projet pour respecter les échéanciers et les requis minimums.
- Lignes directrices de développement logiciel pour s'assurer du respect des bonnes pratiques et recommandations. Par exemple, utiliser des noms représentatifs pour les noms de variables ou de fonctions, avoir une orthographe uniforme surtout dans le choix de la langue et dans le respect de cette dernière. Aussi, chaque fonctionnalité développée devra avoir un test correspondant pour s'assurer de son bon fonctionnement.
- Des mécanismes de révisions ont aussi été mis en place pour vérifier la conformité de notre solution. Chaque membre de l'équipe GoExplore a été assigné à la tâche pour laquelle il semble le plus compétent en se basant sur ses expériences passées et ses connaissances.
- Chaque merge request doit être approuvée par au moins un autre membre de l'équipe ou encore des sessions de revue de codes auront lieu tout au long du déroulement du projet. L'intégration continue assurera également que l'analyse statique et que les tests passent intégralement. Une révision collective de la solution proposée aura lieu quelques jours avant les différentes remises. Elle consistera principalement à passer en revue la documentation du projet, le montage vidéo des différents requis et finalement de la révision de code prenant en compte les fonctionnalités attendues et l'assurance qualité.

5.2 Gestion de risque (Q11.3)

Comme dans tout projet, il est nécessaire de bien réagir lorsqu'un ou plusieurs risques se présentent face à nous. En effet, l'équipe GoExplore devra faire preuve d'habileté et d'altruisme afin de répondre au mieux aux différents imprévus. En début de projet, nous avons pris la peine de lister quelques risques afin d'éviter au maximum l'élément de surprise en cas de situation anormale :

- Le principal risque de ce projet est celui lié aux échéances. En effet, il est possible de prendre du retard facilement en cas de manque de communication au sein de notre équipe. Afin d'éviter au maximum ce risque, l'équipe GoExplore a mis en place un serveur Discord où auront lieu des réunions hebdomadaires afin de rester au courant de l'avancée des différents membres dans les tâches qui leur sont confiées. Aussi, les scrums serviront à demander de l'aide à autrui en cas de blocage d'un ou plusieurs membres de l'équipe sur une tâche. Ce risque est à prendre au sérieux, car du retard dans les échéances pourra entraîner une augmentation des coûts et ainsi rendre le client insatisfait.
- Un autre risque est de développer une solution ne répondant pas aux critères désirés pour un requis spécifique ou bien de manière plus générale. Certaines directives peuvent être mal interprétées par un ou plusieurs membres de l'équipe GoExplore et ainsi entraîner un résultat différent que celui voulu. Pour remédier à ce risque, il est nécessaire de clarifier en équipe toutes interrogations ou incompréhensions avec le client. Il ne faut pas négliger ce risque, car il pourrait entraîner une accumulation de tâches résultant d'un retard, rendre le client insatisfait, ou encore une mauvaise cohésion d'équipe.
- Il est aussi possible qu'un membre de l'équipe se blesse lors d'une manipulation avec les robots aériens et que cela l'handicape pour le reste du projet voir le force à abandonner. Ce risque est moins probable, mais toujours présent. Il est donc important de porter des lunettes de sécurité. De plus, en ce temps de pandémie, un des membres de l'équipe pourrait être positif à la COVID-19, développer des symptômes et ainsi être forcé à travailler à distance. Pour pallier cela, il est nécessaire d'avoir une bonne communication entre chaque membre et d'être capable de s'adapter très rapidement.

5.3 Tests (Q4.4)

Afin de livrer un produit de qualité et étant conforme à tous les requis nécessaires, il faut mettre en place différentes catégories de tests.

5.3.1 Test unitaire

Tests qui ont pour objectifs de tester une petite partie du code, dans le but de vérifier si elle se comporte de façon attendue. En isolant la partie de code à tester du reste du code, on teste sa validité de façon déterministe, c'est-à-dire qu'elle retournera toujours le même résultat. Nous les utiliserons pour tester les fonctions tout au long du projet.

5.3.2 Test d'intégration

Tests de boîte noire qui ont pour objectif de tester l'interaction entre des unités. Ils seront utilisés pour tester la communication entre les différents composants de système tels que les robots aériens, la simulation ARGoS, le serveur et le service web. Ces tests seront effectués lorsque des changements seront apportés ou bien qu'un nouvel élément qui communique a été ajouté.

5.3.3 Test de régression

Il s'agit de l'ensemble de tests d'un programme préalablement testé, après une modification, pour s'assurer que des défauts n'ont pas été introduits ou découverts dans des parties non modifiées du logiciel.

5.3.4 Test d'acceptation

Il permet de vérifier qu'un système répond à toutes ses spécifications. Il est exécuté avant la fin d'un livrable ou lorsqu'une tâche importante a été finalisée. Il correspond au dernier niveau de tests avant la livraison.

5.4 Gestion de configuration (Q4)

Nous allons utiliser Git comme système de contrôle de version en suivant le modèle de travail gitflow. Ce modèle sépare le code source en deux branches distinctes, *main* et *develop*. Lors du développement, les membres de l'équipe doivent coder les fonctionnalités dans branches *features* qui dévie de *develop* [20]. Une fois la fonctionnalité finie, une *merge request* doit être publiée. Cette *merge request* devra passer par un pipeline CI/CD géré par Gitlab.

Chaque sous-projet (interface, station sol, simulation, embarqué) aura son propre entrepôt Git. Dans chaque entrepôt se trouvera un fichier README contenant la documentation et un dossier tests excluant celui de l'interface qui suivra les standards de test d'Angular.

Nous utiliserons principalement l'éditeur de texte Visual Studio Code (VS Code) comme IDE. À la racine de chaque entrepôt Git se trouvera le dossier de configuration de VS Code .vscode adéquat à chaque projet.

Nous établirons aussi une politique d'approbations pour les merge request. Il faudrait au minimum 2 personnes pour approuver une merge request excluant l'auteur.

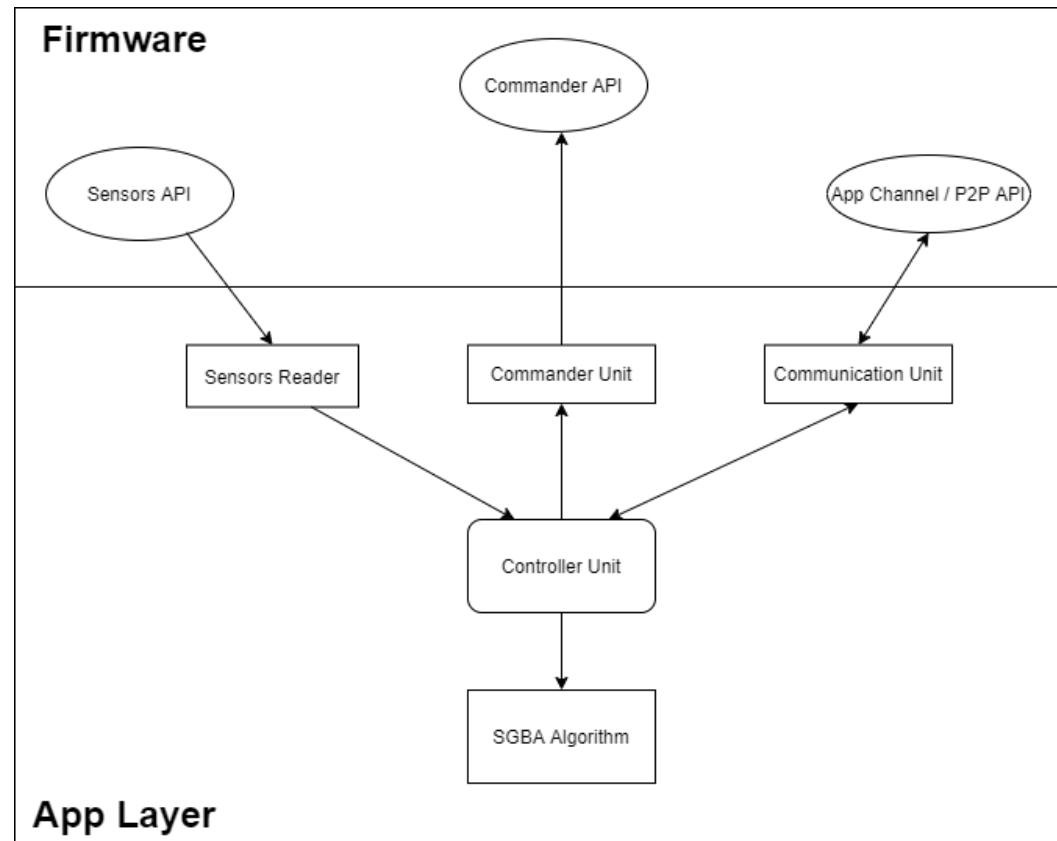
6. Références

- [1] Beltrame, G. (2021). Système aérien d'exploration. [En ligne]. Disponible : <https://moodle.polymtl.ca/mod/resource/view.php?id=145345>
- [2] Kimberly McGuire, C. D. W., K. Tuyls, H. J. Kappen, G. C. H. E. de Croon. (2019). Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. Disponible : <https://www.science.org/doi/10.1126/scirobotics.aaw9710>
- [3] BitCraze. (2021). Crazyradio PA 2.4 GHz USB dongle. [En ligne]. Disponible : <https://store.bitcraze.io/products/crazyradio-pa>
- [4] McGuire, K. (2019). Enabling Swarm Exploration. [En ligne]. Disponible : <https://www.bitcraze.io/2019/10/enabling-swarm-exploration/>
- [5] AcqNotes. (2021). System Design Review (SDR). [En ligne]. Disponible : <https://acqnotes.com/acqnote/careerfields/system-design-review-sdr>
- [6] McGuire, K. (2021). SGBA_CF2_App_layer. [En ligne]. Disponible : https://github.com/tudelft/SGBA_CF2_App_layer
- [7] Collin, J. (2021). Cycle acquisition et entente contractuelle. [En ligne]. Disponible : <https://moodle.polymtl.ca/mod/resource/view.php?id=276640>
- [8] FastAPI. (2021). FastAPI. [En ligne]. Disponible : <https://fastapi.tiangolo.com>
- [9] Pydantic. (2021). Overview. [En ligne]. Disponible : <https://pydantic-docs.helpmanual.io/>
- [10] asyncio. (2021). asyncio. [En ligne]. Disponible : <https://docs.python.org/3/library/asyncio.html>
- [11] Refactoring-Guru (2021). Chaîne de responsabilité. [En ligne]. Disponible : <https://refactoring.guru/fr/design-patterns/chain-of-responsibility>
- [12] Docker. (2021). Dockerfile reference. [En ligne]. Disponible : <https://docs.docker.com/engine/reference/builder/>
- [13] TechEmpower. (2021). Web Framework Benchmarks. [En ligne]. Disponible : <https://www.techempower.com/benchmarks/#section=data-r20&hw=ph&test=composite&l=zijzen-sf&a=2>
- [14] BitCraze. (2021). The Commander Module. [En ligne]. Disponible : https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.06/functional-areas/c-commanders_setpoints/

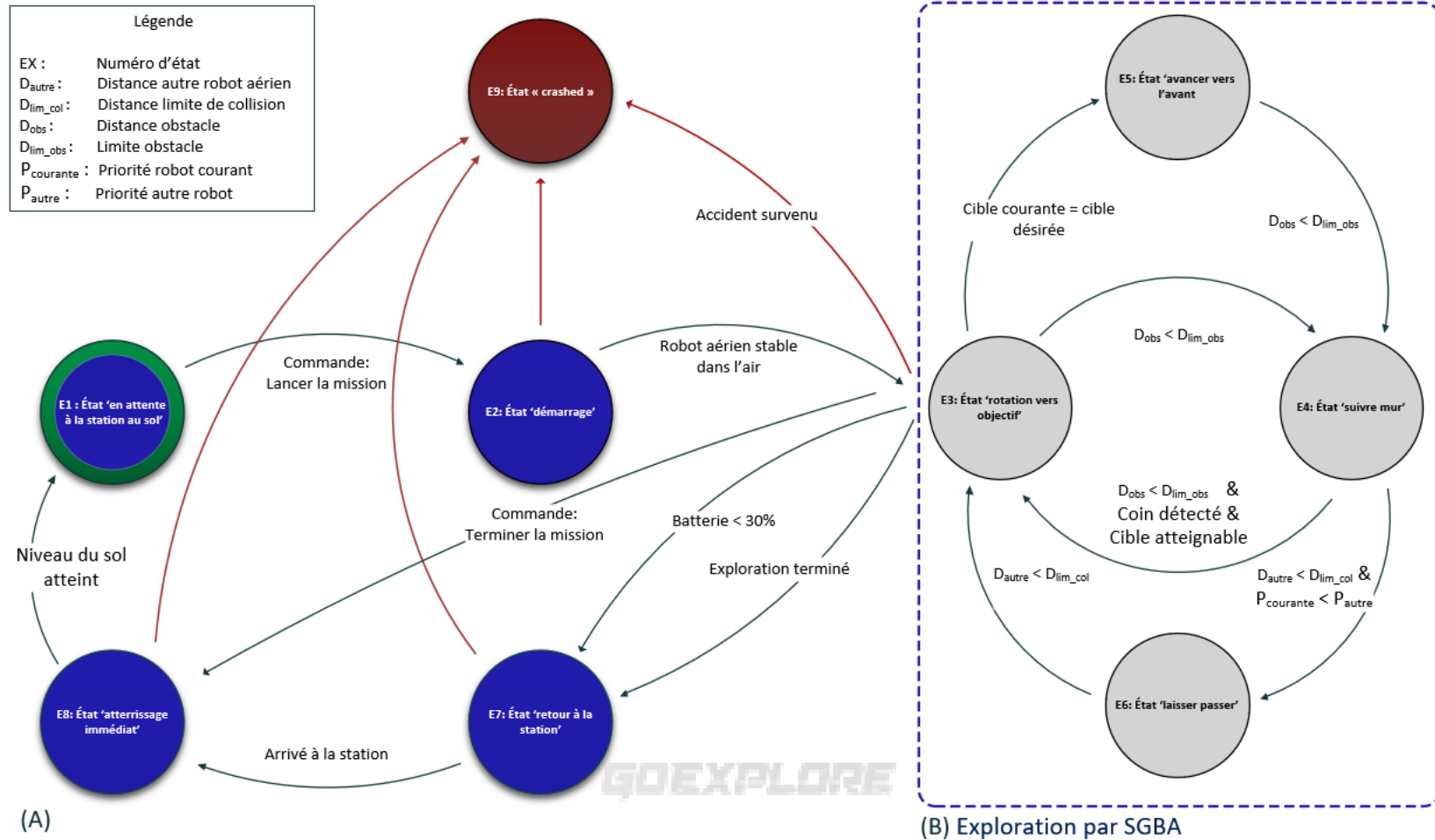
- [15] BitCraze. (2021). State estimators. [En ligne]. Disponible : https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.06/functional-areas/state_estimators/
- [16] Ampow. (2021). Lipo Voltage Chart: Show the Relationship of Voltage and Capacity. [En ligne]. Disponible : <https://blog.ampow.com/lipo-voltage-chart/>
- [17] BitCraze. (2021). Logging and parameter frameworks. [En ligne]. Disponible : <https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.06/userguides/logparam/>
- [18] BitCraze. (2021). Communication with the Crazyflie. [En ligne]. Disponible : <https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.06/functional-areas/crtp/>
- [19] BitCraze. (2021). Peer to Peer API. [En ligne]. Disponible : https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.06/functional-areas/p2p_api/
- [20] Vincent Driessen. (2010). A successful Git branching model. [En ligne]. Disponible : <https://nvie.com/posts/a-successful-git-branching-model/>

7. Annexes

Annexe 1 : Conception architectural du logiciel embarqué



Annexe 2 : Machine à états finis implémentée par le logiciel embarqué



Annexe 3: Exemple d'interface utilisateur de l'opérateur

Robots aériens

Simulation

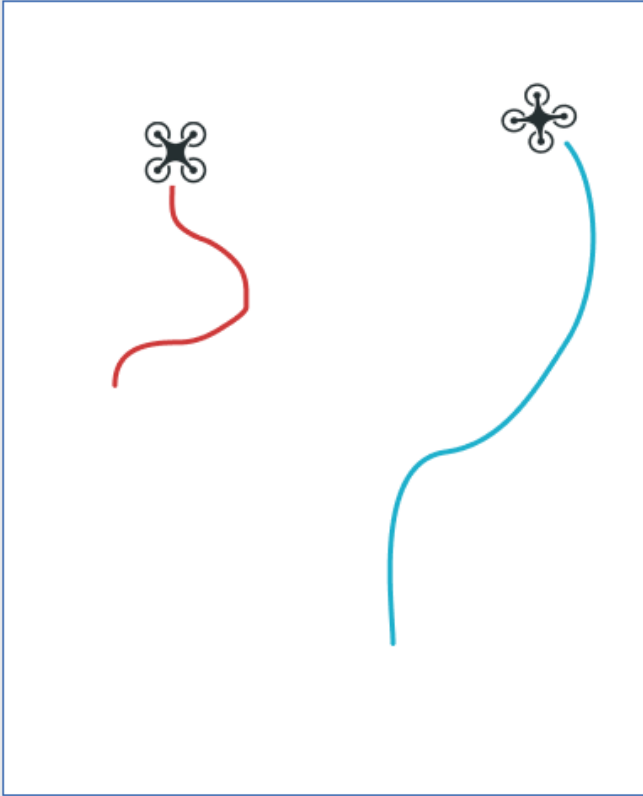
Lancer mission

Terminer mission

Retour à la base

Décollage

Mise à jour logiciel



Carte de l'environnement

GOEXPLORE

Missions

Date :


Heure :



Temps de vol :

Nombre de drones :



Distance totale parcourue :

Type :

 Ouvrir carte générée lors d'une mission précédente

Drone 2:
 État : CRASH

Position :
Orientation :


Identifier

Drone 1:
 État : EN MISSION

Position :
Orientation :

Identifier

GOEXPLORE

- 30 -

POLYTECHNIQUE
MONTREAL 

Annexe 4: Diagramme de Gantt pour la gestion du projet

