Dernière modification: 19 avril 2021

INF3995: Projet de conception d'un système informatique
Hiver 2021
Rapport RR

Hivexplore - Système aérien minimal pour exploration

Rapport final de projet

Équipe No. 102

Misha Krieger-Raynauld

Nathanaël Beaudoin-Dion

Rose Barmani

Samer Massaad

Simon Gauvin

Yasmine Moumou

Table des matières

Table des figures

1 Objectifs du projet

Le but principal du projet est de réaliser une preuve de concept pour l'Agence Spatiale. Cette preuve doit démontrer que l'utilisation de drones équipés de capteurs rudimentaires pour une mission d'exploration d'une pièce d'au plus 100 m^2 est une solution fonctionnelle. Pour ce faire, un prototype de niveau de maturité 4 (NMS 4) de la solution a dû être réalisé.

Ce prototype consiste en une station au sol avec une interface opérateur ainsi qu'une partie embarquée, soit un essaim d'un nombre arbitraire de drones explorateurs. La station au sol doit permettre à un opérateur de monitorer l'état, la vitesse et la batterie des drones. De plus, elle doit lui permettre d'envoyer des commandes pour débuter ou mettre fin à une mission d'exploration et mettre à jour le système à bord des drones. Quant aux drones, ceux-ci doivent pouvoir explorer une pièce d'un bâtiment de dimension moyenne, éviter des obstacles incluant les autres drones, communiquer les uns avec les autres et retourner à la base. La mission d'exploration a pour objectif la collecte des données sur les lieux par les capteurs des drones. Ainsi, à l'aide des données récoltées, une carte de l'endroit exploré est générée par la station au sol et affichée à l'opérateur.

Plusieurs requis et contraintes nous ont été imposés pour encadrer le projet. Une fois la mission commencée, soit à la réception de la commande de début de mission, les drones doivent être autonomes et parcourir le périmètre de la pièce à explorer sans recevoir d'autres commandes de la station au sol (R.F.1). Au niveau des drones, au moins un d'entre eux doit être en tout temps en communication avec la station au sol (R.F.5) avec comme seul moyen de communication la Crazyradio PA connectée par USB (R.M.2, R.M.4). Pour les programmer et pour la communication pair-à-pair (P2P), l'API de Bitcraze doit être utilisée (R.L.1, R.L.2). Par ailleurs, l'utilisation d'un serveur pour la communication entre les drones est interdite (R.L.2). En mission, un minimum de deux drones peut être supposé, mais un nombre arbitraire de drones supplémentaires doit pouvoir être soutenu par le système (R.F.2). Le code embarqué des drones peut être mis à jour par l'API de Bitcraze, mais seulement s'ils sont au sol (R.F.4.1). Lors d'un retour à la base, les drones doivent être à un mètre de la station au sol (R.F.4.2). Toutefois, si leur niveau de batterie est inférieur à 30%, ceux-ci ne peuvent pas débuter de nouvelle mission et doivent retourner à la base s'ils sont sinon déjà en mission (R.F.4.3). La distance des drones à la station au sol doit obligatoirement être déterminée par la puissance du signal RSSI de la Crazyradio tel que reçue par les drones (R.L.3). Pour ce qui est de la conception, il est obligatoire d'expérimenter avec des simulations ARGoS avant de tester avec les drones (R.C.1). Le prototype doit nécessairement être fait avec le « STEM Ranging bundle » sur deux drones Bitcraze Crazyflie 2.1 (R.M.1) et seulement le *Multi-ranger deck* [1] et le *Flow deck v2* [2] peuvent y être installés (R.M.3). Enfin, du point de vue de l'utilisation, l'interface utilisateur doit pouvoir être visualisée sur divers appareils et la mise à jour des informations doit se produire à une fréquence minimale de 1 Hz (R.F.5). Il doit, d'autre part, être possible de démarrer le système et sa simulation avec ARGoS en ne faisant appel qu'à une seule commande (R.C.4).

Pour respecter le contrat accepté suite à la réponse de l'appel d'offres, le prototype final remis à l'Agence devait implémenter tous les requis, à défaut de quoi chaque requis non respecté devait être justifié dans le rapport du CDR. Dans notre cas, il a été établi au CDR et par la suite accepté par l'Agence que les requis R.C.2 (la présence de tests unitaires) et R.F.4.1 (la mise à jour du code embarqué à même l'interface utilisant l'API de Bitcraze) seraient absents du prototype final. Cette décision a été prise afin de permettre à l'équipe de respecter un budget limité et fixe tout en réalisant les fonctionnalités jugées plus importantes par l'Agence. Puisque les justifications pour ces deux requis non implémentés ont été acceptées par l'Agence, la solution présentée respecte à présent l'entièreté des conditions convenues.

2 Description du système

2.1 Architecture logicielle générale

L'architecture logicielle de la solution proposée comprend quatre composantes principales et une logique de machine à états commune. Celles-ci sont représentées à la figure 1 par différentes couleurs pour les distinguer.

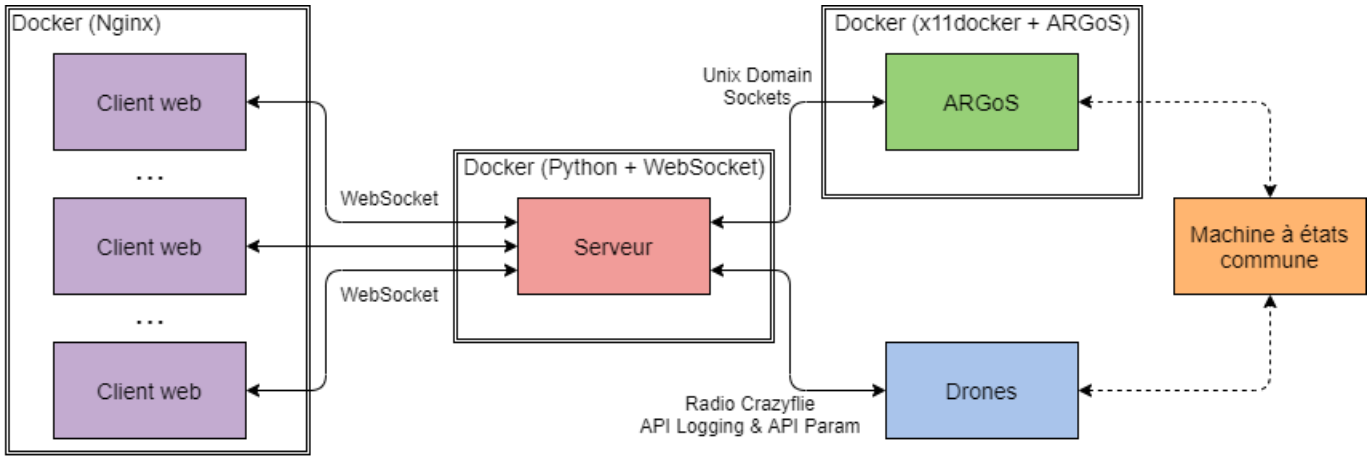

Figure 1 – Diagramme de l'architecture logicielle générale

La première composante est un client Web développé avec Vue.js 3. L'équipe a choisi de développer un client Web pour des raisons de portabilité. En effet, il doit être possible de visualiser l'interface utilisateur sur différents appareils (R.L.4). Notre application Web permet la réutilisation du même client pour un ordinateur et pour un appareil mobile puisque celle-ci a été conçue avec des composantes réactives. La version utilisée est la plus récente de Vue.js, soit Vue.js 3, sortie en septembre 2020. Celle-ci offre un meilleur support Typescript, langue que nous avons choisie pour éviter des erreurs reliées à une mauvaise cohérence des types.

La deuxième composante logicielle correspond au programme en C embarqué sur les drones. Les drones communiquent avec le serveur à travers la Crazyradio. Ils explorent un environnement et recueillent des données avec leurs capteurs. Parallèlement, ils envoient ces données au serveur et reçoivent les commandes reliées aux étapes de la mission de celui-ci.

La troisième composante est la simulation des drones avec ARGoS. Cette composante logicielle permet de valider les fonctionnalités dans un environnement simulé avant de les implémenter sur les drones. Cette composante simule aussi la communication avec le serveur en l'absence des drones et de la Crazyradio. Elle lui envoie des données sur l'environnement simulé et reçoit ses commandes.

La dernière composante logicielle est le serveur. Celui-ci utilise soit la Crazyradio afin de communiquer avec les drones, soit des *sockets* Unix pour communiquer avec la simulation ARGoS. Le serveur permet la connexion simultanée de plusieurs clients pour la visualisation des données et le contrôle de la mission. Le serveur est en charge de recevoir et de traiter les données des drones et de les envoyer aux différents clients Web connectés. Il a aussi la responsabilité de recevoir les commandes provenant des clients et de les transmettre aux drones.

Au niveau de la communication, le serveur et le client s'envoient des informations et des commandes par l'entremise d'une communication WebSocket [9]. Ce protocole permet au client de recevoir les

informations du serveur dès qu'elles lui sont disponibles : le serveur peut communiquer par messages envoyés à des fréquences différentes en fonction de leur importance. Le client n'a donc pas à ce soucier de demander les données au serveur à chaque seconde. Un système événementiel par *callbacks* a été utilisé, offrant une abstraction simple afin que les fonctions concernées par un évènement soient automatiquement appelées lors de la réception dudit évènement encodé en JSON.

Pour sa part, la communication entre le serveur et la simulation ARGoS est gérée par des *Unix Domain Sockets*. Cette méthode permet une communication inter-processus bidirectionnelle élégante et simple. La raison principale du choix de cette technologie est sa simplicité d'utilisation et sa polyvalence. Pour des raisons de symétrie et de simplicité, nous avons également pris la décision d'utiliser un système de *callbacks* utilisant des messages JSON entre le serveur et la simulation ARGoS.

Au niveau du conteneur Docker du client, Nginx est utilisé pour répondre aux requêtes et servir la page client. Par la suite, un client chargeant la page servie par Nginx peut communiquer avec le serveur par l'entremise d'un autre port exposant le serveur WebSocket. Il est ainsi possible d'avoir plusieurs clients sur différents appareils qui se connectent au même serveur sur le même réseau LAN. Le conteneur du serveur, quant à lui, peut soit communiquer avec le simulateur ARGoS, soit envoyer ses commandes avec le périphérique Crazyradio pour interfacer avec les drones. Finalement, étant donné qu'ARGoS nécessite une interface graphique, un conteneur démarré à l'aide de *x11docker* est utilisé afin de pouvoir transmettre l'interface graphique peu importe les pilotes graphiques installés sur l'ordinateur hôte.

Enfin, Docker Compose est utilisé afin d'orchestrer tous les conteneurs du client Web, le serveur et ARGoS. Afin de supporter l'application graphique ARGoS, un script Bash de démarrage a été écrit pour initialiser les variables d'environnement pour le *forwarding* X11 et pour ensuite exécuter la commande `docker-compose`. Le script de démarrage a aussi été conçu pour améliorer l'expérience utilisateur avec une option `--help`. Étant donné que le système peut soit être exécuté avec ARGoS, soit avec les drones Crazyflie, le déploiement des conteneurs doit supporter les deux modes. Ainsi, plusieurs fichiers de surcharge sont utilisés en plus de la configuration commune dans `docker-compose.yml`. Pour récupérer les journaux après avoir fermé les conteneurs, le système a recours à un *named volume* pour y conserver les fichiers de façon persistante. Un autre volume nommé est employé pour tenir le fichier *socket* nécessaire à la communication via *Unix Domain Sockets* lorsque le conteneur ARGoS est lancé.

2.2 Logiciel embarqué (Q4.5)

En se tournant plus particulièrement vers l'architecture spécifique du logiciel embarqué, on peut voir à la figure 2 que trois des quatre composantes sont concernées, soit le serveur, les drones et la simulation ARGoS, ces deux dernières implémentant une machine à états commune. Une ligne pointillée représente la démarcation entre la section ARGoS et la section des drones. Ainsi, si le contexte est celui de la simulation, il faut faire abstraction des boîtes bleues à droite de la ligne, représentant les drones. À l'inverse, si l'on souhaite se pencher sur l'architecture de l'implémentation réelle sur les drones, il faut ignorer les boites vertes à gauche de la ligne, représentant ARGoS.

Il va sans dire que la simulation ARGoS doit émuler le comportement des drones de la façon la plus réaliste possible. Ceci décrit une des grandes difficultés de la simulation et une contrainte existante dans le domaine de la robotique. Afin de réduire le plus possible les différences entre la simulation et le comportement réel des drones, notre solution prône l'utilisation d'une logique commune pour le déplacement, la communication et la prise de décisions dans les algorithmes d'exploration et du

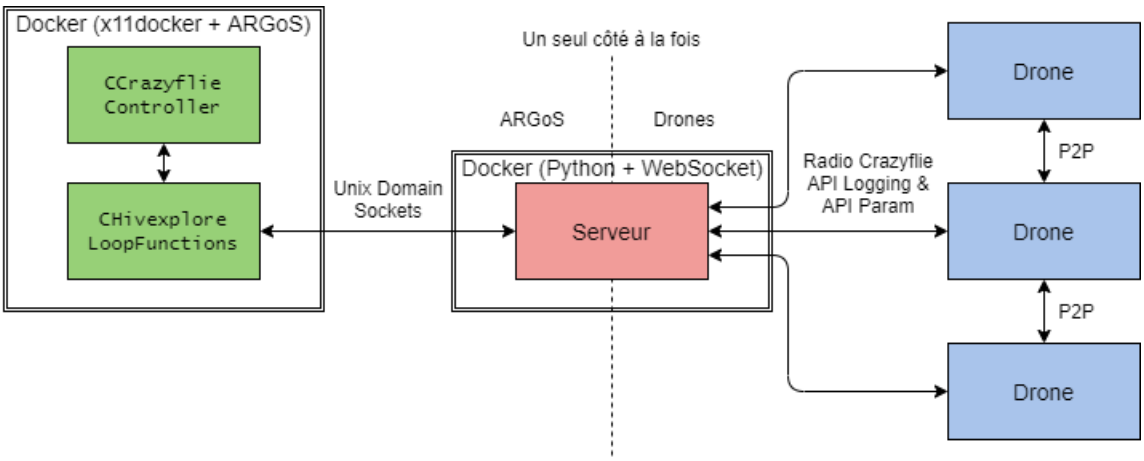

Figure 2 – Diagramme de l'architecture embarquée

retour à la base.

En ce qui concerne la communication entre les drones et le serveur, les drones envoient à chaque seconde leur état à la station au sol, où, à l'aide de la radio, il est possible de recevoir les informations désirées de l'essaim en entier. Pour ce qui est de l'envoi des commandes aux drones, un envoi de type *broadcast* est utilisé. Ainsi, lorsqu'une commande est envoyée - comme « Start mission » - celle-ci est acheminée à tous les drones. Les drones peuvent donc tous commencer la mission de façon autonome et simultanée. Du côté du code embarqué, ces réceptions de *logs* et envois de commandes sont implémentés à l'aide du cadriciel de *logging* de Crazyflie et du cadriciel de *parameter*, configurés avec des macros `LOG_ADD` et `PARAM_ADD`, respectivement (R.L.1). Enfin, la mise à jour des informations des drones sur le client est gérée par l'entremise du serveur à l'aide de WebSockets.

Afin de permettre une communication entre le serveur et la simulation ARGoS, les *loop functions* d'ARGoS ont été employées. Celles-ci permettent d'y ajouter le code nécessaire pour la communication par *sockets* Unix et rendre disponible un point d'entrée dans la simulation pour le serveur. C'est là que se trouve le code qui agit en tant qu'intermédiaire pour la réception et l'envoi de messages entre le serveur et le contrôleur des drones d'ARGoS, simulant ainsi le comportement des cadriciels d'acquisition de données à chaque seconde et d'envoi de commandes disponibles sur le code embarqué.

Ensuite, pour accomplir l'exploration de l'environnement, la communication *peer-to-peer* (P2P) des drones est mise à profit afin de faciliter la coordination de ceux-ci et éviter l'utilisation du serveur comme relais (R.L.2) ; autrement, la dépendance à un système centralisé aurait pu limiter ce que peuvent accomplir une flotte de drones autonomes [3]. Du côté des drones, cette communication est implémentée en utilisant l'API *peer-to-peer* des Crazyflies. Pour la simulation avec ARGoS, les capteurs et actuateurs *Range & bearing* permettent un comportement semblable à la communication P2P des Crazyflies. La coordination avec P2P est puissante puisqu'elle délègue à l'essaim de drones une grand flexibilité au niveau des différents algorithmes. Puisque les drones, à eux seuls, ne sont pas capable de déterminer leur position dans un référentiel commun, il est nécessaire de leur envoyer leur position initiale relative à la base au début de chaque mission. Cet ajout par rapport à la proposition telle que présentée dans le CDR permet aux drones d'avoir un point de référence commun et de diffuser leur position avec beaucoup plus de précision. Cette information est enfin mise à profit dans l'algorithme d'exploration, l'algorithme d'évitement des autres drones ainsi que dans la génération de la carte à partir des données des drones.

Étant donné que les drones ont accès au *Multi-ranger deck* et au *Flow deck v2*, la combinaison de ces modules permet d'avoir la distance dans toutes les directions (avant/gauche/arrière/droite/haut/bas) ainsi qu'un système de positionnement relatif basé sur les variations de positions avec un filtre de Kalman [8]. Les mesures des distances ont pu être simulées avec ARGoS grâce à un capteur de distances. Cependant, l'implémentation de ce dernier ne permettait d'obtenir les distances que dans certaines directions (avant/gauche/arrière/droite). Pour combler le capteur du bas qui est important pour savoir l'altitude du drone, la position absolue a été utilisée comme remplacement dans la simulation.

Les données provenant des capteurs de distances permettent aux drones d'éviter les obstacles durant la mission. Envoyées au serveur, ces données, ainsi que les positions et les orientations des drones permettent de générer la carte de la pièce explorée par les drones, que ce soit les Crazyflies ou ceux simulés sur ARGoS.

Par ailleurs, pour ce qui est de la mesure de la puissance de signal (RSSI) reçue par la radio, cette donnée est facilement accessible grâce au cadriciel de *logging* de Crazyflie. Or, cette donnée n'est pas accessible dans la simulation ARGoS ; elle a donc été simulée en calculant la distance euclidienne entre les drones et la station au sol. Il est important de noter que la mesure du RSSI contient énormément de bruit avec les Crazyflies, et que ce bruit est difficilement simulable avec ARGoS.

Une différence à noter entre les Crazyflies et la simulation ARGoS est la logique de contrôle. Du côté des drones, la logique est basée sur les vitesses. L'actuateur permettant de contrôler le drone en utilisant des vitesses n'étant pas implémenté sur ARGoS, il a fallu implémenter la logique de contrôle sur ARGoS en se servant de positions.

Un dernier aspect important de l'architecture est l'utilisation d'un code identique pour tous les drones, puisque cette solution est plus facilement maintenable et extensible avec un grand nombre de drones (R.F.2).

Pour ce qui est du déroulement de la mission, les drones et la simulation ARGoS suivent une même machine à états, présentée à la figure 3.

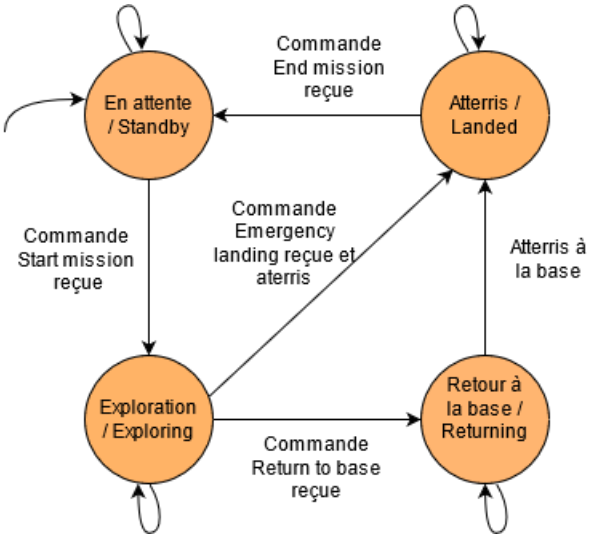

Figure 3 – Machine à états du déroulement de la mission des drones et de la simulation ARGoS

En fonction des commandes envoyées par le client (R.F.4), les drones changent d'état de mission. La logique d'évitement d'obstacles est utilisée pour tous les états du drone qui engendrent un mouve-

ment de celui-ci (R.F.3). Ainsi, en tout temps, les drones s'éloignent des obstacles qui les entourent en appliquant une vitesse dans la direction opposée à l'obstacle détecté. Ils débutent à l'état `Standby` (état affiché sur l'interface utilisateur) et, à la réception de la commande de début de mission, passent à l'état `Exploring`. Une fois à l'état `Exploring`, les drones se retrouvent dans une deuxième machine à états : celle de l'exploration (R.F.1). Cette machine à états est présentée à la figure 4 ci-dessous. Si la batterie d'un drone se décharge et atteint un niveau inférieur à 30%, ce dernier exécute l'algorithme de retour à la base et son statut change pour `Returning`. Par contre, l'état de la mission demeure à `Exploring`. Si tous les drones se déchargent ou si la commande de retour à la base est envoyée, l'état de la mission change pour `Returning`. Finalement, une fois que tous les drones ont atterri, que soit soit par un retour à la base complété ou par la réception de la commande d'atterrissage d'urgence, l'état de la mission change pour `Landed`. Une fois que la commande de fin de mission est reçue, l'état de la mission retourne à `Standby`.

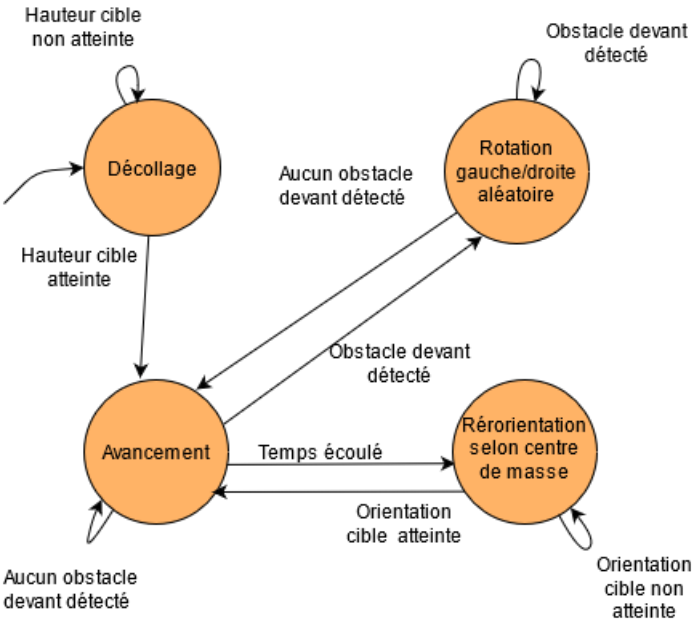

Figure 4 – Machine à états de l'exploration des drones et de la simulation ARGoS

Une fois que la commande de début de mission est reçue, les drones changent d'état et exécutent l'algorithme d'exploration. Ce dernier a la particularité d'inclure une alternance des directions de rotation et un éloignement périodique du centre de masse (R.F.1, R.F.6). Les drones commencent d'abord par décoller. Une fois la hauteur cible atteinte, les drones se retrouvent à l'état d'avancement, `Explore`. Dans cet état, les drones avancent en ligne droite vers l'avant. Dès qu'un obstacle est détecté par le capteur de distance avant, le drone change à l'état de rotation (`Rotate`), dans lequel il effectue une rotation dans une direction donnée jusqu'à ce qu'il n'y ait plus d'obstacle devant lui. Dès qu'il y a suffisamment d'espace vers l'avant, le drone retourne à l'état d'avancement, `Explore`. La rotation s'effectue dans la même direction entre trois et six fois avant de changer à la direction opposée. Cette approche permet d'éviter que des drones tournent en rond ou demeurent dans le même lieu trop longtemps. Durant l'exploration, les drones s'envoient leurs positions par communication *peer-to-peer*. Dans la simulation ARGoS, les capteurs et actuateurs *Range & bearing* simulent ce comportement en envoyant un *ping*. À partir des positions, chaque drone calcule le centre de masse de l'essaim. De façon périodique, les drones se réorientent dans la direction opposée à ce centre de masse et retournent à l'état d'avancement (`Explore`). À la réception d'une commande du

client qui met fin à la mission, les drones sortent de l'état de mission d'exploration (`Exploring`) et vont dans l'état de mission de retour à la base (`Returning`). La machine à états du retour de la base, illustrée à la figure 5, est au centre de ce nouvel état de mission.

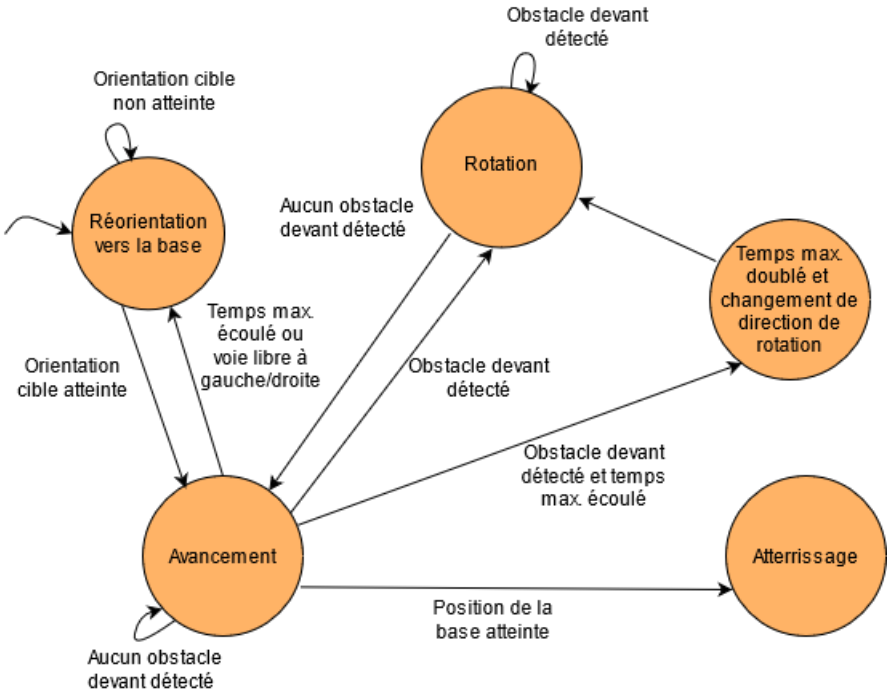

Figure 5 – Machine à états du retour à la base des drones et de la simulation ARGoS

Dans la machine à états du retour à la base, les drones débutent tous dans l'état de réorientation vers la base. Dans cet état, chaque drone calcule l'orientation vers la base en fonction de sa position courante et sa position en début de mission. Ensuite, il se réoriente pour faire face à sa position initiale. Par la suite, il se retrouve à l'état d'avancement. Au sein de cet état, si aucun obstacle ne fait obstruction au chemin vers la base du drone, celui-ci avance jusqu'à ce qu'il atteigne sa position initiale et qu'il respecte la condition de RSSI. Une fois arrivé, il se retrouve à l'état d'atterrissage. Si, pour un drone, un obstacle doit être contourné, un temps limite lui est octroyé pour contourner cet obstacle dans une certaine direction fixe. Une fois ce temps limite écoulé, le drone retourne à l'état de réorientation vers la base. S'il détecte toujours un obstacle vers sa base, le temps limite pour contourner l'obstacle est doublé pour ce drone, et il tente, à présent, de contourner l'obstacle dans la direction inverse. Cette logique permet au drone d'éventuellement trouver une issue afin de contourner l'obstacle. Si, pendant que le drone contourne l'obstacle, il détecte que la voie est libre vers sa base par son capteur de côté après un certain intervalle de temps, il retourne à l'état de réorientation vers la base. Ainsi, les drones peuvent retrouver leur base et contourner les obstacles qui bloquent leur chemin.

Suivant un processus itératif, l'algorithme d'exploration initial qui a été développé pour le CDR était relativement simple et consistait surtout en un évitement des obstacles. Depuis le CDR, la structure flexible de la machine à états d'exploration a permis d'étendre la solution et de construire une logique plus sophistiquée pour une exploration plus optimisée. Cette logique incorpore maintenant une alternance des directions de rotation et l'utilisation du *peer-to-peer* (P2P) pour un éloignement du centre de masse des drones. Il a aussi été possible d'intégrer l'algorithme de retour à la base robuste

face à différentes dispositions d'obstacles. Au final, l'ajout d'heuristiques aux algorithmes a permis de pallier plusieurs limitations de notre ancienne implémentation et est maintenant fonctionnel pour la totalité des scénarios testés.

2.3 La station au sol (Q4.5)

Pour l'architecture spécifique de la station au sol, on peut voir à la figure 6 que les quatre composantes sont concernées : ARGoS, les drones, le serveur et le client. Les choix d'architecture et les protocoles de communication sont détaillés dans cette section.

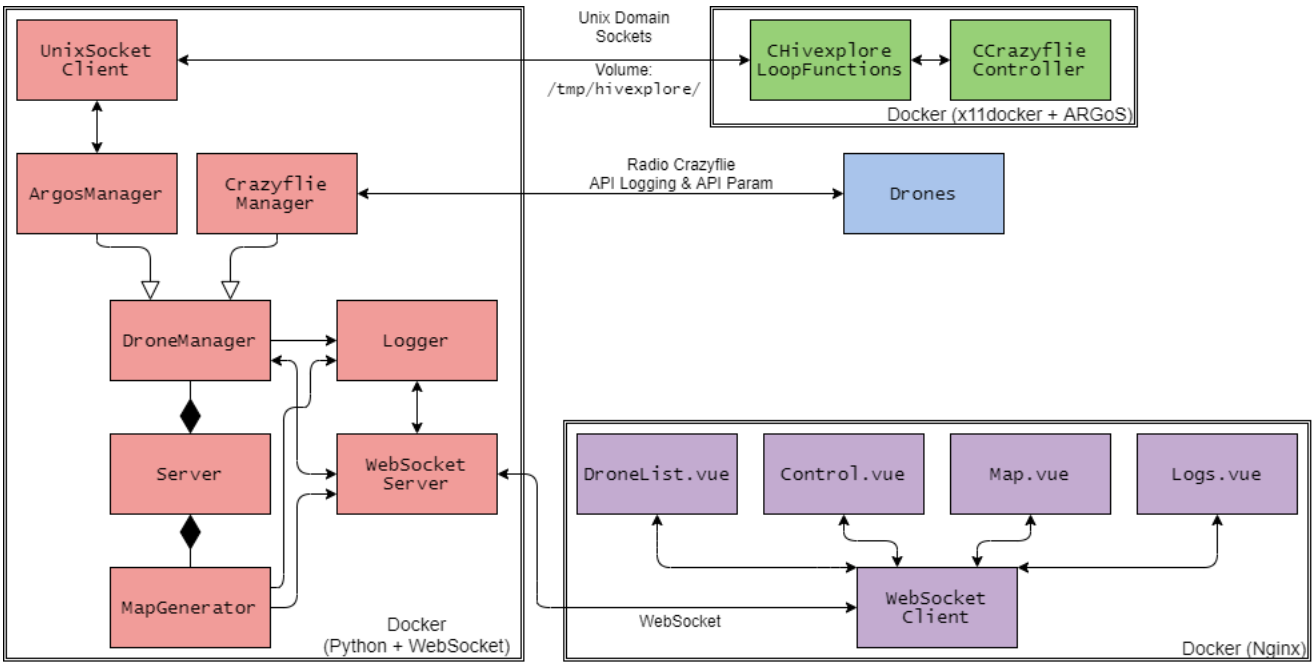

Figure 6 – Diagramme de l'architecture de la station au sol

D'abord, le serveur doit pouvoir choisir de communiquer soit avec les drones Crazyflie, soit avec la simulation ARGoS. Pour communiquer avec les drones, `cflib` [4], la librairie officielle pour Crazyflie, facilite la tâche. Bien qu'il existe des ports en d'autres langues, implémenter le serveur dans la même langue que la librairie officielle est plus simple et évite la possibilité d'encontrer des erreurs plus rares. Le support à long terme est aussi plus évident en se limitant à une distribution officielle. Il a donc été choisi d'écrire le serveur en Python.

Pour ce qui est de la communication du serveur Python avec le client Web, des WebSockets sont utilisés. Cette technologie permet d'établir une communication bidirectionnelle entre le client et le serveur, selon une interface élégante qui simplifie plusieurs des interactions associées à un état. De cette manière, aucune boucle de requêtes ne doit être maintenue sur le client. Pour permettre aux divers systèmes de fonctionner de façon simultanée sans se bloquer, les coroutines de Python sont utilisées à leur plein potentiel. La librairie `asyncio` de la librairie standard de Python permet de créer des tâches asynchrones qui se partagent équitablement le temps de calcul.

Le serveur est notamment en charge de la génération des points de la carte à partir de la mise en commun des données reçues de chacun des drones (R.F.6, R.F.7, R.L.6). En utilisant les estimations de position 3D et d'orientation (*roll/pitch/yaw*), une matrice de rotation est appliquée aux points

détectés par les capteurs. En générant la carte sur le serveur, la carte est assurée d'être cohérente entre les clients. Les points générés de la carte sont ensuite envoyés au client pour l'affichage. Du côté du client, la carte est visualisée en 3D à l'aide de la librairie Three.js. Cette librairie a été choisie pour sa popularité, sa simplicité, sa performance, et le fait qu'elle permet l'affichage de la géométrie en 3D. Ceci permet une visualisation très intuitive qui peut être observée de plusieurs angles.

De plus, le serveur est responsable de l'abstraction entre la gestion de la simulation et celle des Crazyflies. Dans le but de diminuer la duplication de code, un réusinage architectural a dû être fait depuis la remise du CDR. Au lieu d'avoir deux classes indépendantes, une pour le contrôle de chaque système, un lien d'héritage a été fait en créant une nouvelle classe mère. Ainsi, `ArgosManager` et `CrazyflieManager` héritent maintenant de la classe `DroneManager`, laquelle s'occupe de la logique de haut niveau, alors que les deux autres classes possèdent des méthodes qui sont spécifiques à leur système et appelées depuis la logique de la classe mère.

Une amélioration qui a été faite quant à l'algorithme des drones a demandé de faire un ajout qui n'était pas initialement prévu dans le serveur. En effet, tel que discuté plus tôt, il a été conclu que les drones avaient besoin de connaître leur position de décollage par rapport à la base. Pour ce faire, un fichier de configuration des drones a été ajouté dans le serveur, contenant l'identifiant de chaque drone et sa position de départ. Ces positions sont envoyées aux drones concernés avant le début de la mission et sont réutilisées pour la génération de la carte, produisant ainsi une carte beaucoup plus cohérente lorsque plus d'un drone sont utilisés.

Du côté des données entre le client et le serveur, un nouveau requis requiert une façon d'assurer une persistance des données des missions. La première partie de ce requis implique de conserver les cartes générées par la mission. Pour ce faire, notre solution propose à l'utilisateur de télécharger la carte à même l'interface Web s'il le désire. Ensuite, la seconde partie du requis concerne la sauvegarde des *logs*. En plus d'être affichés au client au cours de la mission (R.L.5), les logs sont aussi automatiquement enregistrés dans un fichier sur le serveur. Pour ce faire, la librairie standard `logging` de Python est utilisée et une nouvelle classe, `Logger`, sert d'intermédiaire. Cette nouvelle classe s'occupe d'envoyer les *logs* aux clients Web, de les afficher dans la console du serveur ainsi que de les enregistrer dans un fichier sur le système de fichiers de la machine hôte. Dans l'éventualité où un utilisateur voudrait avoir accès aux *logs* d'une ancienne mission, il serait toujours possible de se connecter par l'entremise d'une connexion `ssh` ou `ftp` avec le serveur afin de retrouver les fichiers voulus.

Finalement, une librairie de composantes d'interfaces graphiques pour l'application Web est utilisée afin d'avoir une application visuellement attrayante, uniforme, accessible et adaptative. Pour cette tâche, PrimeVue a été choisie : pour l'instant, c'est la seule librairie d'interface graphique qui supporte Vue.js 3. Celle-ci simplifie également la création d'une application adaptée aux appareils mobiles (R.L.4).

2.4 L'interface de contrôle (Q4.6)

Les technologies utilisées pour l'interface de contrôle ont été choisies dès le début dans une optique d'optimisation de l'expérience utilisateur ainsi que pour écourter le temps de développement. Le requis qui a eu le plus de poids dans le choix de technologie est le requis demandant que l'interface utilisateur puisse être vue à la fois depuis un appareil mobile et sur un ordinateur (R.L.4). Considérant la multitude de plateformes à supporter (Windows, macOS, Linux, Android et iOS), l'utilisation d'une application Web facilite le développement multiplateforme. De plus, le Web offre aux déve-

loppeurs d'excellents outils afin d'adapter leurs interfaces graphiques aux différentes tailles d'écran. Ainsi, l'interface profite de l'espace que fournissent les grands écrans et s'assure d'arranger l'application de manière intelligente afin de la rendre ergonomique sur de plus petits appareils comme des téléphones.

Du point de vue des fonctionnalités, l'interface de contrôle est séparée en quatre grandes composantes graphiques. Un aspect original de l'arrangement des composantes est que celles-ci se retrouvent toutes sur la même page, évitant à l'utilisateur de devoir naviguer dans l'application afin de retrouver les éléments voulus. Ainsi, l'application offre une expérience simple et intuitive en minimisant le nombre d'actions requises dans un désir de minimalisme.

La première composante est la composante de contrôle de la mission. Celle-ci permet de commencer la mission, de demander aux drones de retourner à la base, de faire un atterrissage d'urgence ainsi que de terminer la mission. L'état de la mission peut également être vu à tout moment ainsi que le nombre de drones en connexion avec le serveur.

La deuxième composante graphique est la carte 3D générée par les drones. Cette carte est faite à l'aide de THREE.js, une librairie qui simplifie l'utilisation de WebGL, afin de permettre une visualisation 3D élégante et légère. Cette carte permet de voir la position des drones ainsi que les points que ceux-ci détectent (R.F.5). Lorsqu'un point est généré, une ligne entre le drone et le point est affichée afin de comprendre quel point provient des capteurs de quel drone et d'offrir une expérience de visualisation plaisante.

La troisième composante graphique est une liste défilante horizontale des drones. En utilisant une liste défilante, la taille qu'occupe l'affichage des drones à l'écran est diminuée et offre une interface moins imposante à regarder. Pour chaque drone, on y retrouve des indicateurs pour sa vitesse, son pourcentage de batterie, l'état de sa DEL ainsi que son état courant, le tout disposé comme un tableau de bord automobile. L'état courant affiché permet de comprendre plus précisément la condition dans laquelle se trouve un drone, à savoir un des états dont « Returning », « Crashed », « Landed » et plus encore. Il est aussi possible d'allumer et d'éteindre la DEL du drone à partir de cette composante graphique pour facilement les repérer.

La quatrième et dernière composante graphique est celle de l'affichage des journaux. On y retrouve les informations regroupées selon qu'elles soient spécifiques au serveur, reliées à la génération de la carte ou concernant chacun des drones. Chaque catégorie se retrouve dans une languette différente pour permette à l'utilisateur de mieux se retrouver dans les journaux sans être envahi. De plus, il est possible d'activer le défilement automatique ou de cacher complètement les journaux afin de réduire la quantité d'information affichée.

L'interface de contrôle est conçue afin de facilement supporter un plus grand flux d'information. Dans le cas où l'Agence voudrait avoir plus de drones avec une mise à jour des données plus rapide, aucun changement dans l'interface ne serait nécessaire. De plus, l'utilisation de Vue.js 3 [6], un cadriciel Web moderne et simple, permet à l'Agence spatiale de maintenir l'application sans avoir à gérer du code utilisant un cadriciel trop complexe ou bientôt désuet. Enfin, le choix de la librairie d'interface graphique PrimeVue est compatible avec Vue.js 3 et offre un support intégré de composantes graphiques réactives aux différentes tailles d'écran.

Pour ce qui est de la réactivité de l'interface, chaque composante a été développée en priorisant avant tout l'expérience utilisateur sur un appareil mobile. Les éléments affichés sont automatiquement redimensionnés, réorganisés ou réduits selon l'espace écran disponible. Par exemple, d'un à quatre drones peuvent être affichés à la fois en fonction de l'espace disponible afin de garantir que l'information présentée ne surcharge pas l'utilisateur d'information trop dense. D'autre part, la carte

3D assurée par Three.js supporte nativement les gestes tactiles avec lesquels les utilisateurs d'appareils mobiles sont déjà à l'aise, leur permettant de manipuler la carte de manière très intuitive.

Enfin, dans l'optique de permettre à d'autres membres de l'Agence spatiale de voir et monitorer l'état de la mission et de pouvoir visualiser la carte générée en temps réel, la station au sol rend accessible un port sur le réseau local. De cette façon, d'autres appareils que la machine hôte sont en mesure de contrôler l'essaim par l'entremise du client Web en se connectant au port Nginx. La cohérence des informations affichées aux différents utilisateurs est bien sûr maintenue afin d'éviter toute confusion et d'assurer une expérience uniforme. Par exemple, si un utilisateur change l'état de la DEL d'un des drones, tous les autres utilisateurs voient instantanément cette information être mise à jour.

Bref, l'interface utilisateur a été conçue pour être réactive, intuitive, innovatrice et minimaliste, tout en gardant une allure agréable et joyeuse.

2.5 Fonctionnement général (Q5.4)

En règle générale, le fonctionnement complet de l'application est documenté dans des `README.md` situés dans le répertoire du projet. Un total de cinq `README.md` sont présents : un à la racine pour décrire le fonctionnement général, ainsi que quatre autres pour chacun des sous-projets `client`, `server`, `drone` et `argos` afin de donner de l'information supplémentaire utile au développement. Ces fichiers de documentation servent de documentation principale et donnent toute l'information pertinente dans une optique autant de production que de développement. Toutefois, une version extraite et abrégée est présentée ci-dessous, mais il est *fortement* recommandé de se référer aux `README.md` du projet.

En guise de prérequis, il est nécessaire d'avoir un ordinateur avec une distribution Linux sur laquelle sont installés Bash, Docker, Docker Compose et `x11docker`. Le script de démarrage à une seule commande se trouve à la racine du projet, nommé `start.sh`. Celui-ci prend obligatoirement un paramètre, qui peut être soit `drone`, `argos` ou `build`. En cas de doute sur les options, la commande `start.sh --help` peut être utilisée pour faire afficher de l'information d'aide.

L'option `build` est utilisée seulement si une des images Docker existe déjà et que des changements au code ont été effectués depuis, nécessitant alors une reconstruction de l'image. Sinon, cette option n'est pas normalement requise puisque les images sont automatiquement construites si elles n'existent pas déjà.

Le système a deux modes principaux, pouvant fonctionner soit avec la simulation ARGoS ou directement avec les drones Crazyflie. Pour le mode se connectant avec la simulation ARGoS, le système peut être démarré en utilisant la commande `start.sh argos` sans configuration préalable. Pour démarrer le système dans le mode se connectant avec les Crazyflies, par contre, les étapes ci-dessous doivent d'abord être suivies avant de partir l'application :

1. S'assurer que les permissions USB sont valides pour que la radio puisse fonctionner correctement. Voir le fichier `server/README.md` pour plus de détails.
2. Connecter la Crazyradio à un port USB.
3. S'assurer que le code embarqué des drones est à jour. Voir le fichier `drone/README.md` pour plus de détails.
4. Assigner une adresse unique à chaque drone. Voir le fichier `server/README.md` pour plus de détails.

5. Configurer la position de chaque drone. Voir le fichier `server/README.md` pour plus de détails.
6. Exécuter la commande `start.sh drone`.

Une fois la commande de départ du système exécutée, tous les conteneurs Docker de l'application sont démarrés en même temps (R.C.4). L'interface graphique peut alors être accédée sur l'ordinateur hôte à l'adresse `localhost:3995` dans un navigateur Web. Il est aussi possible d'accéder à l'interface graphique de l'application à partir de n'importe quel autre appareil connecté au même réseau LAN (R.L.4). Les instructions du fichier `README.md` à la racine du projet donnent plus d'informations à ce sujet.

Pour commencer une mission, les drones doivent tout d'abord être connectés au serveur. Avec la simulation ARGoS, ceci est automatiquement assuré par un *socket* Unix. Avec les Crazyflies, ceux-ci doivent être allumés pour qu'ils se connectent à la Crazyradio connectée à l'ordinateur sur lequel s'exécute le serveur. De plus, chacun des drones doit être fonctionnel pour que la mission puisse être commencée, c'est-à-dire avoir un niveau de batterie supérieur à 30% et ne pas avoir le statut « Crashed » - ils devraient donc normalement être en « Standby » avant une mission. Si un drone parmi tous ceux qui sont connectés n'est pas fonctionnel, il n'est pas possible de lancer la mission et un message de diagnostic apparaît dans la section de contrôle de la mission pour en aviser l'utilisateur.

À tout moment, une DEL peut être allumée ou éteinte sur chacun des drones grâce à un bouton à bascule propre à chaque drone appelé « LED » dans la section « Drones » de l'interface graphique. Ceci est notamment utile au début de la mission, lors de la configuration de la position des drones afin d'identifier les drones avec leur adresse pour les positionner.

Pour commencer une mission, il suffit de peser sur le bouton « Start mission » dans l'interface de contrôle. Tous les drones décollent alors immédiatement et commencent à explorer la salle. Si une situation indésirable ou potentiellement dangereuse se produit, il est possible d'ordonner un atterrissage d'urgence lorsque les drones sont en vol. Sur réception de cette commande, tous les drones sont contraints d'atterrir sur-le-champ, peu importe où ils se trouvent.

Sinon, une fois que la carte générée est jugée complète, un retour à la base de tous les drones peut être commandé en appuyant sur le bouton « Return to base » de l'interface de contrôle. En réponse, les drones reçoivent l'instruction de retourner à leur position de départ, qui se situe toujours à moins d'un mètre de la station au sol. Il faut toutefois noter que si le niveau de batterie d'un drone baisse sous le seuil de 30%, le drone retourne automatiquement à la base. Après un atterrissage, que ce soit après un retour à la base normal ou suite à un atterrissage d'urgence, le bouton « End mission » devient disponible lorsque tous les drones actifs ont le statut « Landed ». Ceci termine ainsi la mission et remet les drones actifs à l'état « Standby ».

Suite à la fin d'une exploration, le système permet de commencer une nouvelle mission, à condition que tous les drones connectés soient toujours fonctionnels. Il suffit alors de peser à nouveau sur le bouton « Start mission », comme au premier démarrage. Il faut noter que suivant un atterrissage d'urgence, le retour à la base d'une prochaine mission assure toujours le retour à la position originale malgré l'atterrissage préemptif précédent. Cependant, dans le but de confirmer l'intention de l'utilisateur, un message de confirmation apparaît si une nouvelle mission est démarrée après un atterrissage d'urgence, afin d'éviter d'involontairement reprendre une mission alors qu'un drone n'est pas encore dans un état sécuritaire.

La carte générée par les drones peut enfin être téléchargée une fois la mission terminée (ou à n'importe quel autre moment durant la mission) à l'aide d'un bouton situé dans le coin supérieur droit de la carte. Puisque la carte est réinitialisée à chaque nouveau départ, il peut être intéressant de la sauvegarder à la fin de chaque mission à des fins d'archivage.

Finalement, pour arrêter l'exécution du programme, un simple `CTRL+C` dans la console exécutant le script permet d'arrêter tous les conteneurs de façon ordonnée. Les journaux du serveur sont alors disponibles dans un volume Docker, situé à `/var/lib/docker/volumes/hivexplore_logs/_data` sur la machine du serveur.

3 Résultat des tests de fonctionnement du système complet (Q2.4)

L'équipe est très fière d'attester qu'elle a réussi à respecter tous les requis acceptés et qu'ils fonctionnent tous parfaitement. Pour valider les fonctionnalités et leur intégration au système, le processus de développement consistait à simuler la logique en utilisant ARGoS et, par la suite, à expérimenter sur le matériel physique. Dans la simulation ARGoS, une génération aléatoire d'environnements a permis de tester des dispositions d'obstacles variées et de détecter des problèmes de logique rapidement (R.C.1, R.C.5) [5]. De plus, l'utilisation d'ARGoS a permis de valider le fonctionnement du système avec plus de deux drones (R.F.2). Ensuite, la logique a été testée sur les drones en désactivant l'utilisation des moteurs lorsque possible afin de valider son fonctionnement sur le matériel [12]. Ces deux étapes se rattachent à une approche simultanée de *software-in-the-loop* [10] et *hardware-in-the-loop* [11]. Cette approche a permis d'éviter une utilisation excessive des drones et de les garder en bon état.

En parallèle, des listes de contrôle détaillées - soit pour le pré-vol, le vol et le post-vol - ont été utilisées afin de valider de façon plus rigoureuse le comportement des drones lors des tests dans la volière. Avant chaque test, l'utilisation de la liste de contrôle de pré-vol a assuré un environnement de test reproductible. Ensuite, la liste de vol a permis de valider le comportement des drones durant leur mission. Enfin, la liste de contrôle post-vol a servi à assurer qu'aucune composante matérielle n'ait été endommagée durant les tests. L'intégralité des listes de contrôle se trouve en annexe (R.F.7).

Cette procédure a été utilisée pour l'implémentation de tous les requis algorithmiques acceptés et pour vérifier leur opération durant les tests finaux. Premièrement, l'algorithme d'évitement a su prouver son efficacité en présence de plusieurs obstacles et dans des environnements étroits, ce qui confirme son fonctionnement correct (R.F.3). D'ailleurs, cet algorithme utilise entre autres la communication P2P de Bitcraze pour l'évitement entre les drones (R.L.2). Deuxièmement, l'algorithme de retour à la base des drones permet d'atterrir à moins d'un mètre de la base de manière répétable (R.F.4.2). Il utilise aussi les valeurs de RSSI pour vérifier si la distance de la base est de moins d'un mètre et déterminer si l'atterrissage peut être entamé (R.L.3). De plus, l'algorithme a démontré sa robustesse en assurant un retour à la base précis même lorsque des drones ont dû naviguer dans un environnement complexe. L'incertitude cumulée sur l'estimation de la position durant une longue mission n'est donc pas un problème significatif. Troisièmement, à tout moment lors d'une mission, il est possible de demander un atterrissage d'urgence. Cette commande a souvent été testée et elle s'est avérée très pratique à maintes reprises. Enfin, lorsqu'un drone a un niveau de batterie sous le seuil de 30%, l'interface de contrôle empêche bel et bien le démarrage d'une nouvelle mission (R.F.4.3). Si un drone tombe plutôt sous ce seuil durant une mission, il effectue automatiquement un retour à la base (R.F.4.3).

Pour ce qui est de l'interface utilisateur, durant chaque test, une validation a été effectuée quant à son fonctionnement complet. La mise à jour à une fréquence de 1 Hz de la carte et des informations concernant la mission ont été validées tout au long des procédures de test (R.L.5, R.F.5, R.L.6).

À chaque intégration d'une nouvelle fonctionnalité au code principal, le pipeline CI/CD a permis

de valider automatiquement la compilation, le passage de l'analyse statique (*linting*) et la cohérence des types en Python (*type checking*). Ainsi, chaque nouvelle fonctionnalité a automatiquement été validée afin d'empêcher une régression entraînant des erreurs de qualité ou de compilation. Les ajouts ont dû respecter tous les critères sans quoi ils ne pouvaient être intégrés au code principal (R.C.3). Ce processus rigoureux a permis à l'équipe d'avoir un contrôle sur la qualité du code de chaque composante du projet (R.Q.1, R.Q.2, R.Q.3).

4 Déroulement du projet (Q2.5)

Si tous les requis acceptés par l'équipe ont pu être complétés avec succès et sans problèmes majeurs, c'est grâce à une excellente gestion d'équipe et une planification adaptative sur toute la durée du projet.

Tout d'abord, il est utile de mentionner que l'équipe a commencé le projet du bon pied. Durant les deux premières semaines pendant lesquelles l'appel d'offres n'était pas encore disponible, du temps a tout de même été investi afin d'établir des règles de fonctionnement internes et une méthodologie pour le déroulement des réunions. Ainsi, les fondations du fonctionnement de l'équipe ont été mises en place très tôt. Parallèlement, plus d'une semaine entière a été consacrée à une recherche exhaustive pour trouver les solutions et les technologies les mieux adaptées aux problèmes et particularités propres au projet. Le temps a enfin été bien investi à bien organiser et configurer la structure des sous-projets et les outils de développement associés à chaque langue de programmation. Comme de fait, les choix technologiques étudiés et adoptés en début du projet n'ont pas eu à changer entre l'appel d'offres et le RR, et aucun regret n'a été ressenti à posteriori face aux techniques utilisées. Grâce à ceci, aucun temps imprévu n'a dû être perdu en milieu de projet à faire un réusinage colossal de code à cause de choix qui auraient été pris sans assez de réflexion.

Une fois ces décisions clés prises, l'équipe a suivi un processus itératif afin de laisser l'expérimentation et les contraintes d'intégration des différentes technologies renseigner les décisions architecturales pendant les semaines précédant la remise du CDR. En effet, l'équipe a identifié très tôt qu'un des plus grands défis du projet serait justement la complexité d'intégration des divers composantes et de la communication entre celles-ci. En expérimentant et en assurant le plus vite possible l'intégration entre les diverses parties du projet, plusieurs problèmes majeurs ont été trouvés d'avance et ont pu influencer les décisions architecturales en conséquence. Par exemple, il a été découvert que les drones simulés dans ARGoS ne peuvent pas être contrôlés de la même manière que les drones Crazyflie. Quoiqu'un des buts initiaux de l'équipe ait été d'implémenter une librairie pour la logique commune au code embarqué et au code simulé, notre processus itératif a permis de réaliser que ceci ne serait pas possible d'accomplir dans les délais demandés en raison des différences de contrôle. À l'inverse, si la librairie avait été écrite et implémentée avant de rencontrer le problème, beaucoup de temps aurait été mal investi à faire fonctionner une librairie qui n'aurait pas pu être mise à profit.

Une bonne gestion d'équipe passe aussi par une bonne organisation des réunions. En ayant au minimum deux réunions hebdomadaires lors du projet, l'équipe a assuré une bonne coordination des membres. En effet, les réunions du lundi ont servi à diviser les tâches pour la semaine en fonction des disponibilités de chacun. Ensuite, les réunions du jeudi ont eu pour but de partager les avancements de chacun et de discuter des éléments entravant la complétion des tâches restantes. Elles ont par moments aussi servi à réévaluer la planification et l'assignation des tâches afin de réussir à toutes les compléter avant la fin du sprint. Un autre point fort des réunions est que celles-ci étaient assez efficaces, étant donné qu'un ordre du jour avec une structure récurrente était toujours écrit

avant leur début. Après quelques semaines, tous les membres de l'équipe étaient devenus à l'aise avec le déroulement des réunions, améliorant leur fluidité.

Grâce à cette organisation robuste, un rythme continu de travail a été maintenu sur toute la durée du projet tout en ayant un processus de développement itératif et une revue de code continuelle. Malgré que les semaines avant les remises aient parfois été plus chargées, une charge de travail solide a été investie à chaque semaine, en détaillant en début de projet un plan de développement agressif permettant des semaines tampons en cas de problème. Ceci a permis de garder un bon rythme tout au long du projet, pour enfin soumettre un produit final impeccable.

Cependant, en dépit d'une bonne gestion de projet et d'une planification prévoyante des imprévus, un problème important qui s'est manifesté a été au niveau du nombre d'heures excessives requises pour compléter un projet du calibre de celui décrit par l'appel d'offres de l'Agence. En effet, nous avons présumé au début du projet que les 630 heures prévues par l'Agence seraient suffisantes pour remplir tous les requis de l'appel d'offres - et même certains requis optionnels si désiré - et avons adapté notre planification en conséquence. Cependant, tel qu'il a été rapporté dans le rapport du CDR, le temps nécessaire à l'accomplissement de plusieurs tâches a été sous-estimé et plusieurs ont pris plus de temps que prévu. Au CDR, il a donc dû être concédé d'annuler certains requis, soit les tests unitaires et la mise à jour du code embarqué à distance.

Avec ces modifications au CDR, il avait été prévu que 210 heures seraient suffisantes pour compléter les fonctionnalités restantes, avec un surplus potentiel de 10%. Malheureusement, en comptant les heures réellement dépensées aux sprints suivant le CDR (sprint 5 au sprint 10), on remarque que 460 heures ont été consacrées au développement, soit un excès de plus de 190 heures (460 - (210 + 570*10%) = 193) par rapport à ce qui était planifié. En additionnant le temps de développement investi depuis le début du projet, ceci revient à un total de 820 (360 + 460) heures, largement plus que les 570 heures prévues pour le développement.

L'écart entre le nombre d'heures (270 heures, en incluant le surplus de 10%) prévues pour les sprints 5 à 10 et le nombre d'heures réalisées (460 heures) peut notamment être expliqué par quatre tâches qui ont pris considérablement plus de temps que prévu :

- Réusinage de l'interface utilisateur et la rendre réactive. Prévu : 12h. Dépensé : 45h.
- Démarrage du système à une seule commande. Prévu : 10h. Dépensé : 25h.
- Retour à la base (ARGoS et firmware). Prévu : 22h. Dépensé : 80h.
- Retour à la base automatique si la batterie < 30% + correction de la lecture du niveau de la batterie. Prévu : 6h. Dépensé : 40h.

Plusieurs facteurs expliquent les retards dans ces tâches qui sont emblématiques des difficultés imprévues qui ont été rencontrées en cours de projet. D'abord, les tâches de réusinage de l'interface utilisateur et du démarrage à une seule commande ont été sévèrement sous-estimées, plus particulièrement pour ce qui a trait à l'apprentissage de nouvelles techniques ou technologies. Par exemple, bien que certains membres de l'équipe étaient déjà familiers avec le concept d'interfaces réactives, certains détails plus complexes n'étaient pas bien maîtrisés, à savoir l'utilisation de *media queries* CSS pour régler certaines difficultés restantes pour avoir une interface entièrement réactive ou la modification d'un thème global pour avoir une palette de couleurs jaune et noir. La tâche de Docker Compose est un autre cas où un nombre insuffisant d'heures a été estimé pour tenir compte du temps d'apprentissage de cette technologie, sachant que personne dans l'équipe n'était expérimenté à cet égard. L'utilisation de Docker Compose avec plusieurs modes (un pour se connecter aux drones, et un pour se connecter à la simulation ARGoS), la conteneurisation d'une application graphique indépendante des *drivers* de carte graphique installés, et l'interface CLI avec un script Bash

ont été des éléments notamment plus compliqués que prévu. En général, les tâches concernant des technologies moins maîtrisées ont souvent été sous-estimées dans la planification du projet.

D'autre part, certains requis qui semblent en surface assez réalisables se sont révélés être jonchés d'embûches causées par des complexités apparemment non anticipées par l'Agence dans l'appel d'offres, des problèmes matériels, et des inadéquations entre les technologies prescrites par l'Agence et la réalité. L'exemple parfait est celui de la fonctionnalité du retour à la base. En effet, l'Agence prescrivait initialement l'utilisation de RSSI pour déterminer la position de retour du drone. Cependant, d'après nos nombreux tests, il s'est avéré que la valeur du RSSI n'est pas une mesure assez fiable et sensible à la position pour être utilisée par les Crazyflies pour un algorithme de retour. Il a donc dû être nécessaire de complémenter l'utilisation du RSSI à un retour à la position de décollage afin d'avoir une mesure plus fidèle. De plus, la totalité du concept d'obstacles au retour n'avait pas été prise en compte ; en effet, si un obstacle se trouve entre le drone et la base, celui-ci doit s'assurer de le contourner avant de poursuivre dans la direction vers la base. Ce genre de situation n'est que l'un des plusieurs cas particuliers qui ont dû être gérés. Or, à chaque fois qu'une solution était trouvée, un nouveau problème non considéré se faisait remarquer.

Des limitations et problèmes reliés au matériel ont en effet grandement ralenti ou parfois même bloqué le développement, touchant de très près la tâche de retour à la base automatique lorsque le niveau de batterie des drones descend sous 30%, pour ne nommer que cet exemple. Pour ce cas particulier, il a été découvert que le niveau de batterie en pourcentage tel que rapporté par l'API de Bitcraze n'est aucunement fiable lorsque le drone est en vol, indiquant une valeur bien inférieure à celle lorsque le drone est au repos. Sans lecture fiable du niveau de la batterie en pourcentage, plusieurs requis ne sont pas réalisables, comme le retour à la base automatique en cas de batterie faible ou le fait d'empêcher la mission de commencer si certains drones ne sont pas suffisamment chargés. En effet, le niveau de batterie rapporté en vol par l'API de Bitcraze est si faible qu'il cause un retour à la base immédiat après le décollage. Pour régler ce problème, il a été nécessaire de récolter des mesures expérimentales pour le voltage de la batterie au repos et en vol à plusieurs intervalles afin de pouvoir manuellement effectuer la conversion en pourcentage en corrélant le voltage à des valeurs différentes selon l'état de vol du drone.

Un autre problème qui a souvent freiné notre cycle de développement est le temps de charge nécessaire pour les drones par rapport au temps de vol. En effet, la batterie doit être chargée pendant environ 45 minutes mais ne permet qu'entre 3 et 6 minutes de vol sur une charge complète. Ce très long cycle de recharge a réduit le nombre de tests qu'il était possible d'effectuer dans une journée, surtout lorsque ces tests demandaient une mission complète à deux drones, déchargeant alors complètement les deux batteries.

Plusieurs autres problèmes matériels ont malencontreusement eu pour effet de ralentir chaque étape du processus itératif de l'équipe qu'il serait trop long de lister. À chaque fois, du temps supplémentaire devait être consacré afin de trouver la source du problème, dont celle-ci (logicielle ou matérielle) était souvent inconnue. Par exemple, un problème avec la vitesse retournée par le filtre de Kalman [8] offert par l'API de Bitcraze s'est avéré en réalité être causé par un moteur défectueux. De surcroît, pour chaque problème matériel, le contexte de la pandémie est venu complexifier le remplacement de nouvelles pièces venant de l'Agence, nécessitant toujours un rendez-vous et rendant le drone inutilisable entre-temps. Bref, plusieurs problèmes matériels difficiles à diagnostiquer ont rendu le développement sur les drones beaucoup plus long qu'initialement anticipé et ont causé des retards dans notre planification.

Mis à part les problèmes matériels, la nécessité de développer à la fois pour ARGoS et pour le code

embarqué des Crazyflies a aussi amené son propre lot d'imprévus. Afin de tester les algorithmes, il est impératif que la simulation ARGoS se rapproche le plus possible de l'implémentation sur les drones Crazyflie. Or, les différences entre les Crazyflies et les drones simulés ne permettent pas de reproduire fidèlement la réalité, ce qui a complexifié le développement afin de tenir compte des particularités du code embarqué et du code simulé.

En effet, une différence importante entre le code embarqué et la simulation est la logique de contrôle. Celle-ci est basée sur les vitesses du côté des Crazyflies, alors qu'elle est basée sur les positions dans le cas d'ARGoS. Cette différence a été causée par l'absence de l'implémentation de l'actuateur de vitesse sur ARGoS au moment de l'implémentation de la logique de contrôle des drones. Sans cet actuateur, il n'a pas été pas possible de reproduire le comportement des vrais drones. L'actuateur de la position d'ARGoS a donc dû être utilisé, ce qui a créé des différences entre le contrôle des vrais drones et ceux simulés avec ARGoS.

D'autres implémentations sont également manquantes dans ARGoS, ce qui rend certaines données inaccessibles dans la simulation, comme les mesures des capteurs de distances du haut et du bas, acquises pour l'instant grâce à d'autres informations. Certains algorithmes auraient pu être simplifiés en ayant accès à ces capteurs puisque leurs mesures permettraient d'éviter les obstacles se situant sur ou sous les drones. Il n'existe d'ailleurs pas de capteur de vitesse dans ARGoS, ni de capteur pour obtenir la puissance de signal (RSSI). Il n'est donc pas possible d'accéder directement à la mesure de la vitesse courante des drones et la mesure de RSSI sur ARGoS, d'où le fait qu'il ait été nécessaire d'ajouter de la logique pour les calculer automatiquement.

Bref, comme il est nécessaire de prendre en compte les différences entre le code embarqué des Crazyflies et celui de la simulation afin de reproduire assidûment le comportement des vrais drones dans ARGoS, le développement des deux solutions parallèles a été plus complexe et long que prévu.

En fin de compte, même avec tous ces problèmes et contraintes complexes, le fait est que tous les requis ont heureusement pu être livrés à temps à chaque jalon du projet. Malgré que plus d'heures qu'initialement prévu aient dû être investies, les tâches prévues lors du CDR ont pu être terminées grâce à la semaine 9, justement réservée comme tampon en cas d'imprévus. En revanche, il a été nécessaire de jongler avec les problèmes matériels, la complexité inattendue de certaines tâches, la difficulté à s'organiser en fonction des drones en contexte de pandémie et la charge de travail plus lourde qu'anticipée. Pour ce faire, 460 heures de temps supplémentaire ont dû être consacrées par les membres de l'équipe pour tout terminer pour la remise finale. Ceci revient à un total de 820 heures de développement, ce qui est considérablement trop d'heures par rapport aux 570 heures allouées.

Malheureusement, même ces 820 heures pour le développement ne reflètent pas l'entière réalité du temps passé sur le projet ; ces heures ne prennent pas en compte le temps passé pour les réunions d'équipe et l'écriture de rapports. Lors de la réponse à l'appel d'offres, il avait été estimé que 60 heures seraient suffisantes pour toute la gestion de projet. Toutefois, pour donner une estimation conservatrice, à 4 heures/personne pendant 10 sprints, c'est plutôt 240 heures collectives qui ont été dépensées en réunions et en tâches variées de gestion de projet. En ajoutant ces heures aux heures passées sur les tâches de développement, le total revient à 1100 heures pour tout le projet. Ceci correspond à un excès de (1100 - 630) 470 heures par rapport au budget alloué, soit un imposant dépassement de 74%. Par contre, étant donné que le projet est très intéressant et stimulant, l'équipe a fait le choix éclairé d'investir ces heures afin d'avoir un bon produit final dont elle est fière. De plus, comme il avait été mentionné dans le rapport du CDR, ces heures sont considérées comme des heures supplémentaires non payées, donc le budget reste tout de même respecté.

5 Travaux futurs et recommandations (Q3.5)

Même si l'équipe a réussi à compléter tous les requis acceptés, le dépassement de 74% n'est pas négligeable. Par ses connaissances techniques, ses expériences pertinentes et sa bonne gestion d'équipe, l'équipe se considère très compétente. Ainsi, l'équipe croit que ce dépassement est associé aux demandes trop élevées de l'Agence plutôt qu'un manque d'expérience chez ses membres, d'où la recommandation de réduire la charge de travail si l'Agence compte faire d'autres appels d'offres. Par exemple, plus de temps doit être pris en compte pour le déverminage des problèmes matériels. Aussi, le requis des tests unitaires (R.C.2) est difficilement réalisable pour le nombre initial d'heures allouées à l'appel d'offres. Au-delà de ces deux exemples, l'équipe renforce qu'une simplification des requis est conseillée afin que le budget de 630 heures puisse être respecté.

Par contre, si l'Agence décidait d'aller plus loin avec ce projet, plusieurs aspects pourraient être améliorés. Dans le cas où une équipe de l'Agence continuerait d'améliorer la solution Hivexplore, nous pensons que l'écriture de tests unitaires pertinents deviendrait une priorité. L'architecture de tests est déjà présente et parfaitement intégrée. En effet, le *GitLab CI* exécute déjà les étapes de tests. Il ne resterait donc qu'à implémenter les tests pour le client, le serveur et le *firmware* ; les outils de développement et le CI supportent déjà leur exécution.

Par manque de temps, la mise à jour sans-fil n'a pas non plus été implémentée. Cependant, celle-ci augmenterait grandement la facilité d'opération avec les drones. Présentement, un drone devant être mis à jour doit être accessible physiquement par un humain, tâche qui serait grandement alourdie par un plus grand nombre de drones. Ceci deviendrait aussi un sérieux problème pour un déploiement sur des corps célestes.

Outre les requis qui n'ont pas été acceptés, une autre amélioration possible serait l'intégration du capteur de vitesse ainsi que l'implémentation des capteurs de distance du haut et du bas dans ARGoS. Ceci permettrait de simuler les mesures des vrais capteurs des drones Crazyflie. Suite à la création d'une *issue* sur le GitHub du projet ARGoS [13] par une des membres de l'équipe, Yasmine Moumou, l'actuateur de vitesse pour les quadricoptères a été implémenté dans ARGoS. Avec cette mise à jour récente, il serait à présent possible d'utiliser cet actuateur afin de reproduire la logique de contrôle des drones Crazyflie dans ARGoS.

En utilisant ce nouvel actuateur et en ayant ainsi une logique beaucoup plus semblable dans les deux programmes, il devient même possible d'implémenter une librairie de logique partagée. Cette librairie permettrait de regrouper la logique commune entre les Crazyflies et ARGoS et ainsi de réduire considérablement la quantité de code à maintenir. Les résultats des tests du comportement des drones en simulation deviendraient alors plus fidèles au comportement réel des drones.

Tel qu'expliqué plus haut, l'équipe a également rencontré plusieurs problèmes avec les batteries des drones. Une solution a été adoptée, mais elle n'est pas très précise. Ainsi, il est fortement conseillé de trouver une autre manière de faire un calcul plus précis du pourcentage afin que le niveau des batteries en vol soit cohérent avec leur charge réelle.

Certains des problèmes matériels auraient sinon pu être évités si l'environnement de tests à Polytechnique avait été accessible plus tôt. Par exemple, grâce au plancher adapté aux capteurs, le comportement du module *Flow deck* est beaucoup plus fiable et reproductible dans l'environnement de tests mis à notre disposition par l'Agence. De plus, les dimensions de la volière ainsi que le nombre d'obstacles mobiles disponibles ont permis de recréer des scénarios spécifiques plus aisément. Ainsi, il est fortement recommandé de prioriser le travail dans la volière lors du développement avec les drones Crazyflie pour une prochaine itération.

6 Apprentissage continu (Q12)

En premier lieu, Nathanaël Beaudoin-Dion a identifié qu'il a des lacunes au niveau technique et au niveau de la communication. D'abord, en ce qui concerne le technique, compléter une tâche lui prend un peu plus de temps en général. En effet, son savoir s'étend sur beaucoup de sujets variés. Cependant, ses connaissances ne sont que superficielles concernant certaines de ces technologies. Ainsi, il doit consacrer plus de temps à apprendre ce qui lui manque. Pour régler ce problème, il n'a pas eu peur de demander de l'aide et d'investir le temps supplémentaire requis lors du projet. Par contre, il sait que ce n'est pas assez et qu'il doit continuer à travailler là-dessus. Ainsi, il va prendre plus de temps hors de ses cours d'apprendre plus en profondeur les technologies utilisées lorsqu'il travaille sur des projets externes. En ce qui à trait à la communication, il tend à exprimer ses idées de manière directe et parfois un peu décousue, sans prendre le temps de formuler une idée claire a priori. Ceci s'applique autant à l'oral qu'à l'écrit. Pour remédier à ce problème, il a fait des efforts lors de la deuxième partie du projet pour prendre plus de temps à relire ce qu'il écrivait. Ce faisant, il a mis plus de temps à formuler des idées claires avant d'aborder un problème, ce qui a évité plusieurs malentendus. Toutefois, il sait qu'il doit continuer à travailler sur ce problème étant donné que la communication est un aspect très important au bon travail d'équipe. Pour ce faire, il va continuer à appliquer ses solutions lorsqu'il s'exprime dans son quotidien.

Ensuite, Samer Massaad a des lacunes au niveau de sa concentration lorsqu'il travaille de la maison et a des améliorations à faire quant à son sens de l'organisation. D'abord, à cause de ses difficultés avec le travail à la maison, beaucoup de temps a été dépensé de façon inefficace sur le projet. En temps normal, il aurait été possible de passer plus de temps à l'école ou dans d'autres lieux favorisant le travail. Afin de remédier à cette première difficulté en confinement, Samer a réservé la volière dès que possible afin de travailler dans un environnement propice à la concentration et loin de la maison. Pour ce qui est de ses difficultés à s'organiser, des retards accumulés dans d'autres cours ont causé des indisponibilités pour le projet près des dates de remise. Afin de palier cette lacune, il s'est assuré de faire du travail sur le projet en début de semaine. Ceci a permis au reste de l'équipe de faire une revue de travail plus tôt dans la semaine. Parallèlement, Samer a pu consacrer du temps à ses autres cours le temps que cette révision soit terminée. Une autre solution possible à cette lacune aurait été de faire une planification de sa semaine plus détaillée afin de s'assurer de ne pas prendre du retard dans ses autres cours.

Simon Gauvin, de son côté, a identifié des lacunes quant à son organisation ainsi que la gestion de son temps. En effet, il était difficile de prévoir les moments durant la semaine durant lesquels il pouvait travailler, rendant difficile pour tous les membres de l'équipe de planifier leur semaine. En effet, puisque le développement de plusieurs tâches dépendaient de tâches qui devaient être faites préalablement, il était important de savoir à quels moments quels membres de l'équipe pouvaient travailler afin d'assigner les tâches de manière à ce que personne ne soit bloqué par une autre tâche. En ayant de la misère à identifier à l'avance ses périodes de travail et en ne respectant pas toujours ses engagements, certaines tâches ont souffert d'un blocage qui aurait pu être évité. Afin de remédier à ce problème, un canal Discord a été créé afin que les membres de l'équipe puissent partager leur avancement dans leurs tâches et une partie de la réunion du début de sprint a été accordée à la planification de la semaine. Ceci a aidé à identifier les problèmes à l'avance et pouvoir remanier la distribution des tâches. De plus, Simon a pu partiellement remédier à ce problème en organisant mieux sa semaine avec ses autres cours. Cependant, il est conscient que du travail a besoin d'être fait quant à cette déficience en étant plus assidu et en adoptant l'utilisation d'un agenda.

Yasmine Moumou, quant à elle, a pris connaissance de ses lacunes au niveau de son assiduité par

rapport à la révision régulière du code des autres membres. Initialement, elle priorisait ses propres tâches au point de seulement commencer à réviser le code d'autrui une fois qu'elle était assez avancée. Or, il est nécessaire pour l'avancement global du projet d'apporter des critiques et des améliorations au code des autres tout au long du sprint. Pour pallier ce problème, elle a choisi de se prendre un moment fixe dans la journée et ce, à chaque deux jours pour passer au travers des ajouts des autres membres et apporter des commentaires. Cette méthode s'est avérée assez efficace durant le projet. Cependant, vers la fin du projet, elle a manqué de le faire dû à ses autres cours. Aussi, c'était la première fois qu'elle travaillait avec les drones Crazyflie et ARGoS. Yasmine a donc dû se renseigner sur les drones Crazyflie à l'aide de la documentation présente sur le site et dans les `README.md` du code de base du code embarqué. Aussi, elle a dû prendre connaissance de la structure d'ARGoS. Finalement, elle s'est rendue compte de sa réticence à demander de l'aide et à poser des questions. Pour y remédier, elle s'est établi des limites au niveau du temps à passer à régler un problème avant de demander de l'assistance ou de poser des questions aux autres membres. Cette limite n'a malheureusement pas toujours été respectée, c'est donc un aspect qui serait toujours à améliorer. Une autre piste de solution serait, par exemple, de faire une liste des problèmes rencontrés et de demander de l'aide lorsque la liste dépasse une certaine longueur.

Quant à Rose Barmani, elle reconnaît avoir des lacunes dans l'organisation de son temps ainsi que dans ses habiletés de communication. Ainsi, la planification de ses semaines n'a pas toujours été idéale, ce qui a affecté le temps qu'elle a passé sur le projet ainsi que sur ses cours. N'ayant pas toujours réussi à prévoir le temps nécessaire pour ses tâches, elle a souvent pris du retard dans ses autres cours. Pour remédier à cette lacune, Rose a pris l'habitude de planifier, à l'avance, le temps qu'elle passerait sur chacune de ses tâches en début de semaine. Elle a pu, de cette façon, prévoir si elle allait manquer de temps ou non. Malgré cette planification, il est arrivé des situations imprévues où une tâche a pris plus de temps qu'anticipé, comme l'algorithme du retour à la base. Cet aspect aurait donc pu être amélioré en demandant de l'aide ou en déléguant des tâches lorsqu'elle se rendait compte qu'une certaine tâche serait plus complexe qu'initialement prévu. Pour ce qui est de ses habiletés de communication, Rose est plutôt réservée et ne prend pas souvent la parole lors des réunions d'équipe. Pour remédier à cette lacune, elle a essayé de s'exprimer lorsqu'elle avait une idée ou une suggestion. De plus, elle a été désignée pour animer une partie des rétrospectives de sprint pendant quelques semaines, ce qui l'a aidée à s'affirmer. Rose est consciente que ses habiletés de communication pourraient être améliorées, puisque cet aspect sera toujours important pour les projets en équipe. Elle doit donc continuer à travailler à se dégêner afin d'être à l'aise de s'exprimer et de prendre la parole plus souvent dans les réunions d'équipe.

Enfin, Misha Krieger-Raynauld considère avoir des lacunes au niveau de sa capacité à estimer la charge de travail qu'il accepte d'assumer tout en jonglant avec le temps passé en aidant d'autres membres de l'équipe dans leurs tâches. En effet, beaucoup de ses tâches ont parfois pris plus de temps que prévu à cause d'une estimation optimiste des tâches plus complexes ou en n'accordant pas assez de temps pour apprendre une technologie encore inconnue. À d'autres moments, il a pris des tâches en supplément lorsque des problèmes ont été découverts en milieu de sprint, sans assez de considération sur l'impact que ceci aurait sur son temps disponible. Ceci s'est souvent avéré problématique puisque trop de temps a été investi dans le projet, au détriment de ses autres cours. Cette situation a été exacerbée par le fait que le projet en soit a représenté une charge de travail excessive dans le contexte d'un cours universitaire de quatre crédits. Pour tenter de régler ce problème, la situation a été soulevée aux membres de l'équipe lors des rétrospectives de sprint, dans l'effort d'en faire part aux autres pour mieux équilibrer la chose. Il a aussi tenté plus souvent de déléguer des tâches aux autres pour être moins surchargé avec les siennes. Toutefois, ces efforts n'ont pas toujours porté fruit, et la situation de surcharge de travail s'est néanmoins reproduite à

plusieurs reprises lorsque des bogues ou difficultés se sont manifestés avec les tâches de l'équipe. Plus d'efforts devraient être investis pour remédier à cette lacune, en trouvant par exemple de nouvelles façons de séparer son travail en fragments partageables et en hésitant moins à demander de l'aide. Il serait aussi bien de diminuer les attentes par rapport à soi-même et au résultat escompté, en investissant moins de temps dans l'apprentissage de notions plus compliquées et en les remettant à plus tard.

7 Conclusion (Q3.6)

Somme toute, l'équipe est très fière du résultat final, affectueusement baptisé **Hivexplore**. Tous les requis acceptés ont été respectés et même dépassés en ajoutant quelques fonctionnalités supplémentaires, aucun bogue restant ne ternit l'éclat du produit fini, et le fonctionnement du système est à la hauteur de nos attentes de robustesse et d'expérience utilisateur.

Par rapport à l'appel d'offres, le système final correspond assez fidèlement à ce que l'équipe avait en tête comme vision initiale. Toutefois, certaines hypothèses quant au processus de conception et aux solutions envisageables avaient été émises en tout début de projet qu'il serait intéressant de revisiter. En premier lieu, nous avions estimé lors de la réponse à l'appel d'offres que la Crazyradio serait en mesure de soutenir une connexion fiable avec tous les drones dans un espace de 100 m^2 en tout temps, sans quoi le système se heurterait à des contraintes matérielles très problématiques. Pour établir cette hypothèse, nous nous étions basés sur la lecture de la documentation de la Crazyradio et de l'API de Bitcraze. Heureusement, cette hypothèse se voit maintenant parfaitement confirmée, puisque le système fonctionne en effet avec une seule Crazyradio, la distance entre les drones et la base n'est pas un problème, et les obstacles physiques testés n'empêchent pas la réception du signal.

D'autre part, nous avions aussi présumé durant le PDR que toutes les fonctionnalités physiques des Crazyflies et de leur API pourraient être reproduites fidèlement avec la simulation ARGoS. Au CDR, nous avons cependant invalidé cette hypothèse en constatant que tous les modules du code embarqué et de l'API de Bitcraze ne sont effectivement pas déjà implémentés dans ARGoS. Ainsi, certaines fonctionnalités comme le calcul de distance par RSSI, l'API de *logging* et de *parameter*, et l'envoi de positions par communication *peer-to-peer* ont dû être réécrites ou adaptées pour les simuler avec ARGoS. Au CDR, il était encore supposé que les capteurs des drones nous offriraient suffisamment d'information pour inférer la position absolue de ceux-ci. Depuis, nous avons également invalidé cette dernière hypothèse : malgré que les capteurs des drones leur permettent d'avoir une très bonne estimation de leur position actuelle par rapport à leur position initiale, ce n'est pas le cas pour leur position absolue, qui est décalée par rapport aux autres drones en fonction de la position de décollage de chacun. Puisque cette information est nécessaire pour la logique d'évitement de collisions entre les drones, pour mettre en commun les points de la carte venant de différent drones, et enfin pour visualiser la position précise des drones sur la carte en temps réel, il a été nécessaire de remédier à ce problème. Pour ce faire, le serveur envoie la valeur de position de décalage des drones par rapport à la base au début de chaque mission, contrairement à notre hypothèse initiale.

Enfin, nous avions supposé qu'il serait toujours possible de se transférer les drones entre les membres de l'équipe malgré les restrictions pour contrer la COVID-19. Bien que l'échange du drone n'ait pas été problématique durant la première moitié du projet où les deux drones pouvaient être distribués à deux personnes différentes, la situation de la pandémie est devenue plus problématique vers la fin du projet, où des fonctionnalités nécessitaient que les deux drones soient rassemblés chez la même

personne. Dans ces situations, le couvre-feu et les indisponibilités occasionnelles de chacun ont malheureusement rendu plus difficile l'échange des drones au sein de l'équipe, ralentissant parfois le progrès.

En dépit de ces quelques hypothèses invalidées, le processus de conception que l'équipe a fini par suivre s'est avéré très près de ce qui avait été envisagé lors de la réponse à l'appel d'offres, excluant bien sûr les nombreuses heures supplémentaires requises. L'équipe a suivi un processus itératif, où les grandes décisions architecturales étaient toujours informées par une expérimentation préalable. De ce fait, il a été possible de minimiser l'impact des gros imprévus en testant les hypothèses sur lesquelles reposeraient des fonctionnalités ultérieures le plus tôt possible dans le développement. Le produit final résultant a bénéficié de ce processus et s'en trouve plus robuste, raffiné et agréable à utiliser.

8 Références

[1] "Multi-ranger deck", Bitcraze. [en ligne]. Disponible : https://www.bitcraze.io/products/multi-ranger-deck/. [Accédé : 10-Fev-2021].

[2] "Flow deck v2", Bitcraze. [en ligne]. Disponible : https://www.bitcraze.io/products/flow-deck-v2/. [Accédé : 11-Fev-2021].

[3] St-Onge D., Varadharajan V., Svogor I. et Beltrame G., "From Design to Deployment : Decentralized Coordination of Heterogeneous Robotic Teams", Mai 2020. [en ligne] : https://www.frontiersin.org/articles/10.3389/frobt.2020.00051/full [Accédé : 9-Fev-2021]

[4] Bitcraze, "bitcraze/crazyflie-lib-python", GitHub. [en ligne]. Disponible : https://github.com/bitcraze/crazyflie-lib-python. [Accédé : 9-Fev-2021].

[5] The ARGoS website. [en ligne]. Disponible : https://www.argos-sim.info/. [Accédé : 9-Fev-2021].

[6] Vue.js. [en ligne]. Disponible : https://v3.vuejs.org/. [Accédé : 10-Fev-2021].

[7] "Crazyflie 2.1", Bitcraze. [en ligne]. Disponible : https://www.bitcraze.io/products/crazyflie-2-1/. [Accédé : 11-Fev-2021].

[8] "State estimation," Bitcraze. [en ligne]. Disponible : https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.04/functional-areas/state_estimators/. [Accédé : 03-Avr-2021].

[9] "Web technology for developers", Web APIs | MDN. [en ligne]. Disponible : https://developer.mozilla.org/en-US/docs/web/API/WebSocket. [Accédé : 10-Fev-2021].

[10] Xiang Chen, M. Salem, T. Das, and Xiaoqun Chen, "Real Time Software-in-the-Loop Simulation for Control Performance Validation," SIMULATION, vol. 84, no. 8-9, pp. 457–471, 2008.

[11] M. Bacic, "On hardware-in-the-loop simulation," Proceedings of the 44th IEEE Conference on Decision and Control.

[12] "Controllers in the Crazyflie", Bitcraze. [en ligne]. Disponible : https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.04/functional-areas/controllers/. [Accédé : 26-Fev-2021].

[13] Ilpincy, "Actuator quadrotor_speed methods dont affect drones' movements · Issue #160 · ilpincy/argos3" GitHub. [en ligne]. Disponible : https://github.com/ilpincy/argos3/issues/160. [Accédé : 9-Apr-2021].

9 Annexe

Table 1 – Liste de contrôle pré-vol

Vérifié	Étape	Description
	Porter l'équipement de protection	S'assurer que tout le monde porte des lunettes de protection.
	Démarrer le serveur	Démarrer le serveur à l'aide du script de départ.
	Ouvrir le client Web	Se connecter au serveur à l'aide d'un navigateur Web à l'adresse suivante : `<adresse IP>:3995`.
	Allumer les drones	S'assurer que tous les drones sont stables et au sol avant de les allumer.
	Valider la connexion	S'assurer que tous les drones sont connectés à la station au sol à l'aide du client Web.
	Valider la position des drones	Pour chaque drone, activer sa DEL afin de l'identifier. Ensuite, valider, avec la configuration, qu'il est bien placé par rapport à la base et orienté vers l'axe des X. La position reçue peut être consultée directement depuis les journaux sur le client Web.
	Valider la vitesse des drones	À l'aide du client, s'assurer que la lecture de la vitesse au repos de tous les drones est inférieure à 0.005 m/s. Si elles ne le sont pas, redémarrer le drone concerné.
	Sortir de la volière	S'assurer qu'aucune personne n'est dans la volière.

Table 2 – Liste de contrôle en vol durant les tests dans la volière

Vérifié	Étape	Description
	Vérifier que tout est sécuritaire	Vérifier que l'environnement de tests est sécuritaire. Si un drone se retrouve dans une situation dangereuse lors d'une mission, immédiatement ordonner un atterrissage d'urgence.
	Retour à la base	Après avoir envoyé une commande de retour à la base, vérifier que tous les drones atterrissent à moins d'un mètre de leur position initiale.
	Évitement d'obstacles	Vérifier que les drones réagissent à leur environnement et qu'ils n'entrent pas en collision avec les obstacles.
	Évitement de drones	Vérifier que les drones s'éloignent des autres drones lorsqu'ils se rapprochent trop.
	Génération de la carte	Vérifier que la carte générée concorde avec l'environnement exploré par les drones.
	Atterrissage d'urgence	Vérifier que les drones peuvent faire un atterrissage d'urgence à tout moment.
	Mission en boucle	Vérifier qu'il est possible de faire plusieurs missions en boucle sans devoir manuellement réinitialiser le drone.
	Informations des drones	Vérifier que les informations de vitesse, de la position, de la batterie, de la DEL et du statut des drones concordent avec l'état réel des drones.
	Batterie trop faible	Vérifier que les drones ne commencent pas une mission si au moins un des drones a un niveau de batterie sous 30%.
	Contrôle de la DEL	Vérifier que le contrôle de la DEL est possible.

Table 3 – Liste de contrôle post-vol

Vérifié	Étape	Description
	Vérifier les hélices	Vérifier que les hélices des drones sont stables et fixes.
	Vérifier le *Flow deck V2*	Vérifier que la lecture de la position des drones concorde avec leur position.
	Vérifier le *Multi-ranger deck*	Vérifier que la lecture de l'altitude de tous les drones est de 0.
	Recharger les drones	Éteindre les drones et les charger pour des futures missions.