
Dernière modification: 15 février 2021

INF3995: Projet de conception d'un système informatique
Hiver 2021
Réponse à l'appel d'offres

# *Hivexplore* - Système aérien minimal pour exploration

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

Équipe No. 2

Misha Krieger-Raynauld

Nathanaël Beaudoin-Dion

Rose Barmani

Samer Massaad

Simon Gauvin

Yasmine Moumou

# Table des matières

# 1 Vue d'ensemble du projet

## 1.1 But du projet, portée et objectifs

Principalement, le but du projet est de réaliser une preuve de concept pour l'Agence Spatiale démontrant que l'utilisation de robots 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 doit 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 débuter ou de mettre fin à une mission d'exploration et de mettre à jour le système à bord des drones. Aussi, la station doit afficher le nombre de drones connectés. Quant aux drones, ceux-ci doivent pouvoir explorer une pièce d'un bâtiment de dimension moyenne, éviter des osbtacles incluant les autres drones et communiquer entre eux. La mission d'exploration a comme 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.

Additionnellement, pour ce qui est du suivi du projet, trois livrables sont attendus : le « Preliminary Design Review », le « Critical Design Review » et le « Readiness Review ». Le « Preliminary Design Review » inclut la remise d'un prototype de base, d'une simulation simple ainsi que de la présente réponse à l'appel d'offres. Ensuite, le « Critical Design Review » consiste en la remise d'un système partiellement completé, mais implémentant la grande majorité des fonctionnalités. Finalement, le « Readiness Review », implique la remise du prototype complètement finalisé respectant tous les requis, un rapport technique détaillant le développement du prototype ainsi qu'une présentation orale du produit fini.

## 1.2 Hypothèse et contraintes

Avant de détailler le plan de la réalisation du projet, il importe de mentionner que celui-ci repose sur certaines hypothèses afin de simplifier l'étendue des connaissances préalables nécessaires. En premier lieu, nous présumons que la Crazyradio est capable de se connecter à plus d'un drone à la fois, chose qui nous sera nécessaire dans le but de diffuser les commandes à tous les drones et à récolter les données de chaque drone en simultané. En considérant que la documentation de la Crazyradio mentionne une portée d'un kilomètre, nous supposons également que la connexion avec les drones sera toujours fiable : la pièce mesure au plus 100 $m^2$. Les obstacles dans celle-ci ne devrait pas non plus représenter une entrave à la communication avec les drones.

Ensuite, nous devons aussi poser l'hypothèse que toutes les fonctionnalités physiques des drones et du *firmware* Crazyflie peuvent être reproduites fidèlement avec la simulation ARGoS. En effet, à première vue, ce ne sont pas tous les modules du *firmware* et de l'API Crazyflie qui sont disponibles dans ARGoS. Certaines fonctionnalités comme la communication *peer-to-peer*, le RSSI et l'API de *logging* et de *param* pourraient avoir à être réécrites pour les tester dans une simulation ARGoS. De même, dans l'optique où nous souhaitons réaliser le requis optionnel de l'affichage de la position des drones dans la station au sol, nous supposons aussi que les capteurs des drones nous offrent suffisamment d'information pour déterminer la position absolue de ceux-ci.

Sinon, en se tournant vers des considérations externes à l'équipe, nous devons aussi émettre l'hy-

pothèse qu'il sera possible de se transférer le drone entre les membres de l'équipe malgré les restrictions pour contrer la COVID-19.

Au-delà des hypothèses, plusieurs contraintes nous sont aussi imposées pour encadrer le projet. Une fois la mission commencée, soit une fois qu'ils reçoivent 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 connectée par USB (R.M.2). Pour les programmer et pour la communication P2P, l'API de BitCraze doit être utilisé (R.L.1, R.L.2). Aussi, l'utilisation d'un serveur pour la communication entre les drones est interdite (R.L.2). Il y a toujours un minimum de deux drones, mais un nombre arbitraire de drones peuvent être ajoutés (R.F.2). Leurs systèmes peuvent être mis à jour par l'API BitCraze seulement s'ils sont au sol (R.F.4.1). Au retour à la base, les drones doivent être à un mètre de la station au sol (R.F.4.2) et si leur niveau de batterie est inférieur à 30%, ceux-ci ne peuvent pas débuter une mission et doivent retourner à la base s'ils sont déjà en mission (R.F.4.3). La distance avec la station au sol doit obligatoirement être déterminée par la puissance du signal RSSI que la Crazyradio reçoit (R.L.3). Le prototype doit nécéssairement être fait avec le « STEM Ranging bundle » sur deux drones Bitcraze Crazyflie 2.1 (R.M.1) et seulement le « ranging deck » et le « optical flow deck » peuvent y être installés (R.M.3). Additionnellement, l'interface utilisateur doit pouvoir être visualisée sur divers appareils et les mises à jour des informations doivent être d'une fréquence de 1 Hz au minimum (R.F.5).

Pour ce qui est de la conception, il est obligatoire de faire une simulation ARGoS avant d'expérimenter avec les drones (R.C.1). Des tests unitaires et de régression doivent être faits (R.C.2, R.C.3) et il doit être possible d'effectuer une simulation sur ARGoS de notre système par une seule commande (R.C.4).

En bref, Le prototype final remis à l'Agence doit respecter tous les requis et contraintes stipulés par le contrat suite à la réponse de l'appel d'offres.

Enfin, il existe aussi des contraintes plutôt organisationnelles et externes à l'équipe. Nous sommes contraints à ne pas dépasser la limite de 630 heures-personnes et le projet s'échelonne sur dix semaines, du 1er février au 12 avril. De plus, nous devons avancer dans le projet en priorisant les requis spécifiés par chacune des trois remises du 15 février, 8 mars et 12 avril ; remises qui seront faites sur le répertoire Git. De plus, pour permettre à l'Agence Spatiale de prendre connaissance des avancements, un suivi du déroulement du projet devra être documenté. Finalement, dû au contexte actuel de pandémie, nous le projet doit être réalisé à distance et les membres de l'équipe devront communiquer à l'aide de diverses plateformes pour le travail collaboratif, ce qui limite l'utilisation des drones à un ou deux membres en même temps.

## 1.3 Biens livrables du projet

Au cours du projet, trois principaux livrables sont attendus. Le premier, soit le « Preliminary Design Review », est à remettre le 15 février 2021 et comprend la remise d'un prototype simple, d'une simulation ARGoS et la réponse à l'appel d'offres. Le prototype correspond à une station au sol avec une interface permettant de monitorer le niveau de batterie des drones et d'allumer ou éteindre leurs DEL. La simulation ARGoS doit, quant à elle, simplement simuler deux drones qui se déplacent avec une trajectoire quelconque.

Ensuite, la prochaine remise est celle du « Critical Design Review », attendue pour le 8 mars 2021.

Cette dernière implique la remise d'un prototype assez avancé. En effet, les commandes de communication aux drones pour commencer et terminer une mission doivent être implémentées et l'interface utilisateur doit permettre de monitorer les drones et de visualiser la carte générée. Ainsi, les drones doivent être en mesure d'explorer un lieu et de collecter des données pour générer la carte. Une simulation ARGoS de quatre drones qui utilisent des capteurs de distance pour éviter les obstacles et qui explorent une carte générée aléatoirement est aussi à remettre pour ce livrable.

Finalement, la dernière remise pour le 12 avril 2021, soit le « Readiness Review », correspond à la remise finale du prototype. Celle-ci doit respecter tous les requis et inclure une simulation ARGoS complète. Les fichiers remis par le biais du répertoire Git seront donc le code source du système embarqué, de la station au sol, de l'application Web et de la simulation ARGoS, ainsi que les fichiers de conteneurisation Docker et toute documentation. Accompagnant la remise du code sera également un rapport technique détaillé du développement du système et une vidéo mettant en scène le fonctionnement complet du système. Enfin, pour une présentation orale du prototype et du processus viendra clore ce dernier jalon du projet.

# 2 Organisation du projet

## 2.1 Structure d'organisation

Nous avons organisé le projet en *sprints* d'une semaine en s'inspirant de la méthodologie Agile. La remise de chaque sprint est le lundi midi, à l'heure du début de la séance de cours. Nous avons donc minimalement une rencontre par semaine le lundi après-midi pour clore le *sprint* de la semaine et débuter le prochain. Durant ces rencontres, nous avons un Google Doc partagé que tous les membres de l'équipe peuvent visualiser et modifier. Ce document contient les notes de nos réunions, soit les ordres du jour. Chacune de ces réunions est introduite par un court *standup* durant lequel chacun peut mettre à jour ses coéquipiers de ses avancements de façon formelle.

Les réunions de *sprint* du lundi se divisent en trois étapes : la rétroaction, le *grooming* et l'assignation des tâches pour le *sprint* à venir. La rétroaction correspond à un tour de table rapide de 20 minutes afin de mentionner les éléments s'étant bien ou moins bien passés pour apporter les correctifs nécessaires pour la suite. Ensuite, pendant l'étape de *grooming* de 20 minutes, nous réévaluons la répartition des tâches établie au début du projet, en fonction de ce qui a été accompli à présent, ce qu'il reste à accomplir et les nouvelles difficultés anticipées pour les prochains sprints du projet. Finalement, nous planifions le *sprint* à venir en 30 minutes en évaluant notre progrès, le temps mis sur les tâches du *sprint* précédent et les priorités pour la suite. Le tout en gardant un oeil sur les livrables et les requis à respecter sur le long terme. Nous documentons ces modifications sur le tableau de gestion GitLab de l'équipe.

En plus des rencontres du lundi, nous nous rencontrons le jeudi matin. Le but de cette rencontre, plus flexible, est de faire un autre suivi avec les membres pour s'assurer d'aider ceux qui sont en difficulté. En effet, nous avons établi qu'il est important de privilégier le déblocage des membres dont l'avancement des tâches est bloqué par une difficulté quelconque. Aussi, si d'autres points doivent être partagés avec l'équipe au complet, nous pouvons tenir des réunions le samedi.

Pour communiquer entre nous, nous utilisons la plateforme Discord. Dans celle-ci, nous avons créé plusieurs chaînes textuelles pour mieux organiser nos messages (réunions, ressources, annonces, etc.). De plus, dans le but de travailler ensemble, nous avons créé plusieurs chaînes vocales. Dans

celles-ci, les membres peuvent facilement rejoindre d'autres membres dans des discussions et demander de l'aide. Le tout afin de favoriser la collaboration et l'entraide au sein de l'équipe.

Pour ce qui est de notre méthode de travail, nous restons ouvert à la programmation en binôme pour le tâches plus complexes et les décisions architecturales. De plus, pour chaque *merge request*, deux personnes autres que son auteur doivent approuver la branche afin de pouvoir l'intégrer à la branche principale. Ainsi, nous nous assurons que la grande majorité des membres aient compris et approuvé les changements.

Finalement, pour une meilleure organisation et efficacité, nous avons décidé de répartir entre nous les rôles suivants : animateur, secrétaire, porte parole, avocat du diable, coordonnateur et maître du temps [11]. L'animateur, rôle pris par Misha Krieger-Raynauld, se charge d'identifier les points à discuter et de les inscrire dans l'ordre du jour. Aussi, il s'occupe de gérer les tours de parole et de s'assurer que l'on ne s'éloigne pas du sujet lors des discussions. Pour ce qui est du secrétaire, Samer Massaad, son rôle est de prendre des notes durant les réunions et d'organiser les documents produits, comme les ordres du jour. Quant au porte-parole, rôle assumé par Yasmine Moumou, ses responsabilités sont de représenter notre équipe lorsque nous devons contacter l'Agence afin d'obtenir des clarifications sur les requis. Le rôle de l'avocat du diable, détenu par Simon Gauvin, est de poser des questions et proposer des arguments qui vont à contresens des idées ou des propositions. Ceci permet de souligner les faiblesses, les incohérences ou les problèmes dans nos raisonnements ainsi que dans les solutions proposées afin de les éviter. Ensuite, le rôle de coordonnateur, détenu par Nathanaël Beaudoin-Dion, est de s'assurer que l'équipe respecte les échéanciers dans sa réalisation des tâches. Il s'assure de signaler tout retard et d'encourager l'équipe à aller de l'avant. En bref, il garde un oeil sur les dates de remises et rappelle l'équipe des objectifs à atteindre dans le temps. Cela nous permet de nous assurer que le projet avance au rythme prévu. Enfin, Rose Barmani a le rôle du maître du temps. Sa responsabilité principale est la gestion du temps durant les réunions. Le maître du temps assigne un certain temps aux points de l'ordre du jour durant la réunion et signale au groupe si ces limites ne sont pas respectées. Finalement, pour ce qui est du réseau de communication, nous favorisons un réseau décentralisé, c'est-à dire un pouvoir réparti entre tous les membres de l'équipes pour la prise des décisions et la direction générale. Ainsi nous bénéficions des connaissances de tous et faisons ressortir les meilleures idées. À travers tout cela, nous essayons le plus possible de prendre nos décisions par unanimité pour assurer une meilleure cohésion et satisfaction au sein des membres.

Bien sûr, en réponse aux conclusions des rétroactions hebdomadaires, notre organisation d'équipe est sujet à changement. Si une problématique est soulevée au courant du projet ou si une meilleure approche est mise de l'avant, il est possible que nous ajustions notre structure en conséquence.

## 2.2 Entente contractuelle

Pour le projet, nous proposons le type d'entente contractuelle livraison clé en main, aussi appelé contrat à terme. Ce contrat implique la livraison d'un produit final qui doit être accepté par le client avant que le paiement soit émis. Cette méthodologie correspond exactement à ce qui est attendu par l'Agence. Effectivement, il est très important pour l'Agence que nous remettions un prototype fini pour le 12 avril 2021, sans quoi la proposition sera rejetée. De plus, le contrat livraison clé en main implique une étude approfondie du déroulement du projet préalablement à sa réalisation. Il faut donc planifier et établir les défis technologiques à surmonter, les coûts, la durée, le matériel nécessaire, les acteurs impliqués, le mode organisationnel de l'équipe et plus encore à l'avance. Cet aspect permet une plus grande assurance par rapport au coût finaux et permet au promoteur de suivre le

déroulement du projet. Les critères au niveau du degré de risque pour les coûts et les échéanciers, l'incertitude et la complexité des spécifications, le niveau de de compétition, l'estimation des coûts et l'urgence de la livraison permettent de conclure que le contrat livraison clé en main est le plus approprié pour notre projet.

Dans un premier temps, le degré de risque pour les coûts et échéanciers est assez élevé. En effet, l'Agence Spatiale veut maximiser ses investissements afin de stimuler l'industrie spatiale avec leurs ressources limitées. Ainsi, il est important pour elle d'avoir une solution fonctionnelle et innovatrice au plus bas coût possible et en un délai fixe. Si la proposition du contrat dépasse la limite des coûts et de l'échéancier, celle-ci sera alors rejetée.

Ensuite, pour ce qui est du critère des requis techniques, l'appel d'offres envoyé par l'Agence décrit précisément la majorité des fonctionnalités essentielles à la solution proposée pour le projet. Cependant, un certain degré de liberté est permis à l'entrepreneur pour des requis comme la précision de la carte, puisque que ce projet reste après tout un prototype et agit en tant que preuve de concept.

En ce qui concerne le niveau de compétition, il est relativement faible considérant que, malgré que plusieurs équipes aient reçu l'appel d'offres, l'Agence se réserve le droit d'accepter plus d'un projet.

L'Agence nous laisse la liberté d'établir un budget et une estimation des coûts du projet. Cependant, ceux-ci doivent être basés sur des taux fixes horaires pour les développeur-analystes et le coordonnateur de projet avec un maximum de 630 heures-personnes non négociables. De surcroît aux contraintes matérielles nous limitant à des modèles spécifiques, le projet ne permet pas une grande flexibilité budgétaire.

Finalement, il est très important de réitérer que la remise du prototype fini à l'Agence doit se faire en date du 12 avril 2021. Cette date non discutable est importante à la planification du projet.

Ainsi à l'aide de ces cinq critères, nous pouvons facilement confirmer le type de contrat approprié pour ce projet. Un contrat livraison clé en main, aussi appelé contrat à terme, satisfait parfaitement les exigences présentées par l'appel d'offre de l'Agence Spatiale. En effet, la durée maximale du projet ainsi que son budget sont fixes et ne peuvent pas changer, sans quoi la proposition sera rejetée. Par contre, les requis importants sont connus d'avance et si l'entrepreneur veut modifier ces requis, une négociation sera alors requise.

# 3 Solution proposée

## 3.1 Architecture logicielle générale

L'architecture logicielle de la solution proposée comprend cinq composantes principales. Celles-ci sont représentées à la figure 1 par différentes couleurs pour les distinguer.

La première est un client Web développé avec le cadriciel Vue.js 3. L'équipe a choisi de développer un client Web pour des raisons de portabilité. Effectivement, il doit être possible de visualiser l'interface utilisateur sur différents appareils. Une application Web permet de réutiliser le même client pour un ordinateur et pour un appareil mobile, dans la mesure où celle-ci est conçu avec des composantes adaptatives (*responsive*).

La deuxième composante logicielle correspond au programme en C embarqué sur les drones. Les drones communiquent avec le serveur par la Crazyradio. Ils explorent un environnement et re-

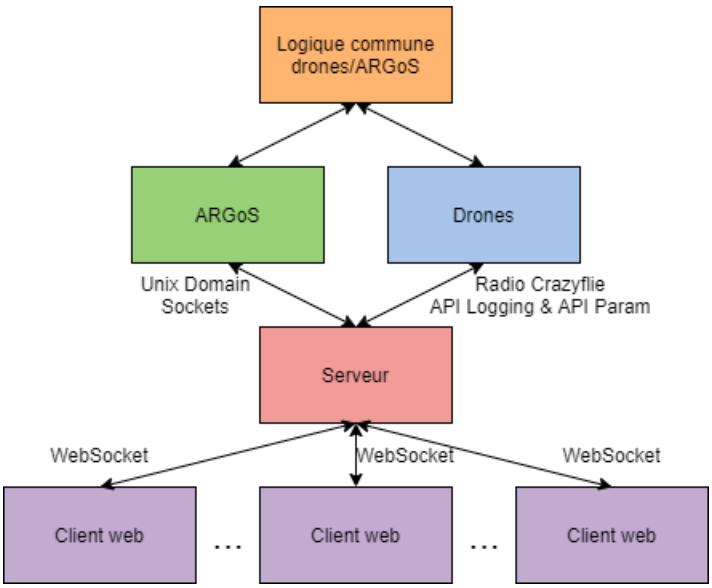



Figure 1 – Diagramme de l'architecture de la station au sol

cueillent des données avec leurs capteurs. Parallèlement, ils envoient ces données au serveur et reçoivent des commandes de celui-ci.

La troisième composante est la simulation des drones avec ARGoS. Cette composante logicielle nous permet de valider nos fonctionnalités dans un environnement simulé avant de les implémenter sur nos 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 quatrième composante logicielle est la librairie utilisée par le code embarqué des drones ainsi que par la simulation sur ARGoS (appelée « logique commune » dans la figure 1). Celle-ci s'occupe de factoriser la logique comportementale commune au code embarqué et au code de la simulation, sans dépendance spécifique aux particularités propres à ces deux environnements.

La dernière composante logicielle est le serveur. Celui-ci est connecté à la Crazyradio PA afin de communiquer avec les drones et communique par sockets Unix à la simulation ARGoS. Le serveur permet la connexion de plusieurs clients en même temps pour la visualisation des données et le contrôle des drones. Le serveur est en charge de recevoir et de traiter les données reçues des drones et de les envoyer aux différents clients Web connectés. Le serveur est aussi chargé de recevoir des commandes provenant des clients et de les transmettre aux drones pour changer l'état de la mission.

La communication entre le serveur et le client est facilitée par le protocole de communication WebSocket. Cette technologie est parfaite pour répondre au requis puisqu'elle 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. Ainsi, il devient facile de respecter le requis R.F.5 et de s'assurer que le serveur acquiert ses données à la fréquence minimale demandée de 1 Hz. Le client n'est donc pas responsable de ce requis.

La communication entre le serveur et la simulation ARGoS est gérée par des *Unix Domain Sockets*. Cette technologie 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.

Enfin, dans le but d'avoir un environnement reproductible et stable pour le produit final, chaque composante logicielle est conteneurisée avec Docker. L'installation et l'exécution de la solution s'en retrouve grandement simplifiée pour l'utilisateur final, ce qui permet de répondre au requis R.C.4. Il sera effectivement possible d'orchestrer les conteneurs pour les différents modules par l'appel d'une seule commande sur Linux.

## 3.2 Architecture logicielle embarquée

En se tournant plus particulièrement vers l'architecture spécifique du logiciel embarqué, on peut voir à la figure 2 que quatre des cinq composantes sont concernées, soit les drones, le serveur, la simulation ARGoS ainsi que la logique commune entre les drones et ARGoS. Une ligne en pointillé représente la démarcation entre la section ARGoS et la section des drones. Ainsi, si le contexte intéressé 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.

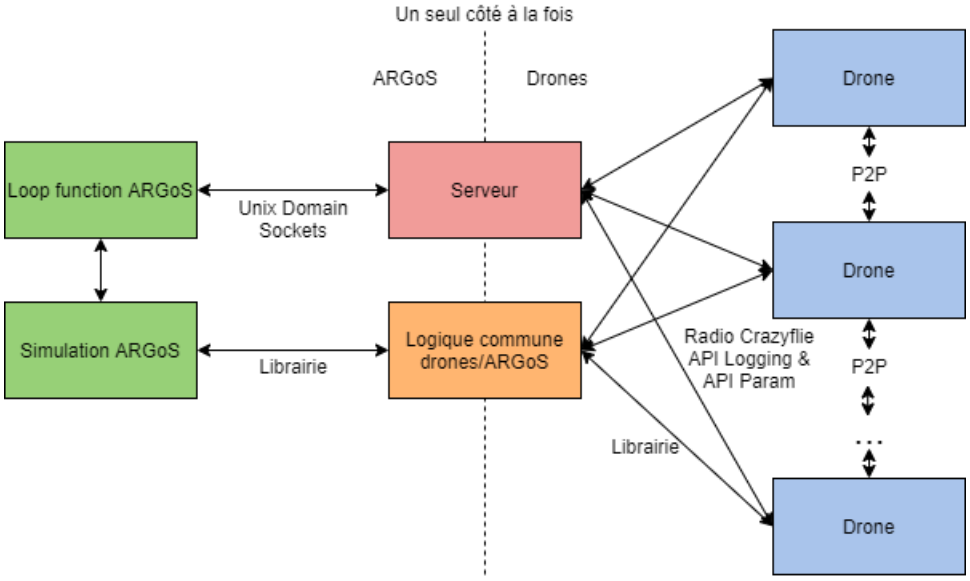



Figure 2 – Diagramme de l'architecture de la station au sol

En ce qui concerne la communication entre les drones et le serveur, selon nos hypothèses, nous supposons qu'un lien radio direct est toujours possible entre la station au sol et les drones. Ainsi, afin de simplifier l'implémentation, l'information de plusieurs drones est reçue en même temps à l'aide d'une radio par l'entremise du serveur. Ainsi, les drones poussent plusieurs fois par seconde leur état vers la station au sol, où, à l'aide de la radio, il est possible de recevoir les informations désirées pour 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 », elle est acheminée à tous les drones. Ceux-ci peuvent donc tous commencer la mission de façon autonome et simultanée. Du côté du *firmware*, ces envois et réceptions de messages sont implémentés à l'aide de l'API du *logger* de Crazyflie et des macros `ADD_PARAM`. Enfin, la mise à jour des informations des drones sur le client est gérée par l'entremise du serveur à l'aide de WebSocket. Ce choix technologique sera approfondi à la prochaine section.

Ensuite, pour faire l'exploration de l'environnement, la communication *peer-to-peer* (P2P) des drones sera mise à profit afin de faciliter la coordination de ceux-ci et éviter l'utilisation du serveur comme relais (R.L.2), puisque la dépendance à un système centralisé limite ce que peuvent accomplir une flotte de drones autonomes [4]. Cette communication peut être implémentée en utilisant l'API *peer-to-peer* de Crazyflie. Par chance, des exemples pour une simulation du P2P sont aussi disponibles pour la section sur ARGoS sur lesquels il sera aisé de se baser. La coordination avec P2P est puissante puisqu'elle délègue à l'essaim de drones beaucoup de liberté pour un algorithme distribué d'exploration.

Étant donné que les drones ont accès au « ranging deck » et au « optical flow deck », la combinaison de ces modules permet d'avoir la distance dans toutes les directions (haut / bas /gauche / droite / avant / arrière) ainsi qu'un système de positionnement local basé sur les variations de position.

Du côté de la simulation ARGoS, il va sans dire que celle-ci doit émuler le comportement 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 prone l'utilisation d'une logique commune pour le déplacement, la communication et la prise de décision. Pour ce faire, une librairie contenant toute la logique des drones peut être liée à la compilation. Les consommateurs de cette librairie - la simulation et le code embarqué - n'ont alors qu'à envoyer les données des capteurs à la librairie pour ensuite en récupérer les résultats pour enfin les traduire en appels à des fonctions de contrôle ou de communication. Ultimement, la librairie contiendra la logique essentielle du système et les consommateurs de celle-ci n'auront qu'à exécuter la décision prise par la librairie. Celle-ci sera écrite en C, par désir d'uniformité avec le code des drones qui est lui-même en C.

Afin de permettre une communication entre le serveur et la simulation ARGoS, ARGoS offre l'utilisation de *loop functions*, lesquelles rendent accessible un point d'entrée dans la simulation. Celui-ci permet d'insérer notre code qui s'occupera de la communication avec le serveur en agissant comme intermédiaire.

Un dernier aspect important de l'architecture est l'utilisation d'un code unique pour tous les drones, puisque cette solution est plus facilement maintenable avec un grand nombre de drones (R.F.2). Dans l'éventualité où une architecture qui requiert des drones avec des rôles spécifiques serait utilisée, n'importe quel drone pourrait assumer les différents rôles de la flotte : les rôles entre les drones pourraient être échangés durant l'exécution de la mission au lieu d'être décidés a priori. Suivant un processus itératif, les drones utiliseront dans un premier temps un algorithme d'exploration à parcours aléatoire en évitant les obstacles. Si le temps le permet, un algorithme plus avancé permettant une exploration plus efficace sera implémenté. Par exemple, les drones pourraient faire utilisation d'un *gossip algorithm* [4] ou d'une prise de décision par algorithme distribué.

### 3.3 Architecture logicielle de la station au sol

Pour l'architecture spécifique de la station au sol, on peut voir à la figure 3 que quatre des cinq composantes sont concernées : ARGoS, les drones, le serveur et le client. Le choix d'architecture et les protocoles de communications sont détaillés dans cette section.

D'abord, le serveur doit pouvoir choisir de communiquer soit avec les drones, soit avec la simulation ARGoS. Pour communiquer avec les drones, la librairie officielle pour Crazyflie, *cflib*, facilite la tâche. Bien qu'il existe des implémentations en d'autres langues, implémenter le serveur dans la même langue que la librairie officielle semble être un bon choix. Il a donc été choisi d'écrire notre serveur

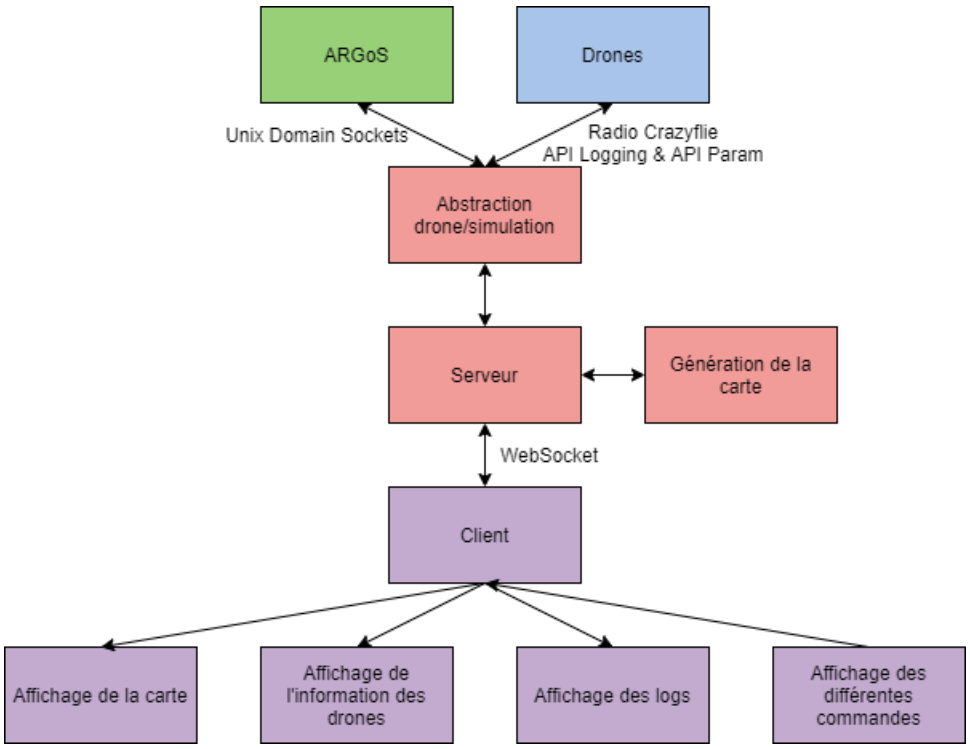



Figure 3 – Diagramme de l'architecture de la station au sol

en Python.

Le serveur est aussi en charge de la génération des points de la carte à partir de la combinaison des données de chacun des drones. En générant la carte sur le serveur, il peut être assuré que tous les clients aient la même carte. Pour notre premier prototype, la carte générée affichera simplement chaque point reçu. En procédant à des itérations sur nos algorithmes et en poursuivant certaines idées comme un algorithme de remplissage ou l'utilisation de l'estimation par la méthode des moindres carrés, un algorithme plus complexe et précis nous permettra de générer une carte plus fidèle à la réalité.

Ensuite, pour communiquer avec la simulation ARGoS, trois choix populaires ont été analysés : la mémoire partagée, les FIFOs (*pipes*) ainsi que les *sockets* Unix. Puisque le système doit pouvoir soutenir une acquisition de données à au moins 1 Hz (R.F.5), la performance de la communication n'est pas un facteur limitant. Ceci dit, l'équipe pense que l'utilisation des Unix *sockets* est le candidat idéal grâce à sa simplicité d'utilisation par rapport aux deux autres options. En étant bidirectionnels et conçus pour l'envoi de messages, la solution serait la plus adaptée pour simuler le comportement de *cflib* lorsque la simulation ARGoS est utilisée.

Pour ce qui est de la communication entre le client Web et le serveur Python, il y a deux options, soit les WebSocket et un API REST. Afin d'envoyer les données les plus récentes à tous les clients à un débit fixe de 1 Hz, l'utilisation de WebSocket est le candidat idéal pour la tâche. En effet, cette technologie permet d'établir une communication bidirectionnelle entre le client et le serveur, selon une interface simple d'utilisation.

Pour le choix de l'interface client, l'utilisation du cadriciel Vue.js 3 a été conservée. Ce cadriciel gagne en popularité ces dernières années alors que ses grands compétiteurs (Angular/React) stag-

nent après avoir atteint un plateau d'adoption [5]. Une raison est que Vue.js 3 offre une courbe d'apprentissage moins escarpée. Puisque la complexité du projet est principalement concentrée sur le code embarqué et sur le serveur, c'est un avantage d'avoir un cadriciel Web simple d'utilisation. Aussi, l'utilisation de Vue.js 3 est souvent recommandée pour des petits projets, car le cadriciel offre une expérience de développement élégante et simple [5]. La version qui est utilisée est la plus récente de Vue.js, soit Vue.js 3, qui est sortie en septembre 2020. Celle-ci nous offre un meilleur support Typescript, ce qui est pratique pour éviter des erreurs reliées à une mauvaise compréhension des types.

Finalement, une librairie d'interface graphique pour l'application Web est utilisée afin d'avoir une application visuellement attrayante. Celle que nous avons choisie s'appelle PrimeVUE : 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).

# 4 Processus de gestion

## 4.1 Estimations des coûts du projet

Pour réaliser notre projet, la charge requise est de 630 heures-personnes. Nous sommes une équipe de six développeurs-analystes dont un est aussi coordonnateur de projet. Le taux horaire pour les développeurs-analystes est de 130$/h et de 145$/h pour le coordonnateur de projet. Bien que, nous estimons que 60 heures totales seront allouées à la gestion de projet, nous calculons les coûts du projet en supposant que le taux horaire du coordonnateur de projet ne change pas en fonction de s'il effectue de la gestion ou du développement ; celui-ci reste à 145$/h. En divisant les heures parmi les 6 membres de l'équipe (soit 105 h/personne), nous obtenons un coût de 15 225$ pour le coordonnateur de projet et 68 250$ pour les 5 développeurs-analystes. Ceci équivaut à un coût total relatif aux ressources humaines de 83 475$.

Soixante des 630 heures-personnes sont allouées aux diverses tâches en gestion de projet et les 570 heures restantes seront consacrées au développement. Ceci revient à une moyenne d'un peu plus de 6 heures par sprint (semaine) de gestion et 63 heures de développement hebdomadaire réparti sur les neuf sprints.

## 4.2 Planification des tâches

Le diagramme de Gantt suivant détaille visuellement la planification et la répartition des tâches tout au long du projet. On y voit rapidement que la répartition des tâches pour le « Critical Design Review » (CDR) et le « Readiness Review » (RR) ne semble pas équilibrée. Deux raisons expliquent ce déséquilibre. D'abord, la majorité des requis et des fonctionnalités critiques doivent être implémentées pour le CDR. Il y a alors nécessairement plus de tâches à compléter. Ensuite, certaines tâches risquent d'être plus compliquées qu'anticipées et causer plus de problèmes, ce qui augmente le risque de délais. En nous laissant moins de tâches vers la fin du projet, nous nous assurons d'avoir suffisamment de temps pour tout imprévu que nous pourrions rencontrer afin de pouvoir garantir que le produit sera livrable pour le RR.

Légende pour l'assignation des tâches :

— MKR : Misha Krieger-Raynauld

— NBD : Nathanaël Beaudoin-Dion

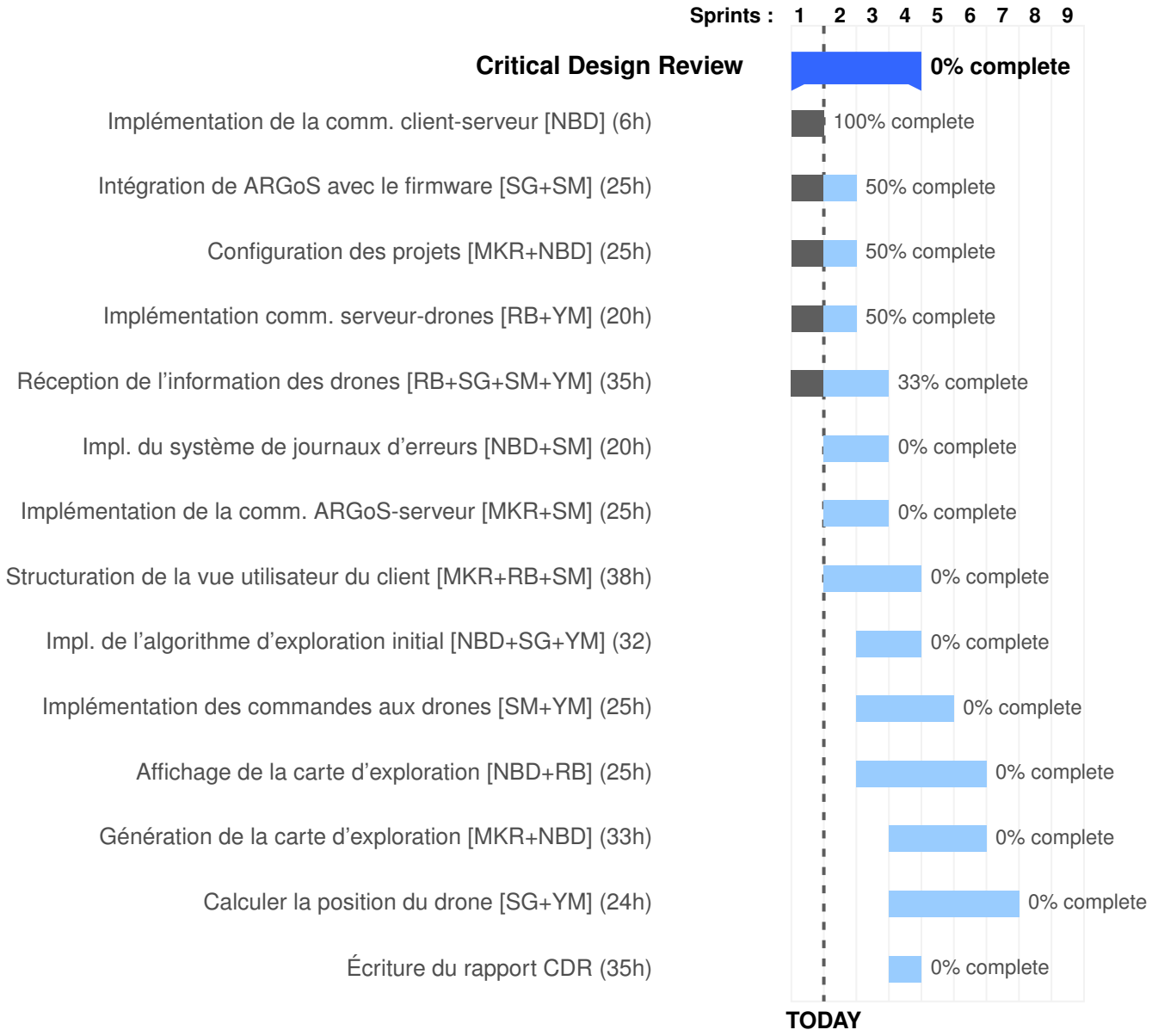
— RB : Rose Barmani

— SG : Simon Gauvin

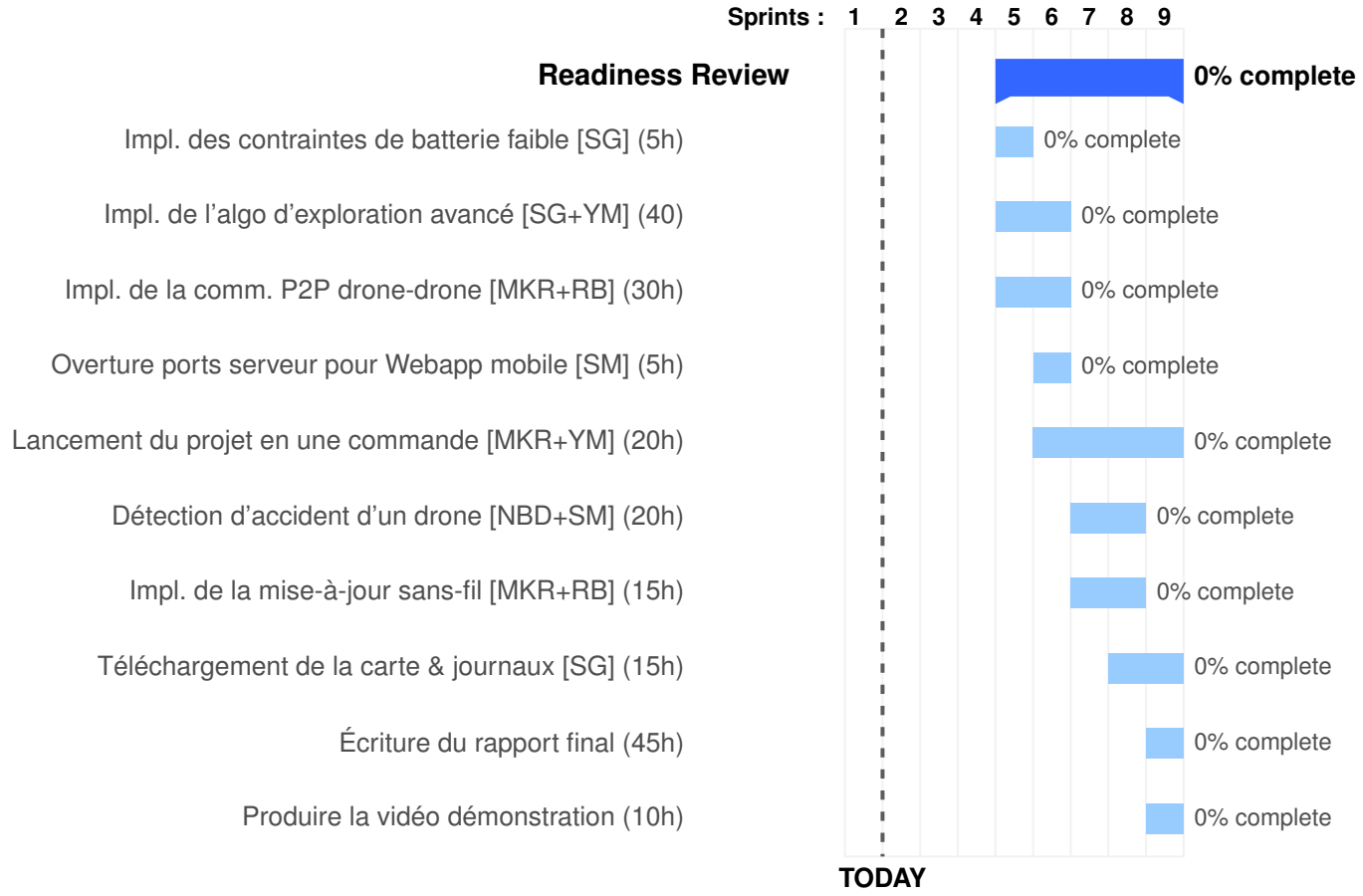
— SM : Samer Massaad

— YM : Yasmine Moumou

Ce premier diagramme met l'accent sur les tâches à commencer avant les livrables du CDR.




Le diagramme ci-dessous sert quant à lui à illustrer le travail débuté uniquement après le CDR, le tout dans le but de livrer le produit final pour le RR.




Au vu des fonctionnalités à réaliser pour le CDR, nous pensons qu'il pourrait être pertinent pour l'Agence de reconsidérer la date du CDR. En effet, celle-ci semble être trop tôt et il pourrait être judicieux de penser à la repousser.

Selon les diagrammes de Gantt, de nombreuses tâches qui sont commencées lors du CDR seront terminées lors du RR. Ceci complique le calcul des heures exactes qui doivent être faites pour le CDR. Cependant, selon nos estimations et nos heures inscrites sur le tableau GitLab, nous devons faire 314 heures de développement pour le CDR et 256 heures pour le RR. Les heures et l'organisation des sprints peuvent être consultées dans notre tableau GitLab nommé « Backlog » à l'adresse suivante : `https://gitlab.com/polytechnique-montr-al/inf3995/20211/equipe-102/hivexplore/-/boards/2361638`

Bien que le total d'heures soit conforme aux 570 heures allouées pour le développement et l'analyse, la charge de travail du CDR est relativement disproportionnée par rapport à celle du RR. Considérant que nous sommes rendus au sprint 2, ceci veut dire qu'il reste seulement trois semaines pour accomplir la majorité du travail pour le CDR, soit environ 270 heures, tandis que cinq semaines sont disponibles pour le RR pour faire 256 heures de travail. On voit qu'il y a donc un certain problème. En guise de comparaison, le CDR nécessite un rythme de travail de 270/3 = 90 heures/semaine alors que le RR n'en demande que 256/5 = 51 heures/semaine.

Selon l'équipe, la raison principale jusitifiant une charge de travail plus grande pour le CDR que le RR est la suivante. En général, ce qui est le plus long à faire dans un projet multi-technologique

comme celui-ci est de se familiariser avec les nouvelles technologies, de définir l'architecture, d'intégrer les divers modules, implémenter leur communication et avoir un résultat minimalement fonctionnel. Comme nous voulons être efficaces dans la manière dont nous travaillons, il est nécessaire de passer plus de temps au début à bien établir l'architecture de la communication entre les divers modules du projet pour avoir une base solide. Sinon, nous devrons faire plusieurs réusinages de code après le CDR, ce qui serait particulièrement inefficace. En théorie, le CDR n'est qu'un prototype qui doit prouver que les outils fonctionnent en harmonie, et seul le RR constituerait un produit fini esthétique.

Cependant, dans le cadre de ce projet, faire un prototype représente la majorité du travail. En guise d'exemple, prenons un des requis du livrable du CDR qui pourrait sembler banal à implémenter à première vue : l'affichage de l'information des drones.

Afin d'afficher les informations des drones - par exemple le niveau de battérie restante des drones sur le client Web - plusieurs éléments sous-jacents doivent d'abord être implémentés. En partant du code embarqué, le drone doit envoyer son niveau de batterie au serveur par l'entremise de la radio. Pour ce faire, nous devons donc apprendre comment l'API de communication entre les drones et un serveur Python fonctionne. Ensuite, par un WebSocket, le serveur doit envoyer cette donnée au client qui la reçoit aussi avec WebSocket. Puis, nous devons afficher correctement cette information sur le client à l'aide de Vue.js et une certaine librairie de composantes UI, un autre outil que nous devons comprendre et appliquer correctement. L'information est mainenant accessible. Ce n'est pas tout, cependant, car nous devons aussi faire que ce soit fonctionnel avec ARGoS. Pour ce faire, nous devons déterminer une architecture d'intégration entre le code des drones et le code sur ARGoS, pour que les deux puisse être interchangés pour les tests sans avoir deux codes complètement différents. Après avoir implémenté cette architecture, l'information de la simulation ARGoS doit être acheminée vers le serveur Python à l'aide de sockets Unix, une autre technologie à apprendre et à bien intégrer. Finalement, dans le serveur, il faut trouver un moyen de faire abstraction de la réception et de l'envoi de messages pour ARGoS et les drones. Une fois cette base implémentée, le niveau de batteries peut enfin être reçu à la fois avec les drones qu'avec la simulation ARGoS. Cependant il reste encore à tester toutes ces composantes et leurs interactions entre le serveur, le client, les drones et ARGoS, en plus d'apprendre trois cadriciels différents de tests.

Ainsi, pour faire un simple affichage du niveau de batterie des drones, nous avons toutes ces différentes choses invisibles au client qui doivent être implémentées et testées. C'est pourquoi les heures allouées pour le CDR sont présentement excessives. Nous suggérions donc à l'Agence de repousser le CDR d'une ou deux semaines, afin d'avoir un ratio d'heures/semaine plus équilibré tout au long du projet.

## 4.3 Calendrier de projet

Le tableau ci-dessous présente un résumé du calendrier général du projet, avec les dates de fin pour chaque jalon principal.

| Calendrier | |
|---|---|
| **Jalons** | **Dates** |
| Preliminary Design Review | 15 février 2021 |
| Critical Design Review | 8 mars 2021 |
| Readiness Review | 12 avril 2021 |

## 4.4 Ressources humaines du projet

D'abord, comme mentionné lors de la section 4.1, l'équipe est composée de six développeurs-analystes dont un membre est aussi coordonnateur de projet. Les membres sont tous des étudiants en génie informatique à Polytechnique Montréal ; cinq d'entre eux sont en troisième année et le sixième est en quatrième année.

Tous les développeurs ont une certaine expérience en développement Web et en programmation embarquée, deux compétences importantes pour ce projet. Au delà de l'expérience acquise lors du parcours scolaire, chacun a également des connaissances supplémentaires acquises lors de la réalisation de projets personnels, de participation dans des sociétés techniques ou lors de stages en entreprise.

En premier lieu, Nathanaël Beaudoin-Dion a une bonne expérience en embarqué. En effet, il est habile avec les mêmes technologies utilisées que dans ce projet, comme la programmation des microcontrôleurs de la famille STM32F4 en C/C++ et la gestion du système d'exploitation en temps réel avec FreeRTOS. De plus, lors de son dernier stage, il a acquis des connaissances dans plusieurs outils de développement comme GitLab, ses pipelines CI/CD, Docker et systemd-nspawn. Finalement, il a aussi une expérience en gestion de projet, étant le directeur du département d'avionique de la société technique Oronos.

Samer Massaad a une bonne expérience avec le développement de tests pertinents et robustes dans le contexte d'un projet Web et d'un serveur. En plus de son expérience en tests, durant son stage, Samer a acquis beaucoup de connaissances sur les différentes méthodes de communication possibles entre un serveur et son client. Pour l'aspect embarqué du projet, sa contribution sera pertinente avec son expérience au sein de la société technique Oronos dans le département d'avionique.

Simon Gauvin, de son côté, a des bonnes connaissances en C++ grâce à ses plusieurs projets personnels, son stage de recherche et à son expérience professionnelle dans l'industrie du jeu vidéo. Sa capacité de résolution de problèmes et de priorisation des tâches est la bienvenue dans l'équipe.

Yasmine Moumou, quant à elle, a de l'expérience en développement Web Angular, acquise dans le cadre d'un stage. Aussi, étant membre d'une société technique de robotique, elle a de l'expérience en apprentissage machine, plus spécifiquement en « computer vision ». Finalement, elle a aussi auparavant mis le pied dans le domaine du jeu vidéo à travers un projet personnel en C++.

Quant à Rose Barmani, elle est chargée de laboratoire pour le cours « Noyau d'un système d'exploitation ». De plus, elle a de l'expérience en gestion d'équipe, acquis au sein du comité Poly-FI. Aussi, son souci du détail est un atout indispensable à l'équipe. Finalemement, elle a également participé à la compétition de robotique CRC.

Enfin, Misha Krieger-Raynauld a beaucoup d'expérience en C++, langue qu'il a utilisée dans plusieurs projets personnels - dont un engin de jeux vidéo - et qu'il a enseignée en tant que chargé de laboratoire pour le cours d'INF1010. Il a également de l'expérience en Python et en développement Web qu'il a acquises à travers son expérience professionnelle en tant que consultant pendant 3 ans. Finalement, il a approfondi son expérience en gestion de projet et de l'utilisation de WebSockets lors de son dernier stage chez Microsoft. Ses connaissances de Linux, des outils de DevOps et les aspects plus poussés du C++ seront un atout à l'équipe.

# 5 Suivi de projet et contrôle

## 5.1 Contrôle de la qualité

Afin de répondre aux standards et requis exigés par l'Agence en ce qui concerne les biens livrables, notre équipe suivra un processus rigoureux permettant d'assurer le respect et le contrôle continu de la qualité tout au long du développement.

En premier lieu, nous misons sur la mise en place de divers outils de développement afin de prévenir des erreurs de qualité avant même leur inception. Lorsque pertinent, des outils d'analyse statique (*linters*) sont employés pour détecter des erreurs de style, de respect des bonnes pratiques et signaler les portions problématiques de code. Par exemple, du côté du client en TypeScript, nous employons ESLint ; du côté serveur en Python, nous avons opté pour Pylint. En parallèle, des outils de formattage automatique du code source garantissent une certaine uniformité au sein de l'équipe de développement pour nous permettre de consacrer du temps aux éléments plus critiques. À cet effet, nous utilisons Clang-format pour le C et le C++, Prettier pour le TypeScript, le HTML et le CSS, et enfin yapf et mypy pour le Python. Pour que ces outils soient faciles d'accès, ceux-ci sont intégrés au système de compilation (*build system*) et sont accessibles à même l'environnement de développement, soit VS Code dans notre cas.

Bien sûr, les outils d'assurance qualité pour le développement ne sont qu'utiles s'ils sont employés avec assiduité. C'est pourquoi ceux-ci sont appelés dans le pipeline CI/CD afin d'assurer que tout code proposé ait bel et bien passé ces vérifications statiques. Si le code proposé d'une *merge request* (MR) comporte des erreurs de *linting* ou de compilation, celle-ci ne pourra être acceptée jusqu'à ce que ceci soit résolu.

Dans cette même optique de contrôle de la qualité en prévention des erreurs, nous avons aussi opté pour des choix technologiques qui nous permettent de remarquer des erreurs le plus tôt possible. Du côté du client, le TypeScript a été choisi plutôt que le JavaScript puisque son typage statique nous permet de détecter des erreurs de programmation et d'augmenter la lisibilité du code. Du côté du serveur, nous utilisons mypy en conjonction avec des annotations de types afin d'assurer une certaine cohérence des types en Python.

Toutes ces mesures nous permettent donc d'affirmer avec certitude que les requis de qualité R.Q.1, R.Q.2 et R.Q.3 seront respectés.

Cependant, ces outils ne sont pas une panacée. Un processus de révision sur GitLab par chacun des coéquipiers apporte de grands bienfaits au niveau de l'intégration, de l'organisation du code, de la planification de l'implémentation et une vérification que chaque ajout est bel et bien la solution optimale pour chaque problème rencontré. Pour ce faire, une intégration très fréquente sera faite en tout temps à l'aide de *merge requests* (MR) atomiques, spécifiques et encapsulées. En intégrant souvent de plus petites modifications au lieu d'intégrer de grosses modifications à un plus lent rythme, notre équipe atteindra une mise en commun plus agile et harmonieuse, pourra facilement retracer des régressions, et ce qui sera sur la branche principale représentera en tout temps un produit déployable pour faciliter la rétroaction. Avec cette approche, les bogues et les problèmes d'intégration sont détectés le plus tôt possible dans le cycle du développement. Par le biais de ces révisions fréquentes de code, chaque membre de l'équipe aura la chance et la responsabilité (grâce à un nombre minimal d'approbations) de comprendre les différents aspects du projet. Ceci aura l'avantage de mener à une meilleure compréhension globale du projet par chacun pour avoir une solution absente de troubles d'intégration et de problèmes de cohérence.

Nous sommes donc confiants que les requis R.Q.3 et R.Q.4 seront bien respectés.

Du côté des techniques de développement, notre approche sera basée sur une utilisation extensive du simulateur ARGoS afin de tester le bon fonctionnement de nos algorithmes avant même de les charger sur les drones. Ceci permettra à chaque membre de l'équipe de contribuer au développement du code embarqué sans nécessairement avoir les drones en sa possession. Grâce à ceci, nous pourrons bénéficier de la contribution et des idées de chacun, menant à un meilleur produit qui pourra être validé par plusieurs membres de l'équipe lors des révisions.

De surcroît, une suite de tests unitaires couplée à notre pipeline CI/CD nous permettra de vérifier que des changements n'apportent pas de régressions et facilitera le processus de révision en y ajoutant une confirmation automatisée. Ce point sera approfondi plus en détail à la section 5.3.

En terminant, avant chaque remise de livrable, nous vérifierons que tous les objectifs ont bien été atteints en faisant appel à une liste de vérification contenant tous les requis ainsi que certains points additionnels. Cette liste de vérification pourra être conservée dans la section Wiki du répertoire GitLab.

## 5.2 Gestion de risque

En dépit de nos attentes et notre engagement à livrer un produit à la hauteur des attentes de l'Agence, il faut admettre que le projet comporte certains risques. Il faut avant tout mentionner le contexte inhabituel de la pandémie qui nous impose une méthode de travail à distance. Alors que certaines parties du projet s'en trouvent facilitées, par exemple pour ce qui est de la tenue des réunions, d'autres aspects pourraient s'avérer plus difficiles. On peut par exemple penser aux tests réels des drones dans leur environnement physique. En devant travailler à distance et en isolement les uns des autres, chaque membre de l'équipe n'aura pas un accès facile aux drones pour tester ce qui a trait au code embarqué. Pour pallier ceci, nous avons planifié que le drone puisse changer de mains pour certains sprints lorsque différents membres de l'équipe souhaitent travailler sur le code embarqué. De plus, notre utilisation poussée du simulateur ARGoS devrait nous permettre de valider une bonne partie du code sur les drones sans avoir à en être en possession, advenant le cas où le drone ne pourrait être transféré à un de nos membres.

Cependant, l'utilisation du simulateur ARGoS ajoute en elle-même une couche supplémentaire de travail. En effet, il pourrait nous être nécessaire de réimplementer certains modules du code embarqué Crazyflie dans ARGoS pour pouvoir tester nos drones correctement ; par exemple pour la communication peer-to-peer ou le calcul de la distance avec RSSI. Ce genre de travail pourrait ralentir le développement de fonctionnalités qui seraient autrement assez simples à mettre en place avec le *firmware* Crazyflie existant. Couplé à des inefficacités résultant du travail à distance et à des problèmes techniques, il se pourrait donc que certains requis ne puissent pas être livrés à temps pour le CDR. Bien sûr, tout risque anticipé serait communiqué à l'Agence dans les plus brefs délais.

Certaines de nos hypothèses pourraient aussi s'avérer fausses. Notamment, nous avons supposé que la Crazyradio est capable de rester en contact avec tous les drones en même temps, sans perte ou dégradation de signal. Si ceci se révèle être faux, nous avons un plan de contingence. Il serait possible d'utiliser la communication peer-to-peer pour acheminer les données à un seul drone qui, lui, serait à l'intérieur de la portée de l'antenne de la station au sol. De cette façon, les drones plus éloignés pourraient toujours envoyer leurs informations malgré l'absence de signal direct.

Un bris d'équipement pourrait aussi venir retarder le développement. Advenant le cas où un des drones serait mis hors d'usage, nous prévoyons continuer le développement à l'aide du simulateur et le drone restant, de sorte que le progrès ne s'en verrait pas ralenti le temps de commander des pièces de rechange.

Enfin, il peut également être pertinent de mentionner un risque au niveau de la santé et la sécurité : les hélices des drones représentent un danger pour les yeux lorsque les drones sont testés dans leur environnnement réel. Pour éviter toute violation des normes de la CNESST, chaque membre de l'équipe portera systématiquement des lunettes de sécurité lorsque les drones sont allumés et prêts à voler.

## 5.3 Tests

De concert avec les requis R.C.2 et R.C.3, notre solution logicielle et matérielle devra être accompagnée d'une suite de tests unitaires et d'une façon de détecter des régressions.

En premier lieu, nous utiliserons des librairies de tests unitaires lorsque pratique pour les fonctionnalités importantes du client, du serveur, et du code embarqué. Du côté du client, nous avons opté de tester notre application Vue.js en TypeScript à l'aide de Jest, que nous avons favorisé par rapport à une combinaison de Mocha + Chai par désir de simplicité. Pour le serveur en Python, nous avons retenu Pytest pour nos tests, puisqu'il nous offre une flexibilité maximale avec ses fonctionnalités modernes tout en étant facile pour les cas plus simples. Enfin, nous utiliserons le cadriciel de tests intégré au *firmware* Crazyflie [1] afin de tester les requis importants et l'intégration du code des drones à notre solution embarquée.

Grâce à ces cadriciels de tests unitaires, plusieurs fonctionnalités pourront être validées de façon unitaire. Pour le client, nous testerons l'interface et l'intégration WebSocket avec le client pour la mise à jour rapide des informations des drones (R.F.5), l'affichage de la carte à partir de la carte envoyée par le serveur (R.F.5), l'envoi des commandes de mission ainsi que l'état actuel de celle-ci (R.F.4) et l'envoi de la commande de mise à jour (R.F.4.1). Pour le serveur, nous testerons notamment l'intégration pour les communications avec le client, la communication et l'intégration avec la simulation ARGoS à l'aide de sockets Unix (R.C.1), la génération de la carte à partir des données des capteurs des drones (R.F.7) et la communication avec le système embarqué sur les drones avec l'API de logging et de paramètres de Bitcraze (R.M.2, R.M.4, R.L.5, R.L.6). Pour le drone, les fonctionnalités unitaires de la détection de l'état d'accident (R.F.5), l'algorithme d'évitement des obstacles (R.F.3), la portion individuelle de l'algorithme d'exploration en intégrant les informations peer-to-peer des autres drones (R.F.6) et l'algorithme de retour à la base (R.F.4.2, R.F.4.3) seront validées à l'aide des tests du *firmware* Crazyflie.

Cependant, les tests unitaires ne sont pas nécessairement adaptés à la validation de tous les requis. Dans certains scénarios, il peut être plus simple de procéder à une vérification manuelle dirigée selon une procédure préétablie sous la forme d'une liste de contrôle ; en particulier pour ce qui a trait aux éléments visuels ou à certains aspects d'intégration. C'est d'ailleurs le cas pour la présentation visuelle du client qui se doit d'être adapative (qu'elle fonctionne bien sur un téléphone, par exemple), que nous validerons à l'aide d'une procédure avec certaines tailles prédéfinies (R.L.4).

Nous utiliserons également la simulation ARGoS en priorité à la programmation sur les drones afin de tester le bon fonctionnement de notre solution avant tout essai avec les vrais drones dans leur environnement physique qui pourrait les endommager (R.C.1). Ceci permettra également à

chaque membre de l'équipe de contribuer au développement des drones sans nécessairement avoir accès à ceux-ci. Pour que nos simulations ARGoS servent également à tester de façon cohérente, reproductible et exhaustive, nous ferons encore une fois usage d'une liste de contrôle détaillant toutes les conditions à vérifier visuellement dans la simulation. Cette vérification pourra également être ajoutée en tant qu'étape manuelle du pipeline CI/CD de GitLab pour agir de rappel contre les oublis. Nous implémenterons aussi une génération aléatoire d'environnements dans la simulation ARGoS pour nous permettre de tester des situations plus rares et détecter davantage de problèmes rapidement (R.C.5).

Grâce à l'exécution et à la validation automatiques de ces tests en les intégrant à notre pipeline CI/CD sur GitLab, nos tests unitaires auront la double fonction de tests de régression. En effet, chaque nouvelle fonctionnalité devra permettre aux tests existants de continuer à passer, sans quoi le code ne pourra être intégré (R.C.5).

## 5.4 Gestion de configuration

Du côté de notre système de gestion de configuration, notre solution est organisée sous un même répertoire Git divisé en cinq sous-projets : le client, le serveur, le code embarqué pour les drones, la simulation ARGoS et la librairie commune. Nous avons opté pour le style *monorepo* - c'est-à-dire un seul répertoire contenant les sous-projets qui forment un tout - plutôt que différents répertoires pour plusieurs raisons. L'utilisation d'un seul répertoire diminue le risque que des changements asymétriques brisent accidentellement l'intégration entre les sous-systèmes à cause de *merge requests* séparées pour des modifications touchant à plusieurs projets. De plus, certains éléments communs et centraux tels que la configuration du pipeline CI/CD, les `README` de documentation, les configurations pour les outils de développement et l'intégration du tout sous un script de composition sont plus faciles à gérer sous un même répertoire plutôt qu'avoir des éléments dupliqués ou isolés d'un projet à l'autre.

Pour la documentation, notre projet mise sur les `README`s afin de communiquer toute information essentielle au développement et à l'utilisation de Hivexplore, organisés de façon hiérarchique pour donner que l'information pertinente à chaque projet. Pour de la documentation optionnelle plus poussée que nous souhaitons écrire à l'interne, nous ferons appel aux pages Markdown du Wiki de notre répertoire GitLab. C'est là que ce trouverons nos listes de contrôle pour nos tests manuels, nos documents de conventions, les informations supplémentaires pour certaines interfaces, ou enfin même des collections de liens et de ressources utiles.

Pour ce qui est du code source, nous priorisons avant tout le respect des conventions propres à chaque langue de programmation. Du côté du client, nous avons suivi la hiérarchie des projets Vue 3 telle que générée par le Vue CLI. Tous les tests se retrouvent intégrés au projet dans un dossier `tests`. Pour le serveur en Python, nous avons suivi l'organisation recommandée pour des *packages*, encore une fois avec son propre dossier `tests`. Enfin, les projets C et C++ suivent la structure déjà donnée par le code du *firmware* Crazyflie et des exemples de simulations ARGoS, respectivement, avec un dossier pour les tests unitaires du code embarqué.

Du côté de l'environnement de développement, nous avons opté pour une configuration intégrée de plusieurs outils à travers des mécanimes d'orchestration de commandes. Pour le client, nous utilisons les scripts npm, et pour le serveur, la simulation ARGoS et le code embarqué, des Makefiles de commandes permettent d'appeler tous les outils allant du compilateur jusqu'au formatteur. Plusieurs de ces outils sont également configurés pour fonctionner à même l'environnement de

développement, soit VS Code dans le cas des développeurs de notre équipe. Avec un *multi-root workspace*, des configurations de débogage et l'intégration des options d'extensions pour qu'elles fonctionnent avec notre projet, nous pourrons ainsi profiter de nos outils à leur plein potentiel et améliorer la cadence du progrès.

À ces outils s'ajoute enfin le choix d'utiliser Docker pour la conteneurisation. En effet, certains programmes comme ARGoS nécessitent beaucoup de dépendances qu'il peut être incommode d'installer sur une machine hôte. Pour ce cas, nous utilisons alors une image Docker de développement pour conteneuriser ARGoS pour qu'il soit identique peu importe la machine hôte et qu'il puisse s'intégrer à VS Code. Pour notre client Vue, les dépendances identiques sont assurées au moyen d'un `package-lock.json` et `node_modules` avec npm. Pour le serveur en Python, nous utilisons de façon analogue un fichier `requirements.txt` et un `venv` pour les dépendances.

La conteneurisation avec Docker des divers modules nous facilitera l'implémentation de l'objectif final, soit le démarrage aisé de l'entièreté de Hivexplore à l'aide d'un seul script sous Linux (R.C.4).

## Références

[1] "Unit testing," Bitcraze. [en ligne]. Disponible : `https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/development/unit_testing/`. [Accédé : 12-Fev-2021].

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

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

[4] 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]

[5] S. Daityari, "Angular vs React vs Vue : Which Framework to Choose in 2021," CodeinWP, 18-Dec-2020. [en ligne]. Disponible : `https://www.codeinwp.com/blog/angular-vs-vue-vs-react/`. [Accédé : 7-Fev-2021].

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

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

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

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

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

[11] "Les rôles et leurs avantages," Travailler en équipe | Attribuer des rôles dans l'équipe | Les rôles et leurs avantages. [en ligne]. Disponible : `https://ernest.hec.ca/video/DAIP/travailler-en-equipe/etudiants/attribuer-roles/roles.html`. [Accédé : 13-Fev-2021].