POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

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

Conception d'un système aérien minimal pour exploration

Équipe No 200

Babin
Lauzon
Lharch
Martineau
Quiquempoix

15 Février 2021

Table des matières

1. Vue d'ensemble du projet

1.1 But du projet, porté et objectifs

Le but de ce projet est de créer un ensemble de logiciels qui permettront à un essaim de drones miniatures de cartographier l'espace d'un bâtiment. Les drones vont cartographier la salle grâce à des capteurs laser et doivent se déplacer de manière autonome. Les drones utilisés sont des Bitcraze Crazyflie 2.1 qui communiquerons entre eux en *peer-to-peer* (P2P) et avec une station au sol munie d'un Crazyradio PA.

La station au sol hébergera une interface web qui permettra d'afficher la cartographie de la salle et d'envoyer des commandes basiques aux drones. L'objectif général du projet est d'avoir un essaim de drones autonomes capable d'explorer l'espace d'un bâtiment et de retourner les données de l'exploration à une console centrale.

Au début de ce projet, nous avons commencé par la PDR (Preliminary Design Review). La PDR consiste à remettre un prototype minimal avec une simulation ARGoS de deux drones qui suivent un parcours. Ce livrable nécessite aussi de remettre une interface Web de la station au sol qui montre le niveau de batterie et qui communique avec les drones pour allumer et éteindre leur DEL (vous pouvez trouver en annexe une vidéo de démonstration de prototype).

Pour le deuxième livrable, qui est le CDR (Critical Design Review), l'objectif est de remettre un système avec toutes les fonctionnalités demandées pour la simulation. Ainsi, nous aurons une simulation des drones qui volent dans un environnement aléatoire en évitant les obstacles grâce à des capteurs lasers. Le système comportera aussi un serveur Web avec les commandes pour décoller et revenir à la base. Le serveur web devra aussi afficher les informations des drones ainsi que la carte générée par ces drones.

Enfin, le dernier livrable est le RR (Readiness Review). L'objectif de ce livrable est de remettre un système complètement fonctionnel avec tous les logiciels et la documentation. Nous implémenterons donc le code des drones simulés sur les drones physiques. De plus, nous aurons tous les boutons demandés sur la partie web.

1.2 Hypothèses et contraintes

Notre première hypothèse, est que nous allons avoir besoin de seulement quatre modules : une interface web, un backend sur la station au sol, une programmation embarquée sur chacun des drones physiques et la simulation des drones.

Nous planifions faire l'interface web en Angular puisque nous possédons déjà des connaissances sur cette plateforme et cela facilitera le développement de ce module. Ensuite, pour ce qui est du backend, il sera responsable de

collecter les données que les drones vont envoyer, de faire les calculs pour la génération de la carte et de transférer les données transformées à l'interface web. Le backend sera codé en python, car nous voulons qu'il soit performant et simple à développer avec un langage facile d'utilisation. Enfin, nous pensons que le logiciel pour les drones (le système embarqué) sera en C/C++. De plus, la simulation sera elle aussi programmée en C/C++.

Pour ce qui est des contraintes, la station au sol devra être capable de communiquer avec les drones par l'intermédiaire de la *CrazyRadio* PA sur un canal de communication de 2.4 GHz. Nous supposons qu'au moins un drone est toujours en communication avec la *CrazyRadio* PA de la station au sol. De plus, la station au sol devra aussi héberger une interface web qui servira d'interface pour l'utilisateur. Cette interface web devra être visualisable par plusieurs appareils (ordinateurs, appareils mobiles, etc.) en même temps via le réseau. Cette interface devra afficher des informations sur les drones, afficher la carte générée par les drones en temps réel et donner quelques options à l'usager pour le contrôle des drones : décoller, atterrir, mettre à jour le logiciel des drones et retour à la base. Les informations qui devront être présente sur l'interface web devront être mis à jour à une fréquence minimale de 1 Hz. Pour la réception et le traitement des informations recueilli par les drones, elles devront être effectuées par la station au sol. De plus, pour faciliter la génération de la carte, la position et l'orientation des drones au début de la mission doivent être connues par la station. De plus, la distance des drones par rapport à la station au sol devra être estimée à l'aide de la puissance du signal (RSSI) reçu par le *CrazyRadio* PA. Le backend devra générer des registres pour permettre la vérification du bon fonctionnement du système (collecte et traitement des données).

Le prototype doit être réalisé à l'aide de deux drone *Crazyflie* 2.1 avec le « *STEM Ranging bundle* » complètement installé. Toutefois, le système doit être capable de supporter un nombre arbitraire de drones. Les drones devront avoir une programmation embarquée pour fonctionner automatiquement sans contrôle de l'utilisateur une fois que la mission est commencée. L'algorithme d'exploration n'a pas de contrainte, mais nous planifions utiliser un algorithme d'exploration aléatoire. Une fois la mission commencée, les drones devront être capables d'explorer une pièce d'une superficie maximale de $100\,m^2$ et devront pouvoir éviter les obstacles automatiquement. Un seul drone sera constamment connecté à la station au sol à l'aide de la *Crazyradio* PA. Les autres drones devront faire de la communication *peer-to-peer* à l'aide de la librairie P2P de *BitCraze*. Enfin, une contrainte que nous pourrions avoir serait un problème de surcharge si plusieurs personnes se connectent à l'interface web. On pourrait avoir un problème de surcharge si trop de clients font de requêtes en même temps, le serveur ne pourrait potentiellement pas pouvoir répondre à tout le monde avec des délais raisonnables. On pourrait aussi avoir des problèmes de concurrence si deux clients s'amusent à débuter et arrêter la mission en même temps.

1.3 Biens livrables du projet

Nous allons générer plusieurs artefacts durant le projet. A chaque livrable, nous avons un certain nombre d'artefacts à remettre. Pour le livrable de *Critical Design Review* (CDR), le premier artefact à remettre est le serveur web interfacé avec la simulation *ARGoS* affichant les différentes métriques des drones et la carte générée par les drones. Le deuxième artefact est le code embarqué sur les drones qui envoie les mesures du *ranging dec*k à la station au sol. Le dernier artefact est la simulation *ARGoS* des 4 drones qui explore un environnement tout en évitant les obstacles. Les artefacts du premier livrable sont à remettre le 8 mars 2021. Pour le livrable final, nous avons tous les logiciels, embarqués et non embarqués, avec la documentation comme premiers artefacts. Ensuite, nous devons remettre les drones qui explorent un environnement en évitant les obstacles ainsi que l'interface web a la station au sol qui affiche la carte. Enfin, une vidéo qui montre le fonctionnement du système en simulation et avec les vrais drones. Ces artefacts sont à remettre pour le 12 avril 2021.

2. Organisation du projet

2.1 Structure d'organisation

Notre équipe se compose de 5 membres. Nous avons un coordonnateur de projet qui est Philippe Babin et 4 développeurs-analystes. Nous avons décidé que le rôle de coordonnateur est purement symbolique puisque notre équipe travaillera en réseau décentralisé. Toutefois, le coordonnateur du projet est celui qui a les drones et qui va s'occuper des taches qui demandent la manipulation des drones. Les autres membres de l'équipe vont s'occuper du frontend, backend et aussi la partie de simulation avec *ARGoS*.

2.2 Entente contractuelle

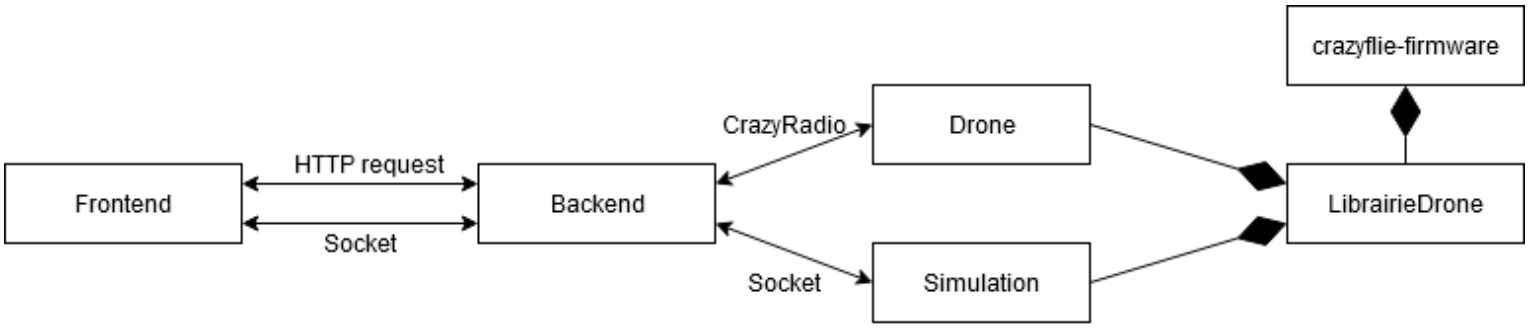
Dans le cadre de ce projet, le type de contrat qui convient le mieux avec le contexte est le contrat livraison clé en main – Prix ferme. Nous avons choisi ce type de contrat, car c'est un contrat ou le contracteur doit livrer un produit final et le paiement se fait après la livraison du produit par le contracteur et l'acceptation du client. Dans notre cas, le produit final est un système qui se compose de drones avec un logiciel embarqué, une interface web et enfin un logiciel *backend* qui s'occupe de générer la cartographie. Nous devons aussi faire une estimation des couts du projet avec un budget en fonction des lots de travail et une limite pour la charge de travail du projet qui est 630 heures par personne, ce qui veut dire que le prix du projet est ferme. Pendant ce projet, nous avons deux livrables à remettre donc seulement deux dates cibles et ce type de contrat requiert un

suivi minimal des travaux par le promoteur, ce qui explique notre choix. Enfin, pour le contrat livraison clé en main – Prix ferme, nous avons besoin de connaitre exactement ce qui est demandée et d'avoir des spécifications détaillées pour pouvoir obtenir le contrat. L'appel d'offre et le document des exigences techniques nous donnent toutes les spécifications dont nous avons besoin pour faire ce projet. [1]

3. Solution proposée

3.1 Architecture logicielle générale

Figure 1: Diagramme générale du système



Pour concevoir le système, nous l'avons divisé en 3 gros groupes. Le premier est la partie *frontend*. Le frontend est la partie web du système. Il s'occupe de l'affichage, et ainsi il affiche les données des drones ainsi que la carte.

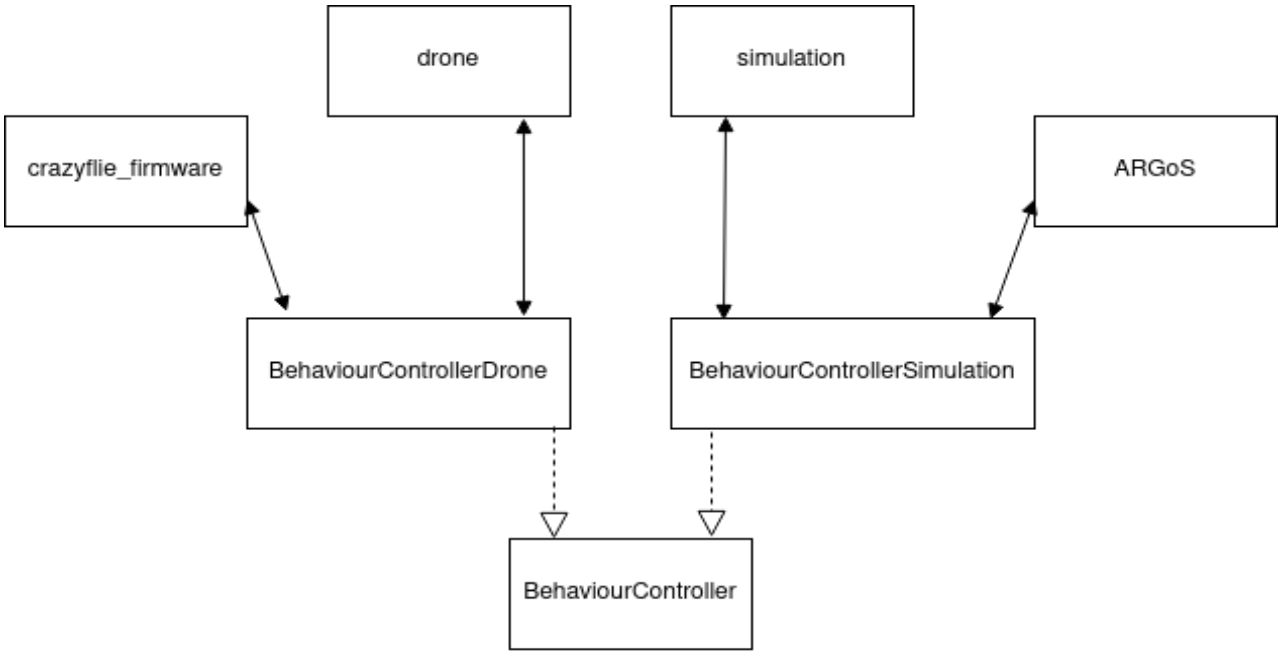
Ensuite, nous avons la partie drone et simulation. Ce module s'occupe de contrôler les drones que ce soit dans la simulation ou en réel. Nous avons aussi décidé de faire une librairie (*LibrairieDrone*) qui permet de rassembler le code contrôlant les drones en réel et dans la simulation. On rassemble le code afin de réduire la répétition car le contrôle des drones est similaire entre les drones physiques et les drones simulé. La partie simulation permet aussi de générer aléatoirement un environnement virtuel *ARGoS* afin de tester le déplacement et la cartographie de la salle des drones.

Finalement, nous avons la partie *backend*. Ce module s'occupe de faire la liaison de communication entre le *frontend* et les drones (ou la simulation). Il communique avec le frontend de deux manières. La première est avec un socket, ce qui va permettre d'envoyer rapidement les données des drones ainsi que les points de la carte a toutes les secondes. La deuxième communication avec le frontend se fait à travers des requêtes HTTP. Ces requêtes seront utilisées pour communiquer les commandes aux drones comme le retour à la base. De plus, le module backend communique avec les drones grâce à la *CrazyRadio.* Avec la simulation, puisqu'il n'y a pas vraiment de drones ni de

CrazyRadio, on simule la communication avec un socket afin de transférer les données entre la simulation et le backend de manière rapide et efficace.

3.2 Architecture logicielle embarqué

Figure 2: Diagramme logicielle embarqué



Pour ce qui est de l'embarqué, soit la simulation et le drone, nous avons opté pour une architecture ressemblant au schéma plus haut. Ces deux côtés seront reliés à une librairie qui permettra une certaine abstraction par rapport aux commandes exécutées par les robots. Cette librairie fera la même action, mais avec une implémentation différente pour la simulation et le drone physique. Nous pourrons donc réutiliser le même code pour ces deux parties. Elle aura aussi besoin de deux autres librairies qui permettent d'interagir avec les robots, soit crazyflie_firmware pour le drone physique et ARGoS pour la simulation. Nous devrons nous créer une liste d'actions faisables par la simulation et le drone et les utiliserons pour créer des mouvements plus complexes. Une fois cette librairie terminée, nous allons pouvoir utiliser presque exactement le même code sur la simulation et sur le drone.

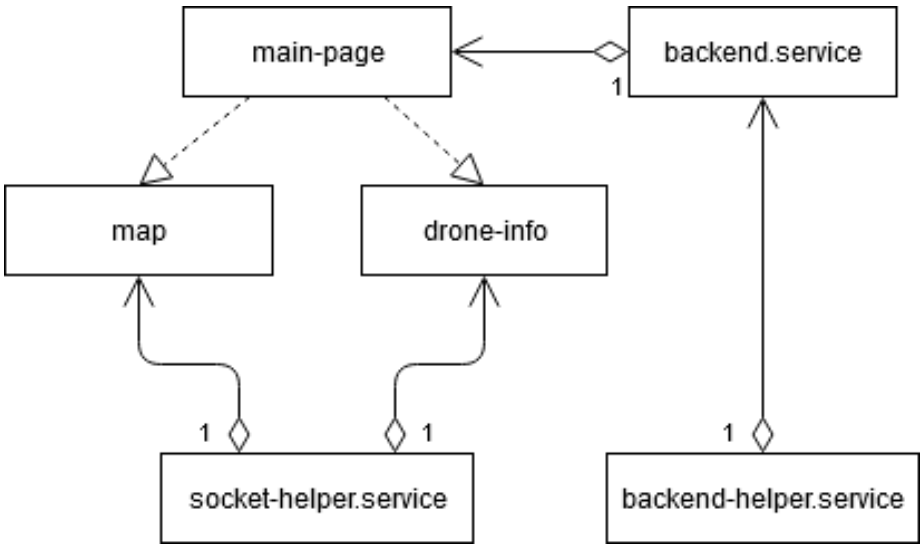
3.3 Architecture logicielle station au sol

Pour le logiciel de la station au sol, nous avons décidé de séparer la tache en deux. En premier, nous avons une application Angular 11, qui s'occupe de faire l'affichage de la carte et des informations du drone. Cette application

Angular est donc l'application web du projet. Ensuite, nous avons décidé d'implémenter un *backend* en Python. Ce *backend* va nous permettre de faire la liaison entre l'application web et les drones ou la simulation. Il va nous permettre de faire les calculs des points de la carte que ni les robots, ni l'application web ne peuvent faire de manière efficace.

On peut alors observer deux diagrammes, le premier pour la partie *frontend* et le deuxième pour la partie *backend*.

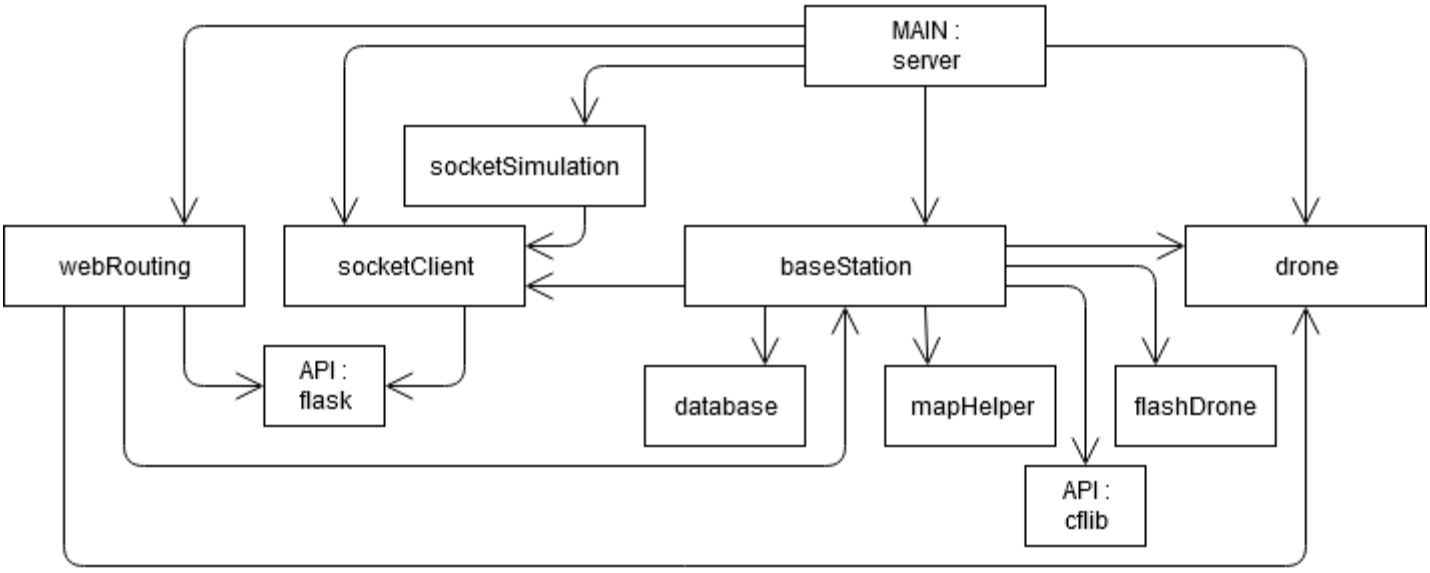
Figure 3: Diagramme station au sol, partie frontend



Pour la partie frontend, nous aurons une page principale, nommée *main-page*. Cette page principale comporte deux objets qui sont la carte (*map*) et les informations des drones (*drone-info*). Ces deux objets s'occuperont d'afficher respectivement la carte et les informations des drones comme leurs niveaux de batterie par exemple. Nous avons décidé de séparer la carte et les informations des drones dans des composants différents car cela leur permet d'être réutilisable à travers d'autres pages de l'application si le besoin en est.

Ensuite, nous avons les trois services qui sont *backend.service*, *backend-helper.service* et *socket-helper.service*. Ces trois services vont nous permettre de communiquer avec le serveur *backend*. Nous avons créé ces services afin de séparer la partie communication de la partie affichage. Nous avons aussi séparé car ces communications peuvent être utilisées à travers tous les différents modules de l'application, et qu'il est préférable de centraliser le code de communication dans un service à part. On retrouve donc deux services principaux distincts qui sont *backend.service* et *socket-helper.service*. Le premier permet de gérer les requêtes HTTP et le deuxième permet de gérer le socket. Les requêtes HTTP vont donc nous permettre de communiquer les commandes aux drones (débuter mission, revenir à la base, atterrir et mise à jour logicielle). Le socket va nous permettre de recevoir plus rapidement les données à mettre à jour toutes les secondes. Ces données sont les informations des drones ainsi que les points de la carte à générer.

Figure 4: Diagramme station au sol, partie backend



En ce qui concerne la partie du serveur (*backend*), nous avons décidé d'avoir une fonction principale (*server*) que l'on peut voir sur le diagramme avec l'annotation MAIN. Cette fonction principale s'occupe alors de lancer les deux sockets, qui sont pour la communication avec le client (*socketClient*) et pour communiquer avec la simulation ARGoS (*socketSimulation*). Le socket de *socketClient* utilise l'API de *flask* afin de gérer les sockets avec *Socket.io*. Nous avons choisi d'utiliser *flask,* car c'est simple et rapide d'utilisation et il permet de communiquer avec le socket de *socket.io* qui est sur le *frontend*. Le socket de *socketSimulation* utilise les librairies socket natives de Python, car nous n'avons pas à parler avec une instance de *socket.io*. De plus, nous avons aussi le module *webRouting* qui utilise aussi l'API de *flask*. *webRouting* s'occupe de gérer les requêtes HTTP faites par le frontend. Il permet de lier les requêtes aux différents modules du serveur.

Ensuite, nous avons la partie du *baseStation*. Ce module nous permet de gérer les commandes qui seront envoyée aux drones. Il permet de recevoir les données des drones, puis s'occupe de les envoyer aux modules correspondants. Le *database* s'occupe d'enregistrer les données en format csv. Le *mapHelper* permet de trouver les coordonnées des points de la carte grâces aux distances qu'envoie le drone. *flashDrone* nous permettra de flash les drones avec du nouveau code. Et finalement, *socketClient* enverras les données des drones vers le frontend à travers le socket. De plus, le module *baseStation* utilise la librairie de bitcraze *cflib*, ce qui nous permet de contrôler l'antenne *CrazyRadio*, et de communiquer avec les drones.

4. Processus de gestion

4.1 Estimations des coûts du projet

Après la planification et la répartition des tâches entre les membres, nous avons défini le nombre d'heures pour chaque membre de l'équipe. Le coordonnateur du projet va passer approximativement 115 heures et 30 minutes pour le projet et avec un budget de 145 $ par heure nous avons un total de 16 747.5 $. Pour les développeurs-analystes, nous avons un total de 427 heures. Le budget pour les développeurs-analystes est de 130 $ par heure, donc le total est de 55 510 $.

Le budget total du projet est donc de 72 257 $.

4.2 Planification des tâches

Tableau 1: Planification des tâches

Tâches	Temps alloué en heures	Personne(s) responsable	Pour quelle remise.
Drones :			
Éviter les obstacles, et les autres drones.	13	Nour et Theo	CDR
Communiquer avec backend (un seul des drones. Master and slaves ?) (Fréquence de 1Hz pour communiquer les distances)	20	Nicolas	CDR et RR
Communiquer en P2P	20	Philippe et Nour	CDR et RR
Algorithme d'exploration	20	Theo et Nour	CDR
Décoller / Commencer la mission	8	Nicolas	CDR
Atterrir finir la mission	8	Nicolas	CDR
Mise à jour logicielle	33	Philippe et William	RR
Revenir à la base.	20	William	CDR
Implémentation du code embarqué sur les drones physiques.	20	Philippe et Theo	RR

Retour à la base à 30% de batterie	3	William	CDR
Total pour les drones:	165		
Serveur :			
Récupérer les points des drones (CrazyRadio)	20	William, Philippe et Theo	CDR et RR
Transférer les points au frontend	3	Theo	CDR
Récupérer les données des drones (autre que les points) (CrazyRadio)	13	William et Philippe	CDR et RR
Générer des logs pour assurer le bon fonctionnement	13	Nicolas Lauzon	RR
Enregistrer les points de la carte	5	Nour et Nicolas	CDR
Communiquer les commandes aux drones	13	Philippe, Nicolas et Theo	CDR et RR
Envoyer les commandes aux drones (4 commandes) lorsque le frontend le demande. (Ajouter l'API).	3	Nour	CDR
Calculer la position du mur avec les infos reçues par le drone.	13	Nicolas et Theo	CDR
Estimer la distance des drones avec la puissance du signal (RSSI)	20	Nour et Philippe	RR
Total pour le serveur:	103		
Frontend :			
Récupérer les données a toutes les 1 secondes	2	Theo	CDR
Afficher la carte (frontend)	33	Theo et Nicolas	CDR et RR
Ajouter les boutons de commande (les 4 boutons)	5	Nour	CDR
Afficher les informations des drones (tableau)	3	Nour	CDR

Total pour le frontend:	43		
Docker :			
Créer / maintenir script d'automatisation	33	Philippe, William, Theo, Nicolas et Nour	CDR et RR
Mettre en place les dockerfiles / docker-compose	20	Theo, Philippe et William	CDR et RR
Simulation :	20		
Générer la simulation Argos aléatoirement (salle aléatoire).	20	William et Nour	CDR
Total pour les Docker:	53		
Rapport :			
Rédiger le CDR	53	Philippe, William, Theo, Nicolas et Nour	CDR
Rédiger le RR	53	Philippe, William, Theo, Nicolas et Nour	RR
Écrire les rapports hebdomadaires.	20	Philippe, William, Theo, Nicolas et Nour	À faire à chaque remise
Total pour les rapports:	126		
Autres :			
Faire des tests de régression.	33	Philippe, William, Theo, Nicolas et Nour	CDR et RR

Figure 5: Répartition du temps en heure pour chaque membre de l'équipe

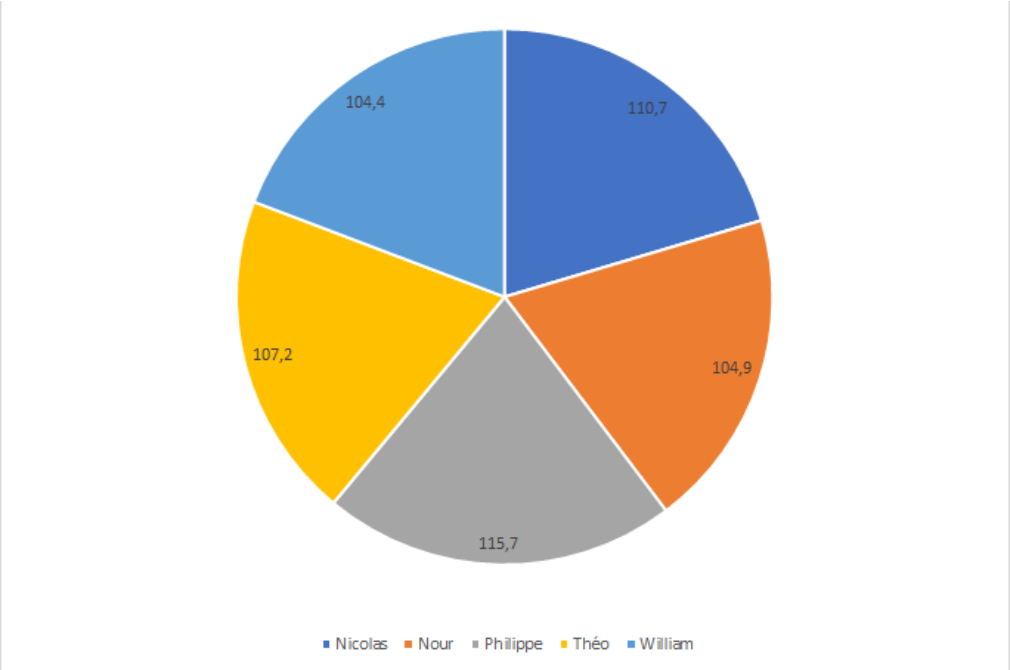
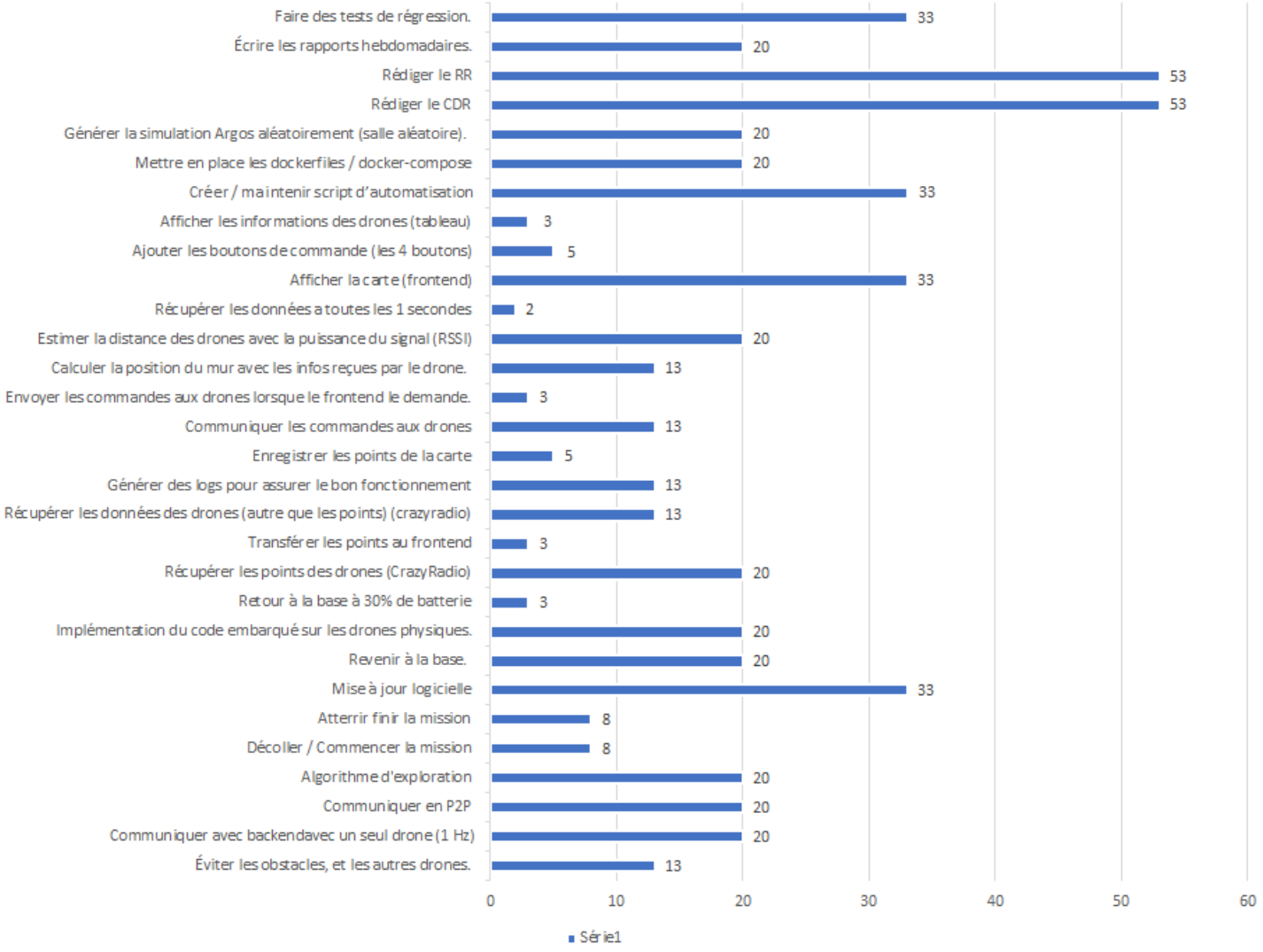


Figure 6: Répartition du temps de travail en heure pour chacune des tâches



4.3 Calendrier de projet

Tableau 2: Remises des livrables

Livrable	Date de remise	Description
Critical Design Review	8 mars 2021	Remise d'un système avec fonctionnement partiel
Readiness Review	12 avril 2021	Remise d'un système avec toute la fonctionnalité du système

4.4 Ressources humaines du projet

Pour Nicolas, il possède des connaissances en Angular qui viennent du deuxième projet intégrateur. Il possède aussi de l'expérience avec des scripts Windows et sur l'utilisation des machines virtuelles. Il possède aussi de l'expérience sur les problèmes matériels et est capable de faire de la soudure sur les systèmes informatiques.

Pour William, ses connaissances en *Unity* sont sans aucun doute un grand atout pour ce projet. En effet, la simulation *ARGoS* ressemble beaucoup à une simulation *Unity*, surtout dans la façon dont le code est exécuté. De plus, William a travaillé sur quelques projets personnels en C++ ce qui lui a permis d'être familier avec les concepts importants.

Theo, quant à lui, a de nombreuses expériences avec Angular, notamment lors d'un stage. Étant le membre avec le plus d'expérience en Angular dans l'équipe, il a été convenu qu'il soit le spécialiste frontend de l'équipe. De plus, il possède quelques expériences avec le langage Python. Il va donc pouvoir prêter main forte pour le développement du backend en Python. Il possède aussi quelques expériences en C++, ce qui va lui permettre d'aider au développement des Drone et de la simulation.

Pour Philippe, il a plus de 1 an et 3 mois d'expérience en Java et en tant que QA. Cela sera très utile pour l'architecture ainsi que pour la mise en place de tests unitaires et d'intégration. De plus, il a de l'expérience au PolyFab Normand Brais se trouvant à Polytechnique Montréal, et a fait de nombreux projets en rapport avec le système embarqué.

Pour Nour, elle a des connaissances en Angular qu'elle a acquis du deuxième projet intégrateur. Elle a aussi quelques connaissances dans le langage python ainsi que le langage C et C++. Elle pourra donc participer au développement du backend ainsi qu'au développement du système embarqué pour les drones.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité

Chaque fonctionnalité sera complémentée d'un ou de plusieurs tests unitaires correspondant pour assurer le bon fonctionnement de ces nouvelles fonctionnalités. Pour assurer la qualité de notre code avant chaque remise, nous avons décidé de finaliser notre développement au minimum deux jours avant la remise. Cela permet de nous donner du temps pour peaufiner le code et corriger les petits bugs qui peuvent rester.

En plus des tests unitaires, nous allons faire des tests de régressions en équipe à chaque fin de sprint. Pour ce faire, nous allons avoir une personne dans

l'équipe qui teste le système en entier en partagent son écran pour que tous les membres de l'équipe évaluent le système.

Nous allons aussi faire des tests d'intégration tout au long du développement pour assurer le bon fonctionnement de chaque fonctionnalité du système.

Pour contrôler la qualité tout au long du développement, le code que nous écrivons est révisé par chaque membre de l'équipe lorsque nous faisons des merge request. Nos merge request doivent être approuvés par tous les membres de l'équipe. Cela va aussi nous aider à comprendre le code des autres et peut nous permettre de proposer de meilleures solutions après avoir compris le code de la personne qui a programmé la fonctionnalité.

5.2 Gestion de risque

Dans le contexte actuel de la pandémie du Covid-19, plusieurs risques liés à la gestion de projet et au travail à distance doivent maintenant être pris en compte dans notre gestion de risque.

Un premier potentiel risque est si une hélice sur un drone brise et que nous sommes en attente de recevoir une hélice supplémentaire pour la réparation. Dans ce cas, plusieurs options sont disponibles selon l'avancement du projet. Premièrement, si nous sommes simplement en train de développer la détection et le transfert des points pour générer la carte, nous pouvons continuer de travailler avec un seul drone. De plus, nous pouvons aussi bouger le drone endommagé dans nos mains, car ce que nous testons est la détection et la transmission des points et non l'algorithme de vol. Une autre option est de continuer le développement en utilisant la simulation Argos. Deuxièmement, si nous sommes en train de développer la communication entre les drones et avec la station au sol, il n'est pas nécessaire que les drones soient en vol. Dans ce cas, le fait qu'une hélice soit endommagée ne change rien à notre développement et ne nous ralentira pas le temps de recevoir une nouvelle hélice.

Un second potentiel risque similaire au premier est si une autre partie d'un drone ne fonctionne plus. Dans ce cas, comme pour le premier risque, tout dépendant de la pièce endommagée, nous pouvons tout de même continuer le développement de fonctionnalités qui ne touchent pas à la partie endommagée en attendant de recevoir la ou les pièces de remplacement. De plus, tout dépendant de la situation, nous pourrions aussi développer en utilisant Argos en attendant de recevoir la pièce de remplacement si nécessaire.

Un troisième potentiel risque est si la personne qui a les deux drones a des problèmes avec son ordinateur et ne peut plus tester et développer avec les vrais drones ou si on veut qu'une autre personne dans l'équipe expérimente avec les drones. Dans ce cas, puisqu'une seule personne aura en sa possession les deux drones, il sera alors nécessaire qu'une personne dans l'équipe se déplace pour aller chercher les drones. Pour le déplacement, une personne dans l'équipe à une voiture et peut se déplacer si nécessaire.

Un quatrième potentiel risque existe s'il y a des changements au requis du projet. Dans ce cas, la meilleure façon de régler ce problème sera de se

rassembler en équipe et refaire notre planification pour tenir en compte des changements. Cela inclut un changement dans les tâches dans notre planification sur le Board sur Gitlab. Ensuite, la tâche sera assignée à une personne de l'équipe.

Un cinquième potentiel risque est que si une personne dans l'équipe à un problème personnel comme une maladie ou un problème familial et ne peut pas compléter sa tâche ou ne peut simplement pas travailler pour une certaine durée de temps. Pour régler ce problème, nous devrons faire une rencontre d'équipe d'urgence pour discuter de la situation et assigner la tâche à une autre personne pour compenser pour la personne qui ne peut pas travailler.

5.3 Tests

Pour tester le code embarqué des drones, nous allons utiliser la simulation d'Argos pour assurer le bon fonctionnement de nos drones. Pour s'assurer du bon fonctionnement, nous devons nous vérifier que l'algorithme d'exploration fait son travail et ne manque aucune partie de la salle à explorer. De plus, nous devons nous assurer que les drones sont capables d'éviter les obstacles et les collisions avec l'environnement et avec les autres drones.

Nous allons utiliser Jasmine pour tester l'interface web qui est codé en utilisant Angular. L'utilisation de Jasmine nous permet d'avoir une couverture de code de 100 % et nous permet donc de nous assurer que notre code est testé en entier. Pour ce qui est de la partie *backend*, puisque celle-ci est codée en Python, nous allons utiliser *Unittest* qui est fourni par défaut avec Python pour faire les tests unitaires et les tests d'intégration. Finalement, pour tester le code embarqué sur les drones, nous allons utiliser *Cmock* pour faire les tests unitaires pour le code C. Les tests unitaires et les tests d'intégrations seront complétés par la ou les personnes qui ont développé la fonctionnalité.

Nous allons aussi faire des tests de régression en équipe à chaque fin de sprint pour tester directement le système. Une personne dans l'équipe exécutera le code normalement et partagera son écran pour que le reste des membres de l'équipe puissent valider le bon fonctionnement du système.

5.4 Gestion de configuration

Afin de s'assurer que notre code est lisible et facilement maintenable, nous avions l'intention d'indiquer en haut de chaque header file l'utilité de la classe. Aussi, nous allons laisser des commentaires au-dessus de chaque fonction pour expliquer leurs paramètres et leur utilité. Ces mesures devraient rendre le code bien lisible pour tout le monde et faciliter sa maintenance.

Ensuite, nous voulons ajouter un Read me à la base du projet pour indiquer comment lancer chacune des parties. Il expliquera par exemple comment lancer le server, le *frontend*, la simulation et comment télécharger le code sur le drone.

Cela devrait permettre à n'importe qui de pouvoir lancer toutes les parties du projet très facilement sans devoir trop chercher.

Nous avons séparé notre projet en 3 dossiers qui correspondent à chacune des parties du projet, soit Drone, Server et Client. Cela nous permettra d'avoir un code bien organisé et facilitera le développement.

Pour s'assurer que seulement du code de qualité se ramasse dans les remises, nous optons pour un système de *merge request*. Cela veut dire que du code doit être vérifié par tous les membres de l'équipe avant d'être mis sur notre branche de remise, soit la branche master. Les imperfections et erreurs sont donc captées et communiquées à l'auteur pour qu'il fasse les changements nécessaires. Une fois les changements terminés, les autres membres peuvent regarder à nouveau et s'ils jugent que le code est parfait, ils acceptent la *merge request* et le code est jumelé avec le reste. Si jamais nous avons des questions par rapport à une partie du code, nous pourrons nous organiser des rencontres afin d'expliquer ou même prendre des décisions côté architecture et design.

Comme dit précédemment, notre code est séparé en trois dossiers. Pour s'assurer que ce code peut rouler sur n'importe quelle machine, nous avons ajouté un *dockerfile* dans chacun de ces dossiers. Ceux-ci créent un environnement contenant toutes les dépendances nécessaires et les instructions sur comment construire les éléments nécessaires. Ces trois *dockerfiles* sont reliés par un fichier appelé *docker-compose*. Celui-ci construit les trois conteneurs d'un coup permettant d'avoir un environnement complet en une ligne de commande.

Nous comptons aussi avoir un script pouvant lancer chaque partie du projet. Nous pourrons lui passer des arguments qui lui indiqueront quelle partie du code à lancer. Ces arguments seront indiqués dans le Read me ce qui permettra à n'importe qui de lancer n'importe quelle partie du projet.

6. Références

[1] J. Collin, *Cycles d'acquisition et ententes contractuelles*, 2021. [En Ligne]. Disponible : https://moodle.polymtl.ca/course/view.php?id=1703

ANNEXE

Démonstration des manipulations sur les drones *:* *https://youtu.be/lV7QmEIYYrA*

Démonstration de la simulation ARGoS : *https://youtu.be/tkKLtBYJh7E*