



**POLYTECHNIQUE  
MONTRÉAL**

UNIVERSITÉ  
D'INGÉNIERIE

Département de génie informatique et génie logiciel

**INF3995**

**Projet de conception d'un système informatique**  
***Conception d'un système aérien minimal pour exploration***

**Rapport final de projet**

Équipe No. 100

Laurent Bolduc  
Jacob Brisson  
Jordan Lecourtois  
Matthew Paoli  
Philippe Savard  
Simon Tran

21 avril 2021

# Table des matières

1. Objectifs du projet	2
1.1 - Objectif commercial	2
1.2 - Exigences du système	2
2. Description du système	3
2.1 Logiciel embarqué (Q4.5)	3
2.1.1 – Architecture générale du logiciel embarqué	3
2.1.2 – Modifications majeures comparées à la CDR	4
2.2 La station au sol (Q4.5)	6
2.3 L'interface de contrôle (Q4.6)	7
2.4 Fonctionnement général (Q5.4)	10
2.4.1 - Les répertoires du projet et installations	10
2.4.2 - Démarrage du système	12
2.4.3 - Lancer les tests	14
3. Résultats des tests de fonctionnement du système complet (Q2.4)	14
3.1 Requis fonctionnels	14
3.2 Requis matériels	15
3.3 Requis de conception	15
3.4 Requis logiciel	16
3.5 Requis de qualité	17
3.6 Requis bonus	17
4. Déroulement du projet (Q2.5)	17
4.1 Réussites du déroulement du projet	17
4.2 Échecs du déroulement du projet	18
5. Travaux futurs et recommandations (Q3.5)	19
6. Apprentissage continu (Q12)	19
6.1 - Laurent Bolduc	19
6.2 - Jacob Brisson	20
6.3 - Jordan Lecourtois	21
6.4 - Matthew Paoli	21
6.5 - Philippe Savard	22
6.6 - Simon Tran	22
7. Conclusion (Q3.6)	23
8. Références	24
9. Annexe	25

# 1. Objectifs du projet

## 1.1 - Objectif commercial

L'objectif commercial de l'équipe est la réalisation d'une preuve de concept pour un système aérien minimal pour exploration. Le système devait se concrétiser sous la forme d'un prototype fonctionnel de niveau de maturité NMS 4. Ce prototype a pour but de maximiser l'efficacité d'exploration de terrain tout en minimisant les coûts d'exploration.

Au cours des douze dernières semaines, l'équipe devait réaliser la tâche demandée dans un temps maximal estimé de 630 heures de travail. L'équipe avait comme obligation d'émettre de façon hebdomadaire un compte rendu d'avancement pour le projet. Les ressources humaines disponibles pour la réalisation du mandat sont les six membres de l'équipe de projet, signataires du présent rapport.

Le mandat initial était composé de trois étapes (livrables) de réalisation. En ordre chronologique, les livrables étaient la PDR (Preliminary Design Review), la CDR (Critical Design Review) et finalement la RR (Readiness Review). Ce rapport fait le point sur l'état du prototype réalisé et remis en date du mercredi 21 avril 2021.

## 1.2 - Exigences du système

Dans cette section, nous présentons une vue d'ensemble sur les exigences du système réalisé.

Dans un premier temps, le prototype doit être en mesure d'effectuer, à l'aide d'un essaim composé de plusieurs drones, l'exploration d'un bâtiment moyen d'une aire maximale de 100 m<sup>2</sup>. Les drones doivent être en mesure d'effectuer cette exploration de manière autonome (sans interaction directe de l'utilisateur) à l'exception d'une interface de contrôle Web accessible sur l'appareil de l'utilisateur.

Dans un second temps, l'interface accessible par l'utilisateur doit lui permettre d'interagir directement avec l'essaim de drones. L'utilisateur doit pouvoir envoyer certaines commandes de contrôle aux drones. Ces commandes devront être minimalement composées d'une option de démarrage du système (décollage), de l'arrêt du système (atterrir, retour à la base) et de la mise à jour du système. L'interface Web devra également permettre à l'utilisateur de visualiser les données d'exploration (drones, obstacles, etc.) en temps réel autant pour une mission simulée qu'une mission réelle. Ces données doivent être mises à jour sur une base périodique d'au moins un

rafraîchissement par seconde. La communication avec les drones doit se faire avec la station au sol à l'aide d'un canal de communication de 2.4 GHz avec les drones.

## 2. Description du système

### 2.1 Logiciel embarqué (Q4.5)

#### 2.1.1 – Architecture générale du logiciel embarqué

La figure 9.1 dans l'annexe représente l'architecture générale du logiciel embarqué. Premièrement, le fichier *Main* reçoit les commandes provenant du backend et se charge d'appeler les bonnes commandes en fonction du caractère reçu. Il y a deux options lors de la réception de paquets, soit il s'agit d'une demande d'information ou bien il s'agit d'une demande d'activation de commande.

Dans le cas où le backend demande de l'information de la part du drone, le *Main* va appeler le *InformationHandler*. Par la suite, le contrôleur d'information va faire appel à des fonctions du *SensorCommands* afin d'obtenir l'information demandée. Par exemple, si le *Main* reçoit le caractère 'b', il va faire appel au contrôleur d'information qui va savoir qu'il s'agit d'une demande d'information concernant le niveau de batterie du drone. L'appel de la fonction pour obtenir le niveau de batterie est fait en passant par *SensorCommands* et l'information est acheminée au *Main* afin d'être envoyée au backend.

Dans le cas où nous voulons activer une commande, le *Main* va plutôt faire appel au *CommandHandler*. Le but de ce contrôleur est de reconnaître les trois commandes que l'utilisateur peut activer : décoller (*Take off*), retour à la base (*Return to base*) et atterrir (*Land*). Afin d'accomplir ces commandes, le contrôleur utilise les commandes de mission qui sont situées dans *MissionCommands* et qui lui, utilise *MouvementCommands*. Dans le fichier des commandes de mouvement, nous retrouvons les fonctions qui permettent au drone d'accomplir toutes les tâches que nous lui demandons. Notamment, la définition des fonctions de déplacement et d'évitement d'obstacles se situe dans ce fichier. Toutefois, pour plusieurs de ces fonctions, nous devons connaître plusieurs informations sur les capteurs. C'est pour cette raison que les commandes de mouvement font appel à des fonctions situées dans le *SensorCommands*.

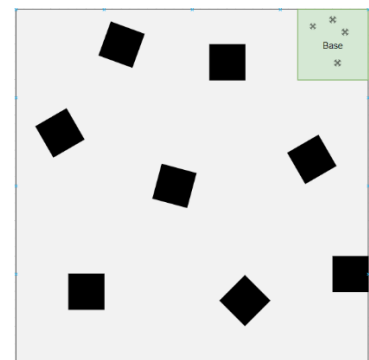
## 2.1.2 – Modifications majeures comparées à la CDR

Lors de la réalisation de la RR, notre équipe a réalisé que certaines fonctionnalités telles que l'exploration de l'environnement et le retour à la base, tous deux implémentés pour la CDR, ne répondaient plus aux requis fonctionnels actuels. Nous avons donc pris la décision de changer intégralement ces fonctionnalités autant pour la simulation que pour les drones réels.

La méthode initiale, telle que présentée dans le rapport de la CDR, tenait pour acquis que l'environnement exploré serait composé de pièces rectangulaires (comme une maison) dans lesquelles les drones pourraient explorer. Ceci était une erreur d'interprétation des requis puisque la carte réelle est plutôt composée de boîtes distribuées de façon aléatoire dans l'arène avec une orientation quelconque. Nous observons un exemple de cette nouvelle arène dans la figure 2.1.

La nouvelle implémentation de ces fonctionnalités n'utilise plus cette méthode. Elle est plutôt basée sur la décomposition de la carte en sous-régions adressables (carrés) qui peuvent contenir trois valeurs : inexplorée, libre et occupée par un obstacle. Afin d'optimiser les performances des algorithmes utilisés (voir plus loin), nous avons décomposé la carte d'exploration en 2500 carrés (résolution de 50 x 50 carrés). La taille d'un carré dans la simulation et dans l'arène réelle est donc de 20 centimètres de côté (considérant une arène de maximum 10 x 10 mètres).

**Figure 2.1.1 – Arène de simulation**



### Exploration

Afin de permettre l'exploration rapide de l'entièreté de la carte, les drones exécutent, à chaque cycle, un algorithme qui permet de déterminer la direction dans laquelle le drone va récupérer le plus d'informations. En effet, les drones possèdent une carte interne représentant l'état actuel de l'exploration en plus de connaître leur position dans cette carte, à partir de leur position initiale et de leur déplacement. Ainsi, le drone peut calculer le nombre de cases libres ou de cases inexplorées se trouvant dans chaque direction afin de déterminer la direction qui permet le plus grand gain d'informations. Dans le but de favoriser l'exploration de l'entièreté de la carte, les cases libres ont un poids supérieur dans le calcul de la prochaine direction.

L'algorithme est donc grandement en fonction du fait que les capteurs des drones fournissent des valeurs précises et constantes quant au déplacement effectué ainsi qu'aux distances fournies par les capteurs. En effet, le drone doit, à chaque cycle, remplir

la carte avec les dernières informations recueillies, pour pouvoir ensuite prendre une décision quant à la prochaine direction optimale.

En pratique, avec nos nombreuses expérimentations, nous en sommes venus à la conclusion que l'utilisation des capteurs avec les drones ne permet pas de remplir de façon précise et exacte la carte, d'une part parce que le drone ne connaît pas sa position exacte, due au fait qu'il dérive et que les capteurs de distance ne retournent pas des valeurs extrêmement précises. Ainsi, en raison du manque de temps et de ressources, nous avons dû changer l'algorithme sur les drones physiques.

En effet, les drones physiques implémentent désormais un algorithme qui est davantage réactif et permet une exploration moins optimale de l'environnement. Ce nouvel algorithme est séparé en deux parties : l'évitement des obstacles et le changement de direction.

Pour l'évitement des obstacles, le drone va simplement aller dans la direction opposée à l'obstacle si ce dernier se trouve à moins de 35 cm. Par exemple, si le drone détecte un mur à moins de 35 cm de son côté gauche, il définit sa position comme étant vers la droite.

Pour ce qui est du changement de direction, les drones vont changer de direction à tous les 80 cycles. Pour déterminer la prochaine direction à suivre, les drones sélectionnent la direction pour laquelle le capteur retourne la plus grande distance et qui est perpendiculaire à la direction actuelle. Par exemple, un drone se dirigeant vers l'avant, il ira ensuite dans la direction où il y a le plus d'espace entre la gauche et la droite.

## **Retour à la base**

Dans le cas du retour à la base, nous avons opté pour une approche hybride entre la force du signal RSSI (pour les drones réels) et un algorithme de recherche des chemins en fonction de la carte explorée. L'algorithme de recherche utilisé est une version personnalisée du *Wavefront Planning Algorithm* en utilisant le modèle "*von Neumann neighborhood*" pour la recherche de cases adjacentes [18]. Le signal RSSI n'a pas été répliqué dans la simulation. Dans cette section, nous allons survoler rapidement l'algorithme choisi ainsi que sa nouvelle implémentation dans la simulation.

Afin d'utiliser le *Wavefront Planning Algorithm* correctement, trois critères doivent être remplis. Premièrement, nous devons connaître la position de la base dans la carte d'exploration. Deuxièmement, le carré de la base doit avoir été exploré par les drones durant l'exploration. Finalement, nous devons connaître la position des obstacles dans la carte. Pour les drones réels, il est possible d'utiliser la force du signal RSSI pour ajuster la position de la base si le drone ne l'a pas explorée. Or, puisque le signal RSSI n'existe

pas dans la simulation (n'a pas été implémenté), notre équipe a choisi d'opter pour une position de base connue. Ainsi, la position initiale des drones se situe toujours à environ 1 m de rayon de la base (la base a pour position  $x = 4.5$  et  $y = 4.5$  dans la simulation). Ainsi, nous enregistrons la position initiale du drone comme position de la base qui sera dans tous les cas explorés. Dans le cas des obstacles, la position de ceux-ci sera découverte lors de l'exploration. Toutes les zones non explorées sont considérées comme des obstacles et ne sont pas une option valide pour le chemin de retour.

Une fois l'exploration complétée (nous connaissons les obstacles), l'algorithme est exécuté. Celui-ci générera une carte des distances de  $50 \times 50$  permettant aux drones de la simulation de déterminer, pour tous les points explorés, la meilleure direction pour retourner à la base. C'est à cette étape de l'algorithme (sélection de la prochaine direction) que nous nous assurons que la direction choisie est réalisable (pas trop près d'un obstacle). Finalement, le drone effectue un déplacement dans la direction choisie par l'algorithme.

En pratique, les mêmes problèmes que pour l'exploration en simulation sont survenus lors de l'implantation sur les drones physiques : la carte n'est pas remplie à cause de l'imprécision des capteurs, ce qui empêche le drone de déterminer le chemin à suivre pour retourner à sa base. Nous avons donc également dû changer l'algorithme pour les drones physiques, malgré le fait qu'il fonctionne de façon optimale en théorie et sur les drones simulés.

L'algorithme utilisé dirige le drone vers le coin inférieur gauche de l'arène à l'aide de déplacements vers l'arrière. Pendant ses déplacements, le drone évite les obstacles rencontrés en se déplaçant vers la gauche. Une fois l'obstacle évité, il reprend son déplacement vers l'arrière en direction du coin visé. Le drone répète ce comportement jusqu'à ce qu'il obtienne un signal RSSI indiquant qu'il se situe dans un rayon approximatif d'un mètre par rapport à la base.

## 2.2 La station au sol (Q4.5)

Pour ce qui est de la station au sol, elle constitue le centre de communication entre les différentes composantes de notre système. La station au sol est par définition composée de l'application interne et de l'interface utilisateur, mais puisque l'interface utilisateur constitue également l'interface de contrôle, son utilité sera incluse seulement lorsque nécessaire dans la description de cette composante.

Les requêtes vers l'application interne sont encapsulées dans des méthodes statiques du fichier *BackendREST.ts* de l'interface utilisateur. Ces méthodes effectuent des requêtes *REST* à l'application interne. En premier lieu, une méthode *liveCheck* est

appelée dès le premier rendu de l'application, puis chaque fois que l'URL de l'application interne est modifiée par l'utilisateur. Cette méthode a pour simple but de retourner un code OK lorsque la connexion au backend est bien établie. Les autres méthodes ne peuvent pas être appelées si la connexion est invalide. Il existe une méthode *scan*, permettant d'effectuer un balayage pour se connecter aux drones, une commande *updateStats* pour obtenir les informations des drones à jour telles que la vitesse, l'état, les valeurs des senseurs, la position et le niveau de batterie. De plus, il y a les méthodes *takeOff* et *land* pour démarrer et arrêter l'exploration, la méthode *land* ayant un paramètre exprimant si l'atterrissage est un retour à la base ou non. La méthode *reset* permet de réinitialiser les données de l'application interne. L'application interne se déconnecte alors de tous les drones et supprime l'information qu'elle avait en mémoire. Cette méthode possède aussi un argument optionnel permettant de demander à l'application interne de se connecter à la simulation ou aux drones réels, afin de modifier son état sans l'exécuter de nouveau. Toutes les méthodes *REST* mentionnées adaptent leur comportement afin de correspondre à si l'exécution est en mode simulation ou non.

Pour ce qui est de l'architecture de l'application interne (voir figure 9.2 de l'annexe), le fichier *App.py* exécute une application *Flask* et contient la logique de toutes les méthodes mentionnées ci-haut, ainsi que les données sauvegardées des drones et de l'état du système [8]. Le fichier *Appchannel.py*, appelé uniquement dans les méthodes de *App.py*, établit la connexion entre l'application interne et les drones physiques par des fonctions de communications. L'alternative du côté de la simulation Argos est supportée par l'utilisation de *sockets* dans *App.py* [16]. Le fichier *DroneDTO.py* contient les informations se rapportant aux drones dans le format selon lequel elles seront envoyées à l'interface utilisateur pour l'affichage. Le fichier a deux sous-définitions *Drone.py*, se rapportant à la définition précise des drones physiques et *Dronesim.py*, se rapportant à la définition des drones de la simulation. Finalement, le fichier *StatusDTO.py* contient l'état de l'application interne, soit si elle est en mode simulation ou non.

Du côté des librairies, les trois principales librairies utilisées sont *socket*, pour le système de communication entre l'application interne et la simulation, *requests*, pour la communication *REST* entre l'application interne et l'interface de contrôle et *cflib*, pour les librairies spécifiques à la communication avec les drones [2], [16]. De plus, la librairie *logging* est également utilisée afin d'avoir un journal des informations désirées dans l'interface de contrôle.

## 2.3 L'interface de contrôle (Q4.6)

L'interface graphique est construite en Typescript en utilisant le cadriciel *React* [7], [13]. Ainsi, il est possible d'y accéder à partir de n'importe quel navigateur Web récent, que ce soit sur un ordinateur, une tablette ou un téléphone intelligent. La librairie *Material-UI* est



utilisée afin d'avoir des composants visuels intuitifs et plaisants au regard [12]. Cette librairie fournit en effet la plupart des éléments graphiques utilisés dans l'application tels que les boutons, les champs de texte, les menus, les icônes. Ensuite, la librairie *Lodash* a été utilisée puisqu'elle fournit de nombreuses fonctions utilitaires [11]. Pour la gestion des dates, la librairie *Date-Fns* a été utilisée [10]. Cette dernière permet la manipulation facile des dates tout en gérant simplement les complexités qui viennent avec ces opérations telles que la gestion des fuseaux horaires.

Les tests unitaires roulent grâce à *Jest* [6]. De plus, la librairie *React-Testing-Library* a été utilisée afin de tester facilement le comportement de nos composants *React* personnalisés [5].

















L'utilisation de *React* comme cadriciel nous a encouragés à préférer le patron de conception de la programmation fonctionnelle à celui de la programmation orientée objet. Effectivement, avec l'arrivée des *React Hooks* dans la version 16.8 de *React*, l'utilisation de classes n'est plus nécessaire pour la création des composants [7]. Ainsi, notre code ne contient que très peu de classes et préfère l'utilisation de composants fonctionnels. Néanmoins, il existe la classe *MapData* qui contient les informations concernant les cartes. Elle contient les méthodes qui permettent de transformer l'objet de la carte en objet qui sera persisté dans la base de données (*MapDataDTO* pour Data Transfer Object) et vice-versa. La seule autre classe est *BackendREST* qui contient des méthodes statiques qui effectuent les requêtes *REST* au backend.

Le projet est séparé en plusieurs paquetages. *utils* contient les fichiers qui contiennent des fonctions utilitaires. Le paquetage *pages* contient les composants des éléments visuels. Il y a *Home*, la page d'accueil, *MapsTable*, la section avec la liste des cartes de la base de données dans la page d'accueil, *Main* et *Layout* pour l'interface principale lors de l'exploration, *Logs* pour la fenêtre qui affiche le journal et *Map* qui contient la vue de la carte. Le paquetage *model* contient les interfaces. *Hooks* contient les *hooks React* personnalisés, en particulier *useMouseHandler* qui offre un gestionnaire d'événements de la souris. *Context* contient le *CFCContext*, le contexte utilisé pour implémenter le patron du *context-provider*. Ce contexte permet d'obtenir les états de l'application ainsi que les fonctions d'interaction avec le backend à partir d'un composant *React* en utilisant le *hook useContext*. Finalement, le paquetage *components* contient les autres composants visuels tels que le *CFDrawer*, le tiroir vertical à gauche et le *ControlDrawer*, le tiroir horizontal en bas.

La figure suivante présente l'interface de la page d'accueil de l'application. C'est là que nous voyons la liste des cartes déjà explorées et qu'il est possible de créer une nouvelle exploration.

**Figure 2.3.1 – Centre de contrôle des Crazyflies**

Centre de contrôle des  
Crazyflies  
INF3995 - Équipe 100

Type	Nom	Date de dernière modification	Actions
	04/15/2021, 1:51 PM	jeudi 15 avril 2021 à 14:00:54	
	04/15/2021, 1:37 PM	jeudi 15 avril 2021 à 13:37:04	
	04/15/2021, 1:12 PM	jeudi 15 avril 2021 à 13:12:39	
	04/15/2021, 1:08 PM	jeudi 15 avril 2021 à 13:08:39	
	04/15/2021, 1:01 PM	jeudi 15 avril 2021 à 13:01:27	
	04/14/2021, 5:10 PM	jeudi 15 avril 2021 à 12:15:16	
	04/15/2021, 11:42 AM	jeudi 15 avril 2021 à 11:42:05	
	04/15/2021, 7:19 AM	jeudi 15 avril 2021 à 10:19:43	

NOUVELLE EXPLORATION...

La figure suivante présente la page principale de l'application sur laquelle nous pouvons voir la carte, la liste des drones et interagir avec ceux-ci. Dans la barre du haut, il y a un bouton permettant de réduire la taille du tiroir gauche, le titre de la carte, un bouton permettant de modifier le titre, une indication qui précise si nous utilisons les drones simulés, un bouton pour sauvegarder et un bouton pour retourner à la page d'accueil. Dans le tiroir gauche, il y a la liste des drones affichant le nom, la batterie, la vitesse et l'état de chaque drone. Un bouton "Balayer" permet de détecter les nouveaux drones. Une bascule permet d'activer le rafraîchissement automatique afin d'obtenir les nouvelles informations des drones chaque seconde. Un bouton permet d'afficher le journal et il y a un champ de texte permettant de spécifier l'URL du serveur. Dans le tiroir du bas, il y a les boutons permettant de décoller, d'atterrir, de retourner à la base et de mettre à jour les drones. Finalement, dans la partie centrale, il y a la carte avec les outils qui permettent de manipuler la caméra. Les points bleus représentent les drones, les carrés rouges et les murs découverts. L'arrière-plan gris représente la zone non explorée. Lors de l'exploration, si le rafraîchissement automatique est activé, le frontend envoie une requête au backend pour obtenir les informations du drone chaque seconde. Le backend renvoie la nouvelle position des drones, la valeur des capteurs, l'état des drones et les niveaux de batterie. Ainsi, pour chaque valeur des capteurs, nous pouvons ajouter un point sur la carte représentant un mur.

**Figure 2.3.2 – Interface de contrôle**

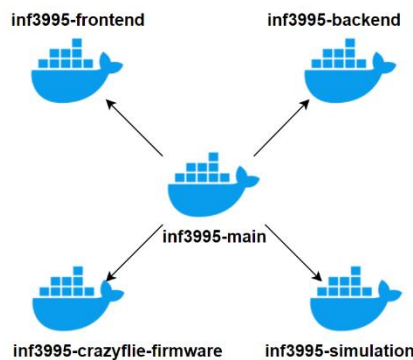
## 2.4 Fonctionnement général (Q5.4)

Dans cette section, nous allons présenter les étapes essentielles de configuration au bon fonctionnement du système. Les outils bien documentés (installation de Linux, git, etc.) ne seront pas détaillés dans ce rapport. Veuillez prendre note que le système est conçu pour un système d'exploitation Linux Ubuntu 18.04 LTS ou ultérieur et pourrait ne pas fonctionner correctement si installé sur une autre plateforme. Prendre note que toutes les étapes pour l'installation et l'utilisation de nos systèmes sont également documentées dans le fichier README.md du répertoire *inf3995-main*.

### 2.4.1 - Les répertoires du projet et installations

Lors du développement du système, notre équipe s'est mise en accord sur l'utilisation de sous-modules dans le répertoire git. Cette décomposition permet de décentraliser le développement et nous permet de mieux arrimer les différentes composantes du système. Le répertoire principal, *inf3995-main*, possède les quatre sous-modules suivants : *inf3995-simulation*, *inf3995-backend*, *inf3995-frontend* et *inf3995-crazyflie-firmware*.

### Figure 2.4.1 – Conteneurs Docker



Chacun des sous-modules présentés est indépendant. Cela lui permet donc d'être modifié, compilé et exécuté en parallèle des autres répertoires. Cela nous permet également d'exécuter plusieurs conteneurs Docker construits et personnalisés afin de supporter la portion du système qu'il contient. Par exemple, le conteneur de la simulation contient toutes les dépendances, bibliothèques et autres éléments nécessaires au bon fonctionnement de la simulation. Nous y reviendrons plus loin dans cette section.

### Cloner le répertoire principal *inf3995-main*

La première étape, afin de correctement lancer le système, est d'acquies à l'aide de *git* le répertoire principal du projet. En raison de la décomposition en sous-modules, il est plus aisé d'exécuter un clonage récursif sur tous les sous-modules. La commande pour faire cette action est la suivante :

```
git clone --recurse-submodules https://gitlab.com/polytechnique-montr-al/inf3995/20211/equipe-100/inf3995-main.git
```

Si le clonage s'est bien déroulé, les fichiers correspondants aux sous-modules ne devraient pas être vides.

### Installation de Docker

Afin de continuer l'installation, assurez-vous d'avoir correctement installé Docker et Docker Compose sur votre machine. Ils seront des composants essentiels pour la suite de la procédure. Vous trouverez l'ensemble de l'information concernant leur installation dans la documentation officielle de Docker ( <https://docs.docker.com/engine/install/ubuntu/> ). Assurez-vous de compléter les étapes [d'après installation sur Linux de la documentation de Docker](#). Ceci nous évitera des problèmes de permission lors de l'exécution des commandes.

### Installation de x11docker

Puisque notre système utilise des conteneurs Docker, nous aurons besoin d'une interface graphique permettant de visualiser depuis un poste de travail la simulation. Notre système utilise x11docker afin d'afficher l'interface de la simulation. Afin d'installer x11docker correctement, suivez les instructions de la documentation : <https://github.com/mviereck/x11docker>

La simulation n'a été testée qu'avec la version 6.6.2 de x11docker. Il est possible qu'elle ne fonctionne pas correctement avec une autre version.

## 2.4.2 - Démarrage du système

Afin de démarrer le système, considérant les étapes précédentes complétées, il suffit d'exécuter la commande :

```
./start.sh
```

depuis le répertoire principal *inf3995-main*. Cette commande exécute en réalité les deux commandes suivantes :

```
docker-compose build
./run-simulation.sh & docker-compose up
```

Lorsqu'on exécute cette commande, nous construisons les conteneurs des composantes un à un, puis lançons la simulation à l'aide du script *./run-simulation.sh*. Une fois le script terminé, vous devriez être en mesure de vous connecter en local à l'adresse <http://localhost:3000> et observer l'interface d'accueil présentée à la section précédente. Une fenêtre x11docker devrait également apparaître montrant la simulation prête à être démarrée.

### Démarrage de l'exploration pour les drones réels

Afin de démarrer une exploration avec les drones réels, appuyez sur le bouton "NOUVELLE EXPLORATION" en bas de l'écran. Dans la fenêtre modale qui s'affiche à votre écran, donnez un nom à votre exploration et laissez le bouton pour simulation désactivé.

**Figure 2.4.2 – Nouvelle exploration pour les drones réels**



En confirmant votre configuration, l'interface de contrôle devrait apparaître.

Allumez ensuite les drones. Lorsqu'ils sont prêts, appuyez sur le bouton "balayage" de l'interface graphique afin de vous connecter aux drones. Lorsque les drones sont

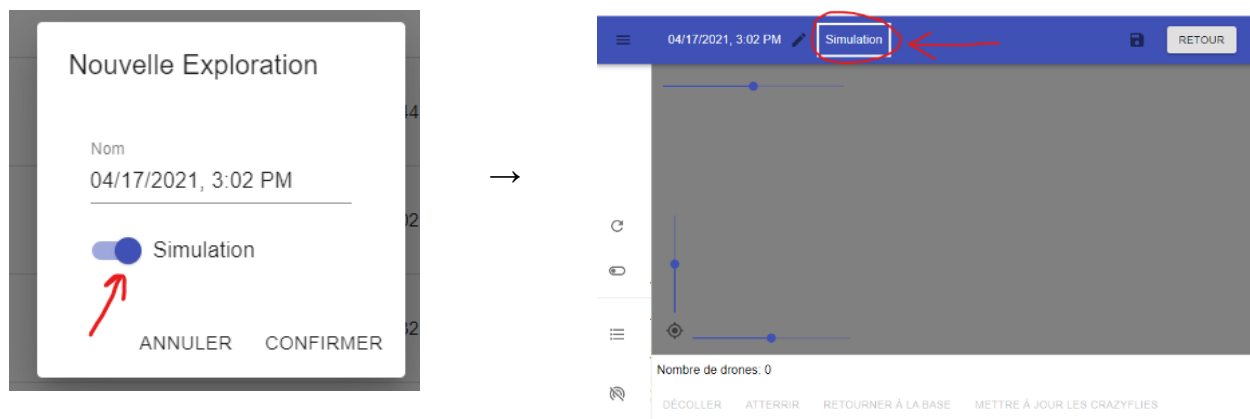
détectés, ils devraient apparaître en rouge dans le tiroir vertical. Cela signifie que les positions initiales n'ont pas été entrées. Cliquez sur chaque drone afin d'ouvrir la fenêtre permettant d'entrer les positions initiales.

Le bouton décollage est maintenant disponible. Appuyez dessus pour démarrer l'Exploration. Il est possible d'appuyer sur la bascule "Rafraîchissement automatique" afin d'activer cette fonction. Appuyez sur le bouton "Retour à la base" puis "Atterrir" pour compléter l'exploration. Il est possible de sauvegarder la carte à l'aide du bouton de sauvegarde en haut à droite. De plus, il est possible de sauvegarder une image SVG de la carte avec le bouton de téléchargement en haut à droite de la carte.

### Démarrage de l'exploration pour les drones simulés

Afin de démarrer une exploration avec les drones simulés, appuyez sur le bouton "NOUVELLE EXPLORATION" en bas de l'écran. Dans la fenêtre modale qui s'affiche à votre écran, donnez un nom à votre exploration en prenant bien soin d'appuyer sur le bouton "Simulation". Lorsque l'interface de contrôle apparaît, vous devriez observer un indicateur "Simulation" en haut de l'écran vous indiquant que vous êtes bien en simulation.

**Figure 2.4.3 – Nouvelle exploration pour la simulation**



Appuyez sur le bouton "Balayage". Dans la simulation, appuyez sur le bouton de démarrage afin de démarrer la connexion entre la simulation et le poste de contrôle. Lorsque celle-ci est effectuée, les étapes suivantes sont les mêmes qu'avec les drones réels.

### 2.4.3 - Lancer les tests

Pour rouler les tests du Frontend, il faut utiliser la commande "**yarn test**". Cela calculera en plus la couverture de test du code. Présentement, environ 67% des lignes du code sont testées ce que nous considérons comme adéquat. Nous utilisons les *snapshots Jest* afin de tester l'apparence de l'application [6]. Cela permet de générer le code HTML et de le sauvegarder de sorte que si l'interface change, le test échoue. Cela permet d'éviter de modifier l'apparence par erreur.

Pour le backend : `./run_tests.sh`

Ici, nous avons testé les fonctionnalités comprises dans les classes utilisées pour représenter les drones réels, les drones simulés ainsi que la communication dans *AppChannelCommunicate*. Il a été plus difficile de tester les fonctions de l'application principale *App.py*, puisque c'est un fichier qui fournit un *API* et que les fonctions utilisent des variables globales qui ne sont donc pas encapsulées dans une classe. Il n'est donc pas possible de tester certaines fonctions mais celles-ci ne font qu'appeler des méthodes de classes testées. Les fichiers de tests sont compris dans "backend/src/tests".

Pour le firmware : `./run_test.sh`

La majorité des fonctionnalités sont testées. Nous n'avons pas testé les fonctions des fichiers *Main.c* et *MissionCommands.c*, puisque ces fonctions ne font qu'appeler d'autres fonctions des fichiers testés. Pour les tests, les fichiers sont contenus dans "crazyflie-firmware/test/project" et nous avons créé des fichiers *Mock* contenus dans "crazyflie-firmware/test/testSupport", afin de simuler le comportement des fichiers à tester sans la présence d'un drone exécutant le code.

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

### 3.1 Requis fonctionnels

- R.F.1: À la réception de la commande "Take off", les drones se mettent à explorer de manière autonome la pièce voulue sans avoir besoin d'autres commandes pour le faire.
- R.F.2: Le système fonctionne pour un nombre arbitraire de drones. Toutefois, il y a quelques changements à effectuer. D'abord, l'adresse physique du drone doit être changée en écrivant la *eeprom* à une adresse unique. De plus, cette adresse doit être ajoutée dans le firmware.

- R.F.3: Les drones utilisent leurs capteurs pour éviter les obstacles et les autres drones ainsi que pour changer de direction.
- R.F.4: Les commandes “Take off”, “Land”, “Software update” et “Return to base” sont disponibles sur l’application Web et effectuent le comportement voulu pour les drones réels et simulés (sauf pour le “Software update” des drones simulés).
  - R.F.4.1: La mise à jour des drones est possible à travers l’envoi de paquets binaires en utilisant l’API de Bitcraze et ce, uniquement lorsque les drones ne sont pas en vol [2].
  - R.F.4.2: Le retour à la base rapproche les drones de la station au sol jusqu’à ce que la puissance du signal RSSI soit minimale et donc que ceux-ci soient à moins d’un mètre de la station.
  - R.F.4.3: Les drones ne décollent pas avec un niveau de batterie inférieur à 30% et reviennent à la base à ce pourcentage.
- R.F.5: Le nombre de drones, leur état, leur vitesse et leur niveau de batterie sont affichés sur l’interface Web et ce, pour les drones simulés et réels. De plus, la carte générée est montrée en temps réel. Les informations sont rafraîchies chaque seconde.
- R.F.6: Les drones explorent la pièce en suivant l’algorithme expliqué dans la section 2.1.2 et la carte représente les points observés par ceux-ci. La précision de la carte est excellente pour les drones simulés, alors qu’elle est plus faible en ce qui concerne les drones réels.
- R.F.7: À partir du Frontend, il est possible d’entrer manuellement les positions des drones en temps réel. L’intégration de leurs mesures capturées est alors possible.

## 3.2 Requis matériels

- R.M.1: L’installation des *STEM Randging bundle* est complète et fonctionnelle pour les deux drones utilisés.
- R.M.2: Nous envoyons des paquets binaires aux deux drones réels avec une seule *Crazyradio PA* et ce, avec une classe *AppchannelCommunicate* fournie par l’API de Bitcraze [2].
- R.M.3: Nous n’avons pas ajouté de nouveaux modules sur les drones autres que ceux fournis.
- R.M.4: La station au sol est un ordinateur portable avec une *Crazyradio PA* connectée par porte-USB [3].

## 3.3 Requis de conception

- R.C.1: Nous avons validé nos algorithmes d’exploration et de retour à la base sur la simulation avant de les implémenter sur les drones réels.



- R.C.2: Des tests pour le frontend, backend ainsi que le firmware ont été ajoutés afin d'assurer la qualité ainsi que le fonctionnement du code. Or, toutes les fonctionnalités ne sont pas testées.
- R.C.3: Le frontend contient des tests "snapshot" qui vérifient que l'interface n'est pas modifiée par erreur. Le backend et le firmware contiennent des tests unitaires qui valident les fonctions implémentées précédemment. Encore une fois, les fonctionnalités ne sont pas toutes testées.
- R.C.4: Comme mentionné ci-haut, lors de l'exécution du script `./start.sh` la simulation ainsi que les autres composants du système sont démarrés.
- R.C.5: L'environnement virtuel pour les drones simulés dans ARGoS est généré avec des formes ressemblant à des boîtes. Celles-ci sont placées de manière aléatoire dans l'arène.
- R.C.6: La mise à jour logicielle avec les drones simulés n'était pas requise et n'a pas été implémentée.

### 3.4 Requis logiciel

- R.L.1: Les drones sont programmés avec le *Bitcraze Crazyflie Firmware* (en langage C) fournit en début de projet et la station au sol communique avec eux à l'aide de l'*API* de Bitcraze (en langage Python) [1], [2].
- R.L.2: Dû au fait que les drones sont aptes à se détecter comme obstacle, la communication interdrone n'est pas nécessaire. Toutefois, la communication *P2P* avait été établie, mais demeure inutilisée [4].
- R.L.3: Le signal RSSI est utilisé afin d'estimer la distance entre le drone et la station au sol, plus précisément la *Crazyradio PA* [3].
- R.L.4: L'interface est une application Web qui est compatible avec tous les navigateurs récents. De plus, l'application est déployée sur Firebase et peut être consultée à partir d'un appareil mobile. Il est possible d'entrer l'IP de la station au sol afin de contrôler les drones depuis un appareil qui n'est pas dans le même réseau que la station avec une simple redirection de ports.
- R.L.5: Il y a un bouton pour afficher les logs à partir du frontend. Les logs sont générés dans le backend et enregistrés dans un fichier `.log` qui est lu lorsqu'il y a une requête pour obtenir les logs.
- R.L.6: Les mesures du *ranging deck* sont envoyées par les drones à la station au sol à chaque fois que celle-ci demande leurs informations. Cela se produit chaque seconde.

### 3.5 Requis de qualité

- R.Q.1: Le format du code est uniforme pour tous les fichiers du même composant (frontend, backend, simulation et firmware). Les différences entre les composants sont expliquées par le fait que ceux-ci sont programmés en langages différents.
- R.Q.2: La majorité des fichiers ont des longueurs raisonnables.
- R.Q.3: Les noms sont clairs et représentatifs des variables et des classes.
- R.Q.4: L'organisation des fichiers du code est logique et simple à comprendre.

Bref, nous remarquons que l'affichage de la carte sur la station au sol n'est pas très précis lorsque nous utilisons les drones réels. Puisque l'affichage se fait presque parfaitement avec la simulation, nous déduisons que le problème avec les drones réels est dû à l'imprécision physique des capteurs et non à notre code. De plus, les tests n'ont pas été faits pour la totalité des fonctionnalités à cause de l'architecture utilisée pour le fichier *App.py* du backend et parce que notre système global est orienté fonctions et non objets. Aussi, le protocole *P2P* de Bitcraze n'a pas été implémenté pour l'évitement des drones réels puisqu'ils s'évitent la majeure partie du temps en se détectant comme des obstacles [4]. Un autre échec est que la simulation est non fonctionnelle avec un nombre de drones supérieur à 7, étant donné que les drones explorent à des hauteurs différentes. Les drones dépassant ce nombre vont donc être bloqués dans le plafond de l'arène et ne pourront pas explorer.

### 3.6 Requis bonus

Nous avons eu le temps d'ajouter quelques fonctionnalités qui n'étaient pas requises. Par exemple, il est possible de sauvegarder la carte dans la base de données et de la récupérer plus tard pour continuer une exploration. De plus, puisque le frontend est déployé sur *Firebase Hosting*, nous pouvons utiliser l'interface de notre application sans avoir accès au code et ce, à partir de n'importe quel appareil. Si l'appareil n'est pas dans le même réseau que la station au sol qui possède la radio, il suffirait d'effectuer une redirection de port et d'entrer l'IP de la station au sol pour la contrôler à distance. Ensuite, nous avons implémenté un bouton qui permet d'exporter la carte en format SVG.

## 4. Déroulement du projet (Q2.5)

### 4.1 Réussites du déroulement du projet

Nous avons réussi à satisfaire les exigences techniques du mandat initial pour les jalons de la "Preliminary Design Review" et pour la "Critical Design Review", et ce, dans les

délais prévus initialement. De plus, comme indiqué à la section précédente, la majorité des requis du projet ont été satisfaits et le comportement global du système est celui attendu, soit de pouvoir contrôler un essaim de drones réels ou simulés à l'aide d'une interface Web. Pour la gestion du projet et l'implication des membres de l'équipe, les tâches assignées respectivement chaque semaine étaient accomplies dans le temps souhaité sauf lorsqu'il y avait des problèmes techniques. Dès le départ, il y avait un respect de l'échéancier et une répartition des tâches impliquant tous les membres, ce qui a permis une meilleure efficacité et l'émergence d'une relation de confiance dans l'équipe. Cela a aussi permis une grande entraide entre les membres et un climat où chacun n'avait pas peur de fournir des critiques constructives aux autres et de montrer son point de vue lors des rencontres. Enfin, le mode de fonctionnement décentralisé, où chaque membre est au courant des avancements et des problèmes des autres, a été maintenu lors du projet. Un coordonnateur a été nommé uniquement pour permettre une meilleure coordination des différentes composantes du système (station au sol, application Web, drones et simulation) et des avancements de chacun, comme expliqué lors de la CDR. Celui-ci s'occupait aussi majoritairement du respect de l'échéancier et d'établir les tâches pour chaque semaine.

## 4.2 Échecs du déroulement du projet

En ce qui a trait aux problèmes rencontrés lors du projet, il a été difficile d'établir une estimation représentative de la réalité pour le temps associé à chaque tâche. En effet, le temps total de travail prévu et réévalué lors de la CDR était de 535 heures. En regardant le temps cumulé des tâches réalisées, nous obtenons en réalité une valeur d'environ 500 heures. Cette valeur exclut le temps passé à réaliser les rapports pour les remises ainsi que le temps passé pour l'établissement des tâches hebdomadaires, puisque ces heures ne faisaient pas partie de la somme totale évaluée à la CDR. Le temps réel final est donc inférieur à celui planifié. Cela est dû à la grande difficulté d'établir avec exactitude les tâches à réaliser chaque semaine. En effet, il y avait toujours des tâches qui s'ajoutaient à celles planifiées en début de semaine et qui n'étaient donc pas classifiées. Leur temps de réalisation n'a donc pas pu être comptabilisé. De plus, malgré le fait que certaines tâches ont été vraiment plus longues que prévu à cause de problèmes techniques, comme l'implémentation du "socket" pour les drones simulés ou les algorithmes d'exploration et de retour à la base, toutes les petites tâches non comptabilisées et les tâches beaucoup moins longues que prévu font que le nombre d'heures est plus petit que prévu. Or, ces tâches plus longues que prévu ont retardé l'avancement du projet et ont donc causé le non-respect de certains requis, comme indiqué dans la section précédente. Notamment, l'implémentation des algorithmes sur les drones réels a été grandement retardée par le changement total des algorithmes prévus, comme expliqué dans la section 2.1, après la CDR, ce qui a causé les problèmes de qualité de l'exploration et du

retour à la base sur les drones réels. Enfin, ce retard et le non-respect de certains requis par le fait même expliquent aussi le retard sur le temps de travail prévu.

## 5. Travaux futurs et recommandations (Q3.5)

Pour ce qui est des tâches futures, il y a des éléments importants à ajouter afin d'être parfaitement conformes aux exigences. En effet, la manière de définir la quantité de drones qui vont être utilisés, que ce soit dans la simulation ou concernant les drones physiques, n'est pas implémentée dans l'interface. Du côté de l'application interne, l'architecture mise en place ne prend pas en compte le rôle du sous-système qui est un sous-système de communication style *backend*. Ce sous-système ne devrait pas avoir d'état associé, cela en fait donc une mauvaise pratique pour ce type d'architecture. Un autre aspect à mentionner est la précision de la carte formée par les informations obtenues des capteurs des drones physiques. La représentation de la carte formée ainsi est loin d'être parfaite et donc pourrait être améliorée. Finalement, une bonne dernière tâche serait l'implémentation d'algorithmes plus complexes, comme celui utilisé dans la simulation, sur les drones réels, puisque ces derniers ont présentement des algorithmes simples de déplacement pour explorer et retourner à la base.

Du côté des recommandations, il y a des fonctionnalités qui sembleraient grandement améliorer l'expérience d'utilisation et de développement entourant le projet. D'abord, la possibilité d'envoyer des commandes à certains drones, donc des commandes spécifiques à des drones, faciliterait la communication entre l'application interne et les drones. Ensuite, la création d'une carte en trois dimensions, plutôt que deux, ne serait pas si complexe à implémenter et améliorerait la qualité de l'information fournie à l'utilisateur. Par après, l'utilisation de capteurs plus précis pour les drones physiques, tels des caméras ou un système de géolocalisation, permettrait d'avoir plus de facilité à formuler un support visuel compréhensible. Finalement, l'utilisation d'une caméra non comme senseur, mais plutôt comme outil de surveillance, permettrait d'ajouter à la tâche d'exploration d'une pièce une couche de sécurité additionnelle.

## 6. Apprentissage continu (Q12)

### 6.1 - Laurent Bolduc

#### 1. *Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*

Une lacune identifiée au niveau de mes connaissances est directement liée à ma capacité de déboguer le répertoire du projet. Dû à la complexité de l'exécution des différents sous-

systèmes et de leurs interactions, entre autres à travers Docker, il m'est arrivé à maintes reprises de ne pas être capable de travailler adéquatement.

*2. Méthodes prises pour y remédier.*

Une méthode utilisée afin d'y remédier a été de communiquer avec mes partenaires afin d'avoir plus d'aise à identifier les problèmes, tout en enrichissant ma compréhension du système. De plus, lorsque cela ne suffisait pas, contacter les chargés de cours et se renseigner en ligne furent deux méthodes également très utiles.

*3. Identifier comment cet aspect aurait pu être amélioré.*

Puisque j'avais déjà travaillé avec Docker auparavant, il m'aurait été grandement avantageux de réviser mes connaissances avant le début du projet. De plus, ne pas utiliser deux postes de travail permettrait de réduire les problèmes potentiels de moitié.

## 6.2 - Jacob Brisson

*1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*

Mes lacunes pour ce projet étaient mes savoirs par rapport à la création d'une application qui utilise des langages de programmation différents et qui doit relier plusieurs composantes à travers le réseau. De plus, l'utilisation de "Docker" m'était inconnue au début du projet.

*2. Méthodes prises pour y remédier.*

Pour améliorer mes connaissances par rapport à "Docker", j'ai utilisé les connaissances développées dans le cours INF8480 - Systèmes répartis et infonuagique. Aussi, en ce qui concerne les connaissances réseau, j'ai effectué plusieurs recherches sur le Web en début de projet pour mieux comprendre le fonctionnement des "sockets" et de la connexion de plusieurs composantes d'une application sur le réseau.

*3. Identifier comment cet aspect aurait pu être amélioré.*

Pour améliorer ces aspects avant le début du projet, j'aurais pu essayer de me familiariser avec la technologie de "Docker", qui était utilisée dans le milieu où je travaillais cet été et où des tutoriels étaient disponibles. Enfin, j'aurais pu utiliser les connaissances acquises lors du cours INF3405 - Réseaux informatiques afin de les appliquer dans un projet personnel avant le projet.

## 6.3 - Jordan Lecourtois

### 1. *Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*

Avant le début du projet, je n'avais jamais eu à travailler avec un projet complexe sur Git, incluant des sous-modules et un flux de travail incluant des *pull request*.

### 2. *Méthodes prises pour y remédier.*

Afin de remédier à cette lacune, j'ai appris à utiliser des outils de gestion de répertoire Git, en plus de communiquer avec mes partenaires qui étaient davantage à l'aise avec l'outil de gestion Git.

### 3. *Identifier comment cet aspect aurait pu être amélioré.*

Cet aspect aurait pu être amélioré en utilisant les outils de gestion de répertoire Git à ma disposition et en m'informant davantage grâce à la documentation et aux différentes plateformes d'information disponible en ligne.

## 6.4 - Matthew Paoli

### 1. *Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*

Au cours du projet, j'ai rencontré plusieurs difficultés en raison de mon manque de connaissances avec le langage de programmation Python ainsi que de la manière de communiquer entre le frontend et le backend.

### 2. *Méthodes prises pour y remédier.*

Afin de m'améliorer avec Python, j'ai pris la responsabilité d'écrire différentes fonctions dans le backend ainsi que de travailler avec le *Crazyflie Python API*. De plus, pour me familiariser avec la manière de communiquer entre le frontend et le backend j'ai lu sur plusieurs aspects, notamment les requêtes GET et POST [2].

### 3. *Identifier comment cet aspect aurait pu être amélioré.*

Une manière que j'aurais pu employer afin de m'améliorer sur ces sujets aurait été de travailler davantage sur le frontend à la place du firmware. De plus, dans des projets antérieurs, tels que le cours LOG2990 (Projet 2), j'aurais pu essayer de mieux comprendre comment on doit procéder pour communiquer entre les deux.

## 6.5 - Philippe Savard

### 1. *Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*

Au cours du projet, j'ai remarqué un manque important au niveau de mes connaissances pour certaines technologies telles que Docker, ARGoS, git, etc. En effet, lorsque notre équipe a entamé le projet, je n'avais pas toutes les connaissances préalables quant à l'utilisation efficace de ces outils. En y repensant, cette lacune dans mes savoirs est un facteur récurrent lors de la réalisation de différents grands projets de conception (par exemple INF1900, LOG2990 et INF3995).

### 2. *Méthodes prises pour y remédier.*

Une solution à cette lacune est d'effectuer une analyse de la documentation des outils utilisés afin d'en apprendre davantage sur leur fonctionnement. Il s'agit en fait d'un apprentissage continu qui varie en fonction de la tâche à réaliser. Cette pratique m'amène à me mettre à jour sur les outils technologiques utilisés dans la pratique. Une solution alternative (souvent préférable selon le cas) est l'aide par les pairs. Il s'agit de poser des questions constructives aux membres de l'équipe afin de combler mes lacunes dans mes savoirs à l'aide des connaissances d'autrui. Par exemple, notre équipe a utilisé des sous-modules afin de séparer les différentes branches du *git*. Afin de combler mes lacunes de connaissances à ce sujet, j'ai demandé à un membre de l'équipe plus à l'aise avec *git* de m'expliquer comment utiliser efficacement cet outil. J'ai donc appris comment utiliser les sous-modules *git*.

### 3. *Identifier comment cet aspect aurait pu être amélioré.*

Lors de la lecture de la documentation pour certains outils, notamment Argos, certains éléments nous semblaient vagues ou peu documentés. Il ne suffisait donc plus de simplement lire la documentation et d'en discuter entre mes collègues, car cela ralentissait le développement du projet. Une bonne pratique dans ce genre de situation aurait été de contacter directement l'équipe de développement de l'outil en question afin d'obtenir plus d'informations sur son fonctionnement.

## 6.6 - Simon Tran

### 1. *Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*

Au début du projet, j'ai noté que je n'avais pas beaucoup d'expérience avec Python. Ainsi, j'ai parfois eu de la difficulté à travailler sur le backend. De plus, je ne connaissais pas Docker et j'ai dû apprendre à l'utiliser, puisque j'étais le membre de l'équipe qui a configuré les différents projets pour qu'ils fonctionnent avec Docker.

## 2. *Méthodes prises pour y remédier.*

J'ai lu de la documentation sur Python afin de me familiariser avec le langage. Pour apprendre à utiliser Docker, en plus de lire la documentation, j'ai écouté des vidéos et suivi certains tutoriels sur le Web.

## 3. *Identifier comment cet aspect aurait pu être amélioré.*

J'aurais pu utiliser Python et docker dans des projets personnels avant le début du projet afin d'avoir plus d'expériences avec ces technologies.

# 7. Conclusion (Q3.6)

Lors de la réponse à l'appel d'offres, notre vision générale du système était encore peu développée. Certaines composantes, notamment les drones, n'étaient pas disponibles rendant la planification initiale difficile. N'ayant pas une idée claire de comment nous allions approcher certains problèmes (exploration, retour à la base, communication entre la simulation et le backend, etc.), plusieurs des solutions proposées initialement ont été complètement modifiées ou remplacées par de nouvelles idées. Par exemple, l'algorithme initial pour l'exploration et le retour à la base utilisait une méthode de recherche où les drones suivaient les murs et changeaient d'orientation par rotation. Dans notre version finale, telle qu'expliquée dans ce rapport, cette méthode n'est plus utilisée. Beaucoup de temps a également été consacré à des fonctionnalités dont nous avons sous-estimé la complexité en raison de notre manque d'expérience. Un bon exemple de ce comportement est la connexion par sockets. Au début, nous croyions que l'implémentation d'un moyen de communication entre le backend et la simulation allait être relativement simple, mais beaucoup de complications ont été rencontrées et par conséquent plusieurs heures de travail supplémentaires ont été dépensées. En résumé, beaucoup de temps a été investi sur des solutions qui n'ont pas été retenues ou sur des solutions avec des estimations de temps moins longs. Afin de remédier à ce problème, il serait avantageux que dans le futur un plan de conception plus précis soit établi en considérant l'entièreté des requis. De plus, une réévaluation de notre démarche de conception nous a permis de prendre conscience de problèmes dans notre démarche. Nous avons observé qu'il aurait été bénéfique pour notre architecture globale de prendre plus de temps pour y réfléchir avant d'entamer le projet. Cela nous aurait potentiellement évité certaines erreurs de conception qui ont causé bien des maux de tête, notamment pour le développement du backend. Finalement, une erreur que nous avons commise lors de notre démarche de conception est la fréquence à laquelle nos fonctionnalités ont été testées. Effectivement, nous avons eu tendance tout au long du projet à tester nos avancements uniquement une fois qu'une fonctionnalité était complétée. Par conséquent,



ceci a parfois retardé le moment auquel nous réalisons que la fonctionnalité ne répondait pas aux requis. Alors, encore une fois, plusieurs heures de travail ont été gaspillées. Afin d'éviter ce problème, il aurait été judicieux de rédiger un plan de test pour chacune des remises, en plus d'implémenter les tests avant les fonctionnalités.

## 8. Références

- [1] Bitcraze. (2021). *Crazyflie-firmware*. <https://github.com/bitcraze/crazyflie-firmware/tree/1a4cc9acf3edefb3833b72e05095be9c8b0fc887>
- [2] Bitcraze. (2021). *Crazyflie-lib-python*. <https://github.com/bitcraze/crazyflie-lib-python/tree/master/cflib>
- [3] Bitcraze. (2021). *Crazyradio PA*. <https://www.bitcraze.io/products/crazyradio-pa/>
- [4] Bitcraze. (2020). *Peer to Peer API*. [https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.02/p2p\\_api/](https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.02/p2p_api/)
- [5] Dodds, K. C. (2021). *Testing Library* [Logiciel]. <https://testing-library.com/docs/react-testing-library/intro/>
- [6] Facebook Open Source. (2021). *Jest* (Version 26.6.3) [Logiciel]. <https://jestjs.io/>
- [7] Facebook Open Source. (2021). *React.js* (Version 17.0.2) [Logiciel]. <https://reactjs.org/docs/getting-started.html>
- [8] Flask. (2021). *Quickstart*. <https://flask.palletsprojects.com/en/1.1.x/quickstart/>
- [9] GeeksforGeeks. (2019). *Socket Programming in C/C++*. <https://www.geeksforgeeks.org/socket-programming-cc/>
- [10] Koss, S. (2021). *Date-fns* (Version 2.21.1) [Logiciel]. <https://date-fns.org/>
- [11] Lodash. (2016). *Lodash* (Version 4.0.0) [Logiciel]. <https://lodash.com/>
- [12] Material-UI. (2020) *Material-UI* (Version 4.11.1) [Logiciel]. <https://material-ui.com/>
- [13] Microsoft. (2021). *TypeScript* (Version 4.2) [Logiciel]. <https://www.typescriptlang.org/>
- [14] Okonetchnikov, A. (2021). *Lint-Staged* (Version 10.5.4) [Logiciel]. <https://github.com/okonet/lint-staged>

[15] Prettier. (2020). Prettier (Version 2.2.1) [Logiciel]. <https://prettier.io/>

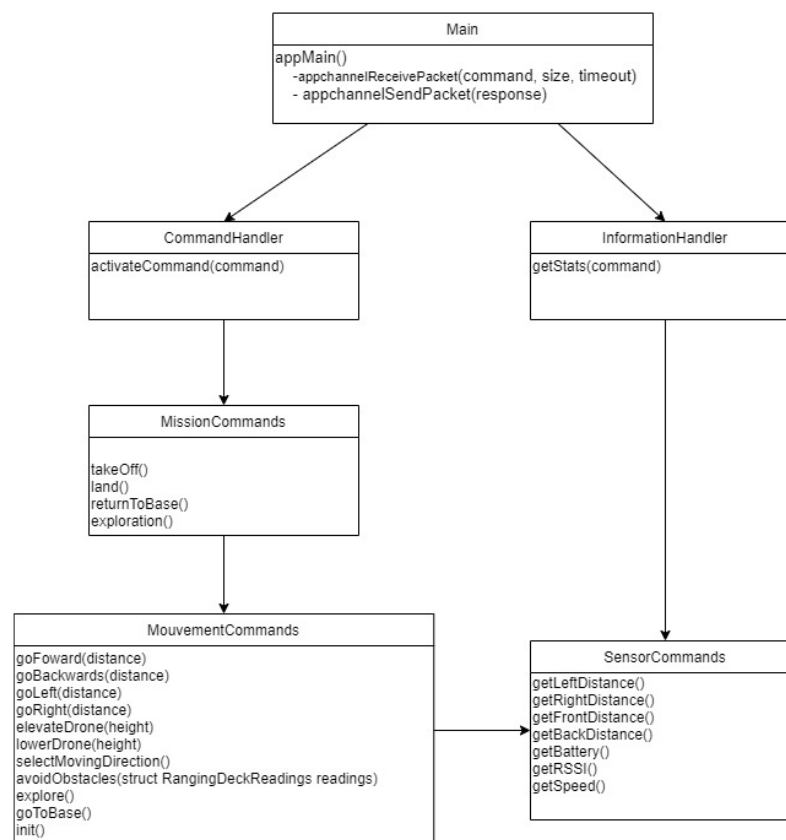
[16] Python. (2021). *socket* — *Low-level networking interface*. <https://docs.python.org/3/library/socket.html>

[17] Typicode. (2021). Husky (Version 6.0.0) [Logiciel]. <https://typicode.github.io/husky/#/>

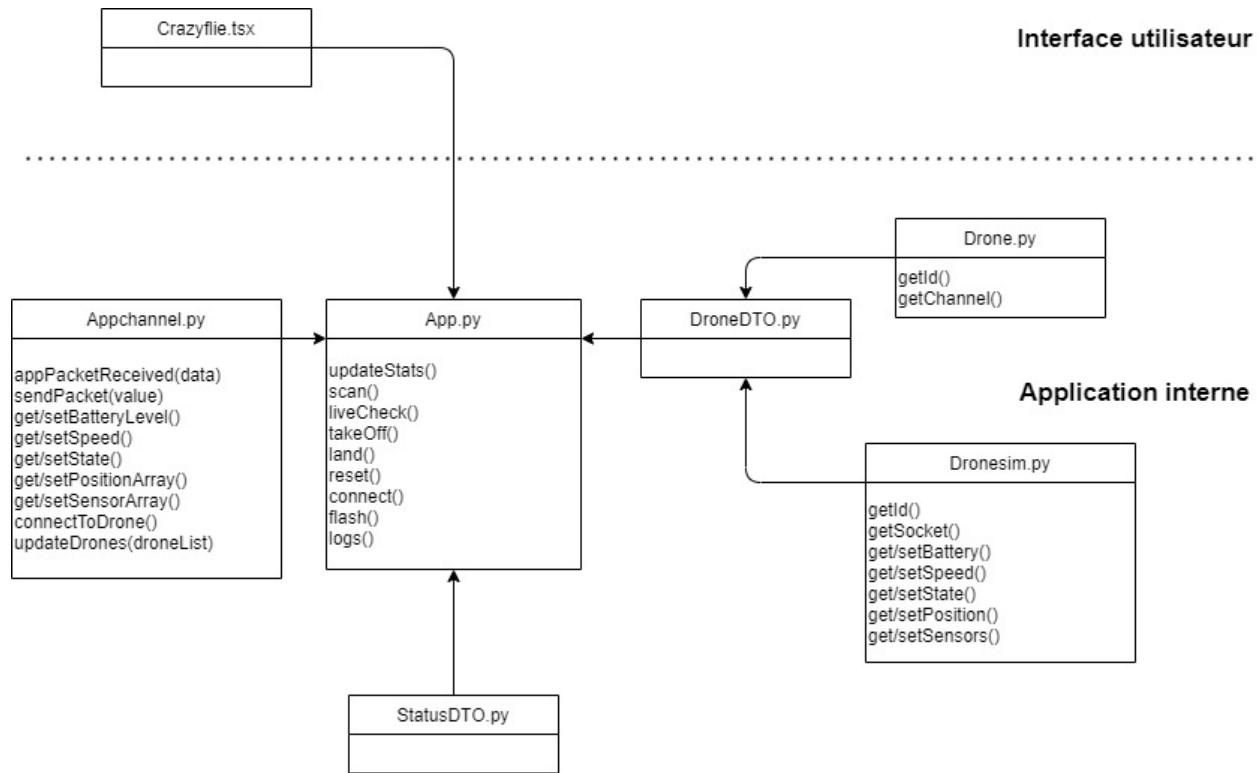
[18] Varshavskaya, P. Wavefront Planning Algorithm. Tufts University. <https://www.cs.tufts.edu/comp/150IR/labs/wavefront.html>

## 9. Annexe

**Figure 9.1 – Architecture du code embarqué**



**Figure 9.2 – Architecture de l'application interne**



**Figure 9.3 – Architecture globale du système**

