Département de génie informatique et de génie logiciel

INF3995

Projet de conception d'un système informatique

Conception d'un système aérien minimal pour exploration

Rapport final de projet

Équipe N°103

Mazigh Ouanes

Tarik Agday

Nabil Dabouz

Mohamed Yassir El Aoufir

Paul clas

21 Avril 2021

Table des matières

1. Objectifs du projet

Ce présent document agit à titre de rapport final à l'appel d'offres de l'Agence Spatiale: Système aérien minimal pour exploration, dans le cadre du cours INF3995. L'appel d'offres fournit au début du semestre nous exigeait de fournir une preuve de concept d'exploration en utilisant de petits drones Crazyflie de la marque Bitcraze dans l'espoir d'étudier la possibilité de répondre aux besoins de l'Agence Spatiale en termes d'exploration planétaire. Concrètement, nous cherchions à établir une communication et de contrôler ces drones à distance en plus d'explorer un terrain inconnu. Cette preuve de concept devait d'ailleurs se concrétiser en atteignant le 4e niveau de maturité qui est détaillé comme étant la « validation de principe reposant sur l'intégration d'applications et de concepts en vue de démontrer la viabilité ».

D'un point de vue commercial, l'Agence Spatiale a certaines attentes au niveau de son investissement et tient à jouer un rôle dans la stimulation de l'industrie. De ce fait, les projets auxquels elle prend part se doivent de maximiser le retour sur investissement tout en maintenant un niveau de qualité et d'innovation à la hauteur de leurs attentes.

Notre équipe a donc travaillé sur cette preuve de concept lors des quatre derniers mois et nous vous proposons une solution répondant aux objectifs de ce projet. En bref, notre solution est un système se divisant en cinq composantes : une station au sol, 2 drones (incluant une radio de communication), une interface client et un système embarqué permettant de programmer les drones.

Voici un résumé de ces cinq composants :

- **Service Web:** permettant l'envoi d'instructions de la part de l'utilisateur aux drones. Ce service affiche aussi certaines informations récupérées par ces derniers pendant leurs missions d'exploration.
- **Réseau de communication:** assurant un lien entre la station au sol et les drones. Le réseau est un canal de communication de 2.4GHz.
- **2 Drones Bitcraze Crazyflie 2.1:** fournis par l'agence spatiale, ils devront communiquer avec la station au sol à l'aide du réseau établi. Ils répondront aux instructions (vol, atterrissage, mise à jour système et retour à la base) indiquées par l'utilisateur.
- **Station au sol:** Interface Web assurant une communication avec les drones à l'aide de leurs antennes du réseau établi.
- **Base de données NoSQL:** stockage des données récupérées par les drones lors des missions d'explorations. Inclus les données suivantes : positions des drones, obstacles détectés, niveau de batterie, orientation des drones, etc.
- **Système embarqué:** Partie logiciel fournie aux microcontrôleurs des drones assurant l'implémentation de notre code.

2. Description du système

2.1. Logiciel embarqué (Q4.5)

Notre solution logiciel embarqué est grandement influencé par le répertoire open-source de Bitcraze. Pour faciliter l'ajout de fonctionnalité et la clarté de notre code embarqué nous avons séparé notre code en plusieurs fichiers clefs. Tout d'abord, voici une vue d'ensemble du controlleur général du Crazyflie et de l'interaction entre chacune des grandes composantes qui le composent :

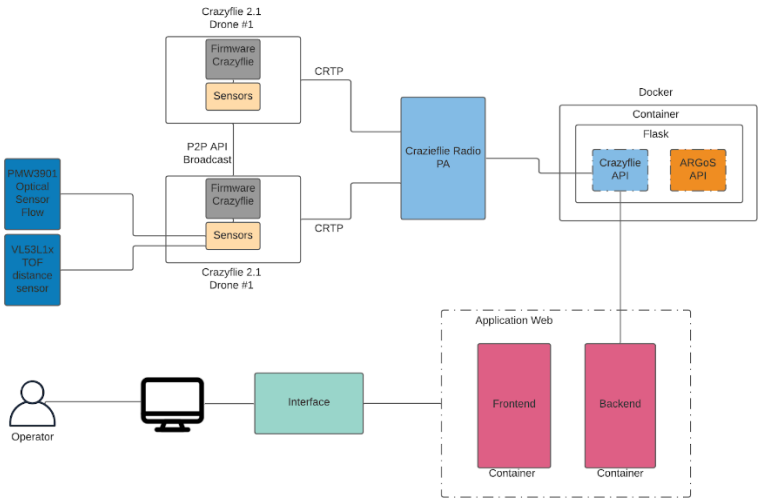

Figure 1 Archicture globale du système embarqué

Compréhension de la structure du logiciel embarqué :

- app.c : Ce fichier est le gestionnaire principal du code embarqué et le seul fichier qui communique avec notre serveur et le fichier app.py
- appData.h : Ce fichier déclare la structure des différentes données utilisés dans notre logiciel embarqué mais surtout AppData qui est une structure de donnée qui permet de communiquer les informations des drones aux serveurs puis client.
- batteryUnit.c : Ce fichier permet de set le treshold de la batterie en pourcentage et de get la valeur des batteries du drone.
- lowLevelCommands.c : permet d'expliciter aux drones les différentes instructions que celui-ci doit faire suivant les différentes commandes transmises. 5 instructions fondamentales d'actions sont utilisés pour contrôler les mouvements des drones :
    - Land qui permet de progressivement diminuer la vitesse de rotation des rotors et faire atterrir le drone
    - Hover permet de garder la vitesse de rotation des rotors à une certaine altitude donnée en ne spécifiant aucun mouvement selon un repère cartésien en x,y et z
    - Vel_command permet d'indiquer la vitesse des rotors et donc de déplacement du drone.
    - Shut_off_engines permet d'arrêter les rotors du drone
- sensorsUnit.c : Ce fichier de code permet de récupérer les données provenant du RSSI, des batteries, du multiranger deck et du flow deck.
- SGBA.c : Ce fichier permet d'implémenter partiellement la machine a état du Swarm Gradient Bug Algorithm tel que spécifié dans la figure suivante :

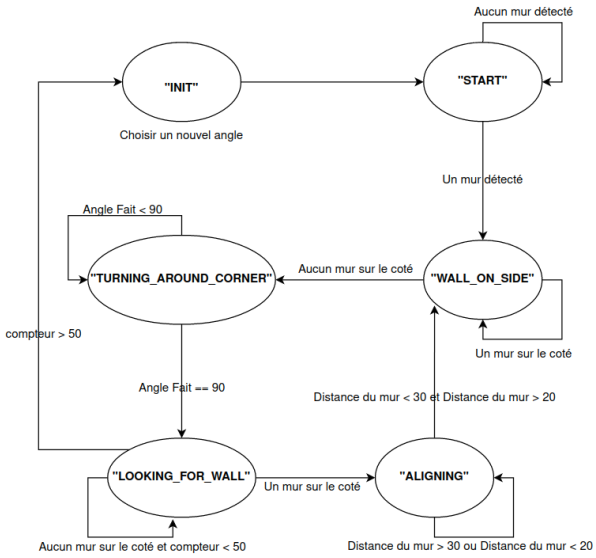

Figure 2 Machine à état du suiveur de mur

- stateMachine.c : Ce fichier permet de spécifier les différentes situations associées à 5 états que nous avons établis en utilisant notre étude des différentes situations de vol pour communiquer au client l'état du drone. Nous avons deux fonctions qui établissent les différents états des drones. setNextAction() permet d'envoyer les instructions aux drones nécessaires pour son état actuel. HandleLasCommand() permet

Premièrement, le fichier app.c reçoit les commandes provenant du backend et se charge d'appeler les bonnes commandes en fonction de la structure de appData recu. La structure de AppData est déclaré dans le fichier appData.h C'est la fonction appMain() qui se charge d'implémenter toute la logique de notre code embarqué.

InitApp() permet d'initialiser les commandes, l'état et la communication entre le serveur et les drones. Elle s'occupe également de la communication Peer2Peer entre drone.

handleLastCommand() est déclaré dans le fichier stateMachine.c et permet de récupérer la dernière commande envoyé par le backend aux drones et d'effectuer les commandes nécessaires soit d'attérir, de prendre vol, d'explorer ou de revenir à la base.

UpdateSensorsData() permet de mettre a jour les données du capteur

SetNextAction() prends l'état du statemachine envoyé par handleLastCommand() et indique aux drones les actions à appliqués.

commanderSetSetpoint() cette méthode met à jour la structure des données utilisés par la librairie Commander.

handleCommunication() permet d'intercepter les commandes envoyés aux drones.

Notre application supporte jusqu'à un total de 5 drones. Pour lancer notre application il faut s'assurer que les Crazyflie écoutent sur la channel 56, et que leur adresses vont de E7E7E7E701 à E7E7E7E705

Dans le cas où moins de 5 drones vont être utilisés, il faut utiliser les plus petites adresses. Tel que pour n drones choisis, il faut avoir des adresses comprises entre E7E7E7E701 et E7E7E7E70n. Par exemple, pour l'utilisation de 3 drones, il faut utiliser les adresses suivantes : E7E7E7E701,E7E7E7E702 E7E7E7E703

2.2. La station au sol (Q4.5)

Notre station au sol permet à l'utilisateur d'interagir avec notre système. De manière plus précise, notre système est composé de composants/services différents. Il est à noter que chacun de ses services est conteneurisé dans un conteneur docker. L'ensemble de ses services sont orchestrés et synchronisés à l'aide d'un docker-compose.

Nous allons ici présenter ces différents services et leur fonctionnement en se basant sur le schéma suivant :

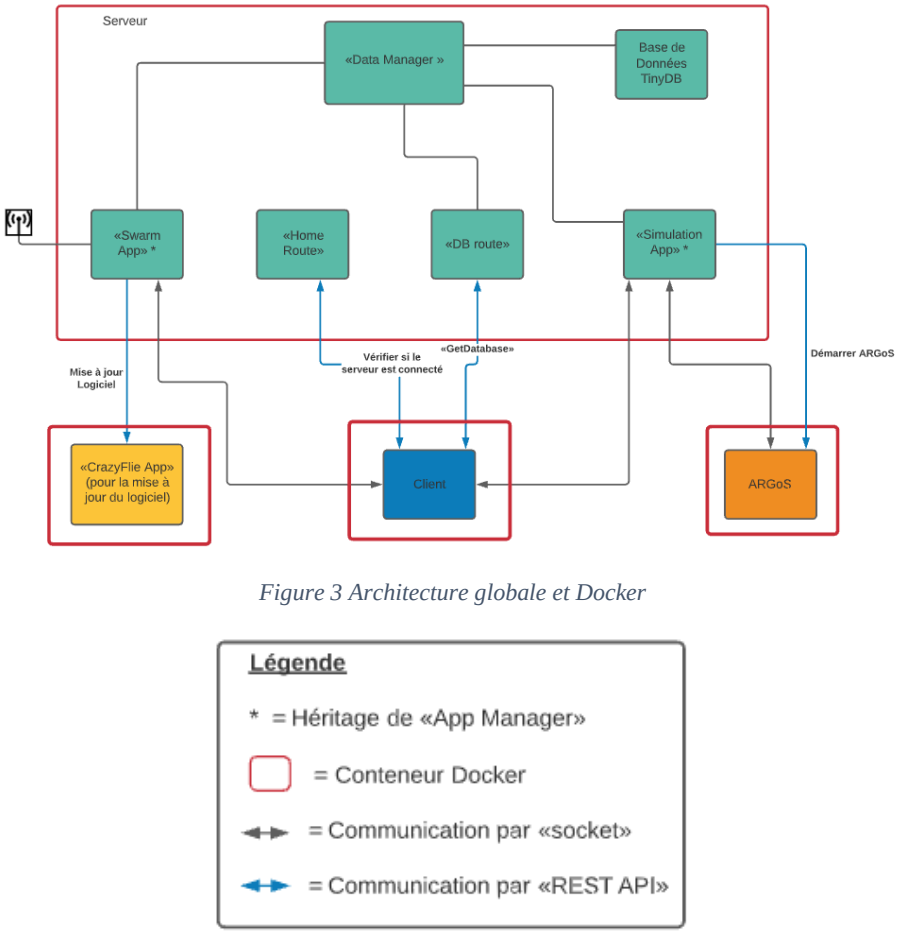

Figure 3 Architecture globale et Docker

2.2.1. Crazyflie App

C'est un petit serveur codé en Python3 et utilisant Flask. Il permet la fonctionnalité de mise à jour logiciel ("software update") des drones. Ce serveur supporte uniquement les communications par "REST API".

2.2.2. ARGoS

Dans ce conteneur, nous avons un serveur Flask utilisant "REST API". Il permet le lancement et l'arrêt de la simulation ARGoS. Une fois la requête de lancement de l'application reçue du serveur principal, ce serveur s'occupe de la préparation de l'environnement de la simulation à l'aide d'un script "launch.sh" puis lance un processus pour la simulation ARGoS en utilisant la librairie Python "subprocess". L'interaction avec le logiciel de simulation ARGoS (envoie de données et réception de données) est faite à l'aide de SocketIO et à travers la classe "CrazyflieLoopFunction". Voici un diagramme illustrant l'architecture globale du conteneur et ses interactions avec le serveur principal.

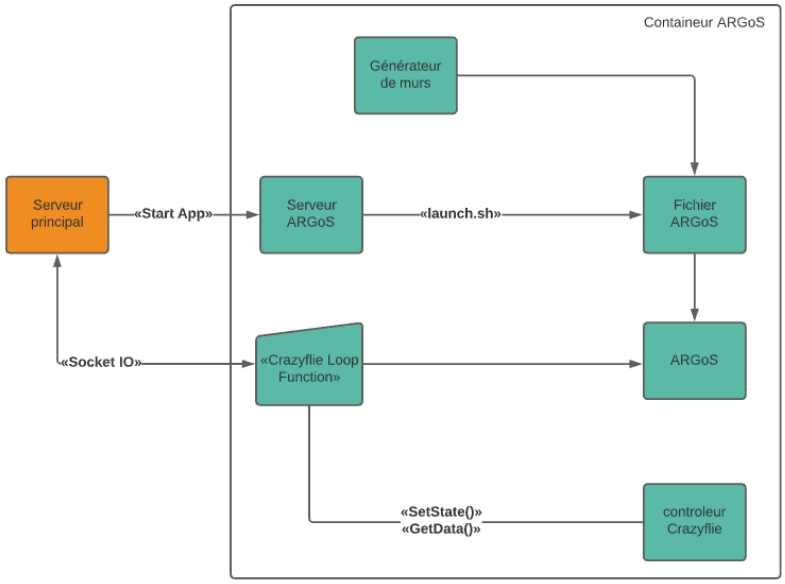

Figure 4 Architecture conteneur ARGoS

2.2.3. Client

C'est notre page Web. C'est ce qui permet à l'utilisateur d'interagir avec le reste du système en boîte noire. Nous utilisons la librairie React JS pour l'implémentation du client. Le client interagit uniquement avec le serveur principal, il communique avec ce dernier par "REST API" lorsqu'on se situe dans la page d'accueil pour savoir si le serveur est connecté et pour accéder aux routes de la base de données. Et il communique avec le serveur principal par Socket lorsqu'on se trouve dans la « Swarm App » ou dans la « Simulation App ».

2.2.4. Serveur Principal

Le serveur principal de notre application est codé en Python3, utilise Flask et SocketIO. Ce service est une sorte de pont dans notre application, c'est lui qui permet de relier les différentes composantes pour former un système unique. En effet, le client communique uniquement avec le serveur principal, ainsi, si le client veut par exemple lancer le simulateur, il va communiquer avec le serveur principal qui lui va communiquer avec le service ARGoS. Nous retrouvons ce même raisonnement lorsque le client veut réaliser la fonctionnalité "software update ".

Le serveur principal supporte les communications par "REST API" et par "Socket".

Le serveur principal possède deux applications, la «Swarm App» et la «Simulation App », Du point de vue du client, il n'y a pas de différence entre l'utilisation de chacune de ces deux applications.

2.2.4.1. Swarm App

C'est l'application dans notre serveur qui se charge de gérer la logique nécessaire à l'utilisation des drones physiques. Elle permet en plus de communiquer avec le service « CrazyFlie App » pour lancer la mise à jour du logiciel des drones. Cette classe hérite de l'interface AppManager.

Voici un diagramme illustrant cette partie de l'application :

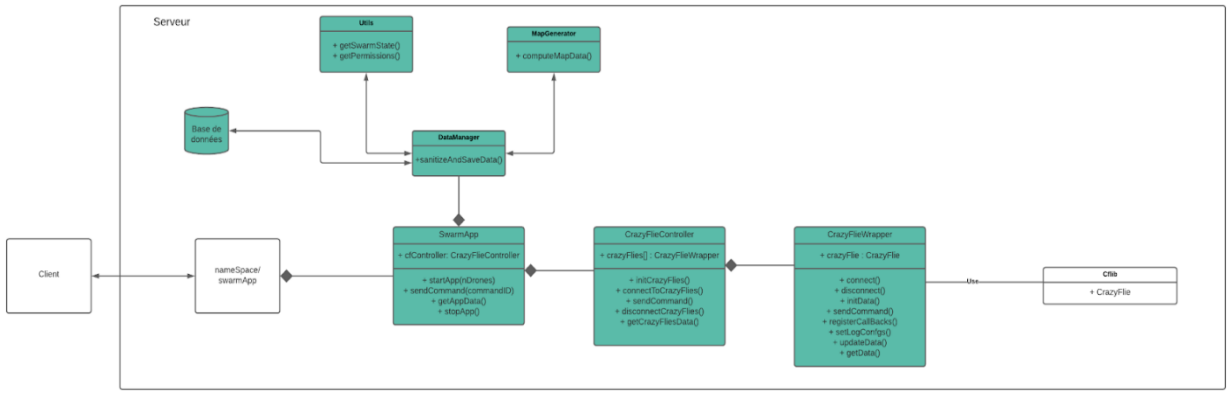

Figure 5 Architecture SwarmApp

- La classe «CrazyFlieWrapper» permet d'interagir avec les drones physiques en utilisant la librairie cflib. Nous avons donc un objet de cette classe instancié par drone physique utilisé.
- La classe «CrazyFlieController» permet de gérer l'essaim de drones physiques. Ainsi, un objet de cette classe possède plusieurs objets de type «CrazyFlieWrapper»

2.2.4.2. Simulation App

C'est l'application dans notre serveur qui permet de gérer la logique en lien avec l'utilisation de la partie simulation de notre système. Elle permet notamment l'interaction avec le service ARGoS en mettant en relation via Socket ce dernier avec le service Client (comme pour une application de chat), elle permet également de lancer le simulateur via une "REST API" (toujours en communiquant avec le conteneur ARGoS).

Ainsi, cette application agit comme un intermédiaire entre le Client et le conteneur ARGoS, de ce fait, son rôle se résume à la gestion de cette mise en relation et de l'utilisation du «DataManager» qui est identique à celle de la «Swarm App».

Cette mise en relation consiste à :

- Notifier chacun des potentiels clients connectés (service Client et Service ARGoS) lorsqu'un client se connecte ou se déconnecte (BroadCast).
- S'assurer de transmettre les communications au bon client en fonction du client émetteur.

- La gestion des différents scénarios possibles induit par l'utilisation d'une communication continue bilatérale (gestion des erreurs, interruption de la communication lorsqu'un client se déconnecte …)

Cette classe hérite de l'interface "AppManager".

2.2.4.3. DataManager

Le « DataManager » est une classe par laquelle passe toutes les communications du serveur principal. Il est utilisé de la même façon par la "Swarm App" et par la "Simulation App". Il permet à l'aide des sous-classes qu'il utilise de :

- Traiter les erreurs potentielles dans les données
- Générer les données nécessaire à la carte
- Générer l'état de l'essaim de drones par rapport aux états de chacun des drones
- Générer l'état des permissions en fonction de l'état de l'essaim de drones
- Sauvegarder ces données dans une base de données locale pour le système de logs.

Voici un schéma montrant les commandes disponibles en fonction de l'état de l'essaim de drones :

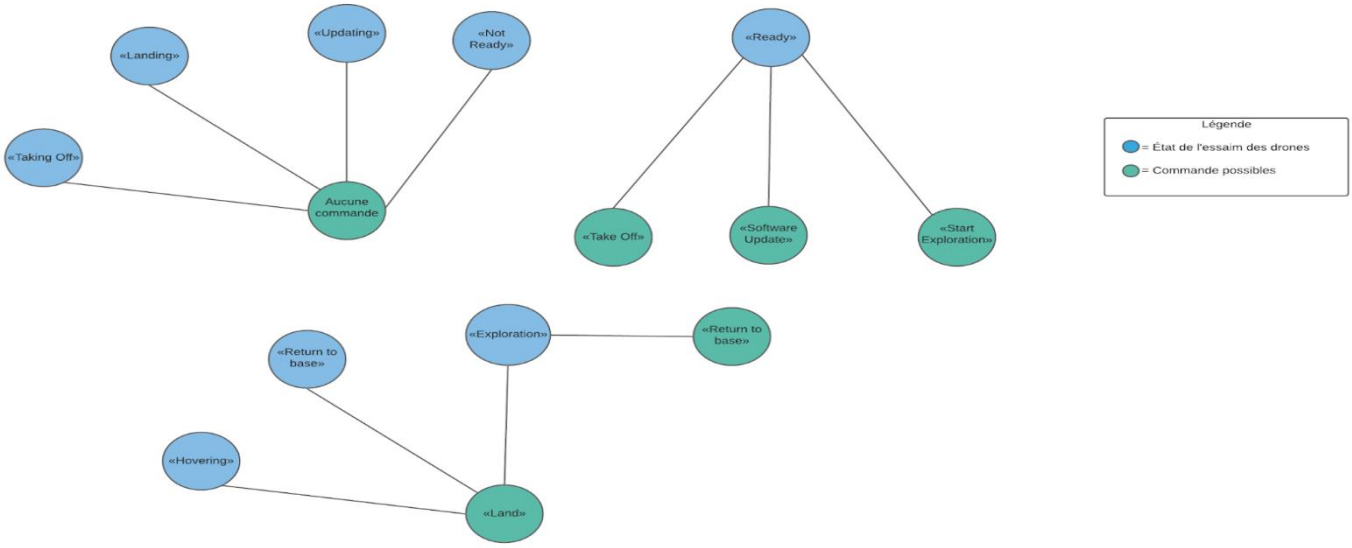

Figure 6 Gestion des permissions

2.3. L'interface de contrôle (Q4.6)

La station au sol possède une interface web qui a été construite sur le modèle CRUD (Create, Read, Update, Delete) appliqué avec une architecture utilisant des sockets. Pour construire cette application web, nous avons choisi d'utilisé la pile FReMP.

FReMP est l'acronyme pour Flask, ReactJS, MongoDB et Python. Le client léger construit avec la pile FReMP s'occupe de gérer et distribuer les différentes requêtes aux différents conteneurs Docker qui contiennent le serveur central de l'application web créer avec ReactJS et la gestion des services Python (Flask), la simulation ARGoS et la base de données MongoDB.

L'interface graphique est construite en javascript en utilisant le framework React. 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 qui facilitent l'expérience utilisateur. 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. Les tests unitaires de notre interface roulent grâce à Jest.

L'utilisation de React comme framework nous a encouragés à préférer le patron de conception de la programmation fonctionnelle à celui de la programmation orientée objet. Notre code ne contient que très peu de classes et préfère l'utilisation de composants fonctionnels. Notre interface dispose de 3 pages, une page « Home » qui permet de choisir entre le lancement de la simulation ARGoS, le lancement des drones physiques et la consultation de logs dans la base de données. La page « UserGuide » qui est possible d'être accéder à partir du boutton Learn more contient des informations sommaire sur le projet. La page « Operator » contient l'ensemble des composantes nécessaires à l'intéraction avec ARGoS et les drones physiques

Voici une revue sommaire des différentes composantes que nous utilisons dans la page Operator et plus spécifiquement la composante MainPanel qui gère les composantes de notre interface de contrôle soit Map, SwarmCommands, DroneList, SwarmStatus, Swarm Console logs, etc.

2.4. Fonctionnement général (Q5.4)

Notre solution contient plusieurs composants tels que mentionnés précédemment et l'intégration de celles-ci se révèle être un objectif de taille dans tout projet informatique. Considérant la nécessaire collaboration entre composantes physiques et matérielles, nous devions assurer une cohérence au niveau des versions de tout matériel tant informatique que logiciel tels que les différents systèmes d'opération, les versions de logiciels, etc. Heureusement, Docker est une solution qui nous assurent la portabilité de notre application et nous l'utilisons pour contenir l'ensemble de notre solution afin que tous utilisateurs puissent y accéder sans grande difficulté.

2.4.1. Docker-Compose

Docker a la capacité d'encapsuler tout requis nécessaire pour le fonctionnement d'une application et nous l'avons utilisé pour répondre à ce besoin. En effet, notre application peut se lancer en utilisant ce produit. Les étapes précises se trouvent dans le fichier «Read me» de notre répertoire de code. Bien que nous ayons aussi programmé l'application de sorte à pouvoir lancer le serveur et le client de manière distincte, l'utilisateur aura plus de facilité à procéder à travers Docker-compose considérant l'encapsulation de toutes les composantes de notre solution au sein d'une même entité. Dans un premier temps, Docker se chargera de «build» les images de notre projet. Cette étape sert à produire les images qui seront utilisées par les conteneurs. Un conteneur étant une instance d'une image pouvant être exécutée, cette étape permettra à l'utilisateur d'avoir le contexte nécessaire pour l'exécution de l'application. Ensuite, l'utilisateur pourra lancer la commande «sudo docker-compose-up» qui agrège la production de chaque conteneur. Dans notre cas, cela se matérialise par la création des environnements requis par nos différentes composantes.

2.4.2. Serveurs

Une fois les commandes de Docker exécutées, les serveurs de la simulation d'ARGoS et celui des drones seront prêts à interagir avec l'application web client. En effet, ces serveurs ne sont visibles que sur la console du code et localement à l'adresse https://localhost:5000 mais jouent un rôle central dans notre application. L'utilisateur pourra y faire appel afin d'avoir une idée sur les informations qui y transigent et pour «flasher» les drones.

2.4.3. Lancement interface web-client

L'interface web client se lance localement sur la machine à l'adresse https://localhost:3000 et l'utilisateur aura deux options : la simulation ARGoS ou les drones physiques avec la «Swarm App». Si vous optez pour la première option, l'application permettant de simuler ARGoS apparaîtra à l'écran (Fig 1.4.1). Celle-ci est configurée de sorte à fonctionner de pair avec l'application web.

Figure 7 Boutons de l'interface SwarmCommands

L'utilisateur pourra dans un premier temps faire appel aux boutons de commande pour contrôler le départ des drones (Fig 1.4.2) une fois avoir appuyer sur le bouton «Play» dans la simulation ARGoS. Le bouton «START MISSION» permet de commencer la mission sans autre intervention de la part de l'utilisateur. Une fois la mission démarrée, l'interface affichera les informations liées aux drones (Fig 1.4.3).

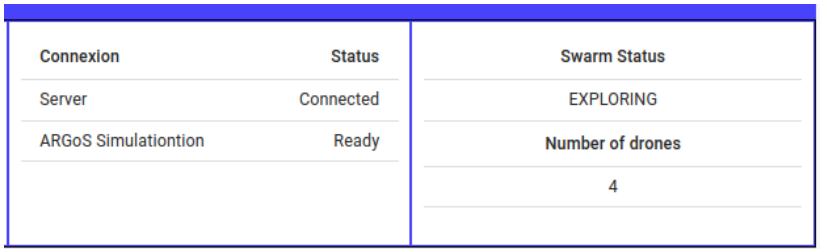

Figure 8 Vue de la composante Swarm Status

Cela permet de connaître en tout temps l'état des drones, le nombre de drones affectés à la mission ainsi que l'état de connexion au serveur et à la simulation ARGoS. Le bouton «TAKE-OFF» permet aux drones de quitter le sol sans pour autant commencer à explorer. Ils feront donc du sur-place en attendant d'autres commandes. Le bouton

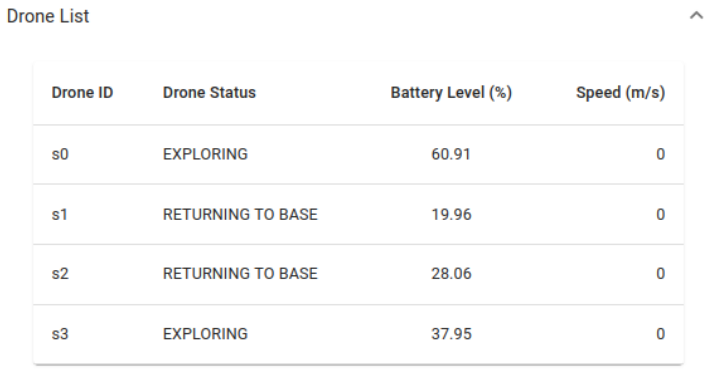

Figure 9 Apercu du dronelist component

«RETURN TO BASE» active le retour à la base des drones alors que le bouton «LAND» permet un atterrissage forcé des drones. Le bouton «SOFTWARE UPDATE» permet de lancer une mise à jour du logiciel des drones. Les boutons «RESET APP» et «START APP» permettent respectivement de réinitialiser l'application et de la lancer. À noter que la simulation ARGoS n'est pas lancée lorsque l'utilisateur choisit l'option des drones physiques. En plus de l'état des différents composants de l'application, l'interface affiche à l'utilisateur la liste des drones, leurs identifiants, leurs statuts, leur niveau de batteries ainsi que leur vitesse (Fig 1.4.4).

2.4.4. Carte

Tout au long d'une mission, les données de positionnements des drones sont récupérées par nos serveurs et sont acheminées au client de sorte à les afficher sur la carte que nous avons programmée. D'ailleurs, au bas de l'application web se trouve un élément nommé «Swarm Console Log» et lorsqu'ouvert celui-ci affiche les données qui proviennent du serveur avant qu'elle ne soient affichées sur la carte. Cette dernière permet donc à l'utilisateur de confirmer la position des murs sur ARGoS et affiche aussi la position des drones et des obstacles rencontrés lors du lancement d'une mission avec les drones physiques. Nous avons opter pour une carte de 10x10 considérant la condition de l'appel d'offre de pouvoir explorer une pièce de 100m2.

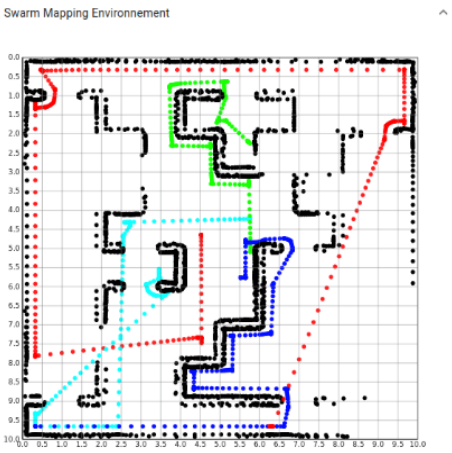

Figure 10 Exemple de cartographie dans l'interface

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

3.1. Requis fonctionnels

R.F.1: Une fois les drones connectés, ils vont s'attendre à recevoir un signal de départ "START MISSION". Dès qu'ils reçoivent la commande, ils se lancent dans une mission d'exploration de la pièce en suivant l'algorithme d'exploration SGBA. (le suiveur de murs permet de parcourir le périmètre de l'environnement)

R.F.2: Notre système fonctionne avec un nombre de drones entre 2 et 5. On peut évidemment avoir plus de 5 drones mais on s'était limités à 5 parce que c'est le nombre maximum 100 m².

R.F.3: Les drones physiques et virtuels évitent parfaitement bien les obstacles sur leur chemin à l'aide des différents capteurs.

R.F.4: Toutes les commandes sont implémentées. Une vidéo qui montre le fonctionnement du "Software Update" est sur le README de l'application.

R.F.4.1: L'option du "Software Update" n'est accessible que si tous les drones sont déconnectés. Ce requis est aussi respecté.

R.F.4.2: Les drones physiques et les drones de simulation sont capables de se rapprocher de la station au sol en se basant sur sa position. (avec une distance de moins de 1 m)

R.F.4.3: Les drones ne peuvent pas décoller avec un niveau de batterie inférieur à 30%. Le retour à la base est déclenché lorsque le niveau de batterie des drones est inférieur à 30%. L'atterrissage est déclenché lorsque le niveau de batterie des drones est inférieur à 10%.

R.F.5: L'interface reçoit une mise à jour de données des drones chaque seconde. (Pour ARGoS, on peut avoir une meilleure fréquence):

- Les différents états des drones sont: Ready, Not Ready, Disconnected, Exploring, Hovering, Taking Off, Landing et Returning To Base.
- La vitesse des drones est en m/s et le niveau de batterie est en pourcentage.
- La carte générée affiche les positions des drones en différentes couleurs et les obstacles détectés en noir.

R.F.6: On utilise pour l'exploration de l'environnement l'algorithme SGBA avec quelques ajustements. La carte générée par ARGoS ressemble parfaitement bien l'environnement exploré tandis que celle générée par les drones physiques contient le trajet exact de chaque drone avec un niveau d'exactitude correct.

R.F.7: Une fois les données récupérées par le serveur, ce dernier fait les traitements et calculs nécessaires pour envoyer les points à afficher sur la carte.

3.2. Requis matériels

R.M.1: Deux drones Bitcraze Crazyflie 2.1 ont été utilisés tout au long du projet.

R.M.2: Une seule Bitcraze Crazyradio PA a été utilisée pour l'implémentation de toutes les fonctionnalités du projet.

R.M.3: Toutes les fonctionnalités du projet ont été implémentées à l'aide d'un seul "ranging deck" et un seul "optical flow deck".

R.M.4: Un laptop avec une Crazyradio PA est utilisé pour la station au sol.

3.3. Requis de conception

R.C.1: Démonstration vidéo de l'exploration des drones sur la simulation ARGoS. La vidéo montre que l'algorithme d'exploration utilisé pour les drones est testé et vérifié à l'aide d'une simulation ARGoS.

R.C.2: Le client et le serveur possèdent des tests unitaires pour chaque composante.

R.C.3: Dans le dossier WebApp, on peut retrouver plusieurs tests pour chacune des composantes principales de l'application. Des tests unitaires ont été effectués sur le client, avec la framework Jest ainsi que de la librairie Enzyme. La combinaison de ces deux librairies react nous ont permis de tester les fonctionnalités principales de notre application. Donc, Jest fut utilisé comme exécuteur de tests et permettait de prendre en charge les bibliothèques d'assertions, puis on a utilisé Enzyme pour créer des tests pour l'interface utilisateur ainsi que la prise en charge des api. Aussi, nous avions utilisé flask unittest. Finalement, nous avions fait des tests de régression qui doivent être pris en charge que nous avions mis en mode «disabled» puisque le gitlab du cours n'avait plus de crédits.. Voir le fichier .gitlab-ci.yml.

R.C.4: Nous pouvons lancer tout le projet, avec une seule ligne de commande. Démonstration vidéo.

R.C.5: L'environnement de la simulation ARGoS est généré aléatoirement à l'aide d'un algorithme de génération de murs implémenté en C++.

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)

R.L.2: Nous avons implémentée le P2P Crazyflie API

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 toutefois nous ne l'utilisons pas pour retourner à la base, nous faisons uniquement un rebroussement de chemin.

R.L.4: L'interface est une application Web qui est compatible avec tous les navigateurs récents.

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 sur le serveur.

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

Des requis de qualité des composantes logiciels étaient nécessaires pour l'accomplissement de notre projet. Voici les requis demandés annoté de nos accomplissement de ces requis:

R.Q.1: Nous avions effectué une généralisation du format de codage et respecté les conventions de codage tel que demandé, incluant, l'indentation, style de commentaires, boucles, etc…

R.Q.2: Nous avions respecté les longueurs et les complexités des classes fonctions et des lignes de codes. En effet, suite au livrable UXT, «User eXperience Test», que nous nous sommes imposé, nous avions effectué un «refactor» de l'application web afin de s'assurer que les lignes de codes ne dépassent pas une taille de plus de 15 mots et que les classes n'ont pas plus de 100 lignes de code au maximum. Nous nous sommes efforcé de respecter cette convention, mais néanmoins, il arrive parfois qu'on ne l'a pas respecté dans une minorité de cas.

R.Q.3: Depuis le début du projet, nous avions mis en place des conventions pour le noms des classes, des variables, des branches, etc… Nous avions respecter les conventions de nommage standard.

R.Q.4: Lors du «refactor» , nous nous sommes assuré de respecter une hiérarchie des fichiers standard.

3.6. Requis bonus

- Guide d'utilisation avec des vidéos explicatifs
- Gestion des permissions pour l'envoie des commandes

4. Déroulement du projet (Q2.5)

Tout au long du projet, nous avons collaboré en équipe. Nous avions compris qu'une forte cohésion d'équipe était nécessaire afin de remettre un livrable complet pour l'Agence Spatiale.

Voici quelques points marquants de notre organisation du projet

- Utilisation de la méthodologie Agile.
- 10 Sprints.
- Utilisation de l'outil de gestion de versionnement GitLab.
- 4 rencontres hebdomadaires
- Plusieurs composantes de projet assignées à des responsable
- Utilisation de Discord.
- Communication accrue & Travail d'équipe.

Tout au long de ce projet nous avions travaillé de manière agile qui est une approche qui met de l'avant la collaboration entre des membres de l'équipe de maniérée autoorganisées et pluridisciplinaires. De manière pratique cette organisation agile est mise de l'avant grâce à l'outil de gestion de versionnement et de documentation des artéfacts logiciel gitlab que nous avions utilisé tout au long du projet. Aussi, nous avions 4 rencontres hebdomadaires depuis le début du semestre en équipe pour s'assurer une bonne cohésion et organisation d'équipe. Nous avions utilisé discord pour communiquer entre nous tout au long du projet.

De plus, nous avions répartis les tâches au sein de l'équipe de manière à ce que chaque composante principale logiciel aie deux responsables. Ainsi, nous pouvions nous entraider mutuellement en cas de problématiques techniques. Nous avions souvent travaillé de pair et effectué des remises sur un compte git. Pour nous, l'avancement du projet était une priorité. Pour chaque tâche hebdomadaire des composantes logicielles nous avions fait des «issues» dans lesquelles nous laissons des notes sur notre travail. Chaque issue avait une branche respective. Puis, une fois un «issue» complété et/ou résolu, nous pouvions faire une merge request sur la branche dev qui sera par la suite mise sur master lors de nos remises de livrables. Le tableau suivant présente les différentes composantes logicielles assignées à différents responsables, membres de l'équipe.

Composantes	Responsables:
Serveur Web	Yassir, Tarik
Interface Opérateur Client	Paul, Yassir
Pont Crazyflie/Web	Yassir, Nabil
Pont ARGoS/Web	Yassir, Nabil
Simulation ARGoS	Nabil, Paul
CrazyFlie Firmware	Yassir, Mazigh
Carte Crazyflie	Tarik, Paul
Navigation Autonome	Tarik, Paul, Nabil
Docker Compose	Yassir, Mazigh
Gestion du projet	Paul, Mazigh
Assurance Qualité	Mazigh, Tarik

Figure 11 Tableau répartition du travail

Somme toute, nous avions respecté notre objectif de fournir une solution au budget de départ. Lorsqu'on a entamé la partie du finale du projet, la Révision de l'état de

préparation, «Readiness Review», nous avions déjà utilisé plus 275 heures allouées à la Révision de la conception du produit, «Product Design Review» et Révision de la conception du code, «Code Design Review». Donc, nous aivons planifié un budget de 330 heures pour finaliser le projet. Nous avions pleinement utilisé ces heures afin de finaliser notre projet tout en respectant les requis demandés. De plus, nous avions ajouté une remise supplémentaire entre le CDR ainsi que le RR tel que présenté lors du rapport du CDR. L'objectif de la UXT, test d'expérience utilisateur «User eXperience Test», fut de nous permettre d'avoir une rétroaction supplémentaire de la part de l'agence spatiale. Celle-ci avait pour but de valider notre travail sur l'application web et de revoir tout autre possible problème en lien avec le projet. Nous avions eu des commentaires valables pour l'aspect assurance qualité du projet que nous avions corrigé.

4.1. Réussites du déroulement du projet

Quoiqu'on n'a pas réussi à perfectionner le projet, nous étions capables de finaliser la plupart des requis dans un temps assez court. Le projet demandait de la recherche, de l'efficacité et de la créativité, ce qui nous a permis de développer nos compétences techniques et habiletés personnelles. Nous avons réussi à développer une application web qui interagit avec différentes composantes d'une façon assez souple et efficace. L'architecture de notre serveur principal est le point fort de notre application. En effet, on a pris le temps au début du projet pour concevoir une excellente architecture qui permet aux développeurs d'avancer sur les différentes fonctionnalités sans devoir comprendre en profondeur la logique derrière. Nous avons aussi réussi à bien comprendre le fonctionnement des différentes technologies ainsi qu'à faire des recherches et d'intégrations du code open source.

4.2. Échec du déroulement du projet

Malgré que nous avions pu remettre tous nos livrables à temps, nous avions sous-estimé la difficulté en lien avec certaines tâches et le temps nécessaire à l'accomplissement de celles-ci. En effet, plusieurs technologies utilisées dans le cadre du cours étaient nouvelles pour la plupart d'entre nous. Il nous aurait fallu davantage de soutien de la part des encadrants du projet pour entamer un projet d'une telle envergure.

5. Travaux futurs et recommandations (Q3.5)

Malgré la difficulté du projet, nous avons énormément appris sur plusieurs technologies. Nous avons décidé de continuer à travailler durant l'été sur le projet et, aussi, nous avions décidé de se procurer des drones Crazyflie afin de continuer à apprendre davantage sur la programmation des drones.

Parmi les fonctionnalités qu'on aimerait développer et améliorer dans le futur:

- Trouver une solution pour l'affichage des obstacles détectés sur la carte
- Amélioration de l'algorithme de communication entre les drones sur ARGoS
- Répertorier les données de la console dans un tableau et ajouter une option de filtrage
- Ajouter un éditeur de code pour modifier le code de l'embarqué
- Faire une traduction de code ARGoS en code embarqué

Nous avons trouvé le projet assez intéressant et ça nous a fait plaisir de travailler dessus, mais nous pensons que le manque de documentation et d'orientation était le plus grand défi qu'on devait surmonter.

6. Apprentissage continu (Q12)

6.1. Mazigh Ouanes

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

Les lacunes que j'ai identifiées cette session ont été surtout reliées à la programmation des drones physiques, à l'embarqué ainsi qu'au pipeline d'intégration continue. En ce qui concerne les drones physiques, la programmation impliquant BitCraze était nouvelle pour moi et représentait un défi de taille. De plus, l'intégration d'autant de composantes au sein d'un même projet était une première pour moi et cela m'a demandé un certain temps d'adaptation.

Méthodes prises pour y remédier.

Pour toutes ses lacunes, j'ai dû prendre du temps pour rechercher de la documentation en ligne. Toutefois, cela n'était pas toujours suffisant et je n'hésitais donc pas à solliciter les chargés de laboratoires afin d'exploiter de nouvelles pistes de solutions aux problèmes que je rencontrais. De plus, j'ai collaboré avec mes coéquipiers de projet afin de m'assurer que mes tâches soient faites en pair.

Identifier comment cet aspect aurait pu être amélioré.

Je pense que la formule actuelle de l'enseignement ne favorise pas le partage de connaissances. Je proposerais donc d'ajouter des tutoriels vidéos pour palier au manque de ressources imposé par l'enseignement à distance. Aussi, je pense que l'on pourrait augmenter le nombre de séances inter-équipes afin de pouvoir échanger sur les problématiques du projet.

6.2. Tarik Agday

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

Ce projet a représenté pour ma part, un défi de taille tout d'abord en ce qui concerne sa gestion considérant la situation actuelle. En effet, la nature de ce projet ainsi que ses contraintes, tels que le nombre de drones restreints ainsi que l'impossibilité de se rencontrer en équipe en présentiel ont mis en lumière certaines lacunes. En effet, j'aurais voulu améliorer mes compétences en matière de systèmes embarquées considérant que j'ai suivi le cours INF3610 aussi à distance et que cela à jouer un rôle dans mon apprentissage. Cependant, la situation ne s'étant pas améliorée , il était plus difficile qu'à la normale pour apprendre avec mes coéquipiers. D'autres parts, l'utilisation de technologies telles que Docker et ARGoS étaient des nouveautés pour moi.

Méthodes prises pour y remédier.

En plus de poser des questions sur ces sujets à mes coéquipiers qui s'occupait de ces parties du projet, j'ai aussi sollicité des élèves d'autres équipes pour partager sur leur système embarqué. J'ai pu m'y adapter notamment avec l'aide du cours d'infonuagique qui fait grandement référence à Docker. En ce qui concerne ARGoS, bien qu'un membre de notre équipe ait travaillé un peu plus que les autres dessus, nous avons pu échanger

sur cette matière et j'ai de mon côté pu en apprendre énormément en consultant la documentation en ligne.

Identifier comment cet aspect aurait pu être amélioré.

Avec le recul, je crois que l'équipe entière aurait dû bénéficier de l'apprentissage de ces nouvelles technologies et nous aurions pu remédier à cela en définissant un moment hebdomadaire dédié uniquement à cet aspect du projet. Je crois que nous aurions pu améliorer notre apprentissage sur ce sujet en sollicitant Mr Beltrame qui est un spécialiste dans le domaine des drones. Mon avis personnel à ce sujet tend à promouvoir les séances théoriques durant les heures de classes à ce sujet si le projet se poursuit pour les semestres à venir.

6.3. Nabil Dabouz

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

Le plus grand défi qu'on a dû surmonter en travaillant sur ce projet est le fait qu'on a utilisé plusieurs technologies et trois projets open source (ARGoS, Bitcraze et SocketIO). Étant donné que j'étais la personne responsable de toute la simulation ARGoS, j'ai trouvé qu'il y avait peu de documentation sur l'utilisation des différentes composantes. De plus, j'ai eu quelques difficultés avec l'intégration de la simulation ARGoS dans le reste du projet à cause des problèmes du docker-compose et la gestion des processus au niveau du serveur ARGoS.

Méthodes prises pour y remédier.

La clé pour ce projet de manière générale fut la recherche de documentation sur les différentes technologies que nous avons utilisées. Pour ce qui est des technologies Open-Source, j'ai souvent considérer d'autres projets similaires trouvés en ligne afin d'approfondir mon apprentissage. D'autant plus, le partage de connaissances avec mes collègues de classe impliqués dans d'autres équipes m'a aussi aidé à comprendre ces technologies.

Identifier comment cet aspect aurait pu être amélioré.

La nature du projet se voulait un peu contraignante et je pense que le cadre enseignant voulait nous pousser à chercher des solutions par nous même. Ceci étant dit, un encadrement plus strict au niveau de la documentation fourni et de l'intégration des différentes technologies aurait certes permis de mieux nous guider dans nos réflexions. En revanche, je pense que le manque de documentation nous a permis de mieux apprendre à cibler, lire, comprendre et utiliser les parties qui nous intéressent d'un code open source.

6.4. Mohamed Yassir El Aoufir

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

Premièrement, l'implémentation du serveur principal a constitué un véritable défi pour moi. Mon objectif était de concevoir le serveur le plus simple d'utilisation possible dans le but de simplifier au maximum le travail de mes collègues et moi-même durant la session. Deuxièmement, je me suis rapidement rendu compte que les estimations de temps que

je faisais pour les différentes tâches étaient erronées, je sous-estimais souvent le temps nécessaire à la réalisation de certaines tâches.

Méthodes prises pour y remédier.

De ce fait, je me suis retrouvé à refactoriser le serveur à mainte reprises pendant la session. Je me suis ainsi rendu compte que je ne passais pas suffisamment de temps sur la conception et que je me précipitais trop rapidement sur l'implémentation (il faut cependant avouer qu'il est complexe d'élaborer une conception «parfaite» lorsqu'on a peu de visibilité sur l'avancement futur du projet). Pour remédier au souci de l'estimation de temps, j'ai pris le temps de consulter mes camarades pour tenir compte de leur avis par rapport à l'estimation des durée des tâches qui m'ont été attribuées. En ce qui concerne l'implémentation du serveur, j'ai décidé que pour mes tâches futures, je prendrai toujours le temps nécessaire pour la réalisation d'une conception efficace avant de commencer l'implémentation. Cette approche a été suivie pour l'implémentation du code embarqué qui a été un succès.

Identifier comment cet aspect aurait pu être amélioré.

Ces aspects auraient pu être améliorés en équipe, en prenant le temps de mieux comprendre les tâches et leur enjeux avant de les attribuer aux différents membres, afin que les estimations puissent être plus précises.

6.5. Paul Clas

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 incomprise au début du projet.

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.

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.

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 réalisions 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

What is Docker? | Opensource.com. (n.d.).(22 janvier 2021), https://opensource.com/resources/what docker

DA14, Top 10 advantages of using React.js (1 février 2021), https://da-14.com/blog/its-high-time-reactjs ten-reasons-give-it-try

Sokolsky, J. J., & Jockel, J. T. (2019). Canada and the Future of Strategic Defense. Perspectives on Strategic Defense, 185-197. doi:10.4324/9780429301506-17

K. McGuire, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown envi ronment" (DataverseNL, 2019); https://hdl.handle.net/10411/YVP873. 43. Bitcraze AB, Crazyflie 2.0 (2019); www.bitcraze.io/crazyflie-2/.

McGuire, K. (2019). Indoor swarm exploration with Pocket Drones.

https://doi.org/10.4233/uuid:48ed7edc934e-4dfc-b35c-fe04d55caee1

Bitcraze AB, Multi-ranger deck (2019); www.bitcraze.io/multi-ranger-deck/.

Bitcraze AB, Flow deck v2 (2019); www.bitcraze.io/flow-deck-v2/.

Bitcraze AB, Crazyradio PA (2019); www.bitcraze.io/crazyradio-pa/.

Noronha, Justin, "Development of a swarm control platform for educational and research applications" (2016). Graduate Theses and Dissertations. 15783. https://lib.dr.iastate.edu/etd/15783

Maulloo, H. (2020, July 06). The FReMP Stack -Building a full stack web application. Retrieved from https://medium.com/@akhilmaulloo/the-fremp-stack-building-a-full-stack-web-application 91308e505250

Crazyflie 2.1, Bitcraze. Available at:https://store.bitcraze.io/products/crazyflie-2-1

Bitcraze webpage, Bitcraze. Available at:https://www.bitcraze.io/

The bitcraze virtual machine, Github. Available at: https://github.com/bitcraze/bitcraze-vm/releases/

AutonomousSequence.py example script, github. Available at: https://github.com/bitcraze/crazyflie-lib python/blob/master/examples/ autonomousSequence.py

Motion commander class, github. https://github.com/bitcraze/crazyflie-lib-python/blob/master/cflib/po sitioning/ motion commander.py

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Caro, G. D., Ducatelle, F., Stirling, T., Gutiérrez, A., Gambardella, L. M., & Dorigo, M. (2011). ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. In Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS 2011) (pp. 5027–5034). Los Alamitos: IEEE Comput. Soc.

Pinciroli, C. (n.d.). ARGoS website. Retrieved from https://www.argos-sim.info/dev_manual.php

IEEE Guide--Adoption of ISO/IEC 90003:2004 Software Engineering--Guidelines for the Application of ISO 9001:2000 to Computer Software. (n.d.). doi:10.1109/ieeestd.2008.4690902

Ahmad, E., Raza, B., Feldt, R., & Nordebäck, T. (2010). ECSS standard compliant agile software develop ment. Proceedings of the 2010 National Software Engineering Conference on - NSEC 10. doi:10.1145/1890810.1890816

Flask Documentation Release 1.1x. (n.d.). Retrieved from https://flask.palletsprojects.com/en/1.1.x/