POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

Documentation du projet répondant à l'appel d'offres no. A2022-INF3995 du département GIGL.

Conception d'un système aérien d'exploration

Équipe No ***105***

Nicolas Charron, Thierry Spooner, Julien Bourque, Audrey Leblanc, Gabriel Bruno

7 décembre 2022

Table des matières

1. Vue d'ensemble

1.1. But du projet, portée et objectifs (Q4.1)

Tout d'abord, il est primordial de remettre en contexte l'ensemble du projet. Il s'agit ici de l'élaboration d'une preuve de concept demandée par l'Agence Spatiale Canadienne, qui aimerait ensuite appliquer ce concept à des robots contrôlés à distance qui exploreraient des objets célestes. Cette preuve de concept est nécessaire pour pallier les défauts des robots d'exploration spatiale actuels. Ceux-ci sont souvent surdimensionnés, ce qui ralentit leur vitesse d'exploration, et par extension leur capacité à explorer les grandes superficies des surfaces d'objets célestes. C'est pourquoi l'Agence Spatiale Canadienne porte un intérêt particulier pour la conception, le développement et la réalisation de robots nouveau genre. Ces robots, plus petits et plus rapides, pourraient paralléliser la tâche d'exploration et ainsi couvrir plus de terrain.

Étant donné que de tels robots n'existent pas pour l'instant, notre tâche est de fournir une preuve de concept constituée d'un "prototype fonctionnel de NMS 4" [1], c'est-à-dire un prototype "reposant sur l'intégration d'applications et de concepts en vue de démontrer la viabilité." [1] Cette viabilité repose principalement sur la capacité de mener à bien le projet d'exploration à distance.

La portée du projet consiste donc en un prototype qui devra pourvoir explorer une salle à l'aide d'un essaim de deux drones et une station au sol, et ce, de manière autonome. Au total, le prototype comportera trois livrables, soit la station au sol, la partie embarquée et la partie simulée. Ces trois parties, une fois complétées formeront la preuve de concept demandée par l'Agence Spatiale Canadienne.

La station au sol se divise en trois parties, soit sa base de données, son serveur et son interface graphique, selon l'architecture classique en trois couches.

La base de données servira à enregistrer toutes les informations captées par les drones en mission d'exploration. Elle sera aussi en communication constante avec le serveur afin de lui fournir les données précédemment enregistrées au besoin, par exemple pour reconstituer une carte complète de la pièce explorée.

Le serveur est le lien entre les composantes du prototype, puisqu'il doit permettre à toutes les parties de communiquer entre elles. Le serveur transmettra les commandes fournies par l'entremise de l'interface utilisateur aux drones afin de les contrôler à distance.

L'interface graphique permet au pilote du système de transmettre ses instructions aux drones explorateurs. Les cartes fournies par la base de données et le serveur ainsi que les informations sur l'état des drones pourront aussi être affichées sur cette interface. En cas de problème, les logs pourront aussi être accessibles au pilote afin de lui permettre de régler tout problème survenant en cours de mission.

La partie embarquée correspond aux drones en tant que tels et à leurs microcontrôleurs. Elle communique entre les drones de l'essaim et la station au sol avec un protocole propriétaire semblable au Bluetooth. Cette communication sert à cartographier l'environnement et à mouvoir les drones dans cet environnement afin d'accomplir leur mission.

Le logiciel de simulation est composé du logiciel Argos [6]. Le but du simulateur est de tester virtuellement les drones afin d'aider les pilotes et les développeurs à éviter des erreurs de manipulation ou de conception qui pourraient endommager les drones. Ce simulateur doit donc reproduire l'environnement dans lequel évolueront les drones. Ainsi, le code devra d'abord être fonctionnel sur ce simulateur.

Le but du projet dans le cadre du cours est donc de fournir un prototype un projet où deux drones contrôlés à partir d'une station au sol ayant une interface graphique, un serveur et une base de données remplissent tous les requis décrits dans le document disponible sur Moodle.

1.2. Hypothèses et contraintes (Q3.1)

La preuve de concept repose sur plusieurs hypothèses, étant donné le niveau de maturité attendu. En effet, faire cartographier une pièce par un essaim de deux drones ne demande pas le même niveau d'effort que de cartographier la surface d'objets célestes. De plus, le fait de ne pas réaliser une solution avec un niveau de maturité complet nous force quelques contraintes.

Tout d'abord, il est demandé que le prototype fonctionne aussi bien dans le simulateur Argos que dans un environnement physique réel. Nous assumons donc que le simulateur représente fidèlement le comportement du drone lors de son utilisation. De plus, nous assumons que le simulateur en tant que tel nous est fourni sans failles de fonctionnement.

Ensuite, il a été expliqué que le drone pouvait avoir des inexactitudes dans l'estimation de son déplacement. Nous posons donc l'hypothèse que ces inexactitudes sont beaucoup plus petites que le déplacement total des drones, ce qui permet de les négliger.

Dans un même ordre d'idées, nous posons l'hypothèse que le mouvement du drone implémenté par le manufacturier prend en compte la dynamique fluide de l'air et les turbulences créées par les drones lors de leurs déplacements, ou bien que les erreurs créées par ces deux derniers éléments sont négligeables par rapport au poids et taille des drones explorateurs.

On suppose aussi que le temps de communication entre la station au sol et les drones est négligeable dans le contexte de la preuve de concept. La communication se fait à l'aide d'un protocole propriétaire similaire au protocole Bluetooth et les drones physiques

voleront dans une pièce dédiée à Polytechnique, ce qui réduira à un point négligeable le délai de transmission et les possibles interférences.

En ce qui a trait aux contraintes, nous avons tout d'abord le temps de développement. Nous devons en effet concevoir la preuve de concept sur une session universitaire, ce qui correspond à environ 12 semaines entre la réception des drones physiques et la remise du rapport final (*readiness review).*

Une autre contrainte est la batterie. En effet, l'exploration faite par les drones doit être assez efficace pour ne pas avoir à recharger les robots trop souvent, ce qui rendrait le processus beaucoup plus coûteux en temps.

Finalement, il faut garder en tête qu'il s'agit ici d'une preuve de concept qui pourrait être adaptée au contexte de l'exploration spatiale. Ainsi, il y a certaines contraintes qui ne s'appliquent dans le contexte du projet réalisé pour l'exploration d'une pièce à Polytechnique, mais qui devront être prises en compte lors de la conception de solutions ayant un niveau de maturité plus élevé, comme les conditions environnementales dans l'espace.

1.3. Biens livrables du projet (Q4.1)

Tout d'abord, le projet se sépare en trois grandes sections qui correspondent à chacune des remises, soit le PDR, le CDR et le RR. Toutefois, la remise d'un rapport d'avancement est demandée en complément aux trois remises mentionnées plus haut à chaque semaine. Ce rapport contient les faits saillants de la semaine présente, l'avancement des tâches non complétées, les tâches finies au courant de la semaine, ainsi que les tâches planifiées pour la semaine à venir.

En ce qui concerne le PDR, celui-ci est attendu le 30 septembre. Il constitue en quelque sorte la réponse de notre équipe à l'appel d'offres lancé par l'Agence Spatiale Canadienne. Le présent document fait partie des biens livrables attendus pour le PDR. Une vidéo montrant que les drones peuvent décoller, atterrir et être identifiés ainsi qu'une autre montrant que la simulation fonctionne aussi sont attendues en complément.

Quant au CDR, il est dû le 4 novembre. Celui-ci comprend la documentation du concept du projet. Il est attendu que les changements faits en cours de route soient énoncés et décrits dans le document remis. De plus, les commentaires faits lors de la précédente remise doivent être appliqués. Une présentation technique de format libre doit expliciter les avancements depuis le PDR et démontrer les progrès réalisés sur la conception du prototype. De plus, de nombreux requis techniques sont demandés à cette étape-ci. Parmi ceux-ci, on retrouve l'affichage de l'état des drones, la capacité d'explorer de façon autonome tout en évitant les obstacles et la disponibilité de logs, le tout disponible dans une interface utilisateur accessible sur plusieurs plateformes et par plusieurs appareils

simultanément. Une fonctionnalité additionnelle dont le choix est laissé à l'équipe est aussi attendue.

Finalement, la date de remise du RR est le 7 décembre. Celui-ci constitue la dernière remise, et devra donc contenir à la fois un prototype fonctionnel et la documentation complète. Une présentation de 45 minutes destinée à un public n'ayant pas le même niveau de connaissances techniques. Des démonstrations vidéo des capacités techniques du prototypes sont attendues en supplément à ladite présentation, tout comme le code nécessaire au fonctionnement du prototype. Un document expliquant la batterie de tests à effectuer pour s'assurer du bon fonctionnement du projet, doivent être remis conjointement avec le code. Finalement, toutes autres instructions nécessaires au bon fonctionnement du projet dans son ensemble devront aussi être remises en format Markdown.

2. Organisation du projet

2.1. Structure d'organisation (Q6.1)

L'équipe qui sera en charge de mener à bien le projet sera formée de 5 membres. Puisqu'il s'agit en réalité d'un projet réalisé dans un environnement d'apprentissage, il est important que chacun puisse développer ses compétences dans tous les rôles reliés à la gestion du projet. Évidemment, au cours de la session, certains se démarqueront dans un certain rôle et pourront approfondir leur implication et leurs habilités dans ce rôle.

Ces rôles sont assez diversifiés. Comme dans toute équipe désirant mener un projet à terme, il est nécessaire d'avoir une personne plutôt en charge de l'aspect organisationnel. Cette personne serait en charge d'organiser les échéanciers et planifier les tâches à réaliser pour chacune des remises. Un autre rôle serait celui de leader technique. C'est celui qui serait en charge de prendre les décisions par rapport au choix de technologies à implémenter et aurait le dernier mot sur l'architecture choisie pour la réalisation du projet. De plus, puisque l'on doit à la fois rendre une interface web et un serveur, il pourrait y avoir deux membres dédiés respectivement au front-end et au back-end du projet. Ceux-ci seraient plus concentrés sur l'aspect technique et aurait beaucoup plus de proximité avec le code.

Finalement, puisque nous sommes une équipe de 5, il resterait un dernier rôle. Plusieurs options s'offrent à nous pour ce dernier rôle. Un rôle crucial pour le dernier membre de l'équipe pourrait être celui de l'assurance-qualité. Notre division des tâches pourrait inclure quelqu'un qui se concentre moins sur l'implémentation des demandes techniques du projet, mais qui s'efforce plutôt d'essayer de briser le projet et de trouver des bogues et autres failles de fonctionnement afin d'ensuite réparer les erreurs ou les comportements fautifs de notre code. Cela pourrait passer par l'écriture de tests et la

conception de pipelines de développement et d'intégration continue sur notre répertoire GitLab.

Évidemment, il existe plusieurs autres rôles assez communs en industrie, et nous devrons assurément cumuler plusieurs rôles. Un exemple de ce genre de rôle serait d'avoir quelqu'un en charge de présider les réunions et les scrums. C'est un rôle présent partout en entreprise et qui est fort utile pour l'organisation du quotidien et qui permet de mettre tout le monde à jour avec une vue d'ensemble du projet. Cela pourrait être une personne désignée en début de projet, ou bien on pourrait alterner parmi les membres de l'équipe. Toutefois, les cinq rôles mentionnés plus haut donnent un bon aperçu d'une possible division et assignation des rôles dans notre équipe de projet.

En plus de ces rôles, notre équipe pourra aussi être divisée en sous-groupes remaniables au fur et à mesure que le projet avance afin de se concentrer sur certains aspects du projet. Par exemple, un sous-groupe pourrait être formé pendant une semaine pour se concentrer sur une fonctionnalité demandée ou bien pour régler un bogue particulièrement persistant. Rendre ces groupes malléables permet aussi d'améliorer la cohésion entre tous les membres de l'équipe et augmente notre flexibilité pour résoudre des problèmes.

2.2. Entente contractuelle (Q11.1)

Le contrat établi pour ce projet est de type *livraison clé en main*, c'est-à-dire à prix ferme. Nous nous sommes arrêtés sur ce type de contrat en raison de la nature du projet qui requiert un rendu final, contrairement aux projets souvent mis en place dans le milieu des TI basé sur des ententes à *temps plus frais*. Il est ainsi possible d'estimer à l'avance les coûts d'implémentation, incluant la main-d'œuvre et le matériel du projet. Cela permet de le borner en imposant des limites aux coûts et en imposant des requis dès le début du projet. Également, nous disposons très exactement d'une session universitaire afin de produire un rendu final. Il est très important que la durée du contrat soit prévue.

Finalement, un contrat de type *partage des économies* est à exclure, car l'objectif du projet est d'établir un prototype d'exploration pour les futures missions spatiales de l'Agence spatiale canadienne. Le but de ces missions est de nature scientifique et non commerciale, elles n'ont pas pour but de générer des profits qui pourraient être repartagés.

3. Description de la solution

3.1. Architecture logicielle générale (Q4.5)

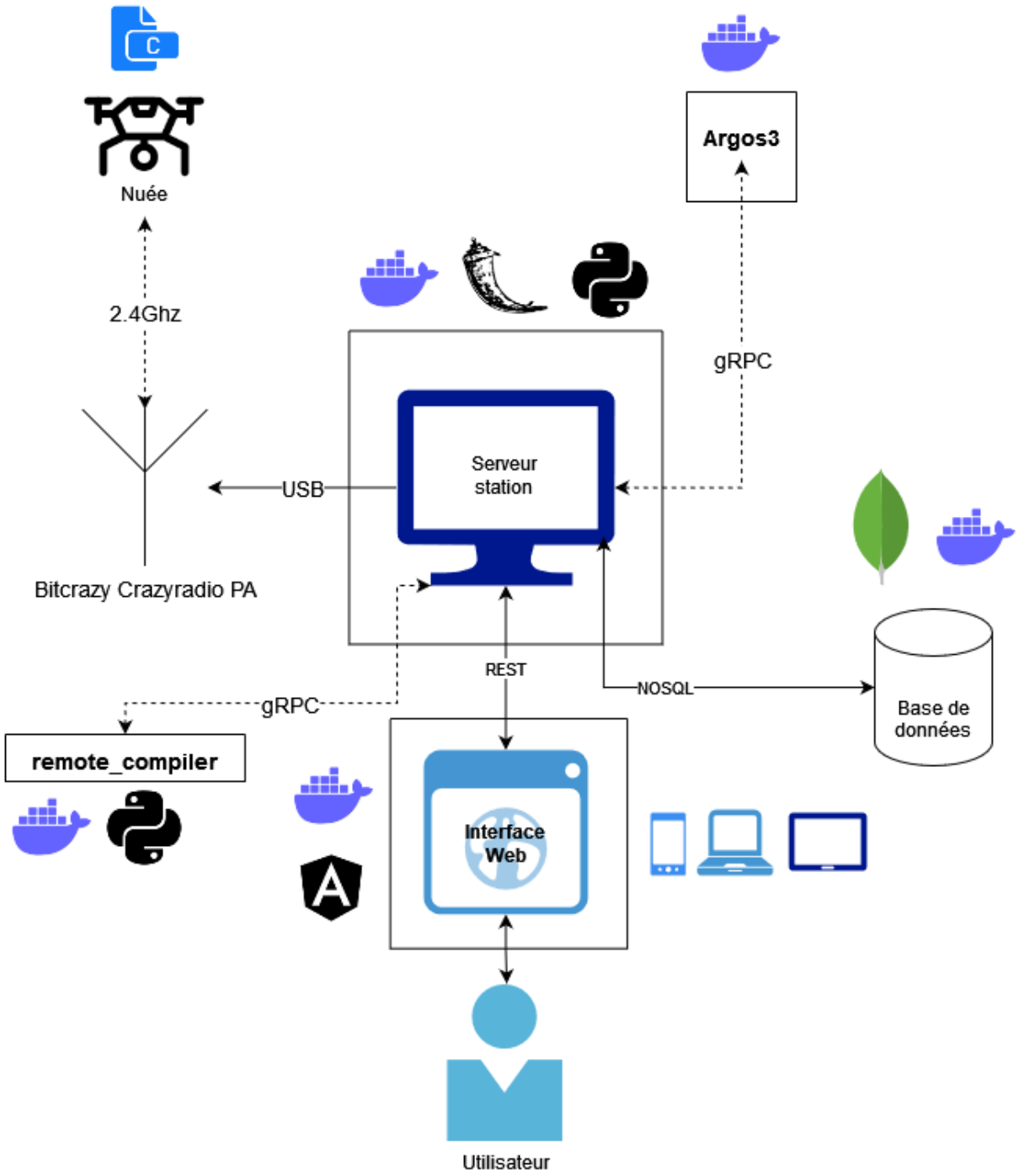


Fig. 1: Architecture générale du projet

La figure 1 ci-haut décrit l'architecture générale du projet. La nuée de drones, initialement limitée à deux Bitcraze Crazyflie 2.1 (R.M.1), communique avec le serveur de la station au sol à travers une Bitcrazy Crazyradio PA connectée par port USB (R.M.2, R.M.4). Pour ce qui en est du langage utilisé pour coder le système embarqué, nous avons choisi C en raison de sa simplicité et afin de rester cohérent avec le code du *firmware*. Les informations partagées par les drones vers le serveur de la station au sol seraient leur état courant de vol, leur position, leur niveau de batterie et les informations qu'ils récoltent sur les objets qui les entourent.

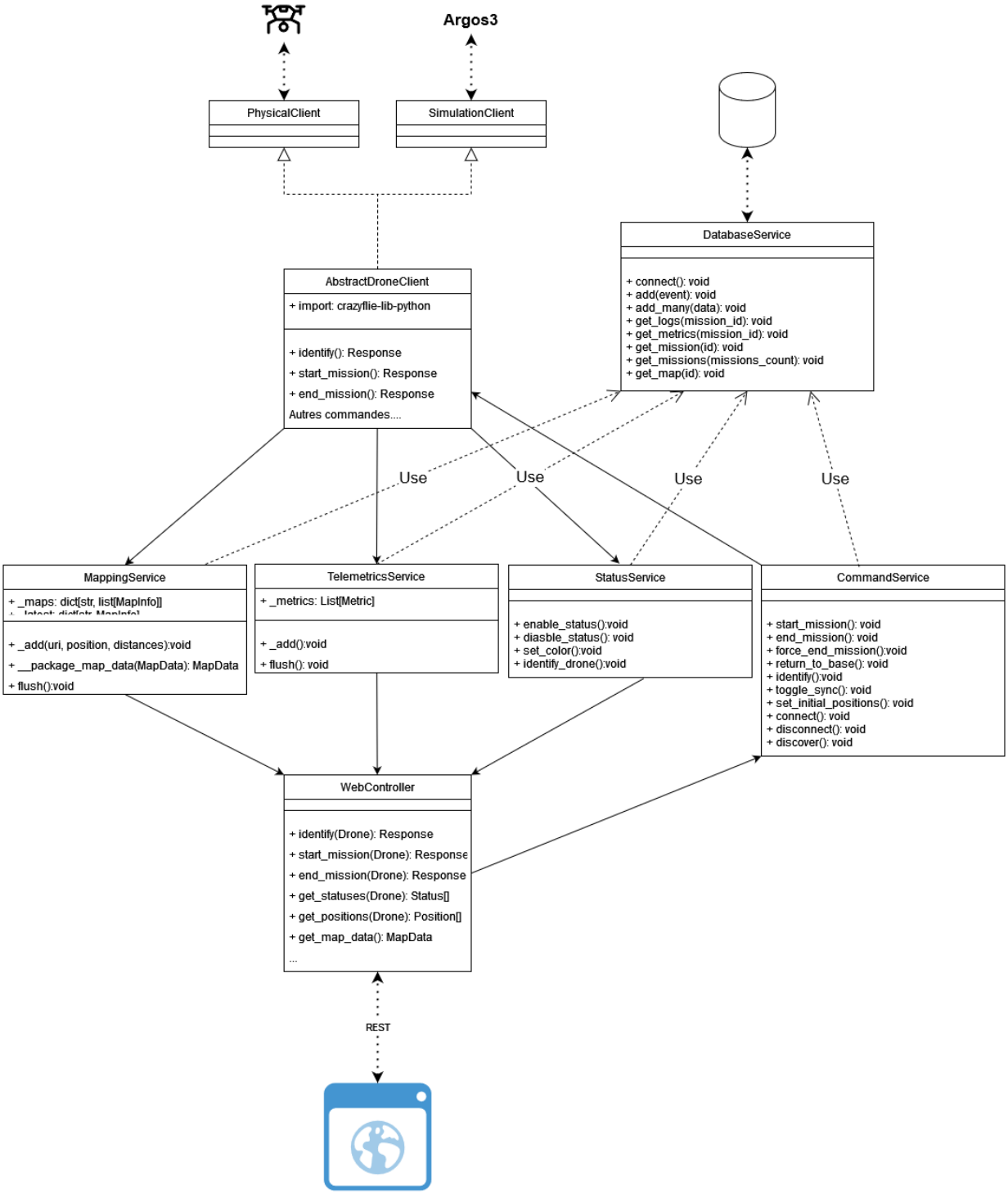
Le serveur peut envoyer des commandes simples à la nuée (R.F.1, R.F.2). Alternativement, il peut se connecter à une simulation Argos3 à l'aide du cadriciel gRPC. Ce dernier permet de simplifier la sérialisation des informations envoyées à la simulation. Il est ainsi possible de communiquer avec les drones simulés d'une manière similaire à celle des drones physiques. Ce même serveur communique également avec une base de données MongoDB [4] pour y enregistrer les données sur la nuée, les données de la carte générée et un historique des commandes envoyées à la nuée. Nous avons choisi MongoDB [4] pour sa simplicité et parce que plusieurs équipiers l'ont déjà utilisé auparavant. De plus nous avons opté pour un serveur Flask écrit en Python, car la haute performance du serveur n'est pas une priorité et certains membres de l'équipe sont déjà familiers avec Flask.

L'utilisateur utilisera une interface web pour lui permettre de visualiser toutes les informations pertinentes à la mission et envoyer des commandes à la nuée. Cette interface sera également disponible pour plusieurs types d'appareils (R.F.10). Nous avons décidé d'utiliser les cadriciels Angular et Bootstrap pour le développement de cette interface, car tous les membres de l'équipe ont déjà de l'expérience avec Angular, notamment dans le cadre du cours LOG2990. Bootstrap nous permettra d'avoir des interfaces plus flexibles, ce qui est très pratique pour concevoir des interfaces utilisateurs devant s'adapter à plusieurs dimensions et types d'écrans.

Le compilateur à distance est conteneurisé à l'aide de Docker et est codé en Python. Il communique avec la station au sol par l'entremise du cadriciel gRPC, comme le fait la station au sol avec la simulation.

Finalement, la simulation Argos3, le serveur de la station au sol et l'interface web seront tous conteneurisés à l'aide de Docker [5] (R.L.4) pour faciliter le partage du code et simplifier le développement sur plusieurs postes.

3.2. Station au sol (Q4.5)



Fig, 2: Architecture du serveur de la base au sol

Comme on peut le voir dans la figure 2, l'architecture du serveur de la base au sol peut être séparé en trois couches principales, soit la couche contrôleur, la couche service et finalement, la couche client.

La couche contrôleur implémente une interface qui permet de communiquer avec le client web Angular. Les commandes reçues par cette interface sont ensuite passées à la couche service à travers le CommandService afin d'être traitées, enregistrées et envoyées à la nuée de drones ou la simulation. Quelques exemples des commandes implémentées par cette interface sont identify (R.F.1), start_mission, end_mission (R.F.2) et get_status (R.F.3).

La couche service constitue la logique principale du traitement des données. Les quelques services présentés dans le diagramme peuvent être séparés en deux catégories. Premièrement, nous avons le service de commande, et deuxièmement, les services de traitement de données.

Le service de commande est le seul service appelé par le WebController. Lorsque celui-ci reçoit des requêtes de l'interface web, il transmet ces requêtes au CommandService pour qu'elles soient traitées. Ce traitement consiste à mettre en place l'enregistrement des logs reliés à cette commande. En d'autres mots, il permet de lier les logs renvoyés par la nuée au ID de la commande envoyée par l'interface.

Pour les services de traitement des données, soit MappingService, PositionService et StatusService, on traite les données renvoyées par la nuée et on les enregistre dans la base de données à l'aide du DatabaseClient. Les données traitées sont ensuite utilisées par le WebController pour les renvoyer à l'interface web.

La couche client comporte 4 classes. Premièrement, AbstractClient, SwarmClient et SimulationClient qui communiquent avec la nuée physique ou simulée et deuxièmement, DatabaseClient qui communique avec la base de données.

L'AbstractClient permet d'abstraire les différences entre la nuée physique et la simulation. Toute l'implémentation du serveur sous cette abstraction est identique pour les deux modes de fonctionnement. Le SwarmClient et le SimulationClient implémenteront les méthodes abstraites du AbstractClient selon leurs besoins particuliers.

Le DatabaseClient est utilisé par le serveur afin de communiquer avec la base de données MongoDB [4], enregistrer les logs, les commandes et toute autre information pertinente.

Le flot de commande peut donc être décrit en quelques étapes:

Tout d'abord, l'utilisateur choisit une commande à partir de l'interface utilisateur. Cette commande est reçue par le WebController, puis est traitée par l'intermédiaire du CommandService. Elle est ensuite envoyée par l'AbstractClient, qui reçoit plus tard les résultats de l'interaction des drones ou de la simulation suite à la commande. Les résultats sont traités par les multiples services de traitement de données. Ces données

traitées sont ensuite reçues par le WebController, puis renvoyées à l'interface utilisateur par ce même contrôleur.

3.3. Logiciel embarqué (Q4.5)

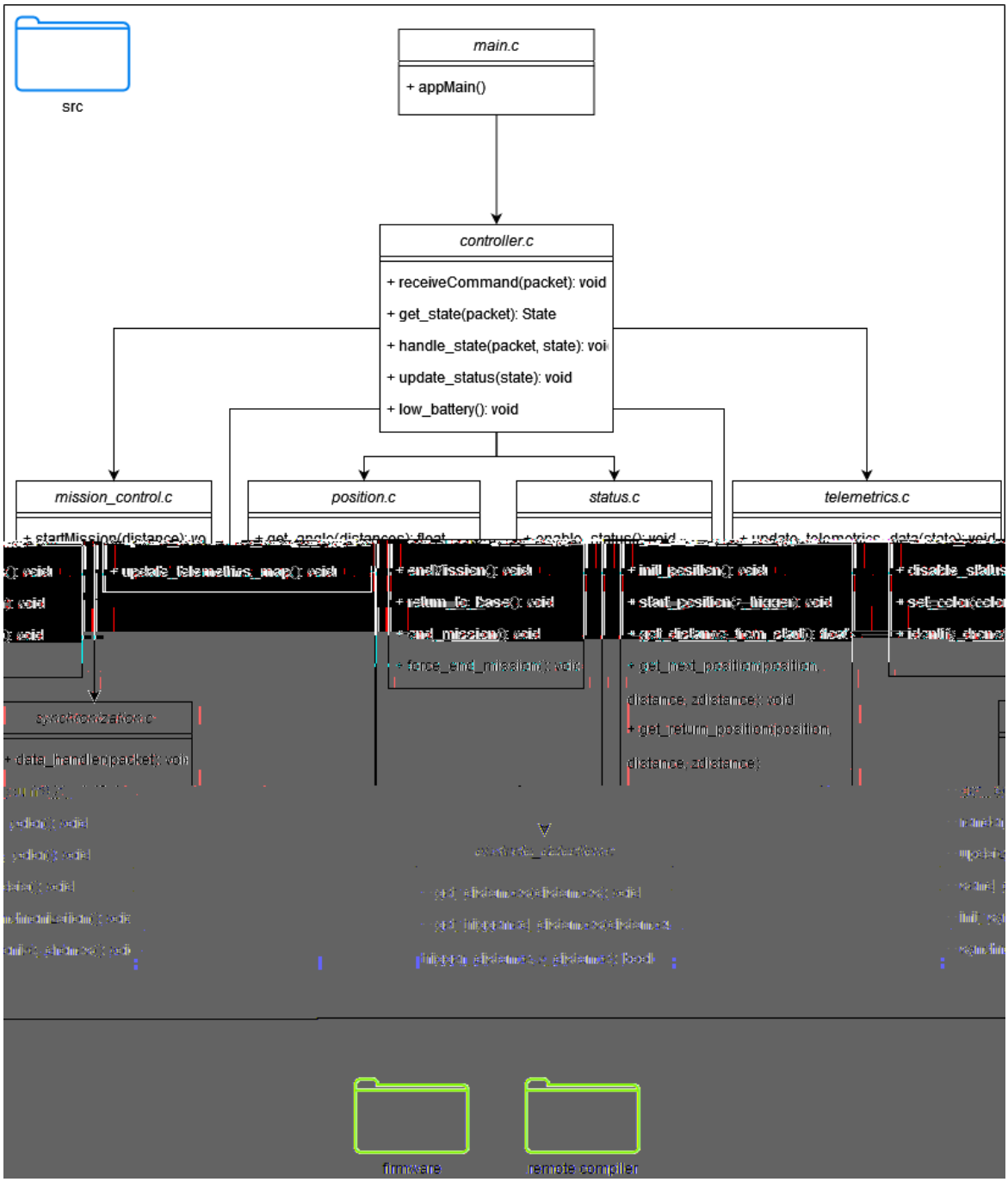


Fig. 3: Architecture du logiciel embarqué

La figure ci-dessus décrit l'architecture proposée de notre logiciel embarqué sur les drones Crazyflie. Nous allons développer nos fonctionnalités à l'aide du crazyflie-firmware afin d'interfacer avec les drones.

Comme premier point, nous avons décidé de séparer le code source développé dans le cadre de ce projet du code firmware crazyflie-firmware. Selon la documentation [2], il est possible de retoucher directement le code du *firmware* afin d'implémenter de nouvelles fonctionnalités. Toutefois, nous avons opté pour la séparation complète de notre code de celui du *firmware* en les séparant dans deux dossiers: *src* pour notre code et *firmware* pour celui du crazyflie-firmware. Ceci s'appelle le *out-of-tree development*, cette approche facilite notre visualisation de l'historique des changements dans notre projet. En effet, en changeant directement le code du *firmware*, il devient difficile de retrouver les changements qui ont été faits et cela pourrait rendre le débogage ardu.

Pour ce qui en est du code C, nous avons décidé de séparer les différentes catégories de fonctionnalités en plusieurs fichiers. Ces fichiers sont semblables à la séparation utilisée dans un langage orienté objet. Le fichier main comporte la boucle de base du fonctionnement du drone et le maximum de fonctionnalités est implémenté dans un fichier catégorisé pour éviter une fonction main monolithique. Un fichier important pour faciliter cette séparation est controller.c, qui implémente la réception de commandes et qui envoie ensuite les requêtes aux fichiers spécialisés dans le traitement de ces commandes.

De plus, le *firmware* est utilisé à son plein potentiel dans le développement de nos routines. C'est-à-dire, nous allons fortement nous appuyer sur son API afin de simplifier notre code *out-of-tree* et pour éviter la duplication de code.

3.4. Simulation (Q4.5)

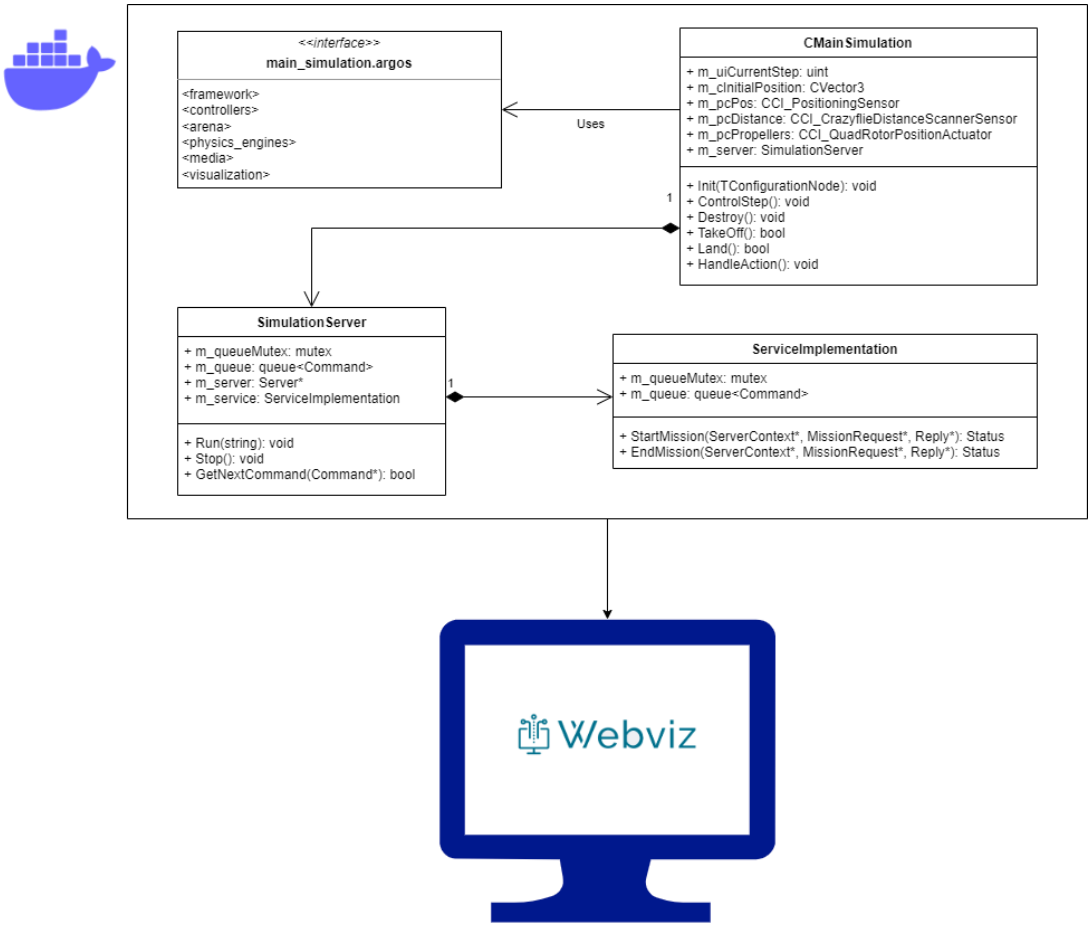


Fig. 4 : Architecture de la simulation

La simulation se fait sous Argos3, et utilise l'interface de visualisation Webviz. La base du répertoire contient un Dockerfile qui automatise le déploiement et permet de la lancer rapidement dans le navigateur. La simulation elle-même est divisée en deux modules principaux: les paramètres de simulation et les contrôleurs.

Les paramètres de simulation sont contenus dans des fichiers au format XML et d'extension *argos*, présents dans le répertoire *experiments*. Chaque simulation est définie dans un fichier de configuration permettant de définir les différentes composantes à simuler, tel que les dimensions de l'arène physique, le nombre de drones, et les paramètres initiaux des drones. Il est également possible de définir des paramètres d'affichage, tels que la fréquence d'actualisation et le type de rendu graphique (OpenGL, Webviz, etc.). Enfin, le fichier de configuration doit spécifier le contrôleur à utiliser.

Le contrôleur (répertoire *controllers)* contient la logique principale du drone simulé, et définit comment il se comporte à chaque instant. Les fichiers CMakeList.txt qui déclarent les librairies Argos à utiliser permettent la compilation du contrôleur sous la forme d'une librairie qui pourra par la suite être chargée par Argos3. La classe CMainSimulation contient les 2 principales fonctions permettant d'exécuter la simulation :

- Init() : Permet d'initialiser les variables de simulation ainsi que la connexion à distance. Cette méthode est appelée au lancement de la simulation. Nous y initialisons aussi les autres classes permettant, entre autres, les interactions avec la station au sol, la simulation de la cartographie et la simulation des données télémétriques.
- ControlStep() : Permet de faire évoluer la simulation. Cette méthode est appelée par Argos3 à un intervalle régulier défini dans les configurations de la simulation (*experiments*). Lors de chaque appel, la position des objets simulés est actualisée, et les commandes reçues de la station au sol sont traitées.

Comme plusieurs drones formeront la nuée, plusieurs instances des contrôleurs seront créées (une par drone). Ceux-ci devront également communiquer directement avec la station au sol. Pour ce faire, nous utilisons le cadriciel gRPC afin de normaliser nos appels et sérialiser les données. gRPC vient avec un compilateur (protoc) permettant de générer du code propre à un langage à partir de la définition d'un protocole de communication (.proto). Les appels sont donc les mêmes sur la station au sol et dans la simulation. À chaque drone simulé est attaché un serveur gRPC auquel la station au sol (ici cliente) doit se connecter. Le port sur lequel le serveur écoute est déterminé à partir de l'identifiant de chaque drone (un nombre : 0 + k pour le premier drone, n + k pour le nième où k est un port de base).

Le serveur gRPC est encapsulé dans une classe SimulationServer (répertoire de communication), c'est cette classe qui est utilisée dans le contrôleur. Il est possible de tirer à partir de celle-ci les actions à exécuter, ainsi que d'y pousser les logs et informations pertinentes à retourner à la station au sol.

Finalement, afin de modéliser au mieux le fonctionnement du drone, le reste de l'architecture du contrôleur tente de reprendre l'architecture du *firmware* embarquée de la figure 3. La différence la plus flagrante est que l'appel à toutes les fonctionnalités est réalisé à partir de la classe MainSimulation et synchronisé à partir de la méthode ControlStep().

3.5. Interface utilisateur (Q4.6)

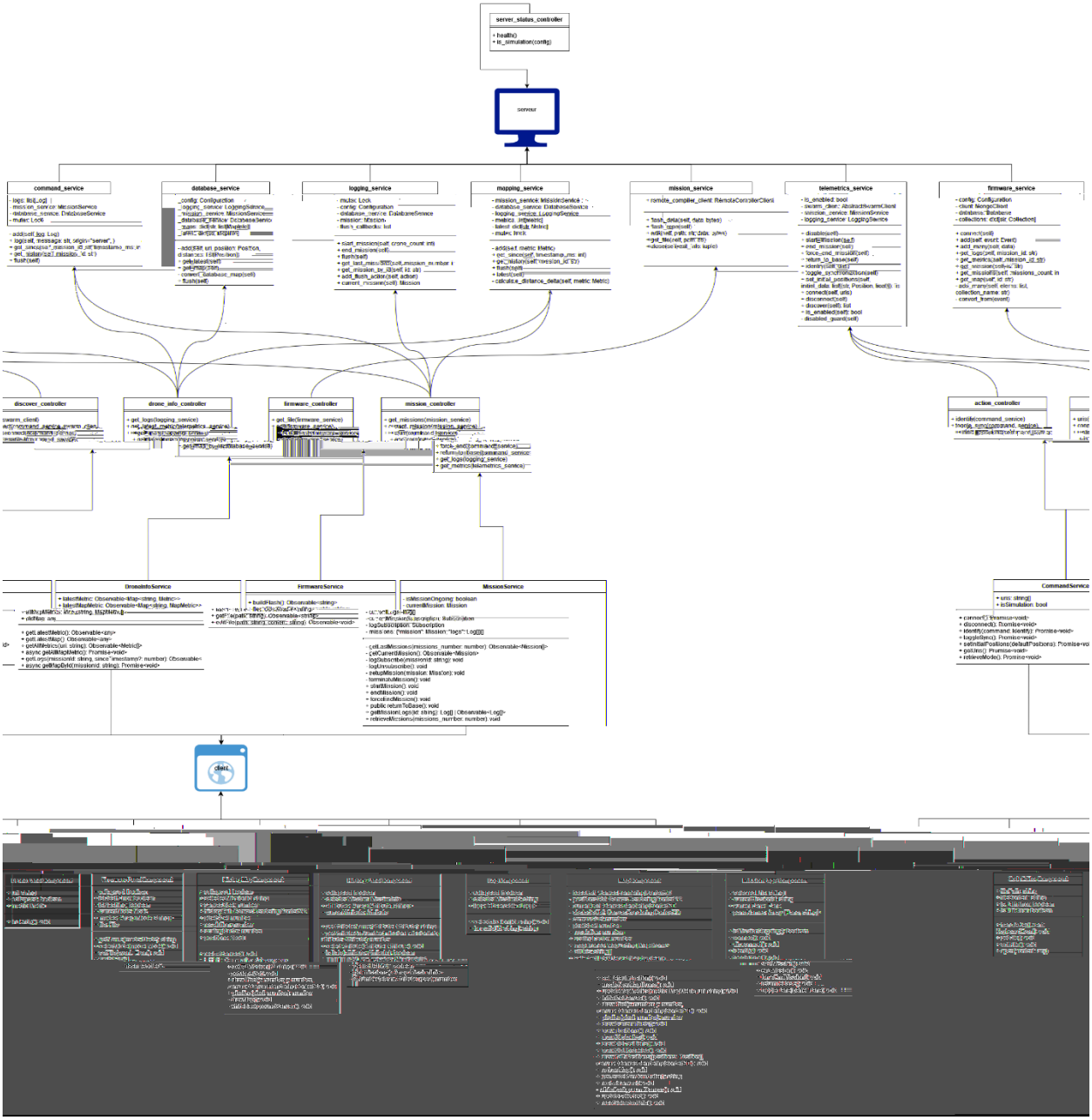


Fig. 5 : Architecture de l'interface utilisateur

La figure ci-haut présente l'architecture et le fonctionnement de l'interface utilisateur. On y aperçoit une architecture assez classique découlant de l'utilisation du cadriciel Angular. Le tout est principalement divisé en services et composantes. Ce cadriciel a été choisi en raison de la familiarité de l'équipe suite au cours LOG2990. La même structure utilisant TypeScript, HTML et CSS a été conservée pour cette même raison. De plus, ce cadriciel s'exporte bien sur différents types d'appareils, comme l'exige le requis R.F.10. Aussi, l'utilisation d'un cadriciel facilitera la conception d'une interface facile d'utilisation et lisible, tel que précisé par le requis R.C.4. Cette interface restera aussi la même, peu importe si

l'utilisateur est connecté à la simulation ou aux drones physiques, comme le spécifie le requis R.L.2.

En ce qui concerne l'expérience utilisateur, celle-ci sera marquée par l'utilisation d'une interface simple comportant de gros boutons et champs d'écriture, afin de simuler le contexte d'un utilisateur devant pouvoir réagir rapidement avec précision dans le cadre constamment en évolution d'une mission d'exploration spatiale. Cela s'appliquera aussi bien sur la version Web que sur les versions mobiles, peu importe la taille de l'écran, puisque l'enjeu est aussi présent lors de manipulations avec la souris que lorsque l'utilisateur essaye d'appuyer sur un bouton spécifique sur un petit écran. Aussi, la communication entre le serveur et l'interface utilisateur se fait à l'aide de Websocket [9] dans le but d'assurer la communication la plus rapide et ininterrompue possible.

Les différentes composantes au bas du diagramme représentent tous les blocs constituant la base de l'interface utilisateur. Par exemple, le MapComponent gère l'affichage sur la carte des données captées par les drones, comme la position des murs ou leur propre position.

Sur le côté, HistoryPanelComponent et HistoryMapComponent permettent d'afficher respectivement les missions passées et les cartes qui leur sont associées, tandis que LogComponent affiche les logs permettant de déboguer en cours de mission. Ce déboguage pourra ensuite être accompli à l'aide du CodeEditorComponent.

La logique derrière les opérations réalisées à partir de l'interface utilisateur est majoritairement implémentée par les différents services. Par exemple, le CommandService permet de transmettre les commandes choisies à travers l'interface utilisateur aux drones afin que ceux-ci réalisent les instructions demandées par le pilote du système. Le MissionService s'occupe de la logique derrière les missions, de leur début jusqu'à l'atterrissage. Il s'occupe aussi de gérer les logs et les anciennes missions.

Les différents contrôleurs font un lien entre le client et le serveur à l'aide d'appels REST. Ils envoient donc les commandes faites à partir de l'interface utilisateur et gérées par le service du client aux services du serveur. Par exemple, le drone_info_controller transmet les mesures prises par les capteurs des drones à la base de données, qui peut ensuite être affichée dans l'interface.

Finalement, les différents services du serveur s'occupent de la logique qui y est adjacente. Par exemple, database_service s'occupe d'ajouter les informations glanées par les drones à la base de données. La plupart des autres services ont des méthodes pour ajouter leurs informations à la base de données.

Le firmware_service et le command_service sont les deux seuls services qui n'utilisent pas la base de données. Le premier sert à modifier le code du *firmware* à partir de ce qui été envoyée à partir de l'interface utilisateur, qu'il s'agisse de lignes de code ou de fichiers. Le deuxième agit directement sur les drones pour leur envoyer les commandes

de haut niveau, comme se connecter à la station au sol, commencer la mission, retourner à la base ou partir la synchronisation entre pairs.

3.6. Fonctionnement général (Q5.4)

Le système s'articule en plusieurs modules. Chaque module contient un *Dockerfile* qui permet de produire un environnement reproductible indépendant de la machine où il est exécuté, ce qui facilite grandement le déploiement

Détaillons maintenant le fonctionnement de chaque module, en commençant par le module Client. Tel que mentionné précédemment, il s'agit d'un « frontend » pour la station au sol utilisant le *framework* Angular. Il permet d'interagir avec les drones ou la simulation par l'entremise du serveur de la station au sol avec lequel il communique par l'entremise d'appel REST et de Websockets [9]. La communication entre le client et le serveur s'effectue par IP, il n'est donc pas nécessaire que le client et le serveur s'exécutent sur la même machine, ni même sur le même LAN.

Le conteneur de ce module peut être lancé individuellement à condition, bien entendu, que le serveur soit préalablement déployé. Le port 5001 doit être exposé à la machine hôte.

Le module serveur est le serveur de la station au sol. Son implémentation est réalisée en Python et utilise les librairies Flask (pour le serveur web) et cflib (pour interagir avec les drones). Tout comme pour le client, il est possible de lancer individuellement le conteneur de ce module. Il est nécessaire d'exposer le port 5000 à la machine hôte afin de permettre au client se s'y connecter. Afin de contrôler les drones, le dispositif de communication « Crazyradio PA » doit être connecté à la machine hôte. Le conteneur doit être exécuté en mode privilégié (*privileged*), le périphérique doit également être monté avec le fichier représentant le « Crazyradio PA ».

Le module Simulation permet quant à lui de simuler et visualiser le vol des drones. Il est commandé à partir de la station au sol. Comme pour les autres modules, il peut être compilé et exécuté directement à l'aide de Docker [5]. Lors de la compilation, de nombreuses dépendances telles qu'Argos3 et Webviz (pour la visualisation web) doivent être compilées, ce qui affecte le temps de compilation de ce conteneur. Plusieurs ports du conteneur doivent être rendus accessibles à la machine hôte : le port 8000 donne accès à l'interface de simulation, le port 3000 est utilisé par l'interface de simulation afin de communiquer (par websocket) au simulateur Argos3, enfin les ports entre 3850 et 3860 permettent de communiquer avec les drones simulés. À noter que les ports des drones n'ont pas à être accessibles à la machine hôte si le conteneur du serveur de la station au sol et le conteneur du simulateur sont sous le même réseau virtuel.

Lors du lancement de l'interface de simulation sur le port 8000, il est possible de lancer la simulation à l'aide du bouton « Jouer ». Cela aura pour effet d'activer les animations et

la communication avec la station au sol. Les drones simulés peuvent alors être contrôlés par l'entremise de l'interface web de la station au sol.

Ensuite, le *firmware* est embarqué sur les drones, il ne s'agit donc pas d'un service comme les autres modules. Le code est plutôt compilé sur une machine de développement et ensuite porté sur le drone à l'aide de la commande « make cload ». Afin de procéder à la programmation d'un nouveau drone, ce dernier doit être placé en mode de démarrage en appuyant sur le bouton marche/arrêt jusqu'à ce qu'un clignotement bleu soit observé. La commande « make cload » peut alors être utilisée. Il faut cependant noter que cette dernière commande ne tient pas compte de l'adresse du drone à programmer. Il est donc impératif de placer les drones un à la fois en mode de démarrage. Une fois la programmation, effectuée, le drone reste programmé et démarre automatiquement.

Il est important de remarquer que ce module est implémenté selon une approche out-of-tree, il faut donc cloner les sous modules git afin que le code puisse être compilé correctement.

Puis, l'interface utilisateur de la station au sol permet de contrôler soit les drones physiques, soit la simulation. L'interface utilisateur s'adapte automatiquement selon le mode choisi et donne accès à la cartographie en temps réel tout comme aux logs permettant le débogage.

Une fois l'initialisation terminée, il est possible de démarrer le mode exploration par le biais duquel la pièce où se déroule l'expérience est cartographiée. Finalement, il est possible de faire atterrir les drones à leur emplacement initial en terminant la mission.

Finalement, afin de lancer le projet en un tout, il convient de cloner de façon récursive le répertoire principal. Une fois les fichiers clonés, il faut compiler et lancer l'ensemble des conteneurs *Docker* à l'aide de la commande « docker compose up ». L'utilisation de « docker compose » permet de définir au préalable les règles, telles que les ports à ouvrir et le réseau virtuel permettant aux conteneurs de communiquer entre eux. Il aurait été possible de placer uniquement les fichiers de configuration dans le dépôt principal sans avoir recours aux sous-modules. Cependant, cela nous aurait forcés à pousser les images de chaque module dans un dépôt (*registry*). L'approche hybride que nous avons adoptée permet de compiler l'ensemble du projet à l'aide d'une seule commande.

En bref, Docker [5] ainsi que les commandes docker-compose et make sont nécessaires pour démarrer le projet. Une fois ces éléments prêts, il faut cloner les *submodules* avec *git submodule init* suivi de *git submodule update*. On répète ces deux dernières étapes dans le répertoire cloné à partir du *firmware*. À partir de ce répertoire, on exécute *docker compose build*, puis on programme le drone en *bootloader* en gardant le bouton de mise en marche appuyé pendant trois secondes. De retour dans le répertoire initial, on exécute *docker compose up*, puis on peut ensuite se connecter à localhost:5000 sur un fureteur Web afin d'avoir accès à l'interface graphique contrôlant le drone.

4. Processus de gestion

4.1. Estimation des coûts du projet (Q11.1)

Le contrat choisi est de type *clé en main*, ce qui veut dire que le prix final du projet est ferme et bien défini. On assume que le salaire pour un développeur analyste est de 130$ par heure, et de 145$ par heure pour un coordonnateur de projet. On estime aussi que le projet demande au total 630 heures-personnes de travail. Avec une équipe de 5, et en assumant une charge de travail égale pour chaque membre, on arrive à 126 heures de travail pour chaque membre du groupe.

On estime qu'un cinquième du temps sera dédié à la coordination et gestion du projet. Le salaire à payer sera donc 145*126 + 130*4*126 = $81900. A ce prix, on rajoute $500 pour couvrir les assurances, et pour financer des activités de cohésion au but de promouvoir la cohésion de l'équipe, pour un total de $82400.

4.2. Planification des tâches (Q11.2)

Tableau 1 : Planification des tâches

Étape	Responsable	Description	Date de remise
1. Preuve de concept: Vendredi, 30 septembre 2022			
1.1 Bouton interface	Nicolas	Envoyer les commandes "Voler" et "identifier" au serveur à partir d'une interface Angular	30/09/2022
1.2 Traitement à la station au sol	Audrey	Création de l'interface REST, du service CommandService et du SwarmClient et SimulationClient	30/09/2022
1.3 Code embarqué	Thierry / Julien	Organisation du projet en C, interface avec le *firmware* et réception des commandes	30/09/2022
1.4 Simulation Argos	Gabriel	Ajouter le CrazyFlie à la simulation	30/09/2022
1.5 Pipeline CI	Julien	Mise en place du pipeline dans les divers projets gitlab	30/09/2022
1.6 Écriture du PDR	Nicolas	Écriture du PDR	30/09/2022
2. Routines: Vendredi, 14 octobre 2022			
2.1.1 Séparation swarm / simulation dans l'interface	Thierry	Fonctionnailté qui permet de toggle entre swarm et simulation. Rendre responsive (R.F.10)	7/10/2022
2.1.2 Boutons lancer et terminer mission interface	Audrey	Implémentation des boutons lancer et terminer la mission dans l'interface et communication avec le serveur (R.F.2), Rendre responsive (R.F.10)	7/10/2022
2.1.3 Lancer / terminer mission dans serveur	Thierry	Interface rest, services et client vers swarm et argos pour 2.1.2 (R.F.2)	7/10/2022

2.1.4 *Firmware* décollage et atterissage	Julien	Implémentation du *firmware* pour décoller et atterir sur commande (R.F.2). Logging (R.C.1)	7/10/2022
2.1.5 *Firmware* algo d'exploration	Nicolas / Gabriel	Implémentation de l'algorithme d'exploration dans le *firmware* (R.F.4). Logging (R.C.1)	7/10/2022
2.1.6 Documentation	Gabriel	Documentation des systèmes	7/10/2022
2.2.1 VIsualisation de logs dans l'interface	Nicolas	Fonctionnalité pour montrer les logs des drones dans l'interface (R.C.1). Rendre responsive (R.F.10)	14/10/2022
2.2.2 Queries getStatus et getPos Interface	Gabriel	Queries 1hz ou plus de l'interface pour getStatus et getPos des drones (R.F.3). Rendre responsive (R.F.10)	14/10/2022
2.2.3 getStatus et getPos dans serveur	Thierry	Implémentation REST, service, client pour getStatus et getPos (R.F.3)	14/10/2022
2.2.4 getStatus et getPos dans *firmware*	Audrey	Implémentation du getStatus et getPos dans le *firmware* (R.F.3)	14/10/2022
2.2.5 Détection d'obstacles dans le *firmware*	Thierry / Julien	Détection et évitement des obstacles à l'aide des capteurs (R.F.5). Logging (R.C.1)	14/10/2022
2.2.6 Mise à jour de la simulation	Julien	Implémenter les comportements jusqu'à maintenant dans la simulation	14/10/2022
2.3.1 Tests	Audrey	Période d'écriture des tests d'intégration du projet	21/10/2022
2.3.2 Écriture du CDR	Nicolas	Commencer l'écriture du CDR	21/10/2022
2.3.3 QA	Thierry	Quality assurance des fonctionnalités	21/10/2022
2.3.4 Vérification du formatage du code	Julien	Vérification du format du code (R.Q.1)	21/10/2022
3. Mapping: 2 décembre 2022			
3.1.1 *Firmware* collection de données mapping	Thierry / Gabriel	Commencer la collection de données par les drones à l'aide des capteurs (R.F.8). Logging (R.C.1)	28/10/2022
3.1.2 Création de la base de données	Julien / Nicolas	Créer la base de données mongo (R.F.17)	28/10/2022
3.1.3 Création du service de base de données	Julien / Nicolas	Communiquer avec la base de données à partir du serveur (R.F.17)	28/10/2022
3.1.4 Élaboration des structures de données de mapping	Audrey / Thierry	Création des structures de données pour la communication des données de mapping entre les applications (R.F.8)	28/10/2022
3.1.5 Visualisation données de vol interface	Audrey	Créer les fondations pour afficher la carte dans l'interface (R.F.18)	28/10/2022
3.1.6 Écriture du CDR	Nicolas	Compléter l'écriture du CDR	28/10/2022
3.2.1 *Firmware* retour à la base	Julien	Faire le *firmware* pour retourner à la base (R.F.6)	4/11/2022
3.2.2 Afficher base de données interface	Thierry	Afficher le contenu de la base de données dans l'interface (R.F.17)	4/11/2022

3.2.3 Afficher la position interface	Audrey	Afficher la position des drones dans la carte sur l'interface (R.F.9)	4/11/2022
3.2.4 Implémenter mise à jour	Nicolas	Implémenter la fonctionnalité de mise à jour à distance. (R.F.14)	4/11/2022
3.2.5 Fonctionnalité mapping	Gabriel / Audrey	Terminer la fonctionnalité mapping (R.F.18)	4/11/2022
3.3.1 État crashed	Julien	Implémenter la fonctionnalité qui montre l'état crashed quand le drone est à l'envers (R.F.13)	11/11/2022
3.3.2 Position initiale *firmware*	Nicolas	Implémenter les commandes de position initiale dans le *firmware* (R.F.12)	11/11/2022
3.3.3 Position initiale interface	Thierry	Implémenter le système de gestion des positions initiales dans l'interface (R.F.12)	11/11/2022
3.3.4 Implémenter l'éditeur de code dans l'interface	Julien / Gabriel	Implémenter l'édition et l'envoi des modifications du code dans l'interface (R.F.16)	11/11/2022
3.4.1 QA	Thierry / Nicolas	Période de quality assurance des fonctionnalités	18/11/2022
3.4.2 Formatage	Julien	Vérification du formatage du code	18/11/2022
3.4.3 Test d'intégration	Audrey / Gabriel	Ajout de tests d'intégration	18/11/2022
3.5.1 Préparation de présentation orale final	Nicolas	Préparation de la présentation finale	25/11/2022
3.5.2 Rédaction du RR	Nicolas	Écriture du RR	25/11/2022
3.6.1 Vérification finale et mises à niveau	Thierry	Préparation à la remise finale	02/12/2022

4.3. Calendrier de projet (Q11.2)

Plusieurs biens livrables seront à remettre tout au long de ce projet. Le calendrier présenté dans le tableau suivant montre les biens livrables attendus à chaque phase du projet.

Tableau 2: Calendrier de projet

Calendrier de projet		
Livrable	Date	Livrables
Preliminary Design Review (PDR)	Vendredi, 30 septembre 2022	- Documentation - Démos vidéo
Critical Design Review (CDR)	Vendredi, 4 novembre 2022	- Documentation - Présentation technique - Démos vidéo
Readiness Review (RR)	Vendredi, 7 décembre 2022	- Documentation - Présentation technique - Démos vidéo

		- Code - Tests automatisés - Instructions de compilation

Le premier livrable, la réponse à l'appel d'offre, est dû pour le 30 septembre. Ce premier rapport montre la documentation initiale du projet. Deux démonstrations vidéo sont aussi nécessaires. La première montre l'interaction entre la station au sol et le drone physique. La deuxième montre l'interaction entre la station au sol et la simulation.

Le deuxième livrable, le livrable intermédiaire, doit être remis pour le 4 novembre. Un deuxième rapport doit montrer les améliorations et les changements effectués depuis la réponse à l'appel d'offre. Une présentation technique détaillant les changements effectués depuis le PDR et le concept final du projet seront aussi à remettre. Sept démonstrations vidéo sont aussi nécessaires. Ces démonstrations doivent démontrer la bonne fonctionnalité des requis R.F.1, R.F.2, R.F.3, R.F.4, R.F.5, R.F.10 et R.C.1.

Le troisième livrable, le livrable final, doit être remis pour le 7 décembre. Le premier livrable est la documentation finale. Un autre livrable est une présentation finale qui détaille les éléments du projet. Des démonstrations pour tous les requis sont aussi nécessaires à remettre. Le code ayant servi à faire le projet doit être remis. De plus, les tests ainsi qu'un rapport de test doivent être fournis.

4.4. Ressources humaines du projet (Q11.2)

Les ressources humaines du projet sont principalement composées des cinq membres de l'équipe qui sont présentement étudiants du cours INF3995. Ceux-ci font tous partie des programmes en génie informatique à Polytechnique Montréal et ont eu des parcours légèrement différents, ce qui leur confère certaines habiletés spécifiques. Par exemple, Thierry a beaucoup travaillé sur le serveur lors du cours LOG2990 et a donc acquis une expertise de plus que les autres membres de l'équipe n'ont pas. Il a aussi un talent pour concevoir des interfaces utilisateurs. Gabriel est plus habile en résolution de problèmes mathématiques et en conception d'algorithmes, ce qui sera pratique pour l'exploration. Julien possède plus d'habiletés avec les services et les conteneurs Docker. Audrey, quant à elle, est organisée et plus que capable de faire le lien entre le *frontend* et le *backend*. Nicolas est plus habitué de faire du *backend* et de la gestion de projet, notamment la rédaction des documents.

Aussi, cette équipe de développeurs aux études est accompagnées de l'équipe académique du cours, soit Prof. Giovanni Beltrame, coordonnateur du cours, Ulrich Dah-

Achinanon, chargé de cours, Maude St-Cyr Bouchard, consultante HPR, ainsi que Sami Sadfa et Guillaume Ricard, chargés de laboratoire.

5. Suivi de projet et contrôle

5.1. Contrôle de la qualité (Q4)

Afin d'assurer le meilleur produit possible, et pour faciliter le développement en général, un processus de révision sera mis en place. L'équipe utilisera Gitlab comme repo, et utilisera ses fonctionnalités de travail collaboratif pleinement: avant de merger une nouvelle branche dans le *main*, il faudra lancer une *merge request* avec une courte mais informative description du but du nouveau code. Au moins un membre de l'équipe qui n'a pas travaillé sur cette branche devra passer le nouveau code en revue, pour le comprendre, demander des clarifications, suggérer des améliorations, et ultimement approuver ou désapprouver la requête.

Chaque membre est encouragé à faire des *commits* fréquents, des *merge requests* qui ne sont pas trop grosses et n'ajoutent pas trop de fonctionnalités à la fois, et de faire des *pulls* depuis le *main* autant que possible pour faciliter l'intégration du code et éviter les conflits. On s'attend aussi à ce que chacun écrive du code de qualité: il faut des commentaires (sans les surutiliser au détriment de code clair), des noms de variables et fonctions descriptifs, une hiérarchie de fichiers et classes bien organisée, et généralement suivre les bonnes pratiques de codages enseignées à Polytechnique Montréal.

Puisqu'une vérification manuelle par les pairs n'est pas exhaustive, des tests seront aussi écrits afin de vérifier automatiquement que les fonctions et le code en général fonctionne correctement. La première méthode sera les tests unitaires. Chaque développeur sera responsable d'écrire des tests qui testent le fonctionnement des méthodes qu'il écrit. Ainsi, tous les cas possibles seront testés et vérifiés. La deuxième méthode de test utilisée est celle des tests d'intégration. Les tests d'intégration permettront de vérifier la cohérence entre les différents modules. Un changement dans l'un des modules pourrait causer des erreurs dans les autres, alors les tests d'intégration pourront détecter ces erreurs. Chaque développeur est libre de rouler localement les tests unitaires et d'intégration. Sur le répertoire Gitlab, les tests sont exécutés automatiquement lors de chaque *merge request*. Le succès de l'ensemble des tests est une condition nécessaire afin de pouvoir compléter la demande.

Aussi, tous les répertoires de code suivent une convention selon le langage. Par exemple, le code Python suit la convention PEP8 [12]. Pour ne pas oublier de faire cette étape cruciale de vérification, les pipelines qui roulent à la suite des *merge requests* comportent une étape validant le formatage du code.

Finalement, il n'y a pas que le code qui est important dans un projet tel que celui-ci, les documents le sont tout autant. C'est pourquoi le contrôle de la qualité du projet implique aussi bien une revue de code qu'une revue de documents. Pour ce faire, chaque section est relue par un membre n'ayant pas contribué à son écriture. Cela est fait afin de s'assurer qu'il n'y ait plus de fautes de français et que le texte soit adapté à son public cible. Il s'agit du même principe de révision que lors des *merge request*.

5.2. Gestion de risque (Q11.3)

L'un des risques principaux du projet est la perte et/ou dégâts accidentels des équipement physiques. Les drones fournis par Polytechnique sont assez petits et délicats, et les chutes ou collisions avec les murs restent un danger omniprésent. Heureusement, endommager le drone n'est pas la fin du monde, car Polytechnique nous fournira un remplacement. Cela va quand même nous coûter le dépôt de sécurité de $350, donc cela reste une chose à éviter. Il faudra donc être vigilant dans nos tests de vols, surtout au début. Les premiers vols du drone devront être à basse altitude et faible vitesse, et préférablement dans un espace ouvert pour éviter toutes collisions horizontales.

Un autre risque potentiel est le manque de cohésion et de communication dans le groupe. Un projet de cette taille et complexité demande une bonne coordination entre les membres du groupe. Idéalement, chaque membre du groupe devrait connaître les tâches sur lesquelles les autres membres travaillent, et être au courant du progrès de chacun, de sorte à être en mesure de prêter assistance à ceux qui en ont besoin. Il est aussi important que chaque membre reste à jour avec le dépôt Gitlab du projet. Même si un membre fait du bon travail individuellement, ça ne veut pas dire grand-chose si lui ou elle est tellement en retard avec la branche *main* qu'intégrer son code avec le reste devient un défi technique majeur. Tous les membres sont encouragés à faire de fréquentes commandes *git pull/git merge*, et les personnes qui ne sont pas confortables avec l'utilisation de Git/Gitlab devraient demander de l'aide aux coéquipiers ou aux chargés. Ce serait aussi une bonne idée d'organiser quelques activités de cohésion, de sorte à promouvoir une atmosphère positive et amicale dans l'équipe, et donner une idée des forces et faiblesses de chacun.

5.3. Tests (Q4.4)

Afin de vérifier le bon fonctionnement du système, plusieurs tests seront nécessaires à faire. Selon les requis, on peut déterminer certains de ces tests en les décomposant parmi les sous-systèmes. Les tableaux suivants montrent les tests à absolument effectuer.

Tableau 3: Tests pour le sous-système de l'interface utilisateur

Tests pour l'interface utilisateur		
# de test	# requis	Test
1-a	R.F.1	Peser le bouton "Identifier" permet de lancer la commande d'identification
1-b	R.F.1	Ne pas peser le bouton "Identifier" ne permet pas de lancer la commande d'identification
2-a	R.F.2	Peser le bouton "Lancer la mission" permet de lancer la commande de lancer la mission
2-b	R.F.2	Ne pas peser le bouton "Lancer la mission" ne permet pas de lancer la commande de lancer la mission
2-c	R.F.2	Peser le bouton "Terminer la mission" permet de terminer la mission
2-d	R.F.2	Ne pas peser le bouton "Terminer la mission" ne permet pas de terminer la mission
3	R.F.3	Il est possible d'obtenir l'état des drones à chaque 1 Hz
4	R.F.8	Il est possible d'afficher la carte en continu
5	R.F.9	Il est possible d'afficher la position du drone sur la carte en continu
6-a	R.F.10	Il est possible d'avoir un format d'affichage sur PC
6-b	R.F.10	Il est possible d'avoir un format d'affichage sur tablette
6-c	R.F.10	Il est possible d'avoir un format d'affichage sur mobile
6-d	R.F.10	Il est possible d'avoir 2 appareils connectés pour la visualisation des missions
7	R.F.11	Toutes les informations de plusieurs drones en mission peuvent être affichées sur la même carte
8	R.F.13	L'interface doit afficher l'état *crashed* d'un drone
9	R.F.14 R.F.16	L'interface doit pouvoir permettre de mettre à jour le logiciel hors des missions
10	R.F.17	L'interface doit afficher les informations de la base de données
11	R.F.18	Une carte d'une ancienne mission peut être affichée
12	R.F.19	Il est possible de lancer la fonctionnalité P2P à partir de l'interface
13-a	R.C.1	Il est possible d'afficher les logs de la mission courante sur demande
13-b	R.C.1	Il est possible d'afficher les logs d'une mission passée sur demande

Pour l'interface, on voit que plusieurs commandes doivent fonctionner lorsque l'on pèse sur un bouton. Les tests doivent donc être développés en fonction qu'une action de l'utilisateur envoie une commande au serveur. Un autre aspect est par rapport à l'affichage sur plusieurs plateformes. Il faut s'assurer d'avoir un design qui s'adapte au format de l'écran. L'aspect d'affichage de la carte est aussi important. Il faut tester que les données envoyées par le serveur soient correctement affichées à l'interface de la carte. Évidemment, il faut avoir accès aux logs. Le dernier aspect est par rapport aux anciennes missions. Il doit être possible d'afficher les informations demandées par l'utilisateur de façon précise.

Tableau 4: Tests pour le sous-système de la station au sol

Tests pour la station au sol		
# de test	# requis	Test
1	R.F.1	Il est possible de lancer la commande "Identifier"
2-a	R.F.2	Il est possible de lancer la commande "Commencer la mission"
2-b	R.F.2	Il est possible de lancer la commande "Terminer la mission"
3	R.F.3	Il est possible d'obtenir l'état du drone
4-a	R.F.6	Il est possible de lancer la commande "Retour à la base"
5	R.F.7	La commande "Retour à la base" est appelée lorsque la batterie est à moins de 30%
6-a	R.F.8	La station au sol collecte les données de position du drone
6-b	R.F.8	La station au sol fait une carte à partir des données de position
6-c	R.F.8	La station au sol met à jour la position du drone sur les données de la carte
7	R.F.9	La station au sol affiche en continu la position des drones
8	R.F.11	La station au sol peut intégrer une carte globale
9	R.F.12	La station au sol détermine la position et l'orientation initiale des drones
10	R.F.14	La station au sol doit pouvoir mettre à jour le logiciel par paquets binaires hors des missions
11	R.F.16	La station au sol doit pouvoir changer le code par un éditeur
12-a	R.F.17	La station au sol peut enregistrer sur une base de données les attributs suivants : date et heure de la mission, temps de vol, nombre de drones, physique/simulation et distance totale parcourue par les drones
12-b	R.F.17	La station au sol permet d'accéder à la base de données et de trier les attributs
13	R.F.18	La station au sol enregistre les cartes

Dans un premier temps, il faut tester que les commandes reçues de la part de l'interface utilisateur font des instructions du côté du serveur. Aussi, il doit être capable de collecter les données de tous les drones de façon précise et qui corresponds à ce qui est envoyé de la part des drones. Le stockage sur la base de données est aussi une partie importante et on doit s'assurer que les bonnes données soient correctement stockées en plus d'être retrouvables par la station au sol. Les autres tests s'assurent de la fonctionnalité de l'éditeur de code et de la base de données.

Tableau 5: Tests pour le sous-système des drones physiques

Tests pour le *firmware* (drones physiques)		
# de test	# requis	Test
1	R.F.1	La commande "Identifier " fait clignoter une DEL

2-a	R.F.2	La commande "Lancer la mission" débute la mission (décollage)
2-b	R.F.2	La commande "Terminer la mission" termine la mission (atterissage)
3	R.F.3	Le drone peut envoyer son état
4	R.F.4	Le drone peut explorer l'environnement de façon autonome
5	R.F.5	Le drone peut éviter les obstacles détectés par les capteurs
6	R.F.6	Le retour à la base ramène à moins de 1 mètre de la base
7	R.F.7	Un niveau de batterie de moins de 30% ramène le drone à la base
8-a	R.F.8, R.F.9	Le drone peut envoyer sa position
8-b	R.F.8, R.F.9	Le drone peut envoyer la position des obstacles
9	R.F.13	Le drone doit pouvoir communiquer son *crash*
10	R.F.14	Le *firmware* doit pouvoir se mettre à jour par paquets binaires hors des missions
11	R.F.16	Le *firmware* doit pouvoir être changé par l'éditeur de code
12	R.F.19	Les drones peuvent communiquer P2P

Les drones doivent répondre aux commandes envoyées par la station au sol. Lors de la réception de cette commande, la bonne instruction doit être exécutée. De façon autonome, le drone doit aussi être capable d'effectuer des actions. Par exemple, il doit envoyer son état et explorer l'environnement sans l'aide de la station au sol pour lui dire de le faire. Le retour à la base a plusieurs contraintes, alors il faut tester que tous les requis sont respectés et que la bonne instruction est exécutée. Les drones doivent pouvoir communiquer entre eux, alors tester qu'une connexion est possible est aussi bien important.

Tableau 6: Tests pour le sous-système Simulation

Tests pour la simulation		
# de test	# requis	Test
1-a	R.F.2	La commande "Lancer la mission" débute la mission (décollage)
1-b	R.F.2	La commande "Terminer la mission" termine la mission (atterrissage)
2	R.F.3	Le drone peut envoyer son état
3	R.F.4	Le drone peut explorer l'environnement de façon autonome
4	R.F.5	Le drone peut éviter les obstacles détectés par les capteurs
5	R.F.6	Le retour à la base ramène à moins de 1 mètre de la base
6	R.F.7	Un niveau de batterie de moins de 30% ramène le drone à la base
7-a	R.F.8, R.F.9	Le drone peut envoyer sa position

7-b	R.F.8, R.F.9	Le drone peut envoyer la position des obstacles

Au niveau de la simulation, il faut que les commandes provenant de la station au sol appellent les bonnes instructions. Le drone doit aussi envoyer de façon autonome quelques informations, par exemple son état et les obstacles détectés. Le retour à la base doit être appelé lorsque la batterie est à moins de 30% ou lorsque la commande est envoyée de la station au sol. Il est important de vérifier que ces deux actions provoquent le retour à la base.

Les tests décrits ici ne sont pas exhaustifs puisqu'un grand nombre de fonctions seront créées lors de l'implémentation de chaque sous-système. Il est bien sûr attendu de chaque développeur qu'il ajoute des tests unitaires pour chaque fonctionnalité. Les tests d'intégration pour leur part feront en sorte de vérifier le bon fonctionnement de l'ensemble des sous-systèmes lorsqu'ils sont connectés.

5.4. Gestion de configuration (Q4)

Plusieurs outils de gestion de configuration et d'organisation du code sont utilisés dans le cadre du projet. Pour bien comprendre pourquoi nous les avons sélectionnés, détaillons d'abord la structure du projet. Celui-ci est constitué de quatre dépôts *git* résumés dans le tableau ci-dessous.

Tableau 7: Fonction des différents dépôts

Nom du dépôt	Fonction
INF3995-105	Dépôt principal du projet, qui contient les règles d'assemblages entre les sous-modules.
INF3995-argos-simulation	Contient les fichiers permettant de décrire une simulation.
INF3995-firmware	Clone du dépôt « firmware » de Bitcraze. Adapté aux besoins du projet.
INF3995-backend	Contient le code de la station au sol ainsi que de son interface web. Contient deux modules distincts.

Le6.rovecip-3( )l14(.)(IN(m)F5(t)( )-19 995

Elle doit être détaillée et segmentée en *commits* de petite taille. Gitlab permet également la mise en place de l'intégration continue. Lors de chaque *Merge Request*, le code est compilé et les tests sont exécutés à l'aide des règles définies dans le *Dockerfile*. Le succès de l'opération est une condition à la complétion de la demande.

À noter : Les pipelines d'intégration continue sont exécutés sur des serveurs appartenant aux membres de l'équipe, il n'y a donc pas de coûts additionnels liés à leur utilisation.

Chaque module est documenté à l'aide d'un fichier rédigé à cet effet (README.md) placé à sa racine. Ce dernier indique les dépendances du module ainsi que les étapes permettant de le compiler, tout comme ses fonctionnalités et interfaces. Le code de chaque module s'autodocumente, c'est-à-dire qu'il se doit d'être clairement lisible et que les commentaires sont utilisés avec parcimonie et sont réservés à des manipulations plus abstraites. Enfin, la plateforme Gitlab intégre également des outils de gestion permettant de définir les tâches à venir et de les assigner aux différents membres. Cela permet de documenter les modifications effectuées. Chaque requis est associé à au moins un « Issue » Gitlab. Ces dernières contiennent la description générale de l'implémentation. Les « Issues » sont réparties entre les différentes remises (*Milestones* sous Gitlab). Les *Merge Requests* sont également associés aux « Issues » concernées. Il est donc possible d'assurer un suivi entre les requis et les tâches effectuées afin de leur répondre.

Enfin, les tests sont implémentés directement dans chaque module. Le client de la station au sol les implémente en utilisant le cadriciel Karma à même les répertoires de développement. Les tests des autres modules se trouvent plutôt dans des répertoires distincts. Les tests du *firmware* et du module de simulation sont réalisés avec le cadriciel Unity. Les tests du serveur de la station au sol le sont plutôt avec PyTest.

Pour tous les modules, l'ensemble des tests doivent être concluants afin qu'une *Merge Request* puisse être finalisée.

5.5. Déroulement du projet (Q2.5)

En ce qui a trait à la remise du PDR, nous croyons que les choses se sont sensiblement bien déroulées. La rédaction du rapport a été satisfaisante. Bien que quelques sections aient eu quelques défauts, l'ensemble a constitué un travail de qualité. Toutes les tâches reliées au code ont été réalisées en concordance avec ce qui était prévu dans l'échéancier. De plus, mis à part une mauvaise compréhension de ce qui était demandé dans le cadre des démonstrations vidéo, tous les requis étaient présents et fonctionnels sur les démonstrations. Aussi, le requis omis en raison d'une incompréhension était aussi fonctionnel, quoique non démontré.

Ensuite, entre la remise du PDR et cette remise-ci, soit celle du CDR, le travail au niveau du code a quelque peu souffert. Alors que nous suivions notre échéancier avec assiduité avant la remise du PDR, certaines fonctionnalités ont connu du retard, notamment celle

des logs. Cela s'explique en partie par le fait que nous avons tous eu environ deux semaines très chargées lors de la mi-session, ce qui a réduit le temps que nous pouvions accorder à la réalisation des requis techniques. Toutefois, nous avons réussi à rattraper notre retard grâce à notre bonne entente et nos habiletés de communication.

Nous avons aussi eu une rencontre avec un conseiller HPR. Cette rencontre nous a permis d'identifier quelques éléments qui ne fonctionnaient pas dans notre équipe, notamment au niveau de l'organisation. Nous n'avons pas encore corrigé ces éléments encore, puisque la rencontre HPR était la semaine même de la remise, mais nous avons déjà des pistes de solution. Nous voulons par exemple organiser des *scrums* dans un endroit plus propice que la salle de laboratoire et aussi prendre des notes de ce qui se dit dans nos réunions.

Après avoir remis le CDR, le processus de travail s'est quelque peu amélioré, mais est globalement resté sur la même lancée. Le retard sur l'échéancier accumulé depuis le CDR s'est poursuivi lors du dernier sprint. Nous étions tous occupés par la fin de session, donc, nous n'avons jamais complètement comblé notre retard. La situation s'est tout de même quelque peu améliorée.

Nous avons aussi pu corriger des éléments qui avaient été soulevés lors de la rencontre avec un conseiller HPR. Nous avons par exemple consacré dix à quinze minutes par séance de laboratoire à faire des *scrums*. Au cours de ces courtes réunions, chacun disait ce qu'il avait fait depuis le dernier cours afin que tous soient au courant des avancées de l'équipe. Nous faisions ces rencontres dans la Station Polytechnique afin de ne pas être dérangés par d'autres équipes travaillant dans la salle de laboratoire. Plus de notes étaient prises par rapport aux décisions prises lors des *scrums*. Nous avons aussi écrit plus de documents permettant de savoir en un coup d'œil ce qu'il restait à faire et quelles tâches étaient assignées à chacun.

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

Une fois toutes les différentes branches intégrées, nous pouvons constater le fonctionnement qui suit. D'abord, les drones physiques répondent à la commande d'identification. Qu'il s'agisse des drones physiques ou de ceux de la simulation, ceux-ci peuvent lancer et terminer leur mission, où ils explorent leur environnement de façon aléatoire. Sur la simulation, cet environnement est généré de manière aléatoire.

Les drones sont aussi capables d'éviter les obstacles et de revenir à la base lorsque demandé par l'intermédiaire de l'interface utilisateur. Ce retour est aussi déclenché automatiquement lorsque le niveau de batterie est trop faible. Enfin, les drones ont de plus une fonctionnalité de *peer-to-peer*. Celle-ci permet d'afficher par les DEL des drones la distance entre ceux-ci et la station au sol.

Nous pouvons aussi avoir l'état des drones mis à jour plus d'une fois par seconde sur l'interface graphique. Cette dernière est assez adaptable pour pouvoir être utilisée sur un écran d'ordinateur, de tablette ou de téléphone mobile, et ce, à partir de plusieurs appareils en même temps. On y affiche aussi des logs de débogage.

Une carte est aussi affichée par l'interface utilisateur. Elle affiche le trajet effectué par les drones, leur position actuelle ainsi que les obstacles détectés par ceux-ci. La position initiale des drones peut aussi être spécifiée. La carte peut aussi afficher les informations collectées par un seul drone spécifique. Ces cartes sont à chaque fois enregistrées, puis peuvent être ouvertes à nouveau pour consultation.

Les cartes mentionnées plus haut sont en effet enregistrées sur une base de données. Cette dernière contient aussi divers attributs pour chaque mission. La liste des dix dernières missions ainsi que leurs attributs enregistrés est accessible à partir de l'interface utilisateur.

Finalement, en cas de problème, le système est capable de détecter les *crashes*. Pour pallier les problèmes causant un éventuel *crash*, il est possible d'envoyer des mises à jour par paquets binaires ou de modifier le code des contrôleurs de drones en passant par un éditeur de code.

Bref, les tests de fonctionnement montrent que les requis obligatoires fonctionnent comme prévu. De plus, on s'aperçoit aussi que tous les requis optionnels que nous voulions remplir fonctionnent à merveille.

Pour finir, notre pipeline a un linter qui permet de s'assurer à chaque *merge request* que notre code respecte les conventions que nous avons choisies, ainsi qu'une étape où les tests unitaires sont exécutés pour effectuer un suivi récurrent de l'état du projet.

7. Travaux futurs et recommandations (Q3.5)

Il n'y a qu'un seul requis que nous n'avons pas réalisé, soit R.F.20. Ce requis consistait à pouvoir tracer sur la carte de l'interface utilisateur une zone de plus de quatre mètres carrés où le drone doit retourner s'il n'y est pas. Ce requis nous semblait comporter trop de critères et être trop difficile à implémenter dans le court laps de temps alloué à la conception du système, d'où sa mise à l'écart lors de la planification du projet. Réaliser cette partie manquante pourrait être un bon premier futur travail sur le système.

Ensuite, nous avons plusieurs recommandations qui pourraient être appliquées lors de futurs travaux sur le système. D'abord, les capteurs des drones ne font pas la distinction entre un mur et un drone. Nous avons donc plusieurs points sur la carte qui sont affichés en tant que murs sans en être. Cela peut porter à confusion, donc une amélioration pourrait être de vérifier si le « mur » détecté est toujours à la même position une seconde plus tard, ou bien d'utiliser un algorithme vérifiant que les points étant des murs sur la

carte ont des dimensions assez grandes pour être de véritables obstacles, et non des drones détectant d'autres drones.

Toujours sur le sujet des capteurs, ceux-ci sont souvent imprécis. Leur fonctionnement n'est pas fiable et l'exactitude des valeurs mesurées est douteuse. Le pire exemple est la position des drones faite à l'aide des capteurs. Sur la simulation, tout fonctionne bien, mais sur les drones physiques, le positionnement est beaucoup plus approximatif. Une solution pourrait être d'implémenter un système de balises pour faire la calibration des drones.

Une autre amélioration contreviendrait directement à l'un des requis du projet, mais nous croyons qu'elle serait tout de même intéressante. Le requis R.L.3 précise que « la station au sol ne doit en aucun cas dicter les mouvements précis des drones », mais nous croyons qu'il serait intéressant de pouvoir explorer de façon manuelle en pilotant le drone avec les flèches du clavier, ou avec un élément graphique contenu dans l'interface.

Une dernière amélioration serait la durée de vie de la batterie des drones. Cela permettrait de faire de meilleures missions d'exploration et rendrait la tâche de conception plus facile. Malheureusement, cela devient un autre projet entièrement.

8. Apprentissage continu (Q12)

Julien Bourque

J'ai trouvé que j'avais un déficit de connaissances théoriques en mathématiques et en physique pour programmer le comportement du drone. Pour combler cela, j'ai lu la documentation du *firmware* du drone en ligne et j'ai fait part de mes incompréhensions à Gabriel, qui a fait un passage en génie physique. Sinon, j'ai aussi eu plus de difficultés avec Python et l'interface utilisateur. J'ai donc fait des tutoriels en ligne, lu de la documentation encore une fois et consulté un guide de bonnes pratiques en développement d'interfaces graphiques. À l'avenir, tout cela pourrait être amélioré en faisant des prototypes de design d'interfaces et en communiquant mes lacunes théoriques au reste de l'équipe afin d'avoir de l'aide.

Gabriel Bruno

Je pense que j'avais une certaine confusion envers les outils de communication entre les services comme gRPC. J'ai eu de la difficulté lorsqu'il a été temps d'ajouter des commandes ou de déboguer les envois de ces commandes entre les services. J'ai fait appel à mes collègues, puisque certains avaient déjà utilisé cette technologie dans le cadre d'un stage. Je me suis aussi bien sûr référé à la documentation en ligne. Afin d'éviter ce genre de problème, j'aurais pu me pencher plus sur cet aspect dès le début

du projet pour mieux comprendre la structure et la hiérarchie des appels de fonction gRPC. Cela m'aurait évité de me perdre dans l'apprentissage de cette technologie en plein milieu du projet.

Nicolas Charron

Deux lacunes parmi celles que j'ai remarquées au cours du projet étaient le fonctionnement du *firmware* et l'écriture de code en Python. J'ai beaucoup moins travaillé sur le *firmware*, donc il m'a été beaucoup plus difficile de devenir familier avec cette portion du projet. Afin de combler ce manque dans mon expertise, j'ai révisé plusieurs *merge requests* en lien avec cette partie. Cela m'a permis de me familiariser avec le code. Une autre idée aurait été d'implémenter une fonctionnalité en lien avec le *firmware*, mais cela n'a pas été possible avec notre distribution des tâches. En ce qui concerne le code en Python, j'ai essayé de suivre des exemples reliés aux drones avec lesquels on travaillait. Toutefois, une meilleure approche aurait été de lire la documentation de Python et de commencer par de plus petits projets.

Audrey Leblanc

J'ai pour ma part eu des difficultés avec les connaissances requises en mathématiques et en physiques, ainsi qu'avec la construction d'un projet complet à partir de Docker. Pour les connaissances théoriques, j'ai moi aussi consulté Gabriel en raison de parcours en génie physique, mais j'ai aussi cherché des exemples de formules ou de transformation de coordonnées sur Internet. Pour Docker, j'ai consulté mes notes de cours d'INF8480, cours que je suis en même temps, ainsi que Julien, qui a de l'expérience avec Docker. Si c'était à refaire, je pourrais faire le cours de mécanique pour ingénieur avant celui-ci et faire plus de tests par moi-même dans les Dockerfile afin d'en découvrir le fonctionnement de façon autonome.

Thierry Spooner

Mes deux principales lacunes ont été le manque de prise en compte de la gestion des exceptions et des messages d'erreurs qui y sont liés ainsi qu'une compréhension peu approfondie de la programmation asynchrone en TypeScript. Afin de résoudre les problèmes reliés à ces lacunes, j'ai agi de manière proactive pour l'implémentation de la gestion d'erreurs après m'être rendu compte du manque d'une telle gestion. J'ai aussi utilisé beaucoup d'exemples qui fonctionnaient déjà pour régler des bogues. Pour remédier à tout cela dans un futur projet, je pourrais m'assurer de vérifier que les erreurs ne se propagent pas dans tout le programme dès la conception dudit programme, et non seulement lors du débogage. Je pourrais aussi moins me fier à des exemples tirés

d'Internet pour plutôt lire la documentation associée aux technologies que j'utilise, surtout dans la conception d'interfaces utilisateur.

9. Conclusion (Q3.6)

Tout d'abord, nous en concluons que la démarche de conception a graduellement divergé de ce qui était imaginé au départ. Toutefois, on pouvait s'y attendre considérant l'envergure du projet en question. Nous avons tout de même réussi à livrer un système avec un fonctionnement duquel nous sommes fiers.

Par rapport aux hypothèses, nous en avions posé quelques-unes qui se sont avérées justes. De plus, certaines suppositions ont permis de simplifier la conception du prototype, notamment le fait que les turbulences dans l'air soient négligeables par rapport au poids des drones. Cela nous a permis de nous concentrer sur le mouvement et l'exploration plutôt que la stabilisation du drone. Parmi nos hypothèses justes, il y avait le temps de communication très faible entre la station au sol et les drones, ainsi que la représentation relativement fidèle du comportement des drones physiques dans la simulation.

Concernant notre vision du système, quelques ajustements ont été nécessaires, mais la vision globale a été respectée dans son ensemble. Parmi les ajustements, on retrouve le remplacement de TCPsockets par gRPC pour effectuer la communication entre les différents services et parties du système et l'ajout de Bootstrap à Angular pour la conception de l'interface utilisateur. Cependant, l'architecture en services et *components* a été gardée pour l'interface, tout comme l'idée d'avoir un contrôleur recevant les commandes dans le *firmware*, puis déléguant le traitement à d'autres classes, par exemple.

Bref, nous pouvons retenir de notre démarche de conception qu'il est très difficile d'exactement prévoir l'architecture d'un système d'envergure dès le début de la conception, mais que cet exercice permet de nous guider pour la suite et de faciliter l'implémentation de la solution.

10. Références (Q3.2)

[1] Beltrame, Giovanni, Ulrich Dah-Achinanon et Sami Sadfa. *Système aérien d'exploration – Demande de proposition*. Montréal : Polytechnique Montréal, 2022. [En ligne]. Disponible : https://moodle.polymtl.ca/course/view.php?id=1703

[2] Bitcraze (2022). Crazyflie-Firmware Overview. [En ligne] Disponible : https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/.

[3] Bootstrap (2022). Get started with Bootstrap. [En ligne] Disponible : https://getbootstrap.com/docs/5.2/getting-started/introduction/

[4] MongoDB (2022). Welcome to the MongoDB Documentation. [En ligne] Disponible : https://www.mongodb.com/docs/

[5] Docker (2022). Docker Docs. [En ligne] Disponible : https://docs.docker.com/

[6] ARGoS (2022). User Manual. [En ligne] Disponible : https://www.argos-sim.info/user_manual.php

[7] Pytest (2022). Full pytest Documentation. [En ligne] Disponible : https://docs.pytest.org/en/7.1.x/contents.html

[8] Bitcraze (2022). Crazyflie Summer Project with ROS2 and Mapping. [En ligne] Disponible : https://www.bitcraze.io/2022/07/crazyflie-summer-project-with-ros2-and-mapping/

[9] Mozilla (2022). The WebSocket API (WebSockets). [En ligne] Disponible : https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[10] ARGoS (2022). API. [En ligne] Disponible : https://www.argos-sim.info/api/index.php

[11] Angular (2022). DatePipe. [En ligne] Disponible : https://angular.io/api/common/DatePipe

[12] PEP 8 (2022). PEP 8 — the Style Guide for Python Code. [En ligne] Disponible : https://pep8.org/

[13] Nielsen Norman Group (2020). 10 Usability Heuristics for User Interface Design. [En ligne] Disponible : https://www.nngroup.com/articles/ten-usability-heuristics/

[14] ESLint (2022). Find and fix problems in your JavaScript code. [En ligne] Disponible : https://eslint.org/

[15] LLVM (2022). LLVM Coding Standards. [En ligne] Disponible : https://llvm.org/docs/CodingStandards.html

[16] PyMongo (2022). PyMongo 4.3.3 Documentation. [En ligne] Disponible : https://pymongo.readthedocs.io/en/stable/

Annexes

Les vidéos de démonstration pour la remise du PDR sont disponibles aux deux liens suivants :

- https://www.youtube.com/watch?v=hs-jucbSDNo
- https://www.youtube.com/watch?v=bV3Axq7wY5M

Les vidéos de démonstration pour la remise du CDR sont disponibles aux quatre liens suivants :

- https://youtu.be/rsf2kfYwevg
- https://youtu.be/_5hsyV-jYjQ
- https://youtu.be/9tELLFvvRRQ
- https://youtu.be/rR7ERyuHU80

Les vidéos de démonstration pour la remise du RR sont disponibles aux dix liens suivants :

- https://www.youtube.com/watch?v=SE6PGoNG7HE
- https://www.youtube.com/watch?v=kQ1fM0RwHMg
- https://www.youtube.com/watch?v=p70tUbzFrpI
- https://www.youtube.com/watch?v=5UfInr4SvYw
- https://www.youtube.com/watch?v=RaWFVwZ_1H4
- https://www.youtube.com/watch?v=Vn_Gu51ttHw
- https://www.youtube.com/watch?v=a7MmSTICbeM
- https://www.youtube.com/watch?v=aGGv6gRTxuw
- https://www.youtube.com/watch?v=fqzUHZ_Tv8M
- https://www.youtube.com/watch?v=Al1XxQnymLo

Une capture d’écran de la figure 5 avec une meilleure résolution est disponible au lien suivant : https://drive.google.com/file/d/1ZbUcfK8G_xS8jGv_-RjVfizuDg2Ppb_A/view?usp=share_link