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. A2021-INF3995 du département GIGL.

Conception d'un système aérien d'exploration

Équipe No ***103***

Andrew Abdo (1954991)
Brando Tovar (1932052)
Estefan Vega-Calcada (1934346)
John Maliha (1984959)
Noe Berguin (1890494)

8 Décembre 2021

Table des matières

1. Vue d'ensemble du projet

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

L'objectif de ce projet est d'implémenter une application web qui permet de contrôler des drones qui vont explorer un environnement inconnu et générer une carte à partir des données recueillies par ces derniers. Ces résultats doivent être rapportés sur une interface utilisateur web. Le projet doit pouvoir être lancé dans un environnement simulé ou physique.

1.2 Hypothèse et contraintes (Q3.1)

C1. Le niveau de batterie de chaque drone devra être récupéré périodiquement par la station au sol afin de s'assurer que le niveau de batterie demeure supérieur à 30% en tout temps. lorsque le niveau de batterie est inférieur à 30% la station ordonne au drone concerner un retour à la base.

C2. Pour simuler l'espace, l'environnement dans lequel vole les drônes ne sera pas connu à l'avance.

C3. Les station au sol ne devrait pas contrôler les déplacement des drones mais uniquement envoyer des commandes de haut-niveau (start, stop, identify, return to base)

C4. Les deux drônes doivent fonctionner avec les CrazyRadio.

C5. La simulation est faite à partir de Argos.

C6. Il est possible de lancer le tout à partir de Docker Compose.

H1. La station au sol reçoit les données récoltées par les drônes en circulation grâce à la station au sol et on se base sur celles-ci afin de choisir la route optimale.

H2. Préférablement, le trajet du drône commencera et se terminera à la base de l'opérateur qui sera situé sur la machine principale. Autrement dit, la station de recharge du drône est localisée à la station au sol. La position cartésienne initiale et finale n'est donc pas forcément la même puisque le véhicule principal sera en déplacement et ne pourra pas être préalablement définie puisque le trajet du véhicule dépend des découvertes effectuées par les drônes.

H3. Le déplacement des drônes sera testé dans un environnement fermé, délimité par des murs et des obstacles. Malgré que l'environnement de la mise en situation serait ouvert et vaste.

H4. Les drones simulés devraient être capable d'explorer l'environnement sans faire de rotation.

1.3 Biens livrables du projet (Q4.1)

- *1er octobre 2021* - Réponse à l'appel d'offres & «Preliminary Design Review»
    - ○ Plan de projet
    - ○ Diagrammes d'architecture logicielle
    - ○ Spécification des requis du système
    - ○ Descriptif des processus de gestion
    - ○ Code source et exécutables
    - ○ Remise du prototype préliminaire

- *29 octobre 2021* - «Critical Design Review»
    - ○ Mise à jour des fonctionnalités remises précédemment
    - ○ Plan et résultats des tests logiciels
    - ○ Code source et exécutables
    - ○ Remise d'un système avec fonctionnement partiel

- *7 décembre 2021* - «Readiness Review»
    - ○ Mise à jour des fonctionnalités remises précédemment
    - ○ Plan et résultats des tests logiciels
    - ○ Code source et exécutables
    - ○ Remise du livrable final complet

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Noé Berguin - coordonnateur de projet
Responsabilité principale: Résolution des bogues sur l'ensemble du système
Autres responsabilités:

- Gestion de la version du système
- Vérifier la progression de chaque membre
- Vérification de la cohérence des modèles
- Planification des sprints et de la répartition des tâches
- Assister les sous-équipes en difficulté

Estefan Vega-Calcada - développeur-analyste
Responsabilité principale: Simulation
Autres responsabilités:

- Application Web
- Gestion du serveur
- Testing

Brando Tovar Oblitas - développeur-analyste
Responsabilité principale: Simulation
Autres responsabilités:
- Application Web
- Gestion du serveur
- Testing

Andrew Abdo - développeur-analyste
Responsabilité principale: Logiciel embarqué des drônes
Autres responsabilités:
- Application Web
- Gestion du serveur
- Testing

John Maliha - développeur-analyste
Responsabilité principale: Logiciel embarqué des drônes
Autres responsabilités:
- Application Web
- Gestion du serveur
- Testing

Il est à noter que les membres de l'équipe se réservent le droit d'assister les autres sous équipes lors des tâches. L'équipe planifie également une rotation des tâches lors de la neuvième semaine afin de diversifier les responsabilités de chacun et d'assurer un partage de connaissance optimal. Les rôles ne sont pas attribués à l'avance et les membres de l'équipe se réservent également le droit d'assister les autres dans l'accomplissement de leurs requis du projet.

2.2 Entente contractuelle (Q11.1)

L'équipe accepte un contrat du type «Livraison clé en main - Prix ferme». Les coûts associés au projet seront donc fixes et prédéterminés (voir section 4.1 estimations des coûts du projet). Cette approche minimise la fréquence de suivi requise par le promoteur et favorise l'autonomie de l'équipe. La connaissance des requis est connue d'avance grâce au document d'exigence fourni par le promoteur, l'équipe ne voit donc pas d'inconvénient à proposer un contrat de ce type dans le cadre de ce projet. Nous proposons une solution qui adhère aux spécifications proposées par l'employeur (voir section 3), il n'y aura donc pas de négociation nécessaire et l'équipe ne souhaite pas établir de changements concernant les exigences actuelles. Ce type de contrat convient à la situation actuelle dû au fait que le projet est d'une durée de 12 semaines (courte durée) ce qui simplifie la prévision des dates d'échéances des nombreuses fonctionnalités. Le tout devra constituer un produit final en date du 7 décembre 2021.

3. Description de la solution

Les requis de la solution proposée comportent un total de 100 points répartis entre les différentes sections du projet.

3.1 Architecture matérielle et logicielle générale (Q4.5)

Pour répondre aux exigences, nous avons décidé de séparer notre projet en quatres applications (voir image-ci dessous). Nous avons choisi d'utiliser le framework Angular pour créer notre interface web, car tous les membres de l'équipe possédaient déjà de l'expérience avec ce framework. Pour le serveur, nous avons opté pour Python, car le API de crazy-radio était en python. Nous avons décidé d'utiliser la librairie Flask pour gérer les requêtes HTTP entre le serveur et l'interface utilisateur.

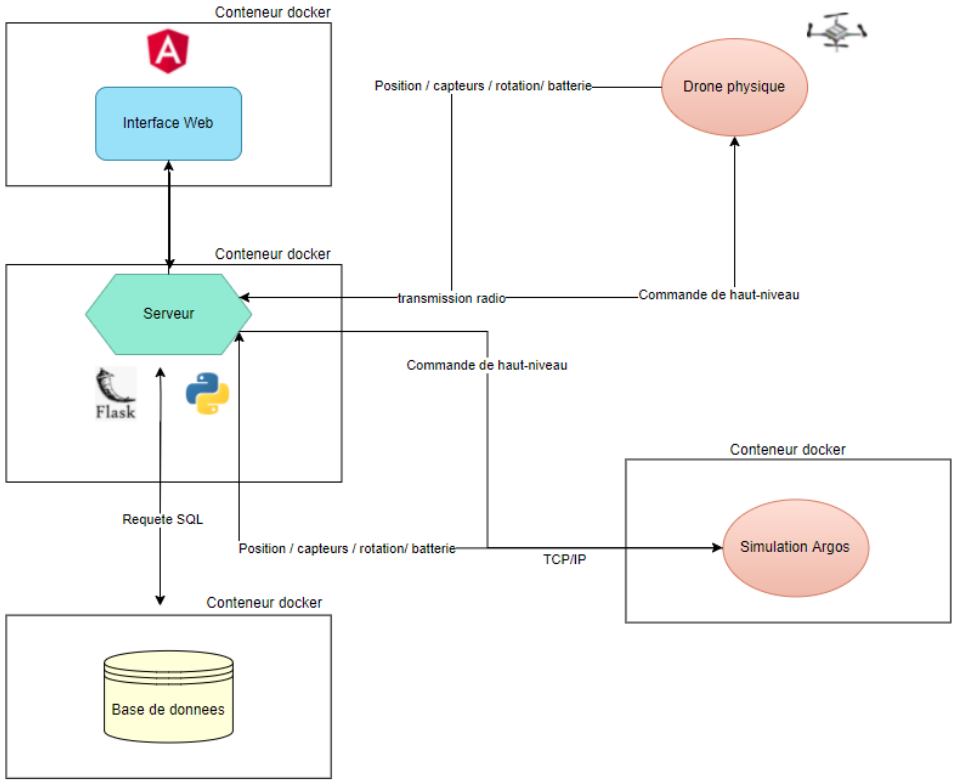


Image 3.1: Vue d'ensemble de l'architecture matérielle et logicielle

Le serveur doit pouvoir communiquer avec les autres applications de façon bidirectionnelle. Les deux bulles rouges représentent les logiciels de drone (embarqué / simulation). On peut regarder l'annexe 3.2 pour observer un exemple de communication de données entre la simulation et le serveur. Ces deux communications utilisent le principe de socket pour transmettre l'information.

Concernant la partie matériel, nous avons l'obligation d'utiliser des drones "crazyflie" et une radio "crazyradio". Tous les membres de l'équipe utilisent un système Ubuntu pour coder. L'équipe utilise également le gestionnaire de version "Git".

Pour le serveur http nous avons hésité entre deux frameworks python, django et flask. Dans notre cas, le serveur http n'avait besoin que de répondre à quelques requêtes donc la solution django qui est plus lourde et demande plus de temps pour apprendre n'était pas l'idéal. Avec flask nous avions l'opportunité de travailler avec un framework plus léger avec une moins grande courbe d'apprentissage qui par-dessus tout, répond à tous nos besoins.

Tableau 3.1 - Requis de l'architecture matérielle et logicielle

Requis	Points	Description	Solution
Requis Matériel #1	-	Le prototype doit être implémenté avec deux drones Bitcraze Crazyflie 2.1 avec le « STEM Ranging bundle » complètement installé, fournis par l'Agence	-
Requis Matériel #2	-	Le seul moyen de communication entre la station au sol et les drones physiques doit être une Bitcrazy Crazyradio PA connectée à la station au sol, fournie par l'Agence	-
Requis Matériel #3	-	Les drones doivent avoir seulement le « ranging deck » et le « optical flow deck » installés pour opérer	-
Requis Matériel #4	-	La station au sol doit être un laptop ou PC avec une Crazyradio PA connecté par port USB	-
Requis Logiciel #1	-	Les drones doivent être programmés en utilisant l'API de BitCraze. Cependant, il est possible d'utiliser une machine virtuelle basée sur l'API à bord du drone (p.ex. la machine virtuelle de Buzz, BVM).	-
Requis Logiciel #2	-	L'interface utilisateur doit être la même lorsque la station au sol est connectée à la simulation ou aux drones physiques. À l'exception des fonctionnalités spécifiques à un des deux systèmes	-
Requis Logiciel #3	-	Le contrôle des drones doit se faire à bord de ceux-ci sur le code embarqué. La station au sol ne doit en aucun cas dicter les mouvements précis des drones, elle ne doit qu'envoyer des commandes de haut niveau (Lancer la mission, Mise-à-jour, etc.) et les informations associées.	-
Requis Logiciel #4	-	Toutes les composantes logicielles, sauf celles embarquées (drone physique), doivent être conteneurisées avec Docker. Ceci favorise la reproductibilité du système entre les étudiants lors du développement, évite les problèmes de compatibilité en les machines, permet d'éviter plusieurs bogues liés à des problèmes de configuration, facilite le déploiement, et permet une évaluation standardisée, reproductible et plus équitable.	-
Requis Fonctionnel #7	2 pts	Le retour à la base et l'atterrissage doit être activé automatiquement dès que le niveau de batterie devient moins de 30%. Les drones ne doivent pas décoller avec un niveau de batterie inférieur à 30%. Le niveau de batterie des drones doit être affiché sur l'interface utilisateur. Pour les drones physiques, le niveau peut être obtenu au moyen d'une table de voltage (ex.https://blog.ampow.com/lipo-voltage-chart/). La simulation du niveau de batterie dans ARGoS est à la discrétion du contractant.	-

Requis Qualité #1	5 pts	Le format (indentation, style de commentaires, boucles, etc.) du code doit être standardisé à travers le projet et suivre des conventions de codage reconnues et non des conventions maisons. Les conventions retenues pour chaque langage devront être spécifiées dans le README à la racine du projet. Le choix des conventions de codage est à la discrétion du contractant (ex. https://google.github.io/styleguide/ , https://developer.gnome.org/programming-guidelines/ stable/c-coding-style.html.en) Votre code et son architec	le code côté serveur respecte la nomenclature: Pip8 l'interface respecte la nomenclature: Angular coding style
Requis Qualité #2	5 pts	Chaque composant du logiciel doit avoir un test unitaire correspondant. Dans le cas où les tests unitaires sont impossibles ou trop complexes (ex : code de contrôle des drones embarqués et simulés), une procédure de test doit être détaillée pour chaque fonctionnalité (une par requis fonctionnel). Ces procédures ont la forme d'une série d'étapes pour déclencher la fonctionnalité à tester ainsi que le comportement normal attendu. Ces procédures de tests doivent être écrites dans un document, nommé "Tests.pdf" qui sera remis avec le livrable final. Elles pourront servir à l'évaluateur pour reproduire le comportement du système.	Chaque composante de l'interface est testée avec les outils déjà fournis avec Angular. chaque composante du serveur est testé avec la library python: unittest

Total de points: 12 points

3.2 Station au sol (Q4.5)

La station au sol est un ordinateur avec une interface Web qui peut communiquer avec les drones à travers un canal de communication. Nous avons décidé de séparer la station en deux blocs. Le premier bloc est une application web qui intégrera l'interface web. Le deuxième bloc est un serveur qui permettra la liaison entre l'interface et les drones. La communication doit être bidirectionnelle.

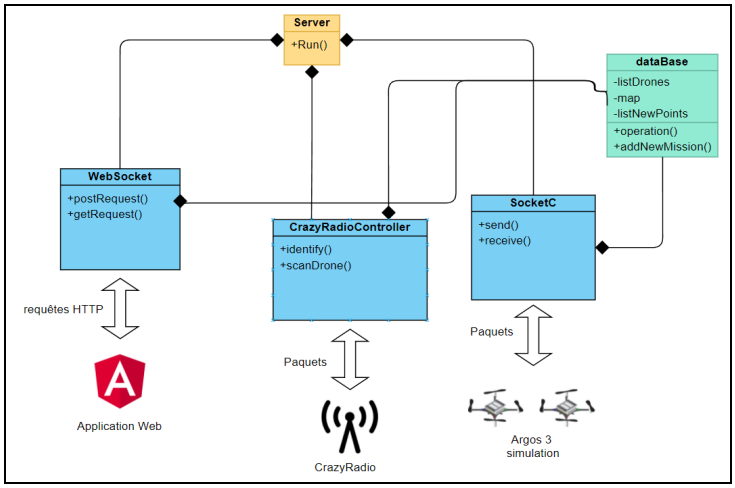


Image 3.2.2:Diagramme de classe - Station au sol

L'application web utilise le framework Angular (voir chap 3.5) et communique avec le serveur via des requêtes HTTP. Nous avons opté pour le framework Angular puisque tous les membres de l'équipe ont un peu d'expérience avec celui-ci suite au projet intégrateur de deuxième année à Polytechnique Montréal. Ceci nous aidait grandement

pour que tout le monde puisse comprendre la logique derrière le client, contrairement à si un des membres décidait de le faire avec React qui n'est pas familier pour tous. Le serveur en lui-même est programmé en Python pour pouvoir utiliser l'API crazyradio et permettre de communiquer avec les drones. La communication entre le serveur et la simulation Argos3 se fera via le principe de socket en utilisant le protocole TCP. Le serveur intégrera également une base de données afin de stocker les données générées. Cette base de données sera développée sous "PostgreSQL" et la communication se fera via des requêtes SQL. Pour la communication Interface/Serveur notre équipe a décidé d'utiliser la librairie open-source Flask qui supporte les requêtes HTTP. Nous avons opté pour la base de données en postgres, car nous avons tous une plus grande expertise dans l'utilisation de celle-ci. Aussi, généralement la syntaxe SQL offre une meilleure structure des requêtes que le NOSQL. Nous avons donc choisis une solution qui permettait d'avoir un développement plus rapide tout en ayant l'efficacité et la performance qu'offre une base de données en postgres.

Tableau 3.2 - Requis de la station au sol

Requis	Points	Description	Solution
Requis Fonctionnel #11	5 pts	*La station au sol doit intégrer les données de tous les drones et créer une seule carte globale de l'environnement exploré. On suppose ici que les positions et orientations initiales des drones sont connues. Si le requis R.F.12 n'est pas complété, les positions et orientations initiales peuvent être spécifiées directement dans un fichier de configuration sur la station au sol.*	L'état, la position, les capteurs et le niveau de batterie des drones seront récupérés périodiquement à une fréquence de 1Hz par le serveur. Les données seront analysées pour générer des nouveaux points et mettre à jour la position des drones. Ces points seront par la suite récupérés sur l'interface. Les points générés sont mis en commun sur une seule carte. Il n'y a pas une carte individuelle par drone. Un tableau regroupant l'ensemble des drones sera disponible sur l'interface. L'état des drones et leurs niveaux de batterie sera affiché dans ce tableau.
Requis Fonctionnel #12	1 pts	*La position et l'orientation initiale respective des drones (relative ou absolue selon les besoins du système) dans l'environnement (physique ou simulé) doit pouvoir être spécifiés par l'opérateur dans l'interface utilisateur avant le début de la mission. Si possible, ces informations peuvent être déterminées automatiquement par la station au sol, dans ce cas, l'opérateur n'a pas besoin de pouvoir les spécifier.*	*Dans l'interface, il sera possible de rentrer manuellement la position des drones. Des valeurs par défaut seront déjà intégrées.*
Requis Fonctionnel #17	5 pts	*Une base de données doit être présente sur la station au sol et enregistrer au minimum les attributs suivants pour chaque mission : date et heure de la mission, temps de vol, nombre de drones, physique/simulation et distance totale parcourue par les drones. L'opérateur doit pouvoir fouiller dans la base de données à partir de l'interface utilisateur. Au minimum, il doit avoir accès à la liste des missions précédentes et pouvoir les trier selon chaque attribut. Lorsque l'opérateur sélectionne une mission précédente, l'interface utilisateur doit en afficher les informations à l'écran pour consultation. Des informations supplémentaires peuvent être enregistrées à la discrétion du contractant.*	*Une base de données sera conteneurisée dans un container docker. Le serveur communique avec la base de données via des requêtes SQL. Il sera possible de stocker les missions dessus, également de supprimer les anciennes missions. Une mission regroupe l'ensemble des points et l'ensemble des logs plus les données comme la date et l'heure.*
Requis Fonctionnel #18	5 pts	*Les cartes générées lors d'une mission doivent être enregistrées sur la station au sol. Il doit être possible à partir de l'interface utilisateur d'ouvrir une carte générée dans une mission précédente pour l'inspecter de nouveau. La carte*	*Une base de données sera conteneurisée dans un container docker. Le serveur communique avec la base de données via des requêtes SQL. Il sera possible de stocker les missions dessus, également de supprimer les*

		générée peut être intégrée (ou non) à la base de données si le requis R.F.17 est complété.	*anciennes missions. Une mission regroupe l'ensemble des points et l'ensemble des logs plus les données comme la date et l'heure.*
Requis Conceptuel #2	4 pts	*Le logiciel complet de la station au sol doit pouvoir être lancé avec une seule commande sur un terminal Linux (ex. docker-compose). Il en va de même pour le lancement de l'environnement virtuel ARGoS. Noter que des arguments peuvent être nécessaires et qu'il est acceptable de devoir appuyer sur "Play" dans ARGoS avant de commencer. La procédure de compilation et de lancement de chaque système (lignes de commandes, arguments, etc.) doit être spécifiées dans un README à la racine du projet.*	*L'ensemble du projet sera conteneurisé sauf le logiciel embarqué sur les drones.*

Total de points: 20 points

3.3 Logiciel embarqué (Q4.5)

Pour la partie logiciel embarqué, il a fallu utiliser le Crazyflie framework pour coder les crazyflies. Ceci se sépare en deux parties soit, d'une part, la crazyflie firmware en C permettant de coder le firmware du drone. Nous avons choisi le langage C puisque personne dans l'équipe n'est familier avec le Buzz et il sera plus facile de trouver de l'aide externe avec un langage aussi répandu. D'autre part, pour envoyer les commandes voulues sur le drone on utilisera une autre libraire basée sur le langage python. Par la suite, le code sur le drone sera exécuté de façon continue et contiendra tout le code nécessaire pour faire les requis. Ensuite, on se sert de cflib pour communiquer entre le serveur et les drones par l'entremise de la Crazyradio en envoyant des paquets qui sont attendu sur les drônes en question. Selon le paquet reçu par les drones, une routine spécifique sera exécutée. Par exemple, le paquet «1» à l'état identification ce qui allumera la del M1 en vert. Par la suite, la del M1 restera allumée pendant 20 sec. Par la suite, le drone retournera à son état initial (idle).

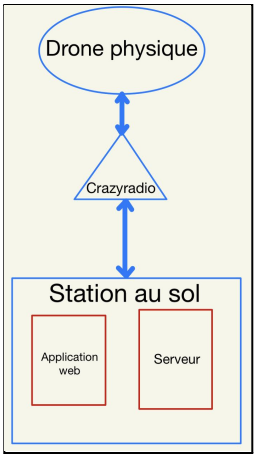


Image 3.3.1: Vue d'ensemble

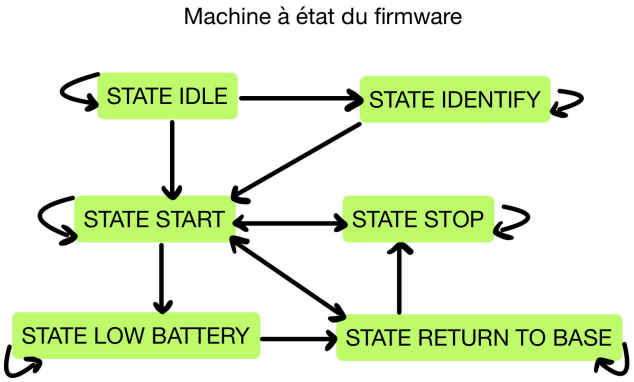
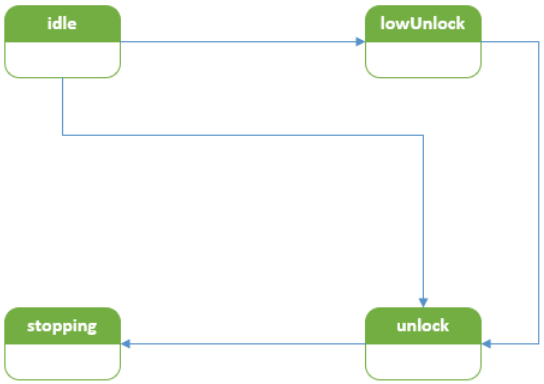


Image 3.3.2 : Machine à état du logiciel embarqué

Image 3.3.3 : Machine à état pour l'exploration



Logiciel embarqué- Diagramme de classe

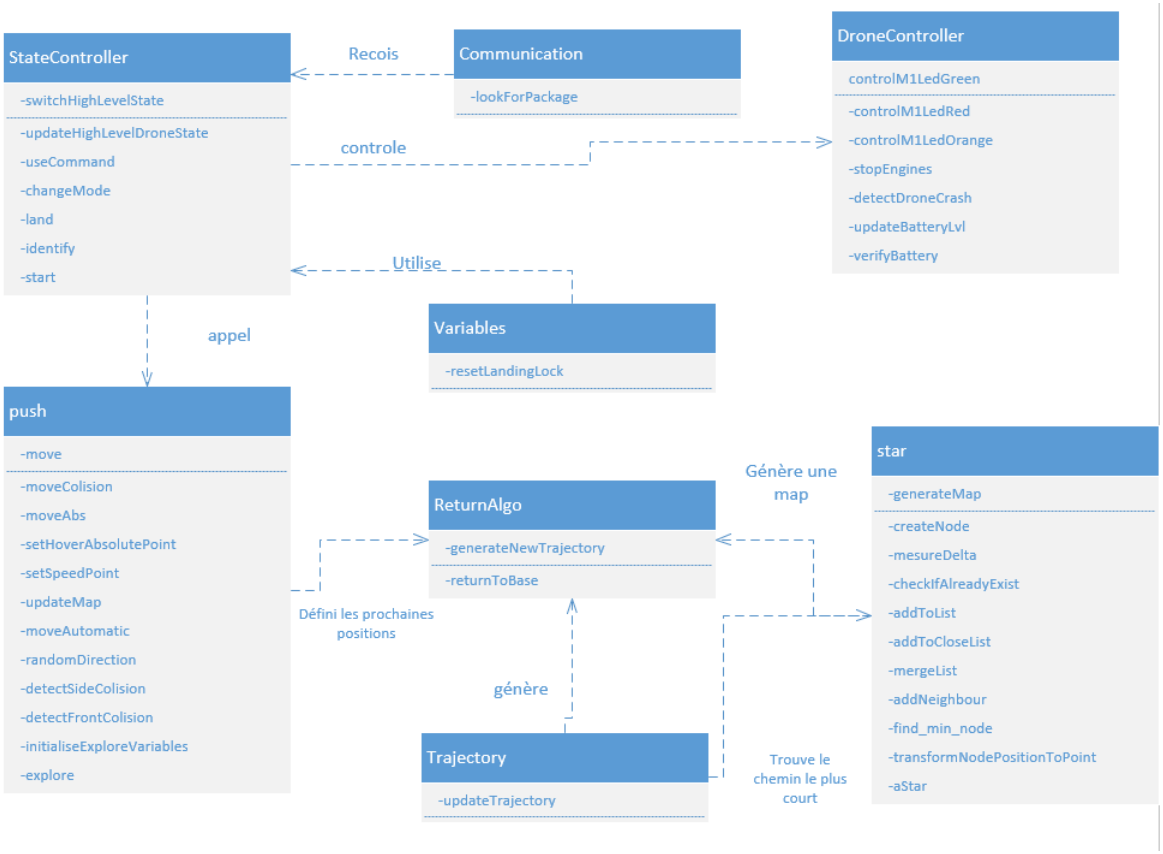


Tableau 3.3 - Requis du logiciel embarqué

Requis	Points	Description	Solution
Requis Fonctionnel #1	3 pts	*Chaque drone physique doit individuellement répondre à la commande "Identifier" disponible dans l'interface utilisateur. Lors de cette commande, au moins une des DELs du drone doit clignoter ou changer de couleur.*	Les uris obtenus par la radio sont utilisés pour cibler un drone spécifique lors de l'identification.
Requis Fonctionnel #2	2 pts	*L'essaim de drones doit répondre aux commandes suivantes, disponibles sur l'interface utilisateur :* *— "Lancer la mission" : décollage et début de la mission* *— "Terminer la mission" : atterrissage immédiat*	Les drones reçoivent les commandes sous forme d'entier. Pour lancer mission la commande sera envoyée sous forme de broadcast à tous les drones.
Requis Fonctionnel #4	2 pts	*Après le décollage, les drones doivent explorer l'environnement de façon autonome. L'algorithme est à la discrétion du contractant. Cela peut être une séquence de mouvements aléatoires* *(random walk)*	Nous utilisons un algorithme de déplacement aléatoire pour explorer l'environnement
Requis Fonctionnel #5	4 pts	*Les drones doivent éviter les obstacles détectés par leurs capteurs (objets statiques ou autres* *drones sans distinction).*	Les drônes utilisent leurs capteurs pour calculer les distances. Lorsqu'un objet est détecté à une distance inférieure au seuil minimal que nous avons établi, les drônes s'ajustent en conséquence.
Requis Fonctionnel #6	6 pts	*Le retour à la base doit rapprocher les drones de leur position de départ pour qu'ils soient à moins de 1 m de celle-ci. Il n'y a pas de requis concernant l'orientation finale. Une commande de "Retour à la base" doit être disponible sur l'interface utilisateur.*	*Durant la mission, chaque drone enregistrera ses positions. Lors du retour à la base, nous appliquerons un algorithme comme Dijkstra pour trouver le chemin le plus court pour optimiser le retour à la base.*
Requis Fonctionnel #8	5 pts	*La station au sol doit collecter les données des capteurs de distance des drones et générer des cartes de l'environnement. Il doit y avoir une carte individuelle pour chaque drone. Ces cartes doivent être affichées en continu lors de la mission (une ou plusieurs à la fois) dans l'interface utilisateur. Les cartes générées doivent ressembler à l'environnement exploré, aucune évaluation quantitative de la précision de la carte n'est requise.*	L'état, la position, les capteurs et le niveau de batterie des drones seront récupérés périodiquement à une fréquence de 1Hz par le serveur. Les données seront analysées pour générer des nouveaux points et mettre à jour la position des drones. Ces points seront par la suite récupérés sur l'interface. Une des particularités ici et qu'il faut ajouter une matrice de rotation sur les points car les drones physiques peuvent faire des rotation.
Requis Fonctionnel #13	4 pts	*Le système doit pouvoir détecter un crash de drone. Dans ce cas, l'interface utilisateur doit montrer l'état "Crashed" pour ce drone. Cette fonctionnalité peut être démontrée en déposant un drone à l'envers sur le sol.*	En utilisant la communication bi-directionnel

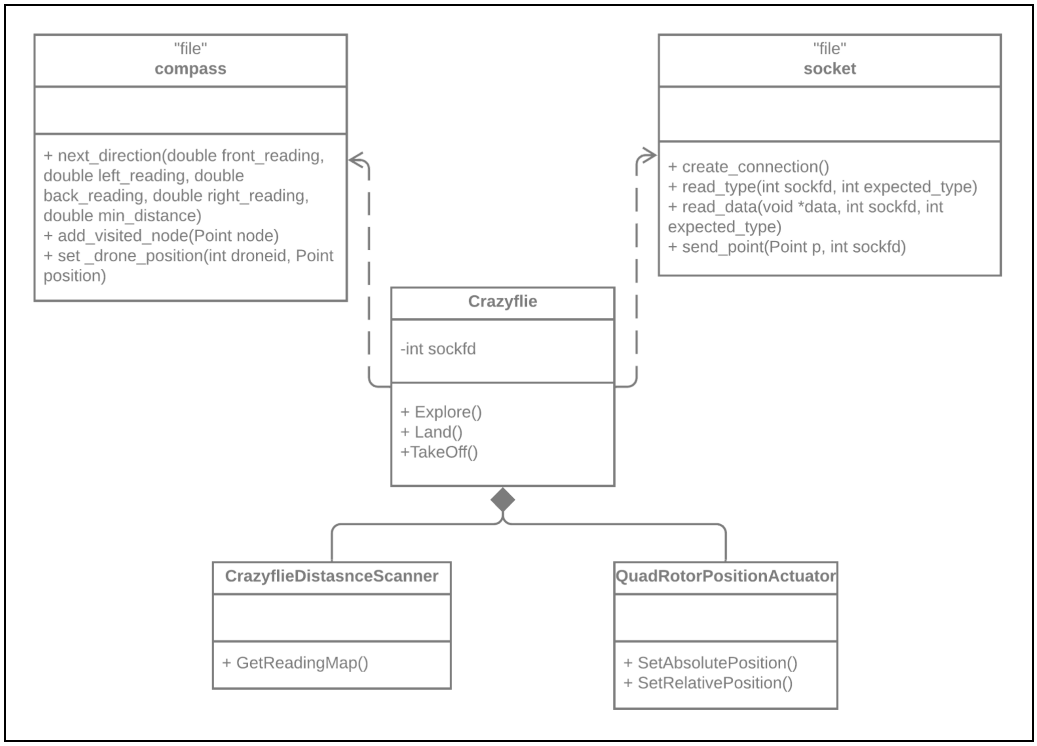
Total de points: 26 points

3.4 Simulation (Q4.5)

La simulation sera conçue principalement avec le langage C++. Ce langage est celui utilisé par le simulateur argos3 et est donc le plus indiqué pour la simulation. Nous utiliserons des sockets pour la communication avec le serveur python.

Le diagramme de vue d'ensemble de la simulation est sous forme de diagramme de séquence et peut être observé en annexe (Annexe 3.4.1). Il explique en détail l'interaction de cette composante avec le serveur et l'application web.

Simulation - Diagramme de classe



La classe Crazyflie est le point d'entrée de l'application qui contient toutes les composantes simulées d'un drone. Cette classe utilisera des fonctions des fichiers compass.h pour la logique des déplacements des drones et du fichier socket.h pour la communication avec le serveur. Ce sont les fonctions de compass qui seront implémentées en C et partager avec la partie embarquée du projet.

Tableau 3.4 - Requis de simulation

Requis	Points	Description	Solution
Requis Fonctionnel #2	2 pts	*L'essaim de drones doit répondre aux commandes suivantes, disponibles sur l'interface utilisateur :* *— "Lancer la mission" : décollage et début de la mission* *— "Terminer la mission" : atterrissage immédiat*	Pour lancer mission et terminer mission nous allons utiliser des sockets (solution standard pour la communication interprocessus). La commande sera donc broadcaster à tous les clients socket.

Requis Fonctionnel #4	2 pts	*Après le décollage, les drones doivent explorer l'environnement de façon autonome. L'algorithme est à la discrétion du contractant. Cela peut être une séquence de mouvements aléatoires (random walk)*	Nous allons utiliser un algorithme de random walk pour l'exploration autonome. Il sera fait en générant un vecteur aléatoire entre 4 possibles qui seront les directions en x positif, négatif et y positif, négatif. Cela facilitera la génération de la carte, car il n'y aura pas d'angle.
Requis Fonctionnel #5	4 pts	*Les drones doivent éviter les obstacles détectés par leurs capteurs (objets statiques ou autres drones sans distinction).*	Tous les déplacements prévus seront vérifiés par une fonction qui lira les capteurs et retournera un déplacement sûr qui permet d'éviter un éventuel obstacle rencontré.
Requis Fonctionnel #6	4 pts	*Le retour à la base doit rapprocher les drones de leur position de départ pour qu'ils soient à moins de 1 m de celle-ci. Il n'y a pas de requis concernant l'orientation finale. Une commande de "Retour à la base" doit être disponible sur l'interface utilisateur.*	À chaque début de mission, la position initiale sera sauvegardée et le retour à la base sera fait en allant à cette position tout en contournant les obstacles rencontrés.
Requis Fonctionnel #8	5 pts	*La station au sol doit collecter les données des capteurs de distance des drones et générer des cartes de l'environnement. Il doit y avoir une carte individuelle pour chaque drone. Ces cartes doivent être affichées en continu lors de la mission (une ou plusieurs à la fois) dans l'interface utilisateur. Les cartes générées doivent ressembler à l'environnement exploré, aucune évaluation quantitative de la précision de la carte n'est requise.*	Les drones enverront leur position et lectures de capteurs mis à l'échelle de la position. Ces données seront utilisées pour la génération des cartes qui sera faite avec canvas.
Requis Conceptuel #3	1 pts	*L'environnement virtuel pour les tests dans ARGoS doit pouvoir être généré aléatoirement (peut être fait simplement dans le fichier de configuration du simulateur). L'environnement simulé doit avoir un minimum de 3 murs (sans compter les 4 murs extérieurs).*	La génération aléatoire de l'environnement sera faite en changeant le seed dans le fichier de configuration. Il sera composé de quelques boîtes positionnées aléatoirement.
Requis Conceptuel #5	5 pts	*Le système doit être conçu pour fonctionner avec un nombre arbitraire de drones, entre 2 et 10 drones. La station au sol et l'interface utilisateur doivent s'adapter automatiquement et se connecter à tous les drones disponibles pour la mission, sans que l'utilisateur n'ait à en spécifier le nombre.*	Chaque nouveau client qui se connecte au socket sera considéré comme un drone supplémentaire et pourra être contrôlé par le client et servir pour la génération de carte.

Total de points: 23 points

3.5 Interface utilisateur (Q4.6)

Pour réaliser l'interface utilisateur, notre équipe utilisera le framework Angular. Ce framework propose de nombreux avantages dont la possibilité de séparer les données, le visuel, et les actions. Il est très optimisé dans la création d'applications web. L'opérateur doit pouvoir contrôler le système physique et le système simulé via notre interface. Les données générées doivent être visualisables sur l'interface. L'utilisateur pourra interagir avec l'interface via les components (voir image 1). l'interface prendra la forme suivante :

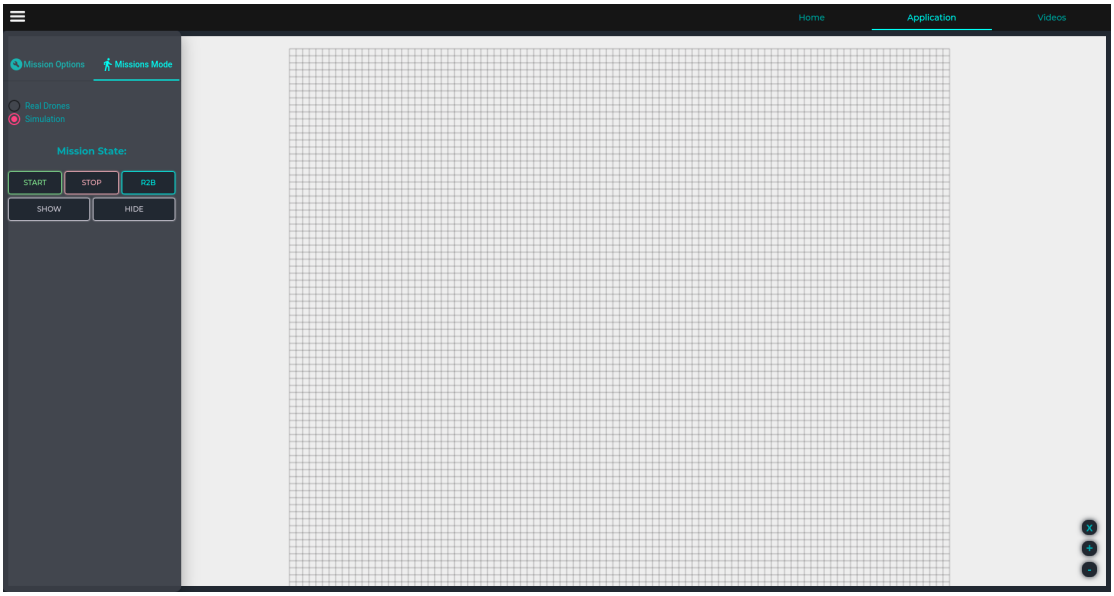


Image 3.5.1: Interface utilisateur

Le component "application" regroupe toutes nos fonctionnalités dans le panneau de droite qui est rétractable. Il permet le contrôle du système physique et de la simulation (au choix). Toutes les options de map comme la visualisation des données générées sont également disponibles dans ce panel. L'utilisateur pourra observer la carte se générer petit à petit et voir les drones se déplacer. L'utilisateur pourra également se déplacer sur la carte via la souris. Les données et les commandes seront transmises par des requêtes http au serveur, voir schéma ci-dessous.

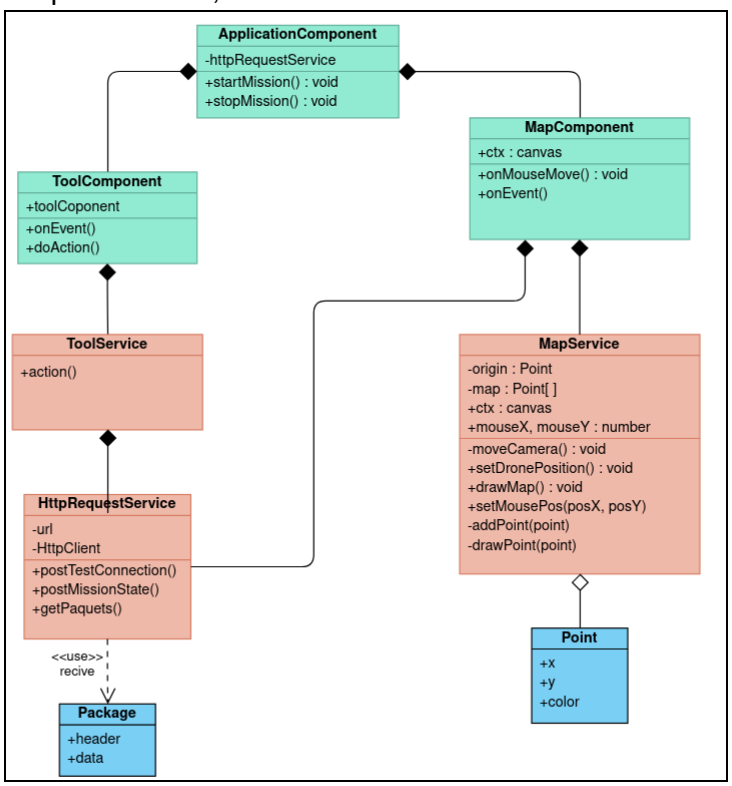


Image 3.5.2: diagramme de classe de l'interface utilisateur

En vert, nous avons les "components". Ils permettent d'interagir avec l'utilisateur et de générer des événements. Cet événement engendre une action qui sera transmise au bon service. Par exemple, le fait que l'utilisateur veut commencer la mission. Un bouton "start "est disponible dans le "tool component" quand l'utilisateur clique sur le bouton, cela génère un événement qui est envoyé vers le "tool service" (en rouge). Celui-ci comprend qu'il doit envoyer une requête au serveur. Pour ce faire, il utilise le service "httpRequestService" qui permet la communication avec le serveur.
Maintenant, voyons comment l'application web va recevoir les données générées par le système physique et le système simulé. Tout d'abord, le "map component" est programmé pour envoyer une requête "getPacket()" toute les 500ms. Cette requête permet de récupérer tous les paquets en attente sur le serveur. Un paquet est composé de deux parties, une partie "header " qui définit son type et une partie "data" qui contient les données. Par exemple, un paquet peut être de type point et contenir une position (x,y) dans la partie data. Un point est ensuite créé à partir du paquet et sera ajouté au "map service".
Concernant les requis, notre équipe s'engagent à ce que l'interface réponde au requis suivant :

Tableau 3.5 - Requis de l'interface utilisateur

Requis	***Points***	***Description***	***Solution***
Requis Fonctionnel #3	2 pts	Pour chaque drone, l'interface utilisateur doit montrer l'état des drones (attente, en mission, etc.), mises à jour avec une fréquence minimale de 1 Hz.	Nous utilisons un mat-grid pour afficher les états des drones qui se mettent à jour automatiquement. C'est le même pour simulation et drône physique.
Requis Fonctionnel #9	3 pts	Lors d'une mission, la position d'un drone dans la carte doit être affichée en continu.	Nous utilisons un mat-grid pour afficher les positions (x,y) des drones qui se mettent à jour automatiquement. C'est le même pour simulation et drône physique.
Requis Fonctionnel #10	4 pts	L'interface utilisateur pour l'opérateur doit être disponible comme service Web et visualisable sur plusieurs types d'appareils (PC, tablette, téléphone) via réseau. Au moins deux appareils doivent pouvoir être connectés en même temps pour la visualisation des données lors d'une mission. Le contrôle peut être restreint (ou non) à un seul appareil.	Nous avons testé en modifiant la taille de l'interface à ce qu'elle s'adapte bien pour la taille d'un téléphone ou d'une tablette.
Requis Conceptuel #1	5 pts	L'opérateur du système doit pouvoir vérifier que le système fonctionne correctement et avoir les informations nécessaires pour résoudre les problèmes. À cet effet, des logs de débogages doivent être disponibles en continu lors de chaque mission. Ces logs comprennent toutes les informations de débogage nécessaires au développement, dont au minimum : les lectures des senseurs (distance et position) de chaque drone (à minimum de 1 Hz) ainsi que les commandes envoyées par la station au sol. Les logs doivent être accessibles, sur demande seulement, dans l'interface utilisateur. Ces logs doivent être sauvegardés sur la station au sol à la fin de chaque mission. Il doit être possible d'ouvrir	Il est possible d'afficher les logs de débogage de chaque mission. C'est un pop-up accessible à partir d'un bouton de l'application. Les missions enregistrées gardent les logs de débogage qu'elles ont eu en cours de mission.

		les logs d'une mission précédente dans l'interface utilisateur afin de diagnostiquer les problèmes du système.	
Requis Conceptuel #4	5 pts	L'interface utilisateur doit être facile d'utilisation et lisible. Elle doit suivre les 10 heuristiques d'interface usager présentées ici : https://www.nngroup.com/articles/ten-usability-heuristics/.	Nous nous sommes assuré que l'interface est user-friendly.

Total de points: 19 points

3.6 Fonctionnement général (Q5.4)

Nous partons du principe que l'utilisateur de notre application possède une machine Ubuntu 20.04 et une installation Docker. Pour faire fonctionner le système, il suffit de suivre les étapes présentées au tableau 3.6

1. **Installation Ubuntu**

Les étapes d'installation de la distribution Linux «Ubuntu 20.04» peuvent être suivies sur le lien suivant:
https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview

2. **Installation Docker**

Les étapes d'installation du logiciel «Docker» sur Ubuntu peuvent être suivies sur le lien suivant:
https://docs.docker.com/engine/install/ubuntu/

3. **Navigation au travers de l'application**

Une fois dans l'onglet application web (annexe 3.6.4), à droite de la page, il faut choisir entre simulation et drone physique (annexe 3.6.1).

L'interface de la partie simulation (annexe 3.6.3) nous présente, à gauche de la page, les boutons start et stop qui permettent de partir de lancer la mission des drones et de les arrêter immédiatement.

L'interface de la partie drones physique nous présente, à droite de la page web (annexe 3.6.2) le bouton scanner qui permet d'obtenir les drones à proximité. Après un certain délai, on peut voir les drones trouvés dans le tableau en bas. Il est possible de cliquer sur les drones trouvés dans le tableau, puis sur le bouton «identifier» qui apparaît en dessous afin d'allumer la LED du drone ciblé.

Tableau 3.6 - Étape de fonctionnement

Étape	Commande
#1	git clone https://gitlab.com/polytechnique-montr-al/inf3995/20213/equipe-103/inf3995-103.git
#2	cd inf3995-103
#3	sudo docker-compose up –build

Tableau 3.7 - Étape de tests (dans le répertoire inf3995-103)

Étape	Commande
#1	python3 project/server/test.py

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

Chaque membre de l'équipe contribue au projet avec un nombre d'heures estimées à 10.5 heures par semaine (au maximum) et cela pendant douze semaines, pour un total de 630 heures-personnes. L'équipe sera composée d'un seul coordonnateur de projet et de quatre développeurs-analystes simultanément durant toute la durée du projet. Au courant du projet, il y aura rotation des rôles, mais les coûts hebdomadaires ne seront pas affectés. L'attribution des rôles peut être observée dans la section suivante (Section 4.2 - Planification des tâches).

Tableau 4.1 - tableaux des coûts

Membres de l'équipe	Coûts hebdomadaires	Coût totaux
coordonnateur de projet (1)	1522.50$	18 270$
développeurs-analystes (4)	5460$	65 520$

- Le taux horaire pour le coordinateur du projet est de 145 $ par heure.
- Le taux horaire pour les développeurs-analystes est de 130 $ par heure.
- Le coût total du projet est de 83 790 $.

4.2 Planification des tâches (Q11.2)

Le tableau suivant présente un plan détaillé des différentes tâches du projet. Les tâches sont classées par phase et possèdent un temps alloué ainsi qu'un membre assigné. Le projet a une durée estimée à 630 heures au total. Toutes ces informations se retrouvent dans le tableau 4.2 ci-dessous.

Tableau 4.2 - Calendrier détaillé

Phase	Tâche	Temps alloué	Membre assigné
Preliminary Design Review		240h	-
2021-09-04 au 2021-10-01	Rédaction de la documentation du projet	25h	Estefan
	Révision de la documentation du projet en équipe	10h	Tous
	Installation des outils nécessaires	20h	Tous
	Mise en place d'une application web Angular	30h	Tous
	Mise en place d'un serveur Python	30h	Tous
	Mise en place de la simulation sous Argos 3	30h	Noé et Brando
	Mise en place du répertoire GitLab	2h	Tous
	Mise en place du logiciel embarqué des drones	25h	John Andrew
	Création d'une image Docker	25h	Noé et Brando
	Mise en place de test unitaire (Angular, Python)	3h	Noé
	Requis R.F.1	20h	John Andrew Brando Noe
	Requis R.F.2	20h	Brando Noe
Critical Design Review		253h	-

2021-10-01 au 2021-10-29	Ajout des dependance dans l'image Docker	3h	Brando
	Rédaction de la documentation	10h	Tous
	Requis R.F.1 (suite)	5h	Tous
	Requis R.F.2 (suite)	7h	John Andrew Noe Brando
	Requis R.F.3	8h	Noe Brando
	Requis R.F.4	15h	Estefan
	Requis R.F.5	5h	Estefan Andrew
	Requis R.F.6	20h	John Noe
	Requis R.F.7	15h	John
	Requis R.F.8	15h	Noe Andrew Brando
	Requis R.F.9	20h	Estefan
	Requis R.F.10	20h	John Andrew
	Requis R.C.1	15h	Brando
	Requis R.C.2	10h	Brando Noe
	Requis R.C.3	10h	Estefan Andrew John
	Requis R.C.4	25h	Noe
	Requis R.C.5	5h	John Andrew
	Requis R.Q.1	10h	Tous
	Requis R.Q.2	35h	Tous

Readiness Review		137h	-
2021-10-29 au 2021-12-07	Ajout des dependance dans l'image Docker	5h	Brando Noe John
	Rédaction de la documentation	10h	Tous
	Requis R.C.5	15h	Noe Andrew
	Requis R.F.11	25h	John Andrew Noe Brando
	Requis R.F.12	5h	Noe Brando
	Requis R.F.13	20h	Estefan
	Requis R.F.17	17h	Estefan Andrew John
	Requis R.F.18	15h	John Noe
	Requis R.Q.1	10h	Tous
	Requis R.Q.2	15h	Tous

4.3 Calendrier de projet (Q11.2)

Tableau 4.3 - Résumé des dates cibles, des versions et de la terminaison des phases

Phase	**Version**	**Date de fin**
Preliminary Design Review	1.0.0	2021-10-01
Critical Design Review	2.0.0	2021-10-29

Readiness Review	3.0.0	2021-12-07

4.4 Ressources humaines du projet (Q11.2)

L'équipe de développement du drône est formée de cinq étudiants, chacun ayant des domaines d'expertise variés. Voici une brève présentation des membres de l'équipe:

Estefan Vega-Calcada
Il en est à sa troisième année en génie informatique à la Polytechnique de Montréal. Suite aux nombreux projets menés en baccalauréat et aux stages qu'il a effectués en entreprise, Estefan possède de l'expérience avec les langages de programmation C++, C#, XML, HTML/CSS et TypeScript. Il est également familier avec le framework Angular. Il possède un intérêt particulier pour le développement web.

Brando Tovar Oblitas
Brando en est à sa troisième année d'étude en génie informatique. Il possède de bonnes connaissances des langages C++, C, Python et Angular. Il possède un intérêt particulier pour les systèmes embarqués et l'intelligence artificielle. Brando est quelqu'un d'autonome et débrouillard qui est capable de s'adapter rapidement à de nouveaux projets auxquels il n'est pas familier.

Noé Berguin
Noé est actuellement en troisième année de génie informatique à la Polytechnique de Montréal. Il possède de solides compétences en programmation dans différents langages. Il a travaillé dans le domaine de la recherche pour polytechnique. Noé est quelqu'un d'autonome et d'assidu qui répond aux tâches qui lui sont confiées.

Andrew Abdo
Andrew est à sa troisième année d'étude en génie informatique. Il possède de bonnes connaissances des langages C/C++ et de Python qu'il a utilisés dans son stage. De plus, il est familier avec le framework Angular qu'il a utilisé lors de son projet de deuxième année. Il est très intéressé par le développement web et les systèmes embarqués.

John Maliha
John est à sa troisième année d'étude en génie informatique à Polytechnique Montréal. Il possède des connaissances avancées en C/C++ et javascript. De plus, celui-ci s'y connaît sur le framework d'angular et de NodeJS, framework qu'il a utilisé pour bâtir plusieurs sites web. John est passionné d'informatique, il apprécie le développement et les systèmes embarqués. Finalement, John est quelqu'un qui travaille fort individuellement ou en équipe.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Afin d'assurer la qualité des biens livrables, l'équipe propose les idées suivantes:

Pour assurer **la qualité des fonctionnalités**, celles-ci seront testées continuellement au long du projet. Chaque composante possède ses propres tests unitaires afin de s'assurer qu'elle fonctionne comme prévu. La phase de tests s'avérera sans aucun doute la plus cruciale afin de détecter les comportements non-désirés du système. Ceux-ci seront disponibles dans un dossier à part.

Pour assurer la **qualité du code**, l'intégration des différentes fonctionnalités sera faite en continu. Les fonctionnalités seront ajoutées sur la branche principale du Gitlab du projet seulement après avoir été testées avec succès. L'intégration en continu permettra d'éviter l'introduction de bogues ou de comportements inattendus dû au fusionnement du code des différentes composantes.

Une révision hebdomadaire du code écrit sera effectuée afin d'éviter une relecture trop massive avant la remise. Une relecture fréquente favorise également la détection d'erreurs. Chaque semaine, l'équipe consacre environ une heure de son temps à un testing général du système afin de s'assurer qu'aucune ancienne fonctionnalité n'ait été brisée par l'introduction du nouveau code.

En **cas de bogue**, le coordonnateur du projet se chargera d'ouvrir une issue sur Gitlab et de l'assigner à un membre de l'équipe. Si nécessaire, celle-ci pourra être travaillée par plusieurs membres de l'équipe sur une branche différente afin d'essayer de la résoudre le plus rapidement possible.

Enfin, **une révision globale et en profondeur** sera effectuée par l'ensemble de l'équipe la semaine précédant la remise des différents livrables dans le but de repérer les dernières failles du système.

5.2 Gestion de risque (Q11.3)

- **Importance**: l'importance du risque est une échelle (allant de 1 à 10) basée sur la probabilité d'occurrence et l'impact du risque
- **Description**: description textuelle du risque et des problèmes envisagés
- **Impact**: l'impact du risque est une échelle qui décrit les dommages que peuvent avoir le risque sur le projet
    - C - critique (affecte le projet en entier)
    - E - élevé (conséquences graves, affecte les fonctionnalités principales du projet)
    - M - moyen (conséquences modérées, mais peut être gérer)
    - F - faible (peu de conséquences)
- **Critère(s)**: aspects du projet affectés par le risque
- **Solution(s) possible(s)**: mesures envisagées pour contrôler le risque

Nouvelles technologies				
Ampleur	**Description**	**Impact**	**Critère(s)**	**Solution(s) possible(s)**
5	L'équipe n'est pas familière avec toutes les nouvelles technologies nécessaires à l'accomplissement du projet. Des nouveaux langages de programmation et de nouveaux outils (ARGOS, Firmware Crazyflie) ont été introduits. La fréquence d'occurrence de ce risque diminue avec le temps, mais il est a considéré, particulièrement dans les débuts	E	Temps: l'équipe pourrait passer beaucoup de temps afin de se familiariser avec les nouvelles technologies Bogues: Le manque d'expertise pourrait augmenter le nombre de bogues introduit dans le système Maintenance du code: demande plus d'effort et de temps afin de garder un code de bonne qualité	Il sera nécessaire pour l'équipe de s'y prendre d'avance afin de se laisser un maximum de temps pour se familiariser avec les nouvelles technologies Les membres devront favoriser le partage de connaissances afin d'économiser un maximum de temps possible dans l'apprentissage

Manque de communication				
Ampleur	**Description**	**Impact**	**Critère(s)**	**Solution(s) possible(s)**
4	Le manque de communication dans l'équipe est un risque à considérer, car il crée des incohérences entre les multiples éléments du projet. Si les membres travaillent sur les différentes fonctionnalités de manière indépendante, le développement du système pourrait être affecté, car les membres de l'équipe peuvent percevoir la structure du code ou les tâches de manière différente.	M	Diminution de la qualité des biens livrables Perte de temps Tensions entre les membres de l'équipe qui sont en désaccord	Réunion hebdomadaire Communiquer avec l'équipe avant d'amorcer une fonctionnalité Parvenir à un consensus

Charge de travail d'une fonctionnalité				
Ampleur	**Description**	**Impact**	**Critère(s)**	**Solution(s) possible(s)**
6	Il se pourrait que l'équipe sous-estime le temps nécessaire à la complétion d'une fonctionnalité	C	Qualité du livrable Fonctionnalité incomplète Non respect du contrat	Estimé à la hausse le temps nécessaire à la complétion d'une fonctionnalité. Prioriser la complétion des tâches les plus complexes

Zone de confort				
Ampleur	**Description**	**Impact**	**Critère(s)**	**Solution(s) possible(s)**
7	Les membres de l'équipe pourraient essayer de rester dans leur zone de confort, c'est-à-dire d'uniquement faire les tâches dans lesquelles ils sont habiles. Le cas échéant, il se pourrait que le projet ait trop évolué et que les membres n'ayant pas exploré les différentes parties du projet ne puissent plus aider les autres dans leurs tâches par manque de connaissances.	C	Manque de temps dû au manque de personnes étant habile à travailler sur une composante du système	Favoriser le partage de connaissance Effectuer une rotation des rôles, des tâches et des composantes travaillées par les membres de l'équipe Jumeler les membres stratégiquement afin de favoriser le partage de connaissance et l'apprentissage (jumeler un membre habile et un membre moins habile)

5.3 Tests (Q4.4)

Serveur
Les tests unitaires sur le serveur ont été implémentés en utilisant unittest. Pour en voir les résultats:

coverage run test.py (pour voir si les tests passent)
coverage report -m (pour voir le pourcentage de couverture)

Simulation
Pour la simulation, nous avons testé en augmentant le nombre de drones et en regardant les résultats. Nous avons aussi donné une seule direction de déplacement pour voir si le drone évitait bien les murs dans chaque direction. La communication serveur argos a été implémentée de manière indépendante de la simulation pour tester son bon fonctionnement.

Embarqué
Pour la partie embarqué, la communication entre la crazyradio a été testé de manière indépendante du serveur sur un fichier python à part. Ainsi nous nous somme assuré que la communication fonctionnait bien avant d'intégrer cette composante au serveur. Pour les algorithmes internes des drones, une connexion avec une manette de PS4 a

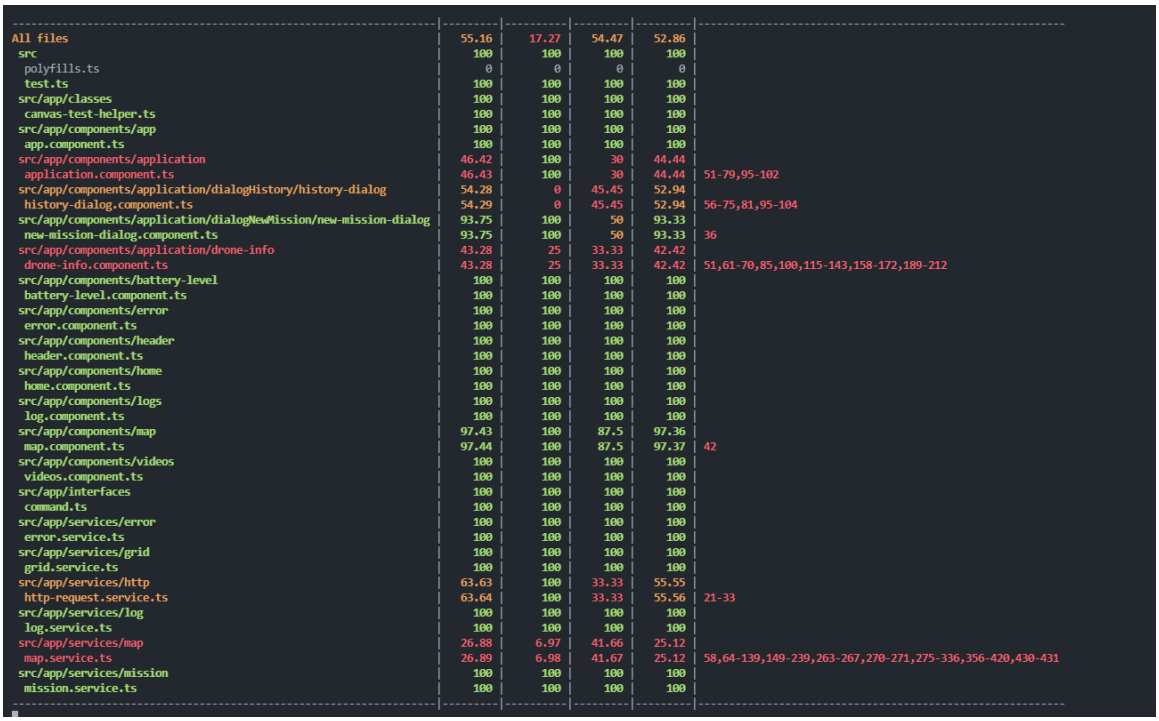
été implémentée afin de tester plus facilement la génération de la map et aussi l'algorithme de retour à la base.

Client

La partie client a été testé à l'aide de Karma et Jasmine. Pour en voir les résultats:

npm test (pour voir si les tests passent)

npm run coverage (pour voir le pourcentage de couverture)



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	55.16	17.27	54.47	52.86	
src	100	100	100	100	
polyfills.ts	0	0	0	0	
test.ts	100	100	100	100	
src/app/classes	100	100	100	100	
canvas-test-helper.ts	100	100	100	100	
src/app/components/app	100	100	100	100	
app.component.ts	100	100	100	100	
src/app/components/application	46.42	100	30	44.44	
application.component.ts	46.43	100	30	44.44	51-79,95-102
src/app/components/application/dialogHistory/history-dialog	54.28	0	45.45	52.94	
history-dialog.component.ts	54.29	0	45.45	52.94	56-75,81,95-104
src/app/components/application/dialogNewMission/new-mission-dialog	93.75	100	50	93.33	
new-mission-dialog.component.ts	93.75	100	50	93.33	36
src/app/components/application/drone-info	43.28	25	33.33	42.42	
drone-info.component.ts	43.28	25	33.33	42.42	51,61-70,85,100,115-143,158-172,189-212
src/app/components/battery-level	100	100	100	100	
battery-level.component.ts	100	100	100	100	
src/app/components/error	100	100	100	100	
error.component.ts	100	100	100	100	
src/app/components/header	100	100	100	100	
header.component.ts	100	100	100	100	
src/app/components/home	100	100	100	100	
home.component.ts	100	100	100	100	
src/app/components/logs	100	100	100	100	
log.component.ts	100	100	100	100	
src/app/components/map	97.43	100	87.5	97.36	
map.component.ts	97.44	100	87.5	97.37	42
src/app/components/videos	100	100	100	100	
videos.component.ts	100	100	100	100	
src/app/interfaces	100	100	100	100	
command.ts	100	100	100	100	
src/app/services/error	100	100	100	100	
error.service.ts	100	100	100	100	
src/app/services/grid	100	100	100	100	
grid.service.ts	100	100	100	100	
src/app/services/http	63.63	100	33.33	55.55	
http-request.service.ts	63.64	100	33.33	55.56	21-33
src/app/services/log	100	100	100	100	
log.service.ts	100	100	100	100	
src/app/services/map	26.88	6.97	41.66	25.12	
map.service.ts	26.89	6.98	41.67	25.12	58,64-139,149-239,263-267,270-271,275-336,356-420,430-431
src/app/services/mission	100	100	100	100	
mission.service.ts	100	100	100	100	

5.4 Gestion de configuration (Q4)

La gestion du code s'effectue principalement sur Gitlab. Le répertoire créé par les chargés de laboratoire sera utilisé comme la version officielle, fonctionnelle la plus avancée du projet. Celui-ci contiendra le code source du projet ainsi qu'un fichier Dockerfile qui permettra aux contractants d'installer un environnement complet à partir d'une image Docker. Dans ce répertoire, on y retrouve quatres dossiers qui séparent les différentes composantes du projet, soit: simulation, serveur, client et drone. Le dossier simulation contient tout le code en lien avec les simulations des drônes, le dossier serveur contient uniquement le code en lien avec le server, le dossier client contient l'interface web et le dossier drone contient tout le code embarqué pour les drônes «Bitcraze Crazyflie 2.1». Éventuellement, l'équipe se servira d'un autre dossier nommé «test» afin de tester les différentes fonctionnalités. Celui-ci contiendra majoritairement les tests unitaires et des fonctionnalités en développement qui pourraient être problématiques. Chaque requis possédera un ou plusieurs tests unitaires associés, qui permettront d'assurer le bon fonctionnement des composantes, à moins d'être jugé non-pertinent. Les fichiers de données seront regroupés par composantes, autrement

dit chaque composante du système possèdera son propre dossier de données. Ceci évitera d'accumuler une grande banque de fichiers de données non-structurée. La documentation de conception (rapport technique) sera disponible dans le dossier principal du projet et contiendra toute l'information relative à la structure et au fonctionnement du système.

En cas de bogue, le code sera exporté vers une autre branche sur Gitlab. La convention pour nommer la nouvelle branche sera «*issue - [le nom du bogue]*». Une tâche, du même nom, sera également créée sur le Gitlab du projet afin de garder une trace des problèmes résolus. Comme mentionné plus tôt dans la section « 5.1 Contrôle de qualité», le coordonnateur de projet sera chargé d'assigner la tâche à un des membres de l'équipe. L'assignation de la tâche sera faite en fonction de l'expertise des membres ou sur une base volontaire.

La version du projet aura la forme suivante: *x.y.z*, où *x* désigne la version majeure, *y* la version mineure et *z* les *patchs*. À chaque remise, la version majeure sera incrémentée, à chaque nouvelle fonctionnalité ajoutée la version mineure sera incrémentée et à chaque semaine nous comptabiliserons le nombre de d'ajustement effectués et nous incrémenterons de manière concordante les *patchs*. Le z prendra également en compte le nombre de *fix* effectués dans la branche de debug et l'ajout des différents tests des fonctionnalités. Chaque membre de l'équipe sera chargé de garder des traces du nombre de fonctionnalités et d'ajustements qu'il a effectué au cours de la semaine. Par la suite, le coordonnateur du projet corrigera la version du système.

Afin d'avantager une communication rapide et simple, l'équipe utilisera l'application Discord sur laquelle nous avons mis en place un serveur possédant différents canaux de communication. Nous avons également mis en place des alertes personnalisées et des rappels afin de permettre aux membres de l'équipe de mieux s'organiser.

À chaque semaine, un rapport d'avancement sera livré suite à une discussion d'équipe qui portera sur les tâches accomplies et à accomplir. Nous planifierons les prochaines étapes du projet en fonction du *feedback* de chaque membre. La création et l'assignation des tâches, l'ajustement de la documentation et la gestion des branches Gitlab du projet sera faite à ce moment.

5.5 Déroulement du projet (Q2.5)

Nous avons implémenté une feuille de temps la semaine avant la remise du CDR pour voir ce que chaque membre de l'équipe contribue et nous permettre de savoir lorsqu'il y a un problème où un des membres reste bloqué pour que quelqu'un d'autre puisse lui venir en aide. Grâce à ceci, nous avons réalisé que nous avions trop de tâches à compléter durant la dernière semaine.

Chaque membre de l'équipe travaillait de façon indépendante. Ceci a créé beaucoup de conflits lorsque nous avons dû merge toutes nos branches et lors de l'intégration des fonctionnalités.

Nous avons également rencontré beaucoup de problèmes avec les drones physiques au niveau de la réception des données. La communication était perturbée par des interférences générées par les drones des autres équipes ce qui perturbait nos phases de test.

Dans l'ensemble, nous sommes satisfait du travail fourni par chaque membre. Chacun a fait le maximum de ce qu'il a pu. La partie simulation s'est déroulée comme prévu et le résultat est excellent. Le serveur fonctionne et communique avec la base de données.

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

L'exploration du côté simulation fonctionne très bien en faisant un random walk avec l'évitement d'obstacle. Du côté drônes physiques, l'exploration se déroule bien avec l'évitement d'obstacle. La génération de la carte sur notre interface fonctionne. Nous sommes en mesure de récupérer toutes les données des drônes. Le retour à la base fonctionne très bien en prenant le chemin le plus court possible.

Il est possible de sauvegarder et de supprimer les missions. Nous pouvons également voir les logs de débogage des missions. Les commandes qu'on envoie du serveur aux drônes ainsi que les données reçues par les drônes sont affichées sur notre interface (états des drônes avec fréquence minimale de 1 Hz).

L'environnement virtuel de la simulation peut être généré aléatoirement avec un minimum de 3 murs.

<u>Liste des requis complétés:</u>

- R.F.1
- R.F.2
- R.F.3
- R.F.4
- R.F.5
- R.F.6
- R.F.7
- R.F.8
- R.F.9
- R.F.10
- R.F.11
- R.F.12
- R.F.13
- R.F.17
- R.F.18
- R.C.1
- R.C.2
- R.C.3
- R.C.4
- R.C.5
- R.Q.1
- R.Q.2

Capture d'écran de l'exploration en simulation

7. Travaux futurs et recommandations (Q3.5)

[Qu'est-ce qui reste à compléter sur votre système? Recommandations et possibles extensions du système.]

Nous avons réussi à compléter tous les requis que nous nous sommes fixés. Nous recommandons pour des travaux futurs de s'assurer de la communication bidirectionnelle entre les drônes physiques et l'application dès le départ. Quant à l'interface, il serait intéressant d'ajouter une page initiale dans l'application qui permettrait de sélectionner une mission dans la base de sélectionner directement le mode de la mission (simulation ou physique) avant d'arriver sur la carte. Lorsque l'utilisateur sauvegarde la mission, il est redirigé vers cette page de lancement de mission.

8. Apprentissage continu (Q12)

Andrew Abdo

Les lacunes que je peux identifier dans mes savoirs et savoir-faire pour le projet serait la gestion de temps et l'apprentissage autonome. C'est difficile de bien répartir son temps mais grâce à un agenda et des objectifs que je me fixe, je réussi à mieux gérer mon temps et avoir de la place pour ma vie personnelle et le travail tout en restant à mes affaires à l'école. Pour ce qui de l'apprentissage autonome, c'est une question de constante pratique et d'amélioration. Par exemple, durant mon stage, j'ai eu beaucoup de pratique puisque j'ai dû développer un algorithme et une application seul, mais d'après moi c'est quelque chose qui est constamment travaillé. Aussi, le fait de devoir travailler en équipe faisait en sorte qu'il était nécessaire de bien communiquer. J'étais plus réservé au départ pour partager mes idées et opinions. Avec le temps, je suis devenu alaise et cela m'a permis de mieux communiquer avec les membres de mon

équipe. Cet aspect aurait pu être amélioré si, dès le départ, nous avions fait quelques activités pour apprendre à se connaître.

Noé Berguin

Au cours de mon cursus universitaire à polytechnique j'ai dû, à de nombreuses reprises, travailler en équipe avec des gens que je ne connaissais pas . Travailler avec d'autres étudiants m'a permis d'en apprendre davantage sur mes lacunes.

Tout d'abord j'ai des difficultés avec la communication. Transmettre mes avancements et les problèmes que je rencontre. Il est important que toutes l'équipe puisse avoir une vue d'ensemble du projet et cela passe par la communication entre les membres. Ma seconde lacune et la gestion du temps. gérer mon horaire afin de garder un bon rythme de travail dans le projet sans délaisser mes autres cours, ma vie sociale, et mon travail à temps partiel. Ma dernière lacune réside dans la coopération. Je travaille généralement de façon autonome une tâche à la fois.

Pour répondre à mes lacunes et ne pas compromettre l'intégrité du projet j'ai pris les devant en me fixant certaines règles. je me suis fixé un emploi du temps personnel pour gérer mon temps. J'ai décidé de parler de mon avancement un maximum avec mes coéquipiers. En équipe, nous avons établi plusieurs périodes de réunion et de travail en équipe ce qui va probablement m'obliger à ne pas travailler de façon autonome.

John Maliha

Au cours de mon parcours universitaire, j'ai été poussé à travailler avec plusieurs personnes différentes. Ceci m'a permis de réaliser mes lacunes tant dans ma méthodologie de travail que mes savoirs.

En effet, j'ai remarqué une lacune sur la communication surtout avec des nouveaux coéquipiers que je ne connais pas. Par conséquent, je cherche à éviter de causer des conflits au sein de l'équipe donc, je ne défends pas mes idées et accepte celle des autres. Aussi, je chercher a trouver des solutions seuls aux différents problèmes que je rencontre dans le développement des features qui me sont assignées. Souvent, poser la question me permettrait de sauver du temps. Ma seconde lacune, consiste à la gestion du temps. En effet, au cours de la session j'ai eu de la misère à gérer mon horaire afin de ne négliger ni le projet ni mes autres cours.

Dans l'objectif de corriger mes lacunes, je dois commencer à planifier mon horaire. Ce qui est à faire pour quand et le moment que je planifie de le faire. Aussi, au fur et à mesure que le projet avance, j'ai appris à connaître mon équipe et je suis devenu alaise alaise ce qui m'a permis de défendre mes idées et de communiquer plus avec eux. Aussi, je me suis informé sur ce que chacun a fait. Nous avons organisé plusieurs rencontres d'équipes qui nous ont permis de mieux gérer l'équipe et le travail à faire.

Brando Tovar

Durant toute mon parcours universitaire j'ai eu l'occasion de faire beaucoup de travaux en équipes et j'ai pu constater des lacunes sur ma manière de travailler en équipe. J'ai pris l'habitude de faire mes parties de travail de manière autonome. De ce fait, mon travail coopératif n'est pas optimal. Ma zone de confort est de réaliser la partie du travail qui m'est assigné et, donc, je ne suis pas le meilleur communicateur d'une équipe. J'ai aussi des lacunes en ce qui concerne la gestion du travail à faire. J'ai parfois du mal à organiser mon travail et à me faire un plan de travail. De plus, lorsque vient le temps de prendre l'initiative en ce qui concerne la gestion d'équipe, je suis rarement le premier à le faire.
Afin de remédier le plus possible à ces lacunes, j'ai essayé de me donner un plan de travail avec les tâches que j'ai à faire. Je me suis informé sur la partie de travail de chacun afin de pouvoir donner des suggestions et ainsi m'impliquer un peu plus dans la gestion d'équipe. J'ai aussi participé à la conception d'une liste de tout le travail à faire pour le projet, afin d'avoir une vue d'ensemble de tout ce qui a à faire par tout le monde. J'aurais pu améliorer davantage mes lacunes en suivant davantage les plans de travail conçus et aussi en communiquant davantage avec mes coéquipiers. J'aurais pu aussi demander à faire plus de rencontres pour m'assurer d'être à jour avec tout le monde.

Estefan Vega Calcada
La lacune la plus importante que j'ai rencontré au courant de ce projet était la gestion du temps. J'ai eu de la difficulté à balancer mes cours avec le projet, ce qui a fait que je n'étais pas très constant sur l'avancement de mes tâches. Une semaine, je pouvais mettre énormément de temps sur mes tâches et une autre, il arrivait que je n'avance presque pas. Une autre lacune est celle de la coopération. Je travaillais principalement sur l'interface utilisateur ce qui a fait en sorte que j'étais généralement à l'écart de l'équipe. Cependant, il arrivait des moments où nous devions tous travailler ensemble et je n'étais pas trop habitué à cela. J'étais beaucoup moins productif ce qui était décourageant.
Pour remédier à ma première lacune, je me suis imposé un horaire fixe en fonction des jours de la semaine. Je consacrais des journées entières spécifiques de la semaine à mes autres cours et d'autres journées au projet. Pour la deuxième lacune, je me suis assuré de compléter mes tâches préalablement, avant de me rendre à la l'école pour les tâches d'équipes. Ainsi, je pouvais entièrement consacrer mon attention sur les tâches communes. Je crois que j'aurais pu améliorer davantage mes lacunes en allant plus souvent à l'école plutôt qu'en travaillant à la maison.

9. Conclusion (Q3.6)

Pour conclure, nous sommes fiers de ce que nous avons accompli et avons trouvé cette application très intéressante. Notre planification des tâches et l'organisation du projet n'était pas tout à fait adéquate dû aux nombreux imprévus que nous avons rencontrés cette session. Notre démarche de conception était fonctionnelle et efficace, mais elle ne correspond pas à ce qui était prévu. En ce qui concerne nos hypothèses de départ, nous avions bel et bien raison de considérer que le drone explore le terrain sans effectuer de rotation. Cela nous a grandement simplifié la tâche. Le reste des hypothèses se sont avérées vraies, mais peu pertinentes à l'avancement du projet.

10. Références (Q3.2)

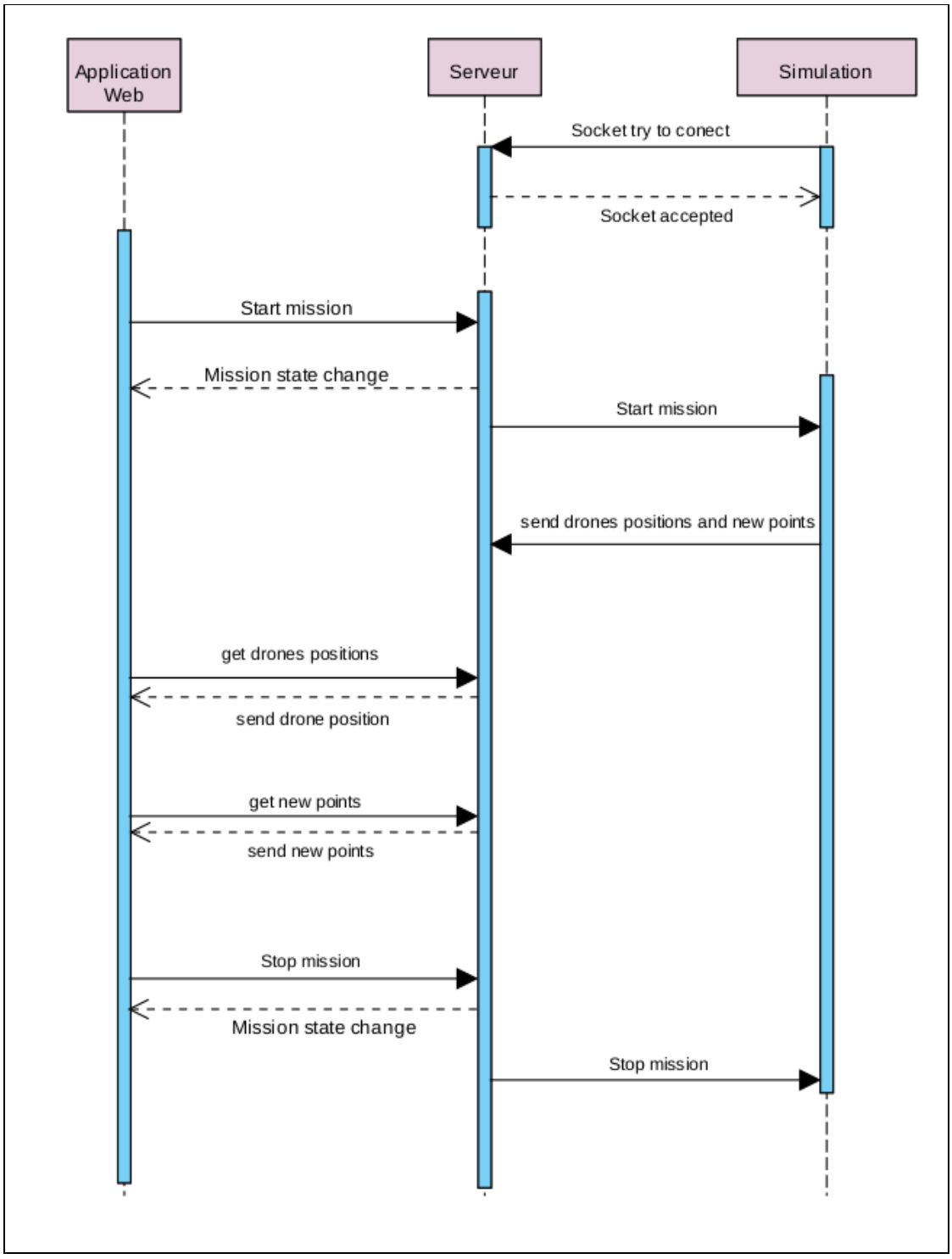
[Liste des références auxquelles réfère ce document. Un site web est une référence tout à fait valable.]

https://en.wikipedia.org/wiki/Software_versioning
https://www.indellient.com/blog/how-to-dockerize-an-angular-application-with-nginx/

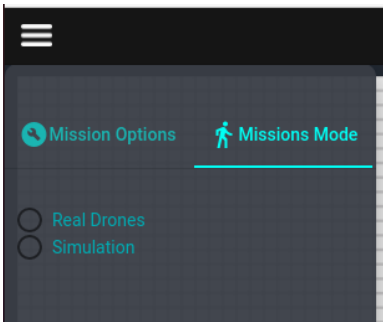
Installation Ubuntu: https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview

ANNEXES

Annexe 3.4.1: Diagramme de séquence de la simulation (Vue d'ensemble)



Annexe 3.6.1: choix du mode de de l'application



Annexe 3.6.2 Bouton scan avec la liste des drones trouvés

Annexe 3.6.3: boutons start et stop mode simulation

Annexe 3.6.4: onglet application