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. H2022-INF3995 du département GIGL.

Conception d'un système aérien d'exploration

Équipe No < ***103*** >

Marc-Olivier Lemieux
Aram Zand
Sanmar Simon
Zakaria Diabi
Sinda Drira

18 mars 2022

Table des Matières

1. Vue d'ensemble du projet

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

Le but du projet est de concevoir un système d'exploration qui permet de gérer un essaim de drones. Ces derniers doivent explorer et cartographier un espace de 100 m2. Ainsi, il est nécessaire de développer une application Web qui communique avec les drones pour recevoir des informations et envoyer des commandes. L'interface de ce logiciel s'occupe de recevoir les données des drones afin d'afficher en direct les renseignements sur ces derniers et d'illustrer la cartographie de la pièce dans laquelle ils se déplacent. De plus, elle permet d'envoyer des commandes spécifiques aux drones (Par exemple : décoller et atterrir) à partir du serveur. En effet, c'est la partie serveur de l'application qui gère les différentes communications entre le logiciel et les drones physiques. De plus, il faut se servir du simulateur ARGoS pour tester le fonctionnement du code dans une simulation avant de l'implémenter dans les robots. Ceci permet de s'assurer que les fonctionnalités se déroulent comme il faut, sans compromettre les drones.

Ainsi, les principaux objectifs généraux de ce projet sont :

- Les drones parcourent un espace de 100 m2 et dévient tous les obstacles.
- L'interface recueille et affiche des données spécifiques concernant les drones.
- L'utilisateur est capable de commander les drones à partir de l'application.

Le système à concevoir sera composé de trois éléments majeurs :

- La station au sol : Le logiciel composé du serveur et de l'interface qui communique avec les drones grâce à la Bitcraze Crazyradio PA.
- La simulation : Le logiciel ARGoS qui permet de simuler les comportements des drones avec le code introduit.
- L'embarquée : Le code qui sera implémenté sur les microcontrôleurs des drones.

Les biens livrables attendus sont :

- Le PDR (4 février 2022) : Remise d'un prototype préliminaire du système. Ceci consiste à la présentation générale de la solution, ainsi que l'implémentation des fonctionnalités de bases, tel qu'un prototype

d'interface, l'identification des drones physiques et le décollage/atterrissage des drones physiques et simulés.

- La CDR (18 mars 2022) : Remise d'un système partiellement fonctionnel, c'est-à-dire la mise en fonction des capacités essentielles au projet, comme l'exploration et l'évitement d'obstacles dans un environnement, la communication des informations entre les drones et la station au sol et l'affichage de ses données sur l'interface. Cette remise abordera aussi les précisions et modifications apportées au projet depuis le PDR, acquises lors du développement.

- La RR (14 avril 2022) : Remise du système complet. Tous les requis sélectionnés devront être livrés, et le système devra être totalement fonctionnel, prêt à être livré au client.

1.2 Hypothèse et contraintes (Q3.1)

Plusieurs hypothèses sont posées pour le projet. Pour commencer, on propose que la partie backend du projet comporte trois serveurs afin de mieux gérer les différents types de communications entre les composantes du système. Toute cette partie serveur sera majoritairement réalisée en python pour assurer un développement homogène. En d'autres mots, éviter les confusions engendrées par l'utilisation de différents langages de programmation. Ensuite, on suppose que l'interface interagit directement seulement avec le serveur principal (serveur interface Web). Ce dernier communique avec les deux autres serveurs (serveur CrazyFlie et serveur ARGoS) pour acheminer les informations nécessaires. De plus, on juge qu'il sera possible d'utiliser les fonctions présentes dans l'API python de Bitcraze pour établir rapidement la connexion entre les drones et l'application Web.

En ce qui concerne les contraintes, il est évident qu'il n'existe pas beaucoup de contraintes techniques étant donné qu'on nous donne la liberté de choisir les requis et les technologies à utiliser. Cependant, pour exécuter les simulations ARGoS, il est nécessaire d'utiliser le conteneurs Docker. Ceci peut être considéré comme une contrainte technique puisque la plupart des membres de l'équipe n'ont jamais utilisé cette plateforme. De plus, la limite de temps de travail pourrait s'avérer contraignante. En effet, certaines personnes sont moins performantes que d'autres. Par conséquent, limiter chacun des membres par 630 heures-personnes pourrait sembler injuste puisque certaines personnes nécessitent plus de temps que d'autres. En dernier, les imprévus durant le processus de développement pourraient présenter une contrainte. En d'autres mots, tout changement ou ajustement apporté dans le développement du

système pourrait causer une perturbation. Cependant, l'équipe doit s'adapter et faire de son mieux pour régler toutes ces contraintes.

1.3 Biens livrables du projet (Q4.1)

I. Preliminary Design Review (PDR):

Ce livrable consiste d'un prototype préliminaire présentant les composantes matérielles et logicielles du projet. Ainsi, le serveur Web sur la station au sol, les deux drones physiques et la simulation ARGoS doivent être fonctionnels et connectés afin de développer les requis nécessaires à cette remise.

En ce qui concerne le Serveur web sur la station au sol et les drones physiques, il faut réaliser le requis suivant :

- R.F.1 : Faire clignoter au moins une des DELs du drône avec la commande 'identifier' dans l'interface utilisateur.

Donc, il faut développer l'application Web (Interface et Serveur) et établir la connexion entre le serveur web et le logiciel embarqué du Crazyflie.

Ensuite, pour la simulation ARGoS avec deux drones qui suivent un parcours quelconque, le requis suivant est à satisfaire :

- R.F.2 Faire décoller et atterrir un drone sur la simulation à l'aide des commandes 'Lancer la mission' et 'terminer la mission' à partir de l'interface utilisateur.

Alors, il est nécessaire d'établir la connexion entre le serveur web et la simulation ARGos3.

Remise : Le vendredi 4 février 2022 à 17 h

II. Critical Design Review (CDR):

Ce livrable consiste d'une structure complète du projet à développer. Il présente un système partiellement fonctionnel. C'est-à-dire, Plusieurs requis sont déployés et prêts à être utilisés. Parmi ces derniers on retrouve :

- Décoller et atterrir les drones à partir des commandes de l'interface.
- Envoyer en temps réel l'état de chacun des drones à l'application web avec une fréquence minimale de 1 Hz.
- Implémenter la séquence de déplacement à la suite du décollage des drones physiques et simulés.
- Implémenter la détection et l'évitement des obstacles dans les drones
- Déployer et rendre l'interface utilisateur responsive aux différents appareils.
- Envoyer en continu des logs de débogage au serveur et les rendre accessibles sur l'interface utilisateur.
- Créer un prototype de cartographie de l'espace grâce à la collecte des données continue des drones.

Ainsi, il faut compléter les fonctionnalités nécessaires afin de satisfaire les requis demandés.

Remise : Le vendredi 18 mars 2022 à 17 h

III. Readiness Review (RR):

Ce livrable consiste à présenter le projet achevé. En d'autres mots, remettre le système complet et fonctionnel. En fait, il est question de soumettre une application complète capable d'envoyer toutes les commandes possibles sur les drones physiques et simulés. De plus, l'interface Web doit afficher les données nécessaires sur les drones et la cartographie des espaces explorés. Donc, le projet doit répondre à tous les requis demandés.

Alors, pour finaliser le système, il faut respecter les caractéristiques suivantes entre autres:

- Introduire la commande de retour à la base pour les drones
- Afficher la position et l'états des drones en temps réel

- Afficher une carte globale combinant tous les espaces explorés par les drones
- Ajouter l'option de zone de sécurité
- Implémenter la mise à jour du système

Ainsi, il faut implémenter et/ou perfectionner toutes les fonctionnalités et les tests du système afin de finaliser le projet pour la remise finale.

Remise : Le mardi 19 avril 2022 à 12h30

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Le travail se fait en équipe de 5 dont le but commun est la réussite du projet.

Il s'agit de trois rôles principaux dans l'équipe :

Développeur embarqué, développeur simulation et développeur Web.

La partie simulation est composée de trois parties : le serveur, l'interface et le contrôleur des drones simulés. Le développeur Web s'occupe de l'interface et le serveur et la connexion entre les deux.

La simulation virtuel des drones et le contrôleur des drones simulés sont faits par le développeur simulation,

Le développeur embarqué fait la partie relative aux systèmes embarqués étant le comportement des drones. Le développeur embarqué doit avoir certaines connaissances en électronique.

Dépendamment des tâches, chaque membre de l'équipe possède un des trois rôles principaux.

La répartition des tâches est de façon équitable. Chaque membre a la responsabilité d'être présent aux rencontres et de travailler régulièrement afin de s'acquitter de ses tâches avant la date limite. Dans le cas où la présence d'un leader est nécessaire, un des membres s'occupe de ce rôle et tout le monde doit le suivre.

2.2 Entente contractuelle (Q11.1)

Le type d'entente contractuelle proposée pour ce projet est un contrat de livraison clef en main. Ce type de contrats est adéquat pour ce projet car la liste des requis est connue, complète et précise. La liste des requis ne sera donc, probablement, pas modifiée au cours du projet. Pour cette raison, il est possible de prévoir le temps nécessaire pour la réalisation du projet et prévoir aussi le coût fixe pour la réalisation du projet. Les coûts du projet sont estimés dans ce présent rapport à la section 4.1. La raison pour laquelle nous avons choisi ce type de contrat est d'avoir une bonne vision d'effort que chacun doit faire et de pouvoir planifier pour la réussite du projet.

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)

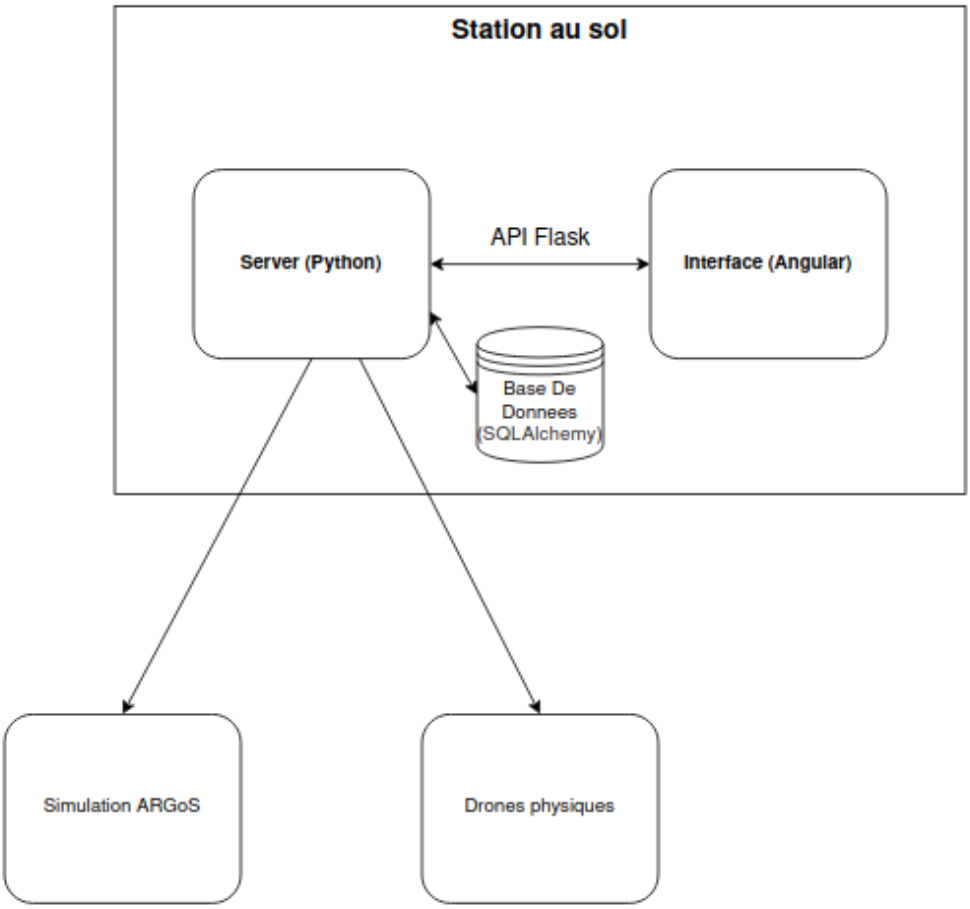

Figure 1: Architecture générale du logicielle

Pour décrire l'architecture et le fonctionnement du système en général, il est essentiel de connaître les différentes couches le composant. D'un point de vue très simplifié, nous pouvons définir l'architecture en trois parties principales, soit la station au sol, la partie embarquée, ainsi que la partie simulée. Explorons brièvement chacune des ces composantes. Tout d'abord, la station au sol est divisée en plusieurs sous-modules, ayant chacun un rôle clé. En effet, l'utilisateur aura accès à la partie interface, qui affichera toutes les informations nécessaires, tel que les informations sur l'environnement, le statut des drones et les boutons permettant d'interagir avec ceux-ci, pour, par exemple, démarrer une mission. Cette interface communiquera avec la partie *backend*, soit le serveur, à l'aide de l'API Flask. Comme le serveur est programmé avec le langage Python, et que Flask est une librairie bien connue, il est naturel d'utiliser cette technologie pour le projet. Ici, le serveur, qui est en fait séparé en trois instances distinctes (voir partie 3.2), servira à traiter les informations entrées par l'utilisateur, et afficher les données nécessaires. Pour stocker et utiliser ces données efficacement, nous implémenterons une base de données SQL. Comme l'équipe possède une bonne connaissance de ce type de base de données, il est logique de mettre

cette expertise en profit pour sauver du temps. Bref, l'architecture de la station au sol peut être décrite par une architecture trois niveaux, où l'interface représente la présentation, la partie serveur représente la logique, puis, la base de données fait office de stockage. Pour ce projet, ce type d'architecture est idéal, car il permet facilement l'évolutivité du programme dans l'éventualité où l'Agence voudrait implémenter plus de drones.

Ensuite, la deuxième grande partie est le logiciel embarqué, soit l'élément concernant les drones physiques. Il est important de noter que le code sera bel et bien embarqué sur les drones physiques, c'est-à-dire que la station au sol enverra seulement les commandes de bases, tel que débuter et arrêter une mission, vers le drones. Ceci est essentiel, car les délais de communications seront non-négligeables lors de déploiement spatial, les drones doivent donc être en mesure d'évaluer leur environnement et remplir leur mission de manière autonome. Les commandes seront envoyées de l'interface vers le serveur, qui ensuite se connectera avec les drones physiques. Ce processus est décrit plus en détail dans la section 3.2 et 3.3.

Finalement, la dernière composante représente la partie simulation. Avant d'effectuer des vols sur les drones physiques, il est impératif d'effectuer des tests virtuels, dans le but d'éviter le plus de problèmes possibles, qui pourraient entraîner des bris et des délais. Pour ce faire, le logiciel ARGoS sera utilisé. Cet outil est bien documenté et il est facile de développer différents paramètres de simulation, ce qui nous sera essentiel. Tout comme la partie embarquée, les commandes de bases proviendront de l'interface, pour être traitées par le serveur, puis dirigées vers le contrôleur de simulation. Pour plus amples détails sur ce sujet, se référer à la section 3.2 et 3.4.

3.2 Station au sol (Q4.5)

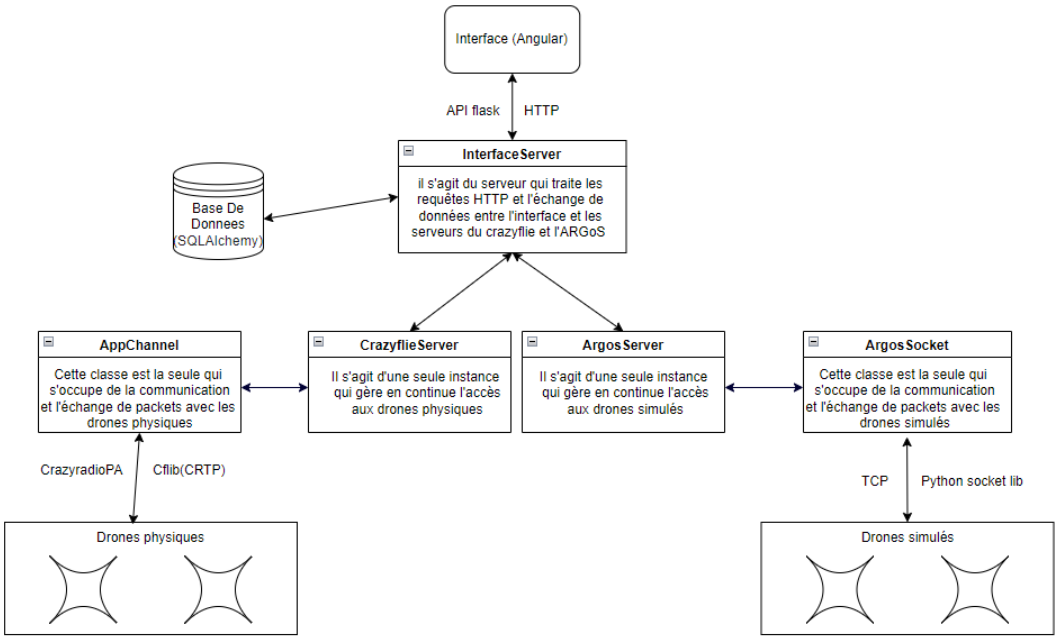

Figure 2: Architecture de base de la station au sol

La station au sol est composée d'une interface web, faite avec le cadriciel Angular, d'un serveur qui se constitue de trois sous-serveurs et d'une base de données SQLAlchemy. Les trois sous-serveurs sont le serveur de l'interface (InterfaceServer),qui gère les requêtes HTTPs avec l'interface et l'acheminement des messages vers les autres serveurs, le serveur des Crazyflies (CrazyflieServer), qui gère les connections avec les drones physiques et le serveur de ARGoS (ArgosServer), qui gère les connections avec le logiciel de simulation ARGoS.

La nécessité d'avoir trois serveurs est due aux moyens de communications différents entre l'interface web, qui utilise l'api Flask et Rest, notamment avec les requêtes HTTP, les drones physiques, qui utilisent Cflib pour la CrazyradioPA, et les drones simulés sur le logiciel ARGoS qui utilise la librairie de Python pour la communication TCP avec les Sockets.

Les serveurs sont tous développés avec le langage Python parce que, premièrement, la librairie Cflib de Bitcraze permettant de communiquer avec les drones physiques à l'aide de la CrazyradioPA est en Python. Deuxièmement, le micro-cadriciel Flask est excellent pour les services légers tels que notre application web. Finalement, l'accessibilité de Python et les maintes ressources et communautés qu'offre ce langage rendront sûrement les problèmes qu'on peut rencontrer lors du développement du projet plus facile et rapide à régler.

En ce qui concerne la base de données, on utilise SQLAlchemy pour stocker et récupérer les données. Afin de gérer cette dernière, le système PostgreSQL est installé. Il offre des types de données plus complexes et permet aux objets d'hériter des propriétés mieux que d’autres systèmes de gestion de base de données relationnelles. De plus, nous avons installé la version Web du logiciel pgAdmin 4 afin de contrôler, maintenir et utiliser les objets de la base de données. En effet, c’est une plateforme d'administration et de développement pour PostgreSQL.

3.3 Logiciel embarqué (Q4.5)

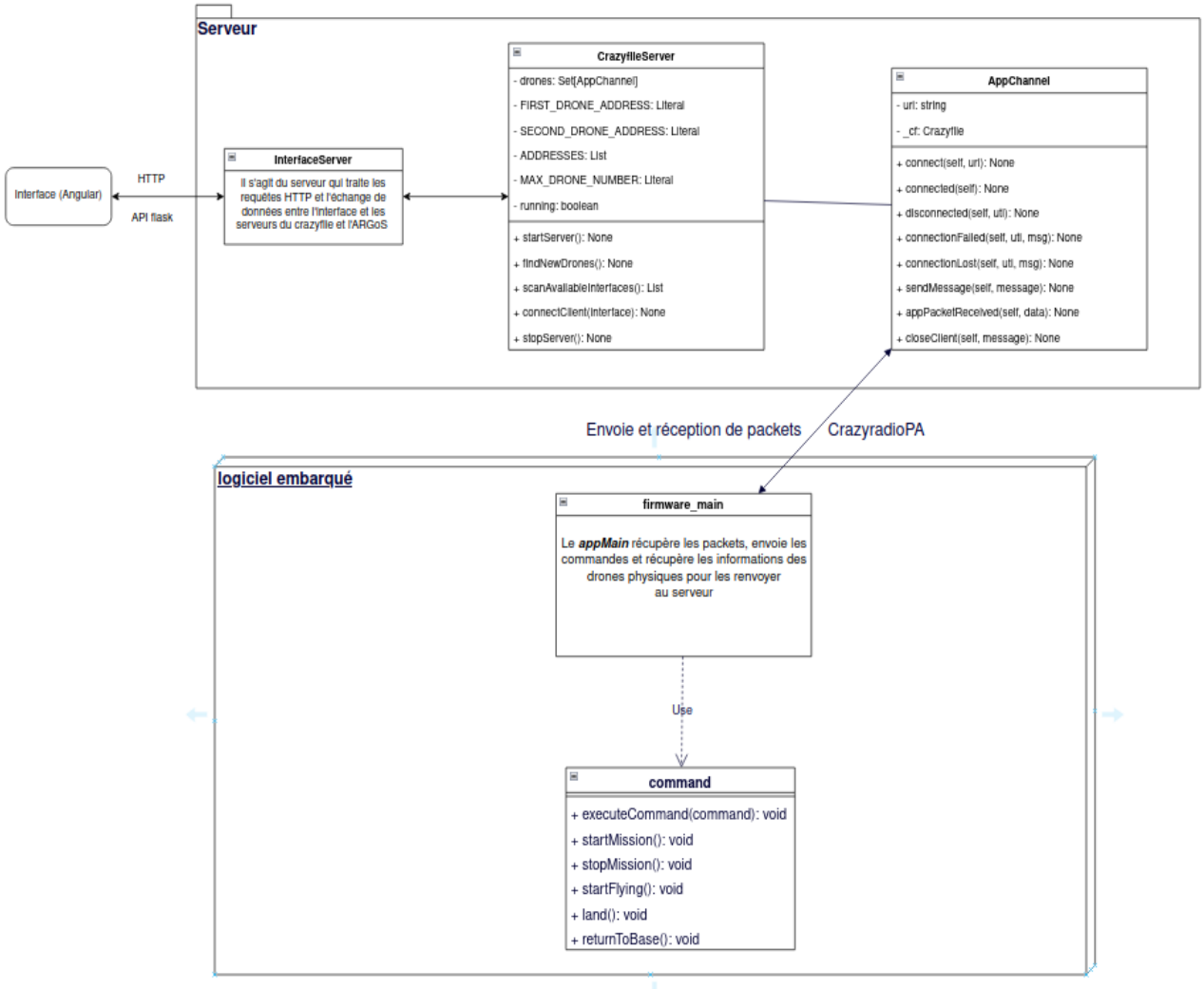

Figure 3: Architecture de base du logiciel embarqué et la communication avec les autres composantes

Avant de parler de l'architecture du logiciel embarqué, il est nécessaire de décrire brièvement les classes, impliquées dans la communication avec les drones physiques, *CrazyflieServer* et *AppChannel* du serveur et leurs rôles.

La classe *CrazyflieServer* est une seule instance qui joue le rôle d'un serveur, elle s'occupe de vérifier si la connexion, avec la CrazyradioPA, aux drones physiques est possible pour ensuite scanner les drones prêts à se connecter. Une fois un ou plusieurs drones sont détectés, elle appelle la méthode de connexion (*connect(self, uri)*) avec l'uri du drone en question comme paramètre. Il vient ensuite le rôle de la classe *AppChannel* qui s'occupe de gérer les différents événements (connection, déconnection, etc…) et de gérer la

communication avec les drones en envoyant et recevant les paquets au biais de la librairie python de Bitcraze (Cflib) permettant d'utiliser la radio CradyradioPA.

Quant au logiciel embarqué qui sera flashé sur les drones physiques, il utilise le App Layer de Bitcraze se compose principalement d'un fichier source *firmware_main.c* représentant le cerveau du drone et qui contient la méthode de routine *appMain()*. Le *appMain()* utilisera les services de la classe *communicationService* pour la communication avec le AppChannel du serveur et avoir les informations de base du drones. La classe *communicationService* joue un rôle de structuration et d'encapsulation des données du drone. Le *appMain()* utilise aussi la classe *command* qui s'occupera de traiter les commandes une fois le filtrage effectué.

Le cœur du code s'effectue donc dans la classe *command*. Lorsque l'exploration débute, le drone décolle, puis effectue en *random walk* dans la pièce, jusqu'à ce qu'il détecte un obstacle. Dans un tel cas, le drone fait demi-tour, s'éloigne de l'objet, puis reprend son exploration aléatoire de la pièce.

3.4 Simulation (Q4.5)

Figure 4: Architecture de base de la simulation

Le côté simulation sera composé de trois couches principales essentielles au fonctionnement attendu, soit l'interface, la partie serveur ainsi que le contrôleur des drones simulées. Tout d'abord, au niveau de la simulation, l'interface permettrait à l'utilisateur de débuter ou arrêter une mission, puis d'afficher les informations nécessaires, comme la charge de la batterie, la position des drones dans l'environnement, et plus encore. L'interface répond donc directement au R.F.2 et R.F.3.

Par la suite, la partie serveur sera essentielle pour gérer les requêtes entre l'interface. Comme décrit dans la partie 3.2, la couche serveur est séparée en deux niveaux, ce qui facilite la communication spécifique au contrôleur ARGoS. En effet, il sera plus facile et efficace de traiter les demandes spécifiques à la simulation dans un serveur dédié à cette partie.

Troisièmement, le fichier contrôleur permettra de réaliser les opérations désirées du côté ARGos. Pour ce faire, la classe *CCrazyFlieSensing* comportera plusieurs méthodes pour gérer l'état des drones, et appeler les méthodes nécessaires aux

différents requis dans la boucle principale, *ControlStep*. Effectivement, les fonctions *Land et Takeoff*, comme le nom l'indique, rendrons possible le décollage et l'atterrissage des drones. Cette fonctionnalité répond aux exigences de base pour le R.F.2, et sera essentielle pour tout le projet. De plus, la méthode *StartMission* démarrera une mission d'exploration à une position spécifique, ou l'essaim devra explorer l'environnement virtuel de manière autonome, tout en évitant les obstacles et en cartographiant l'espace. Pour se faire, les drones iront dans un direction arbitraire générée aléatoirement jusqu'à ce qu'ils rencontrent un obstacle. Dès lors, les drones se dirigeront vers leur droite à un angle de 90 degrés de manière à l'éviter, puis continuerons l'exploration. De plus, cette fonction devra monitorer certains paramètres physiques du drones, tel que son niveau de batterie, pour le ramener à la base si celui-ci est en dessous de 30%. Pour ce faire, la méthode pourrait appeler la fonction *Land.* Toutes ces fonctionnalités rempliront les requis fonctionnels 4, 5, 6, 7, 8 et 12. Les autres méthodes, telles que *ReadMessage, SendMessage* et *Connect* serviront à gérer les communications et la connexion entre le contrôleur et le serveur.

Quant aux langages de programmation, nous bâtirons le serveur ARGoS en python, car il s'agit d'un langage comportant de nombreuses ressources (documentations et librairies), et permet une uniformité entre les serveurs. Pour ce qui est du contrôleur, il sera réalisé en C++, parce que les librairies propres au simulateur sont disponibles avec ce langage.

Finalement, pour ce qui est des communications entre les différentes couches, il est important qu'entre les serveurs et les contrôleurs, une connexion avec des *Web Sockets* sera utilisée, ce qui favorise le transfert bidirectionnel des données.

3.5 Interface utilisateur (Q4.6)

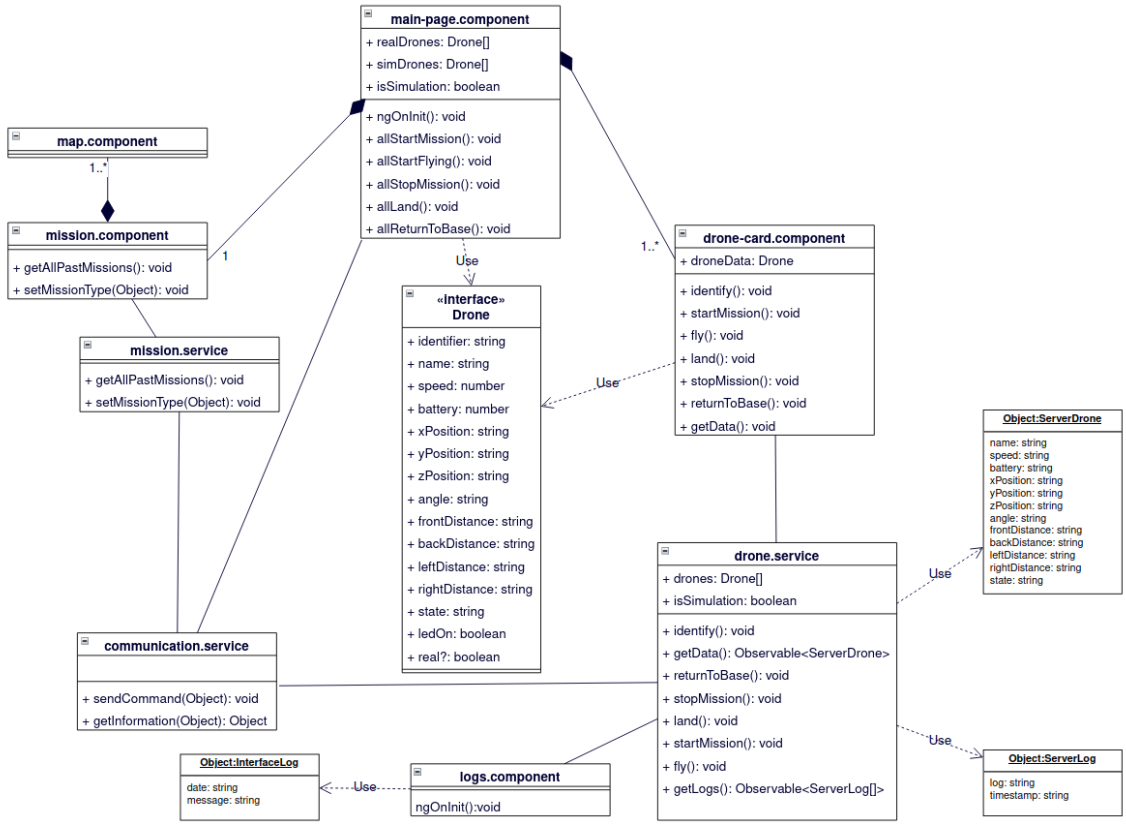

Figure 5: Architecture de base de l'interface

Figure 6: visualisation de l'interface de base

Lìnterface utilisateur est une interface web ergonomique et adaptative aux différents appareils (ordinateurs, tablette ..) développée à l'aide du cadriciel Angular avec les langages typescript, HTML et SCSS.

L'interface sert à commander les drones et à afficher les drones, leurs états (attente, en mission, crashed, etc…) et leurs informations (batterie, vitesse, position etc…) ainsi que les missions qui sont en cours ou passées avec toutes les informations récupérées lors des ces missions, notamment la carte générée lors de l'exploration pendant cette mission.

En effet, l'interface se compose de deux pages, une page principale et une page pour les logs qui est accessible à partir du panel de côté. La page principale affiche le contenu du composant *main-page.component* qui contient deux sections, une section pour les drones qui affiche les drones et tout ce qui est en lien avec ces derniers (boutons, état, données etc...) et une section pour les missions qui affiche les missions et leur avancement ainsi que la carte générée en temps réel lors des missions. La page pour les logs affiche les logs, récupérés des senseurs des drones ou des commandes du serveur, pour des fins de débogage et vérification du bon fonctionnement du système.

Le cadriciel Angular a été choisi de part sa popularité et surtout pour le fait que tous les membres de l'équipe sont bien familiers et expérimentés avec ce cadriciel et les services et librairies qui pourront éventuellement aider. Pour ce qui du langage Typescript, il a été privilégié sur javascript pour les mêmes raisons que celles considérées pour le choix d'Angular, mais aussi pour la structuration et l'organisation que ce langage peut ajouter au côté frontend du projet.

3.6 Fonctionnement général (Q5.4)

Le point d'entrée de l'application est la page principale de l'interface. Cette dernière contient les maps de suivi de l'exploration ainsi que les cartes spécifiques à chacun des drones contenants leurs données et les boutons permettant de lancer les différentes commandes, soit décoller, atterrir, identifier, commencer une mission et retourner à la base. La communication se fait avec REST API en utilisant des requêtes HTTP. Les commandes sont différenciées à l'aide d'un message envoyé par un POST. Au niveau du serveur, dès que le main est lancé, le serveur des drones physiques (*crazyflieServer.py*) et le serveur des drones simulés (*argosServer.py*) sont lancé de façon concurrente en utilisant le

threading de Python pour gérer la connection et la communication avec les drones physiques, par le biais du *appChannel.py*, et les drones simulés, par le biais de Socket, respectivement. Par la suite, le serveur Flask qui traite les requêtes de l'interface et achemine les commandes au bon serveur se lance à son tour. Pour la récupération des logs et des données et états des drones, l'interface effectue des requêtes HTTP GET chaque seconde pour récupérer un tableau d'objets *ServerLog* de type dictionnaire (TypedDict) contenant les logs de débogage et récupérer un objet *Drone* de type dictionnaire (TypedDict) contenant les dernières données reçues des drones. Au niveau de l'interface, les objets récupérés sont représentés par des types *ServerLog* et *ServerDrone* qui ont la même structure et les mêmes champs que les objets *ServerLog* et *Drone* de type dictionnaire du serveur. Par la suite, un type *ClientLog* est initialisé à partir du *ServerLog* récupéré et aussi un objet *Drone* représentant les drones de l'interface est initialisé à partir du type *ServerDrone* récupéré contenant les dernières données et le dernier état du drone en question.

Une fois les commandes arrivées à la simulation, une méthode lit ces dernières et les traduit en différents états, soit décollage, exploration, et atterrissage. Comme détaillé à la partie 3.4, les drones explorent l'environnement selon une méthode bien précise, et évitent les obstacles sur leur chemin. De plus, il est nécessaire d'effectuer plusieurs vérifications en parallèle, tel que le niveau de batterie pour rapatrier le drone à la base si ce dernier descend sous les 30%. De même manière, des commandes spécifiques appellent des méthodes précises, tel que *SendCommand*, pour envoyer les informations nécessaires au serveur, et ultimement, les afficher sur l'interface.

Le système physique fonctionne de manière similaire. Les commandes envoyées changent l'état du drone, qui sont l'attente, l'identification, l'exploration et l'atterrissage. Pour plus de détails sur le fonctionnement de l'exploration, se référer à la section 3.3.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

Pour développer notre système aérien d'exploration, l'Agence spatiale de Polytechnique nous fournit l'ensemble du matériel nécessaire incluant les deux drônes Crazyflie et les deux radios Bitcrazy Crazyradio PA servant à la communication avec le poste de contrôle. De plus, le développement logiciel sera effectué sur nos ordinateurs. Par conséquent, aucun coût associé au matériel n'est endossé par notre compagnie.

De plus, comme en témoigne l'architecture générale de notre solution, l'ensemble des technologies et des services web utilisés par le poste de contrôle possèdent un coût nul.

Enfin les seuls coûts non nuls à prendre en considération sont les dépenses liées aux salaires de nos développeurs et du coordonnateur de projet. Afin de calculer ces coûts nous avons dressé une estimation de la charge de travail nécessaire par personne. Selon nos prévisions un total de 455 heures-personnes sera nécessaire pour nos développeurs-analyste et 3 heures-personnes par semaine pour 10 semaines en tant que coordonnateur du projet. Ces estimations sont effectuées en prenant en compte une taille d'équipe de 5 personnes.

À un taux horaire de 130$ pour les développeurs et 145$ pour le coordonnateur du projet, les coûts totaux du projet s'élèvent à 80 900$.

$$\text{Total} = \text{salaire}_{\text{développeur}} \times \text{heures}_{\text{développeur}} + \text{salaire}_{\text{coordonnateur}} \times \text{heures}_{\text{coordonnateur}}$$

$$Coût\ total = 130\$/h \times 455\,h + 145\$/h \times 150h$$
$$Coût\ total = 80\,900\$$$

4.2 Planification des tâches (Q11.2)

Tableau 4.2.1. Planification des tâches.

Jalon	Tâche	Développeur embarqué (en h)	Développeur simulation (en h)	Développeur Web (en h)	Total (en h)
Preliminary Design Review (PDR - 4 février 2022)	Établir une interface web relié à notre serveur web (Python+Flask)	0	0	15	15
	Établir la connexion entre notre serveur web et le logiciel embarqué du Crazyflie	10	0	0	10
	Mise en place d'une base de données accessible via notre serveur	0	0	5	5
	Établir la connexion entre notre serveur web et notre simulation Argos3	10	0	0	10
	R.F.1 Faire clignoter au moins une des DELs du drône suite à la commande identifier.	10	0	10	20
	R.F.2 Faire décoller et atterrir le drône sur notre simulation grâce aux commandes débuter et terminer la mission	0	10	10	20
	Total	30	10	40	80

Jalon	Tâche	Développeur embarqué (en h)	Développeur simulation (en h)	Développeur Web (en h)	Total (en h)
Critical Design Review (CDR - 18 mars 2022)	Faire décoller et atterrir les drônes Crazyflie à partir des boutons de l'interface.	10	0	5	15
	Envoyer continuellement l'état de chaque drône à notre interface web avec une fréquence minimale de 1 Hz.	15	0	10	25
	Implémenter la séquence de déplacement post-décollage des drônes physique, et pour la simulation.	10	5	0	15
	Activer les capteurs des drônes physiques et dans la simulation pour pouvoir repérer des obstacles.	5	5	0	10
	Mettre en place la stratégie d'évitement d'obstacles.	10	10	0	20
	Déployer notre interface web adaptative	0	0	5	5
	Envoyer en continu des logs de débogage au serveur et les stocker sur une base de données.	5	0	15	20

Jalon	Tâche	Développeur embarqué (en h)	Développeur simulation (en h)	Développeur Web (en h)	Total (en h)
	Afficher une carte de l'environnement de chaque drône en continu sur le poste de contrôle.	10	0	15	25
	Prototype permettant de générer l'environnement virtuel pour les tests dans ARGoS aléatoirement.	0	15	0	15
	Documenter le choix des conventions de codage adoptées par l'équipe.	0	0	5	5
	Effectuer des tests unitaires et les documenter.	20	20	10	50
	Présentation orale détaillant le concept presque final du produit.	4	3	3	10
	Total	84	58	68	215
Readiness Review (RR - 14 avril 2022)	Implémenter l'option de retour à la base pour les drônes	20	15	5	40
	Afficher la position des drônes en continu	15	15	5	35
	Présenter une carte globale combinant tous les drônes	0	0	15	15

Jalon	Tâche	Développeur embarqué (en h)	Développeur simulation (en h)	Développeur Web (en h)	Total (en h)
	Détecter les collisions des drônes et mettre à jour leur état	10	0	5	15
	Rendre disponible les cartes de vols dans la base de données	0	0	5	5
	Implémenter la possibilité de définir une zone de sécurité	10	10	5	25
	Regrouper l'intégralité du système à l'aide de Docker-Compose	0	0	10	10
	Documenter les tests et les instructions nécessaire au lancement du projet	0	0	15	15
	Total	55	40	65	160
Charge totale (en h)					455

4.3 Calendrier de projet (Q11.2)

Tableau 4.3.1 Calendrier du projet.

Semaine	Remise	Avancement du projet
Semaine du 31/01-04/02	PDR	- Appel d'offre complété - Architecture du projet mise en place - Démo R.F.1 et R.F.2
Semaine du 14/03-18/03	CDR	- Drônes physique en mesure de décoller partir en mission et atterrir - Drônes en mesure d'éviter des obstacles - Interface web disponible sur mobile - L'environnement virtuel pour les tests dans ARGoS peut être généré aléatoirement - Logs de débogage accessible pour l'utilisateur
Semaine du 18/04-22/04	RR	- Chaque composant du logiciel possède un test unitaire correspondant. - Retour à la base des drônes - Position des drônes affichées en continu sur une carte sur interface web - Carte globale combinant tous les drônes disponible sur l'interface - L'opérateur peut spécifier la position et l'orientation initiale des drônes - Le système est en mesure de détecter un crash de drone. - Un éditeur de code pouvant modifier le code des contrôleurs de drones est disponible dans l'interface utilisateur - Possibilité pour l'opérateur de spécifier une zone de sécurité.

4.4 Ressources humaines du projet (Q11.2)

Pour réaliser ce projet de grande envergure, une équipe de cinq ingénieurs en génie informatique sera constituée. De plus, ces jeunes professionnels doivent être encadrés par un expert (coordinateur) pour atteindre leur objectif. Voici une brève présentation de l'équipe, incluant leur qualifications et expériences personnelles ayant un impact significatif sur le projet.

Tout d'abord, Marc-Olivier possède des compétences en systèmes embarqués acquises lors de son curriculum au baccalauréat. De plus, lors d'un projet antérieur, il a eu à utiliser des technologies de simulations reliées à la robotique, ce qui pourrait s'avérer utile pour la partie virtuelle du projet.

Ensuite, Sinda est adéquatement qualifiée dans le développement de serveur, compétence travaillée au cours de multiples projets. De plus, son expérience en programmation lui permet de connaître plusieurs langages, dont Python et ses librairies, ce qui sera sans doute utile pour le développement du serveur. Finalement, l'utilisation de l'outil Docker dans un cadre professionnel lui permettra d'accompagner l'équipe tout au long du projet avec ce logiciel.

Pour sa part, Sanmar a de bonnes connaissances aux niveaux des bases de données, qu'il a largement utilisées lors de précédent stages. De surcroît, son expérience avec les systèmes embarqués et le langage C++ sera pertinente lors du développement physique. Puis, son habileté naturelle à l'organisation et au leadership gardera sans doute l'équipe sur le droit chemin.

Par la suite, Zakaria possède de l'expérience pertinente en développement web avec des outils tels que Angular, en ayant réalisé un projet basé sur cette technologie. De plus, sa familiarité avec le système d'exploitation linux est une compétence non négligeable pour le projet.

Finalement, Aram dispose d'une excellente compréhension des méthodes agiles, ce qui aidera certainement l'équipe à implémenter une méthodologie de travail efficace. À un niveau plus technique, ses compétences avec le développement web seront également un atout important à l'équipe tout au long du projet.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

En plus de la réunion hebdomadaire établi par l'Agence avec l'équipe soumissionnaire afin de discuter de l'avancement du projet, trois autres réunions hebdomadaires sont établies entre tous les membres de l'équipe soumissionnaire ainsi qu'au moins une réunion hebdomadaire entre les membres des sous-équipes travaillant sur les différentes composantes du projet. Le plan du projet sera discuté lors des trois réunions entre les membres de l'équipe soumissionnaire et lors de la réunion hebdomadaire établie par l'Agence qui sera planifiée selon un rapport d'avancement des travaux. Lors des réunions entre les membres des sous-équipes, la qualité du code et l'avancement des tâches fera l'objet des réunions. Tout ceci permettra d'augmenter la cohésion de l'équipe soumissionnaire, assurer une bonne qualité de travail et mener à bien les trois livrables, soit le Preliminary Design Review (PDR), le Critical Design Review (CDR) et le Readiness Review (RR).

Le Preliminary Design Review (PDR) sera évaluée selon les critères suivants :

- La présence d'une interface web ergonomique permettant de communiquer avec les drones physiques et simulés.
- Un serveur minimal permettant l'acheminement des requêtes et la communication entre l'interface et les drones physiques et simulés est implémenté.
- Il est possible de faire décoller et faire atterrir les drones en simulation à partir de l'interface à l'aide des boutons ‘’ Start mission ’’ et ‘’ Land ‘’ respectivement.
- Il est possible d'identifier un drone à partir de l'interface à l'aide du bouton ‘’ Identify ‘’ qui aura pour effet d'allumer et éteindre toutes les DELs.
- Une base de données est implémentée et connectée au serveur.

Par la suite, le Critical Design Review (CDR) sera évaluée selon les critères suivants :

- L'interface affiche et actualise l'état des drones chaque seconde.
- L'interface affiche les informations et données tels que le niveau de batterie, vitesse et position de chaque drone.
- Il est possible de faire voler au minimum deux drones et de les commander à partir de l'interface.

- Les drones physiques et simulés sont capables d'explorer l'environnement de façon autonome en utilisant le bouton ‘’ Start mission ‘’ présent dans l'interface.
- Les drones physiques et simulés sont capables d'éviter les obstacles détectés par leurs capteurs.
- L'environnement de simulation sur ARGoS se génère aléatoirement avec un minimum de 3 murs.
- L'interface présente un prototype de génération d'une carte à partir des données collectées par les drones que ce soit physiques ou simulés.

Finalement, le Readiness Review (RR) sera évaluée selon les critères suivants :

- Toutes les informations sur les anciennes missions effectuées sont enregistrées dans la base de données et accessibles à partir de l'interface.
- Il est possible de lancer toute l'application à l'aide de Docker Compose à partir d'une seule commande.
- Tous les requis matériels, logiciels, fonctionnels, de conception et de qualité sont respectés.
- Tous le code de la station au sol du produit final est testé ainsi que les algorithmes contrôlant les drones.

5.2 Gestion de risque (Q11.3)

Tout projet est exposé à une multitude de risques qui pourraient ralentir l'avancement du travail ou même mettre fin au projet. On peut distinguer deux types de risques, des risques qu'on peut prévoir et sur lesquels on a le contrôle, mais aussi des risques qu'on ne peut ni prédire leur occurrence ni les contrôler.

Parmi les principaux risques qu'on peut contrôler et éviter complètement avec une bonne gestion et prise de décision, c'est le non-respect des échéances à cause d'une mauvaise gestion des tâches et l'accumulation des retards. Une tâche non complétée dans le temps qui lui a été consacré peut causer un retard additionnel, probablement de même durée, revenant sur toutes les tâches qui lui sont liées et ainsi de suite pour les tâches qui suivent. Il y aura donc un effet domino qui finira par retarder tout le projet et donc par dépasser les échéances des livrables fixés par le plan du projet. Il est donc primordial de bien respecter le

plan de chaque semaine, mais en cas de surprises, il est possible aussi d'éviter le retard en chaîne si c'est bien géré et pris au sérieux depuis le début. Un moyen de récupération du retard de façon radicale, dans le cas d'urgence, serait d'arrêter des tâches secondaires ou non cruciales à la réussite du projet et transférer leur temps aux tâches obligatoires qui seront affectées par le retard et qui pourront mener à l'échec du projet par leur tour.

Un des principaux risques, non prédictibles et hors du contrôle des membres de l'équipe, serait l'arrivée d'un accident (de voiture, hospitalisation, dépression, etc…) à un ou plusieurs membres. Il est vrai que ce n'est pas possible de prédire l'occurrence de ce risque, mais il est possible d'au moins minimiser la magnitude de l'impact que ça pourrait avoir sur le projet en mettant en place des règles et une bonne répartition de travail en sous-équipes. Par exemple, en s'assurant qu'au moins deux membres maîtrisent et travaillent ensemble sur une même composante pour chacune des composantes du projet (interface, serveur, embarqué et simulation). Ceci aidera à grandement diminuer l'impact de la perte d'un membre car il y aura toujours au moins une personne qui maîtrise la composante affectée.

Pour ce qui est des risques techniques, le principal risque est dû aux requêtes http. En effet, l'utilisation du protocole http pour la communication mets en risque la vitesse et la fluidité de la communication avec le serveur car ça nécessite que le client demande l'information avant de lui envoyer et donc il y a un délai supplémentaire pour établir la communication ce qui n'est pas désirable dans un système de communication temps réel. De plus, cette manière de communication peut être vulnérable à la manipulation des requêtes et l'interception des paquets et des données par une personne malveillante connaissant bien le système. Un autre risque technique serait le dysfonctionnement du logiciel de simulation ou d'une composante d'un drone physique, par exemple des pins qui ne se placent pas bien ce qui aura un effet sur la stabilité du courant.

5.3 Tests (Q4.4)

Comme décrit dans la section 5.1 le contrôle de la qualité représente une partie importante du développement du projet. Par conséquent nous sommes dans l'obligation de mettre en place une stratégie quant au contrôle de la qualité. La mise en place de tests vérifiant le fonctionnement des composants logiciels développés par l'équipe est donc une tâche primordiale au bon déroulement du projet. Les tests nous permettront de repérer les erreurs de conception et les

erreurs de développement. Plus ces erreurs sont repérées tardivement plus elles coûtent chers à réparer, il est donc dans notre intérêt de tester nos composants en continu. Nous allons répartir nos tests selon nos quatres composants majeurs.

5.3.1 Tests interface

Afin de tester les différents services et composantes de notre interface web nous utiliserons les librairies de test Karma et Jasmine. Nous nous assurerons de tester le code relatif à l'interface en continu, et ainsi à ne jamais publier dans notre entrepôt Git du code dont les tests échouent.

5.3.2 Tests serveur

Afin de tester notre serveur écrit en Python et utilisant le micro framework Flask nous ferons appel à l'outil Pytest. Pytest est un outil complet permettant de tester tout type d'application Python. Ainsi nous vérifierons la réception des commandes de l'interface, ainsi que le traitement de ces requêtes. De plus, Pytest nous permettra de s'assurer de la bonne communication avec notre système de simulation ainsi qu'avec nos drônes physiques. De la même façon qu'avec les tests de l'interface, un ensemble de tests réussis est une condition sine qua non à la publication du code dans l'entrepôt Git correspondant.

5.3.3 Tests simulation

L'objectif de la simulation est de permettre de tester le bon comportement de notre système virtuellement avant de le tester dans le monde réel. En effet cette démarche est bien plus prudente car s' il existe une erreur dans le comportement du système il est préférable que celle-ci soit repérée lors de la simulation virtuelle que lors de tests directement sur les drones physiques. De plus, un des avantages de l'utilisation d'une simulation est la grande modularité de l'environnement. Nous pouvons tester un grand nombre de cas et de conditions et le temps dans différents environnements pour s'assurer que la logique derrière notre exploration est sans faille avant de l'implémenter sur les vrais drones. Ainsi la simulation nous permettra de tester tous les requis fonctionnels du drônes tel que le décollage, l'évitement d'obstacles, le retour à la base, le respect d'un périmètre de sécurité et pleins d'autres.

5.3.4 Tests micrologiciel embarqué

Avant d'implémenter le micrologiciel des drones, les nombreux tests effectués sur la simulation nous auront permis de confirmer l'exactitude de la logique d'exploration. Cependant étant donné qu'il existe des différences inévitables entre un environnement virtuel et entre le monde réel il est indispensable également de tester le comportement de nos drônes physiques. De plus, nous

testerons que les positions du drônes affichés sur l'interface correspondent bien aux positions réellement explorées par ce dernier. Pour tester les drones physiques, nous respecterons un protocole bien défini, permettant de minimiser les risques de bris ou quelconque problème. Le protocole, sous forme de liste est le suivant:

- Un seul drone à la fois
- Le vol s'effectue dans un endroit sécuritaire, démuni de gicleurs au plafond, tel que la volière située au local M-4505, à Polytechnique Montréal
- Porter des lunettes de sécurité
- Pour le test d'exploration : Vider la pièce, laissez le drone parcourir un espace restreint.
- Pour le test d'évitement d'obstacle: Diriger le drone vers un obstacle précis, près de nous.
- Noter les résultats et les comportements du drone.

5.4 Gestion de configuration (Q4)

Le système de contrôle de version utilisé pour le projet est git et le dépôt du projet sera organisé avec la plateforme GitLab. Le dépôt GitLab contiendra quatre dossiers :

➔ *Interface* : ce dossier contient le code source de l'application Angular permettant de commander les drones physiques et simulés à partir d'une interface web.

➔ *server* : ce dossier contient tous le code source Python permettant la connexion et la communication avec l'interface web, les drones physiques et les drones simulés avec ARGoS.

➔ *crazyflie-firmware* : c'est un sous-module du App Layer des crazyflie forké à partir du répertoire github *crazyflie-firmware* de la compagnie Bitcraze. Le code source du logiciel embarqué du projet se trouve dans le dossier *INF3995_firmware*.

➔ *sim* : ce dossier contient les fichiers de configuration du simulateur et le code source de la simulation sur ARGoS.

La documentation du code de chaque composante du projet sera mise dans les dossiers associés par le biais de commentaires des classes, méthodes et variables dans le code. La documentation de la solution et de la conception sera mise dans la racine du dépôt GitLab.

5.5 Déroulement du projet (Q2.5)

[CDR et RR seulement]

[Dans votre équipe, qu'est-ce qui a été bien et moins bien réussi durant le déroulement du projet par rapport à ce qui était prévu dans l'appel d'offre initialement.]

Pour ce qui est de l'avancement du projet au niveau des requis fonctionnels, nous sommes plutôt à jour avec ce qui à été présenté lors de la réponse à l'appel d'offres. Effectivement, nous avons réussi à présenter un projet utilisable avec les fonctionnalités de base, reliées au R.F 1, 2, 3, 4, 5 et 10. Cependant, certains points que nous souhaitions déployer lors du CDR ne sont pas tout à fait prêts, tel qu'afficher une carte dans l'interface, représentant le parcours du drone dans l'environnement. De plus, un certain retard à été pris pour les requis de qualité, notamment sur les tests logiciels unitaires. Finalement, pour ce qui est de l'aspect gestion de projet, il reste toujours du progrès à faire, surtout pour ce qui est du poids des tâches sur GitLab.

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

[CDR et RR seulement]

[Qu'est-ce qui fonctionne et qu'est-ce qui ne fonctionne pas.]

Drone physiques

Au niveau des drones physiques, de nombreux tests ont été effectués sur la simulation avant l'implémentation du code sur les drones. C'est-à-dire, la logique développée pour les différentes fonctionnalités est tout d'abord testée sur la simulation. Les tests au niveau de la simulation confirment que la logique d'exploration et d'évitement d'obstacle est exacte. Ensuite, ces fonctionnalités sont testées sur les drones afin de confirmer le comportement. Effectivement, les fonctions tel que fly, land, startMission ont montré un bon comportement sur les drones. Enfin, les tests d'exploration et d'évitement d'obstacle se sont avérés partiellement justes.

Interface et communications

Au niveau de l'interface, Jasmine test framework est utilisé pour effectuer des tests unitaires. Ceci permet de tester le code dans l'interface.

Simulation

Au niveau de la simulation, les tests fonctionnels se sont avérés partiellement concluants. À ce stade, les tests ne touchent qu'à la communication entre l'interface et le drone simulé, ainsi que les algorithmes d'explorations et d'évitement d'obstacles. Pour tester l'exploration, les conditions sont les suivantes: un seul drone dans une salle vide, parcours de la salle jusqu'à épuisement de la batterie, puis, atterrissage. Les conditions de succès sont le respect des murs (ne pas passer à travers), et le parcours continuel, c'est-à-dire ne pas rester pris dans un état intermédiaire. Ce test s'est avéré un succès car le drone est en mesure d'explorer la salle en respectant les contraintes posées. Pour ce qui est du test de l'algorithme d'évitement d'obstacles, l'environnement de test est constitué d'une pièce comportant 5 obstacles, des boîtes, distribués aléatoirement. L'objectif de cette expérience est de vérifier que le drone est en mesure d'explorer la majorité de la pièce, tout en évitant les embûches. Ce test eut un résultat mitigé, car le drone est en mesure, la majeure partie du temps, de compléter sa mission normalement. Cependant, il arrive que ce dernier entre en contact avec les obstacles, ou qu'il reste coincé à un endroit.

Serveur back-end

Au niveau du serveur python, nous allons utiliser pytest pour tester l'application Flask. Grâce à cet outil, il sera aussi possible de tester les différentes communications (communication serveur-simulation et communication serveur-Interface) et l'exécution des commandes reçues par l'interface. Cependant, pour le moment seulement des tests fonctionnels ont été effectués. Ces derniers confirment le bon comportement du code et des différentes communications.

6. Références (Q3.2)

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

Fahmida, Y. (2018). Install PgAdmin4 on Ubuntu. Linux Hint. *https://linuxhint.com/install-pgadmin4-ubuntu/*

Kili, A. (2020, 3 août). *How to install postgresql and pgadmin4 in ubuntu 20.04. Tecmint : Linux Howtos, Tutorials & ; Guides.* https://www.tecmint.com/install-postgresql-and-pgadmin-in-ubuntu/

Krebs, B. (2018, 13 mars). *Using python, flask, and angular to build modern web apps - part 1. Auth0 - Blog.* https://auth0.com/blog/using-python-flask-and-angular-to-build-modern-apps-part-1/

Raible, M. (2019, 25 mars). *Build a CRUD app with python, flask, and angular. Okta Developer.* https://developer.okta.com/blog/2019/03/25/build-crud-app-with-python-flask-angular

GeeksForGeeks. (2022) *Socket Programming in C/C++.* https://www.geeksforgeeks.org/socket-programming-cc/

ANNEXES

[Inclure toute documentation supplémentaire utilisable par le lecteur. Ajouter ou référencer toute norme technique de projet ou plans applicables au projet.]