Dernière modification: 8 mars 2021

INF3995: Projet de conception d'un système informatique
Hiver2021
Réponse à l'appel d'offres

Système aérien minimal pour exploration

Proposition répondant à l'appel d'offres n°H2021-INF3995 du département GIGL

Équipe No. 203

Bal, Samba Bousso

Chritin, Mathurin

Grootenboer, Hubert

Maghni, Issam Eddine

Sangam, Eya-Tom Augustin

Table des matières

Introduction

La robotique fait partie de notre travail depuis maintenant 2 ans (notamment depuis un projet nommé « Projet Initial de système embarqué ») réalisé il y a un an et demi. Cet appel d'offre sur un système aérien minimal d'exploration figurait donc naturellement au sommet de la liste des offres que nous souhaitons décrocher. Nous sommes une équipe de 5 personnes motivés par les systèmes embarqués et nous serions heureux de pouvoir vous faire profiter de notre expertise et de notre engouement pour ce projet. Nous sommes convaincus que nous arriverons à satisfaire vos exigences de la meilleure des manières, en vous offrant de la flexibilité et en restant à l'écoute de vos remarques tout au long de ce projet. Le ci-document répond à votre appel d'offre.

1 Vue d'ensemble du projet

1.1 But du projet, portée et objectifs

L'objectif principal de ce projet est de créer un système informatique de gestion de drones. Ledit système est un ensemble de logiciels qui permettra à un essaim composé d'un nombre arbitraire de drones (quadricoptères) miniatures (<250 grammes) d'explorer une pièce d'un bâtiment de moyenne dimension à l'aide de capteurs lasers et de reporter ces informations à une interface opérateur basée sur le Web. Un opérateur, en utilisant cette interface, doit être en mesure de démarrer le système, l'arrêter, et mettre à jour le logiciel de bord des drones. Le système à concevoir est composé de deux parties (voir Figure 1).

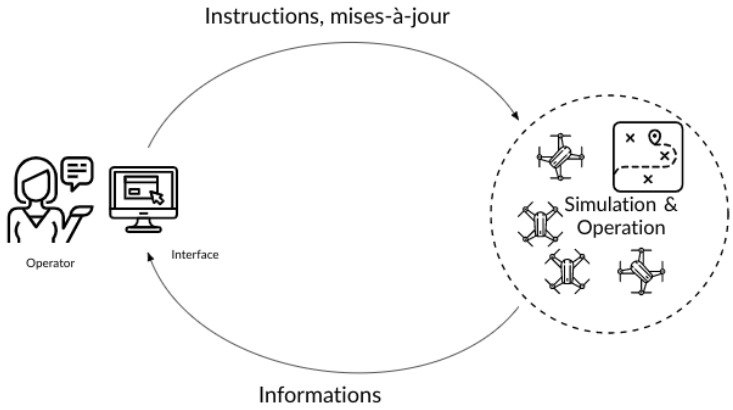

Figure 1 – Système à concevoir

— Station au sol : il s'agit d'un ordinateur avec une interface Web qui peut communiquer avec les drones à travers un canal de communication à 2.4 GHz. Nous supposons qu'au moins un robot de l'essaim est toujours en communication avec la station au sol ;

— Partie embarquée : il s'agit des drones et du logiciel qui roule sur leur micro-contrôleur de bord. Les drones communiquent entre eux et avec la station à travers un canal de communication à 2.4GHz.

L'opérateur utilise un navigateur Web pour contrôler le système et visualiser les données générées par les drones. Le produit complet sera livré accompagné d'une démonstration vidéo.

Un tel système se révélera très utile durant les explorations sur Mars, encore peu connue de tous. Nous imaginons une situation dans laquelle un robot plus complexe, plus lent et limité dans sa capacité de mouvement, se fera diriger par une colonie de drones qui lui indiquera les endroits les plus intéressants à explorer.

1.2 Hypothèses et contraintes

1.2.1 Hypothèses sur l'utilisation du système

- **Utilisation du système** :
  Le système conçu sera utilisé uniquement à des fins d'explorations et non d'espionnage. Selon le milieu exploré, l'utilisateur du système se devra d'obtenir les autorisations nécessaires conformément au lois régissant le lieu d'utilisation.
- **Sécurité** :
  L'opérateur du système devra être âgé d'au moins 16 ans et devra se munir de lunettes de sécurité durant toute interaction avec un des drones.
- **Milieu exploré** :
  Le milieu exploré devra se trouver dans des conditions météorologiques convenables. Idéalement, une pièce fermée à température ambiante.

1.2.2 Contraintes matérielles

- **Les drones** :
  L'opérateur du système devra posséder au moins deux kits Bitcraze Crazyflie 2.0 STEM Bundle. [1] Uniquement le « Flow deck » [2] et le « Multi-ranger deck» [3] fournis dans les kits devront être installés sur les drones.
- **La station au sol** :
  L'ordinateur de la station de contrôle devra être muni d'un système d'exploitation Linux, virtualisé ou non, possédant au minimum 4 Gio de mémoire vive. Un fureteur à jour et un terminal doivent être présent sur l'ordinateur.
- **Communication avec la station au sol** :
  Le seul moyen de communication entre la station au sol et les drones doit être l'antenne Bitcraze Crazyradio PA [4] connectée à la station au sol.

1.2.3 Contraintes fonctionnelles

- **Commandes supportées** :
  Les commandes sont des instructions envoyées depuis l'interface de contrôle aux drones. Trois commandes devraient être supportées par les drones :
  - Lancer la mission :
    Une fois la commande de départ reçue, les drones doivent être capables de parcourir le périmètre d'une pièce d'au maximum 100 m^2 en absence d'autres commandes de la

station au sol.
Durant cette opération, les drones doivent éviter les obstacles sur leur chemin, incluant les autres drones.
L'algorithme d'exploration de l'environnement n'a pas de contraintes. Cela peut être une séquence de mouvements aléatoires. Cependant l'exploration doit se faire de façon à pouvoir générer une carte du milieu exploré.
Les drones ne doivent pas décoller avec un niveau de batterie inférieur à 30%.

— Fin de mission :
Cette commande permet d'arrêter l'exploration. Après cette commande, les drones doivent se mettre en position stable en attendant une autre commande de la station au sol.

— Retour à la base :
Le retour à la base doit rapprocher les drones à moins de 1m de la station à terre. Le retour à la base et l'atterrissage doivent s'enclencher automatiquement dès que le niveau de batterie passe en dessous de 30%. Mise à jour logicielle :

En dehors de ces commandes, le robot devrait toujours se mettre dans un état cohérent si un accident survient. De plus, l'interface doit permettre la mise à jour du logiciel interne des drones seulement s'ils se trouvent au sol. La mise à jour doit être implémentée comme l'envoi d'un paquet binaire en utilisant l'API de BitCraze.
La Figure 2 montre un aperçu des différents états du robot suivant les commandes reçues.

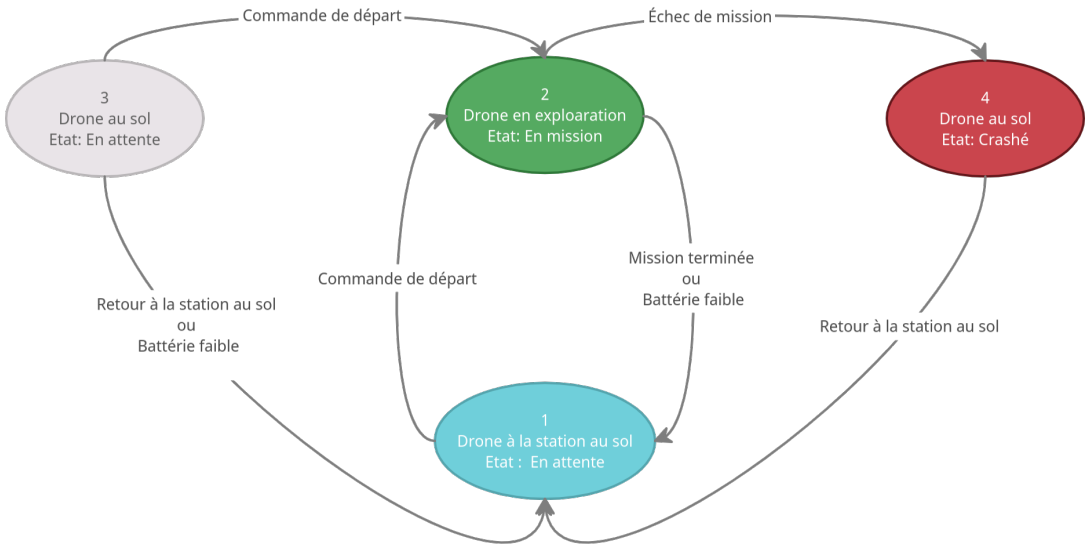

Figure 2 – Diagramme montrant les différents états et interactions possibles avec un drone

— **Interface de contrôle** :
Les données suivantes seront mise à jour avec une fréquence minimale de 1Hz :

1. Nombre drones
2. État des drones (en attente, en mission, accidenté)
3. Vitesse courante des drones
4. Niveau de batterie des drones

5. Carte générée durant l'exploration
6. Position des drones

L'interface utilisateur devra aussi disposer d'un éditeur de code afin de mettre à jour le micro-contrôleur du drone.

1.2.4 Contraintes de temps

La charge requise en termes d'heures pour la livraison du projet est de 630 heures-personnes. Plus de détails sont disponibles à la section 4.3.

Une version finale du projet sera livrée pour le lundi 19 avril 2021 avant midi.

1.3 Biens livrables du projet

Notre application sera composée de trois briques : un tableau de bord accessible par l'opérateur, un serveur maître avec une base de données faisant l'intermédiaire avec le tableau de bord et les drones, et le micrologiciel embarqué sur les drones.

Trois prototypes seront livrés au cours du projet :

Prototype minimal
Le prototype minimal a pour but de démontrer au client la maîtrise des technologies utilisées (*date prévue 2021-02-15*) :

- Interface web minimale avec des boutons de décollage (« Take Off ») et d'atterrissage (« Land »), ainsi que le niveau de batterie, la position et la vitesse du drone en action dans le simulateur ARGoS 3[5].
- Serveur maître basique faisant l'intermédiaire entre le simulateur et l'interface web : relais les messages de mise à jour en temps réel des drones vers le client web, et transfert des commandes du client web vers les drones, le tout à l'aide d'une *socket* TCP. Pas encore de concept de base de donnée.
- Version basique du micrologiciel [6] : récupération des données de la batterie et de la vitesse, encodage en JSON [7] et envoie des données vers une *socket* TCP
- Simulateur ARGoS faisant bouger deux drones.
- Démonstration simple du contrôle des LEDs des drones réels depuis l'interface web.

Prototype intermédiaire
Le prototype intermédiaire a pour but de démontrer au client l'avancement du projet (*date prévue 2021-03-08*) :

- Interface web avancée, fonctionnalité de retour à la base (« Return to base ») implémentés, et visualisation minimale des données captées par les drones sous forme d'un plan en deux dimensions.
- Simulateur avec quatre drones évoluant dans un environnement généré aléatoirement en évitant les obstacles.
- Serveur maître et micrologiciel avancés pour répondre aux besoins de la démonstration décrite par les points susmentionnés.

— Serveur stocke une partie de ses données dans la base de donnée distante

Démonstration finale
La démonstration finale mettra en jeu le produit fini ainsi qu'une vidéo de présentation du système (*date prévue 2021-04-19*) :

— L'interface web finale avec les boutons « Launch mission » et « Return to base » comme seules commandes pour lancer et terminer la mission, et affichage de toutes les informations des drones en temps réel.

— Le serveur maître final pour relayer les commandes de l'interface Web et traiter les données reçues des drones.

— Le tout sous la forme d'une vidéo pour montrer le comportement des drones dans un environnement qui lui est inconnu.

2 Organisation du projet

2.1 Structure d'organisation

L'équipe est composée de 5 membres. Ils participeront au développement de la solution à tous les niveaux (client, serveur et micrologiciel), et seront en tout temps au courant de l'avancement des différents artefacts du projet. Un membre de l'équipe, M. Samba Bal, assumera le rôle du gestionnaire de projet pour organiser le développement. GitLab sera utilisé pour gérer les tâches à effectuer, les assigner aux différents membres et comptabiliser le temps passé sur chacune d'elles.

2.2 Entente contractuelle

Un contrat de livraison clef en main serait adéquat pour ce projet. En effet, le contracteur a une liste des requis complète et suffisamment précise pour ne pas avoir à la modifier grandement au cours du projet. Grâce à cela, on peut prévoir le temps nécessaire pour réaliser le projet et ainsi prévoir un coût fixe pour la réalisation du produit. Le détail de l'estimation des coûts pour ce projet est donné à la section 4.1.

3 Solution proposée

3.1 Architecture logicielle générale

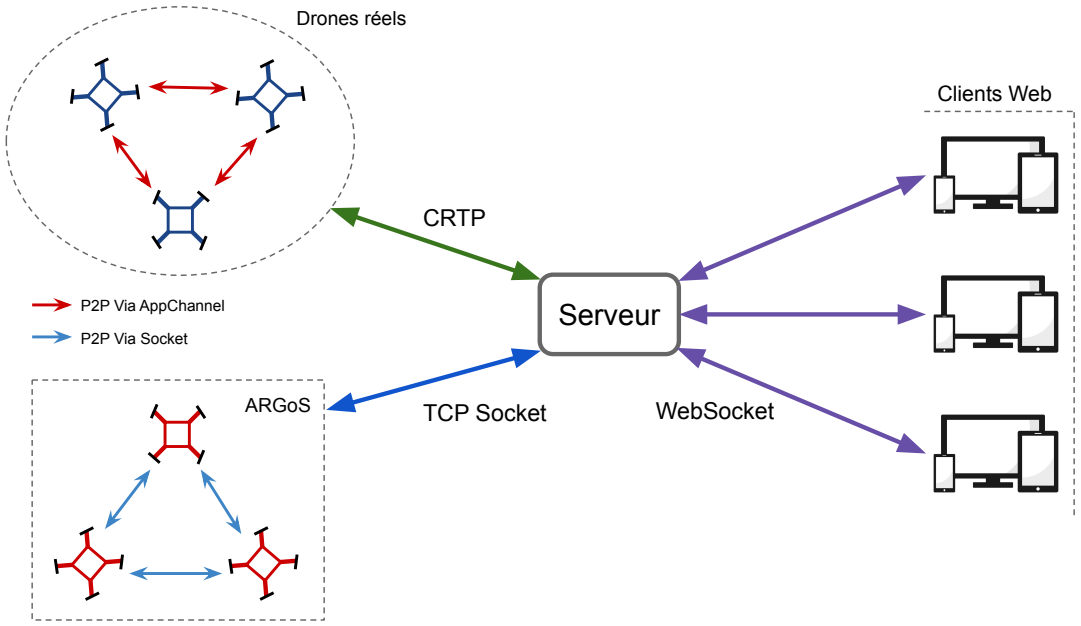

Figure 3 – Architecture globale de la solution

Comme nous montre la Figure 3, la solution retenue fait état de trois grandes entités :

- Un client web : Il s'agit d'une application web utilisée par un opérateur. Il permet ainsi les interactions avec les drones.
- Un serveur maître : Il s'agit d'un serveur centrale qui joue le rôle d'intermédiaire entre le client et les drones. D'une part il transmettra les commandes du client aux drones. D'autre part, il remettra au client les résultats et les informations venant des différents drones. Il interagira également avec une base de données locale pour stocker les différentes missions réalisées par les drones.
- Les drones : Un ou plusieurs drones qui communiquent entre eux mais également avec le serveur central (couple client-serveur). Ils effectuent les activités d'explorations.

Nous aurons donc besoin de deux modules, à savoir, une station au sol qui est un ordinateur disposant d'une interface web et d'une partie embarquée pour les drones. Nous avons par la suite subdivisé la station au sol en deux éléments qui sont le client et le serveur.

3.2 Architecture du logiciel embarqué

Compte tenu de la situation, le simulateur ARGoS sera beaucoup utilisé pour tester le fonctionnement de notre application. Ce simulateur est fourni avec un certain nombre de fonctionnalités

déjà implémentés, mais pour simuler fidèlement le comportement du drone dans la vie réelle, il sera nécessaire de lui apporter des modifications et d'ajouter certaines couches d'abstraction que nous allons tenter d'implémenter de la manière suivante :

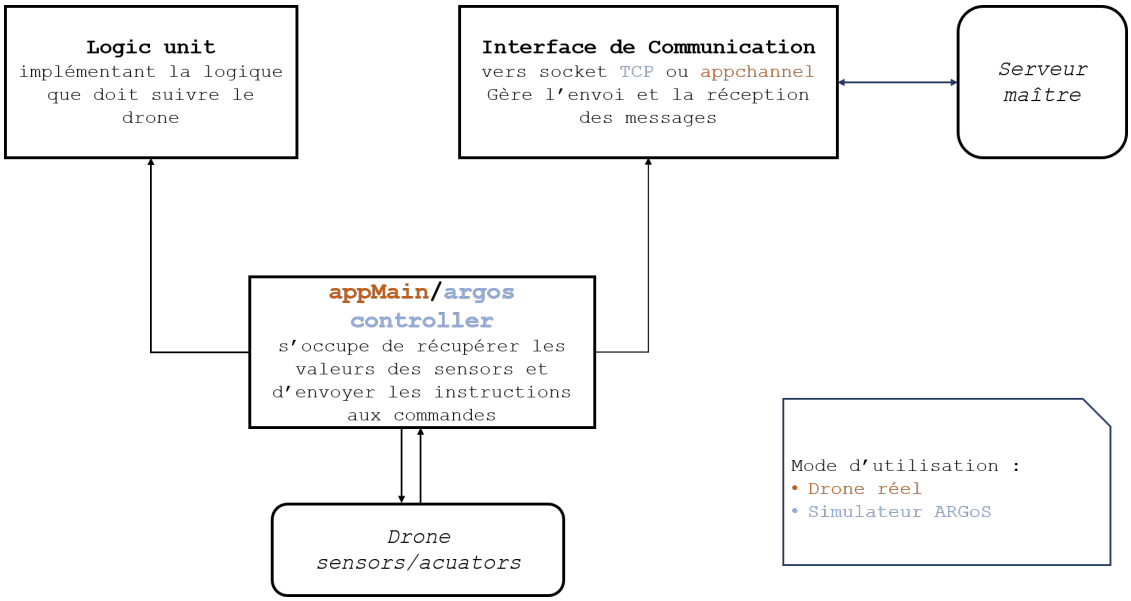

Figure 4 – Architecture du logiciel embarqué

L'objectif ici est de rassembler le plus possible la logique de notre application d'une manière à éviter la duplication de code. Par contre, ARGoS et le drone fournissent des APIs différentes [8]. Pour unifier le processus, nous allons développer sur deux fichiers sources qui implémentent le même fichier d'en-tête. Ainsi, nous pouvons porter notre micrologiciel sur les deux systèmes par une simple permutation des fichiers sources, sans changer le fichier d'inclusion. Par exemple, le simulateur communique actuellement avec le serveur maître à l'aide d'une *socket* TCP, mais le vrai drone utilisera la classe `AppChannel`. Nous allons donc ajouter une couche d'abstraction qui s'occupera de gérer la communication avec le serveur maître peu importe le type de drone utilisé (virtuel dans le simulateur ou réel).

Concernant la logique de fonctionnement du drone, nous décidons de nous tourner vers une machine à états. Cela nous permettra, en fonction des états, d'envoyer les instructions voulues aux drones réels ou à ceux d'ARGoS, dépendamment de ce que nous sommes en train de manipuler. La machine à état compte à ce jour 6 états différents, décrits par la figure 5 et la table 1 :

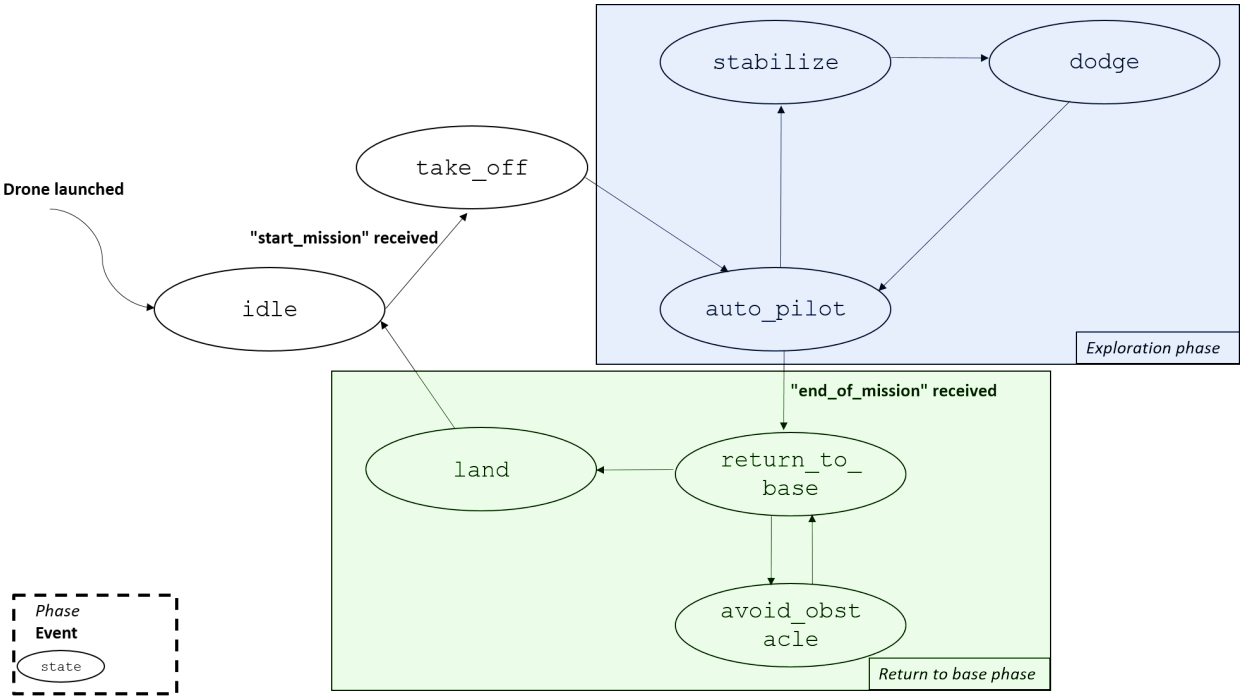

Figure 5 – Machine à états

Les états `go_to_base` et `avoid_obstacle` permettront de faire revenir le drone à son point de décollage.

Nous prévoyons également modifier le comportement du drone en mode `auto_pilot`, afin qu'il suive par exemple les murs pour optimiser l'exploration du milieu. Nous comptons pour cela nous inspirer d'un article de *Science Robotics* [9] : les drones choisissent une direction aléatoire au départ, et essaient d'avancer continuellement dans cette direction en évitant les autres drones et les obstacles rencontrés. Lorsqu'ils rencontrent un obstacle, ils l'évitent en le longeant (dans le cas d'un mur, ils le suivent). Les états `dodge` et `auto_pilot` implémentent pour le moment une forme de suivi d'un mur très naïve, sans se préoccuper des capteurs gauche ou droite, simplement en tournant sur eux-mêmes dès que le capteur avant détecte un obstacle suffisamment proche.

Nous prévoyons également, tout comme le présente l'article, de faire communiquer les drones entre eux en *peer-to-peer* afin qu'ils se transmettent mutuellement leurs positions respectives, et qu'ils puissent ainsi s'éviter efficacement.

Le code embarqué sera essentiellement en C++, tant pour le code simulé sur ARGoS que pour le micrologiciel des drones réels.

État	Description
`idle`	Le drone ne reçoit aucune instruction et reste à terre. Si le drone est en vol, il tombera tout simplement puisqu'il ne reçoit pas d'instruction lui demandant de maintenir son altitude
`take_off`	Le drone décolle, il reçoit une instruction lui demandant de prendre de l'altitude sans se déplacer (mouvement en `z` uniquement)
`land`	Le drone atterit, il reçoit une instruction lui demandant de perdre de l'altitude sans se déplacer (mouvement en `z` uniquement)
`auto_pilot`	Le drone reçoit une instruction le faisait avancer dans une direction donnée. Pour le moment, le drone ne fait qu'avancer droit devant lui en lisant les valeurs des capteurs du ranging deck. Il compare la valeur de son capteur `front` à une valeur seuil (différente entre ARGoS et les drones réels), et si jamais la valeur passe au dessous de ce seuil, le drone passe dans l'état `stabilize`
`dodge`	Le drone passe en mode évitement et se contente de tourner sur lui-même par impulsions de $\frac{\pi}{12}$. Après chaque impulsion, il compare à nouveau la valeur de son `front sensor` pour déterminer s'il doit continuer à tourner sur lui même. Il répète cette opération jusqu'à ce qu'aucun obstacle se trouve devant lui, puis revient dans l'état `auto_pilot`
`stabilize`	Dans cet état, le drone se stabilise. Généralement, on sauvegarde une position ainsi qu'une orientation visée, et on ne change pas d'état tant que le drone ne se trouve pas dans un intervalle donné de part et d'autre de la position/orientation voulue.

Table 1 – États actuels de la machine à états

3.3 Architecture logicielle de la station au sol

Serveur maître

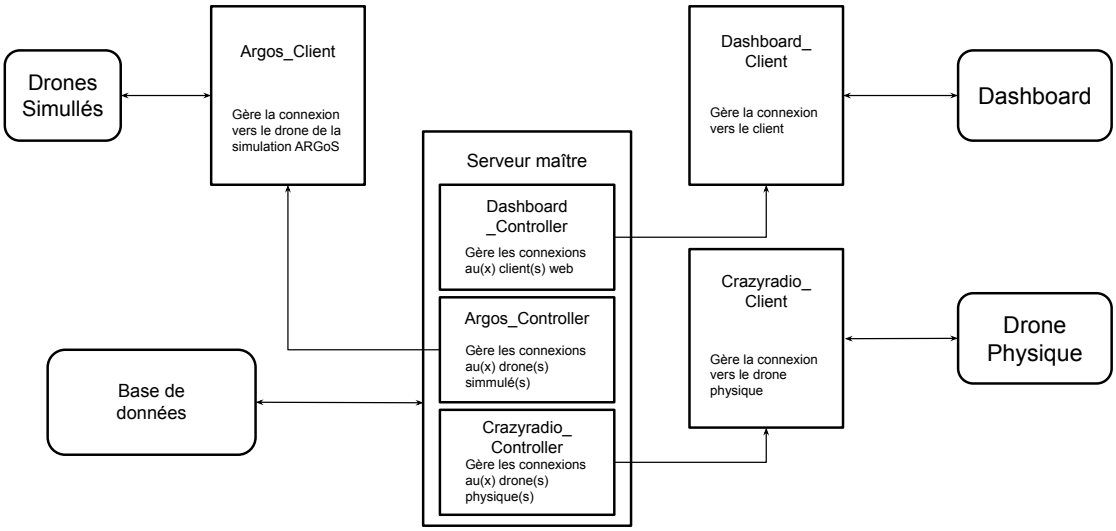

Figure 6 – Architecture globale du serveur maître

Le serveur maître est composé de trois serveurs, un pour gérer les communications avec les drones de la simulation ARGoS, un deuxième pour les communications avec les drones physiques via l'antenne Bitcraze et un autre pour gérer les communications avec les clients du dashboard. Ces trois serveurs sont nécessaires puisque les drones et les clients n'utilisent pas le même support de communication. Les clients utilisent une *WebSocket* [10] pour interagir avec le serveur, tandis que les drones physiques utilisent l'antenne et les drones de simulation utilisent des *sockets* TCP. Étant donné qu'il peut y avoir plusieurs clients et plusieurs drones, les serveurs doivent lancer des instances de gestionnaire de communication pour chaque connexion établie. Ces dits gestionnaires sont responsables de recevoir et d'envoyer de l'information à un seul drone/client. Le serveur maître s'occupe de faire le lien entre ces trois serveurs, et de sauvegarder les informations utiles dans la base de données.

Le serveur maître sera implémenté en utilisant le langage *Python* pour deux principales raisons :

— Les librairies de contrôles à distance des drones à savoir *crazyflie-lib-python*[11] et *crazyflie-clients-python* [12] de Bitcraze utilisent Python. Aucune adaptation ne sera donc nécessaire pour faire fonctionner ces mêmes scripts dans notre serveur.

— Python est un langage de haut niveau très accessible. Beaucoup de librairies et de ressources sont disponibles sur internet pour nous faire gagner en productivité. [13]

La base de données du serveur maître est une base de données non relationnelle nommé *TinyDB* [14]. Elle une implémentation simple d'un base de donnée stocké dans un seul fichier JSON, efficace pour stocker les objets python.

Interface Web

L'interface web s'occupe d'afficher les informations que le serveur lui envoie. Elle possède trois pages qui séparent les différents requis. La première s'occupe d'afficher l'état des drones qui sont connecté au serveur en indiquant leur position, leur vitesse ou leur niveau de batterie. La seconde permet de voir les missions actuelle ou précédente ainsi que la carte de la zone exploré par les drone lors des missions. La dernière dispose d'un éditeur de code permettant de modifier le code que les drone exécutent en pleine mission.

Cette interface Web est basée sur les technologies Web récentes HTML5, CSS3, JavaScript (utilisant le framework Angular).

3.4 Diagramme d'interactions

La figure ci-dessous présente un diagramme d'interaction incluant les parties essentielles de notre système. Il expose par la même occasion les protocoles de communication entre les différents systèmes.

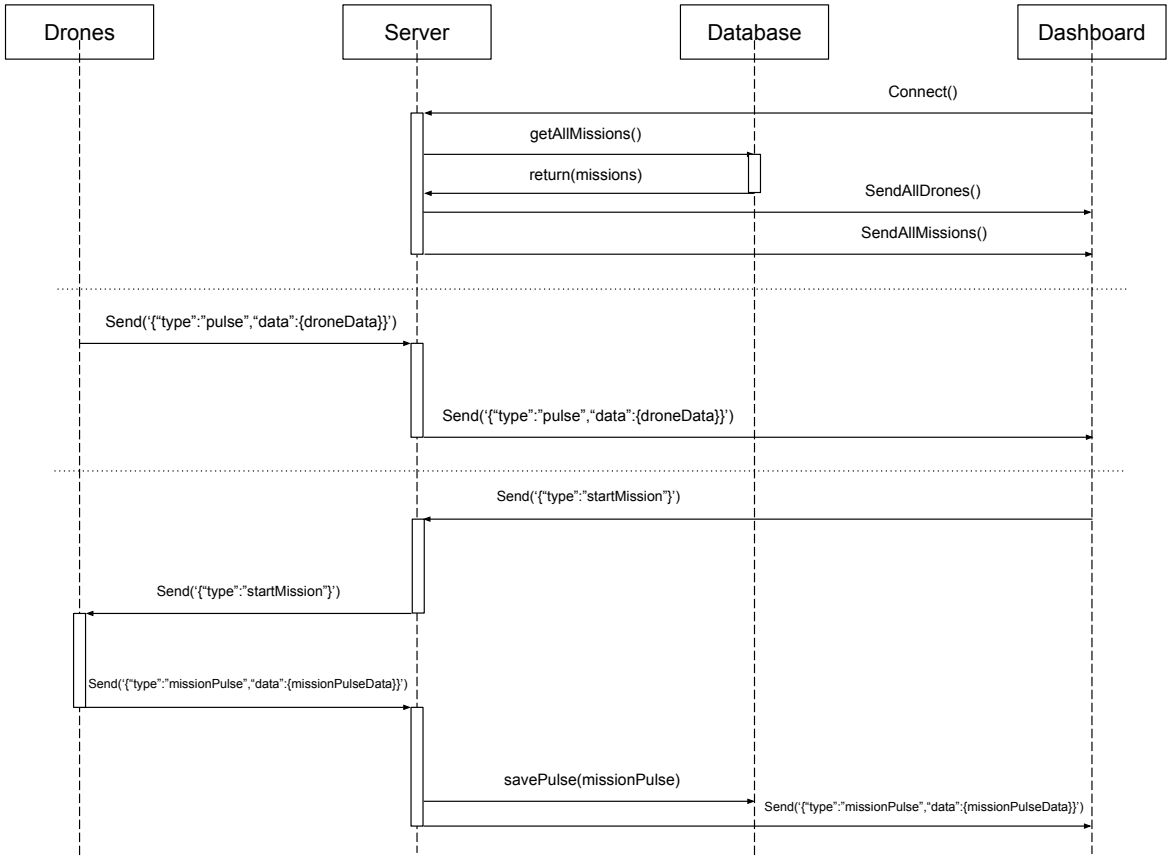

Figure 7 – Diagrammes d'interactions

La première partie nous montre le processus qui se déroule lorsqu'un nouveau client du dashboard se connecte. Une fois que le serveur reçoit une nouvelle connexion d'un dashboard, il récupère toutes les missions sauvegardées dans la base de donnée. Par la suite il envoie toutes les missions récupérées ainsi que tous les drones connectés au serveur à ce moment.

La deuxième partie montre les interactions lorsqu'un drone envoie un "pulse" (une actualisation des informations). Lorsque le serveur reçoit ce message, il met à jour les attributs du drone spécifié et transmet ces informations aux dashboards.

La troisième partie décrit les interactions lorsque l'on lance une nouvelle mission. Une fois que le serveur reçoit la commande de commencer une nouvelle mission, il ordonne aux drones de lancer une mission. Les drones commencent ensuite à envoyer au serveur des données concernant l'environnement qui les entourent (position, détection des capteurs). Le serveur sauvegarde ces informations dans la base de données au fur et à mesure et les envoie aux dashboards.

4 Processus de gestion

4.1 Estimations des coûts du projet

Les ressources humaines constituent la principale dépense du projet. Avec quatre développeurs-analystes et un coordonnateur de projet à temps partiel, il est nécessaire d'avoir une estimation de temps pour en déduire le coût juste.

Nous disposons de 11 heures de travail par semaine et par développeur et 12 heures par semaine pour le coordonnateur. Sachant que le projet s'échelonne sur 11 semaines, nous obtenons un total de 484 heures pour les développeurs et 132 heures pour le coordonnateur.

Le tableau ci-dessous décrit globalement les tâches à effectuer pour développer le système.

En considérant un salaire respectif avoisinant 130$/h et 145$/h, le coût humain s'élève à priori à 80'080$. Nous disposons de deux drones équipés pour un total de 800$. On alloue un 400$ additionnel pour couvrir un possible bris de matériel. Le seul logiciel nécessaire est ARGoS 3 et celui-ci ne nécessite aucune licence d'utilisation.

L'estimation à priori s'élèvera donc à 81'280$, avec une marge de manoeuvre d'une trentaine d'heures.

Tâches	**Coordonnateur [h]**	**Développeurs [h]**	**Total**
Description des exigences du système	10	20	30
Analyse des APIs (Bitcraze)	15	40	55
Analyse du simulateur ARGoS 3		10	10
Analyse des services	10	35	45
Analyse de la base de données		10	10
Implémentation des services	20	135	175
Implémentation des interfaces	10	50	60
Implémentation de la base de données	5	20	25
Adaptation du simulateur ARGoS 3		50	50
Tests des interfaces	8	10	18
Tests des services	8	10	18
Déploiement (Docker)	10	8	18
Tests finaux	25	50	75
Production de la vidéo	3	13	16
Ajustements finaux	6	10	16
Total [h]	130	471	**601h**
Total [$]	18'850$	61'230$	**80'080$**

Table 2 – Estimation des coûts du projet. L'équipe est constituée d'un coordonateur-développeur et de 4 développeurs-analystes.

4.2 Planification des tâches

Ci-dessous, nous définissons concrètement les gros blocs de travaux à effectuer ainsi qu'un nombre d'heures associées. Le nombre d'heure est à titre indicatif et est susceptible de changer durant le développement.

4.2.1 *Familiarisation avec les technologies et interface client*

Date prévue	15 février 2021	
Maîtrise du simulateur ARGoS	12 heures	Mathurin Chritin
Serveur *websocket* en Python	15 heures	Hubert Grootenboer
Construction de l'interface utilisateur avec Angular	16 heures	Eya-Tom A. Sangam
Client *websocket* dans le fureteur	10 heures	Eya-Tom A. Sangam
Serveur TCP en Python	25 heures	Hubert Grootenboer
Client TCP en C++ pour les drones	15 heures	Issam E. Maghni
Ajout de containers Docker pour les trois modules	10 heures	Samba Bal
Programmation des LEDs des drones réels	10 heures	Samba Bal
Tests de communication avec les drones réels	15 heures	Samba Bal & Augustin Sangam
Total	128 heures	

Table 3 – Bloc de travaux n°1 : emphase sur l'interface web de l'opérateur, premiers pas avec les technologies à maîtriser et début de la communication de bout en bout

4.2.2 *Communication, simulation avancée et base de données*

Date prévue	8 mars 2021	
Génération aléatoire de l'environnement sur ARGoS	15 heures	Mathurin Chritin
Communication inter-drones sur ARGoS	25 heures	Issam E. Maghni
Implémentation de l'auto pilot des drones dans le simulateur	30 heures	Mathurin Chritin
Ajout des commandes du drone (Take Off, Return To Base) dans l'interface client	10 heures	Augustin Sangam
Implémentation des commandes du drone dans le serveur maître	10 heures	Hubert Grootenboer
Implémentation des commandes du drone dans ARGoS et dans le micrologiciel	20 heures	Mathurin Chritin & Issam E. Maghni
Lecture et envoi des données des capteurs de distance dans ARGoS3	30 heures	Samba Bal
Protoype de visualisation de la carte générée à partir de données brutes	18 heures	Augustin Sangam
Ajout de la base de donnée pour stocker les données persistentes	15 heures	Hubert Grootenboer
Total	173 heures	

Table 4 – Bloc de travaux n°2 : emphase sur le simulateur ARGoS3, finalisation de la communication de bout en bout et implémentation de la base de données

4.2.3 *Visualisation des données, documentation et validation du système*

Date prévue	19 avril 2021	
Récupération des données des capteurs, uniformisation et envoi au serveur maître depuis les drones réels	30 heures	Mathurin Chritin
Stockage des données de missions dans la base de donnée	20 heures	Hubert Grootenboer
Récupération de missions antérieurs dans la base de donnée	10 heures	Hubert Grootenboer & Augustin Sangam
Traitement des données des capteurs des différents drones dans le serveur maître	40 heures	Augustin Sangam
Visualisation de la carte dans le client	45 heures	All
Tests du serveur maître	12 heures	Hubert Grootenboer
Documenter l'architecture et l'utilisation du système global	12 heures	Mathurin Chritin
Optimisation de l'algorithme de génération de la carte	12 heures	Samba Bal
Enregistrement d'une vidéo décrivant le fonctionnement du système sur drone	10 heures	Samba Bal
Enregistrement d'une vidéo décrivant le fonctionnement de la simulation	6 heures	Issam E. Maghni
Monter la vidéo	12 heures	Mathurin Chritin
Déploiement à l'aide de Docker de tous les artefacts du système	10 heures	Samba Bal
Correction des bogues	35 heures	All
Vérifications finales du système	50 heures	All
Total	304 heures	

Table 5 – Bloc de travaux n°3 : validation et documentation du système, visualisation finale de l'environnement exploré dans l'interface web et présentation du produit fini

4.3 Calendrier de projet

Le diagramme de Gantt ci-dessous présente le calendrier que nous allons essayer de suivre.

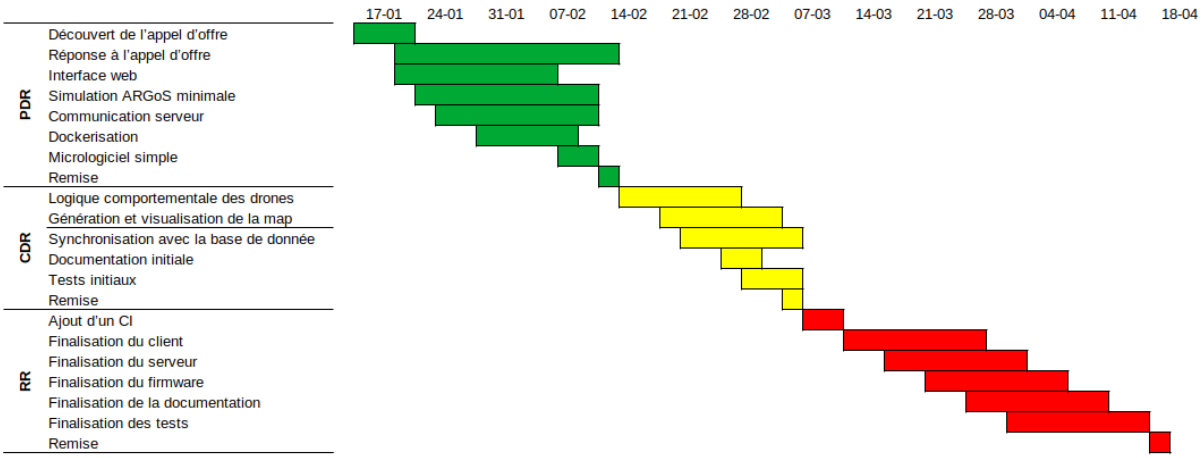

Figure 8 – Calendrier du projet

4.4 Ressources humaines du projet

Dans le but de réaliser ce projet, nous avons mobilisé cinq ressources : nous avons un spécialiste du développement d'interface web, principalement Angular (Eya-Tom Augustin Sangam). Ce dernier travaille de pair avec un développeur Back-End expérimenté (Hubert Grootenboer). L'équipe dispose également de deux développeurs de systèmes embarqués en C++ avec une expérience non négligeable avec les drones (Issam E. Maghni et Mathurin Chritin). Enfin, nous disposons d'un spécialiste DevOps chargé de la gestion du projet (Samba Bousso Bal).

5 Suivi de projet et contrôle

Compte tenu des trois séances hebdomadaires programmées par l'agence spatiale de Polytechnique pour nous rencontrer, des rencontres entre trois à cinq fois par semaine entre tous les membres de l'équipe sont prévues pour faire état de l'avancement de la solution. Des revues de code sont également prévues à chaque deux semaines, durant lesquelles un des membres de l'équipe devra sélectionner un fichier du code source et le soumettre aux autres membres pour qu'ils le passent au peigne fin et fournir une critique constructive. Cette approche bénéficiera tout autant à la qualité du code final qu'à la cohésion entre les membres de l'équipe.

5.1 Contrôle de la qualité

À chaque livrable, une revue du code sera effectuée et l'ensemble des composants de notre application sera inspecté.

5.1.1 Prototype minimal

- Le simulateur est fonctionnel et se lance correctement : en appuyant sur « Play », les deux drones sont à l'arrêt, posés au sol et émettent des « pulse » vers le serveur maître
- Une fois le serveur maître lancé, il ne requiert plus aucune intervention humaine
- L'interface web est fonctionnelle et ergonomique. Elle présente bien le statut des deux drones en vol dans le simulateur et permet à l'opérateur de cliquer sur un bouton faisant décoller le drone, et sur un autre bouton permettant de le faire atterrir.
- Le bouton « Take off » a pour effet de faire décoller le drone dans le simulateur
- Le bouton « Land » a pour effet de faire atterrir le drone dans le simulateur
- Lorsque le drone est en vol, ses informations sont actualisées en temps réel dans l'interface web : niveau de batterie en pourcentage, position `x`, `y` et `z` et vitesse [km/h]
- Il est possible de faire voler plusieurs drones à la fois et de les contrôler ensemble depuis l'interface web
- Les drones réels sont également accessibles depuis l'interface web, et un autre bouton permet de contrôler leurs LEDs.
- Les LEDs des drones du simulateur ne sont pas contrôlables depuis l'interface web. Les boutons Take Off et Land de l'interface web n'ont aucun effet sur les drones réels.
- Lorsqu'on relance plusieurs fois le simulateur, les drones de l'interface web sont actualisés en conséquence (les drones qui ne sont plus utilisés sont retirés, et les nouveaux drones de la nouvelle instance du simulateur sont ajoutés à leur place)

5.1.2 Prototype intermédiaire

- L'environnement du simulateur est généré aléatoirement : des obstacles sont placés différemment à chaque lancement du simulateur
- Il y a quatre drones dans le simulateur qui volent de manière autonome
- Au niveau du panneau de contrôle sur la station au sol, les fonctionnalités « Take Off » et « Return to Base » sont implémentées :
  - Take Off : le drone décolle, et se met à parcourir la zone de manière autonome sans toucher d'obstacles, et sans intervention de la part de l'opérateur
  - Return To Base : le drone retourne tout seul à l'endroit où il a décollé, et ne requiert pas non plus d'intervention de la part de l'opérateur
- Les drones envoient les données du *ranging deck* au minimum une fois par seconde.
- L'interface utilisateur présente un prototype de visualisation de la pièce, sous la forme d'un graphe qui affiche des points représentant des mesures de distance faites par les drones.
- Lorsqu'un drone « crash » ou se crash, la mission en cours est annulée.

5.1.3 Système final

- L'application est complète et respecte chacun des requis fonctionnels, matériels, logiciels et de conception spécifiés dans le document des requis.

— La base de donnée connectée au serveur stocke les missions des drones.

— L'application peut se lancer avec une simple ligne de commande sur Linux avec l'aide de Docker [15].

5.2 Gestion de risque

Le présent document présente un plan organisé des délais et timings à respecter pour le bon déroulement du projet. Le principal risque que nous avons identifié est le non respect de ce planning. Il s'agit ici d'un risque à plusieurs échelles : le non respect d'une tâche assignée une semaine donnée peut être reportée à la semaine suivante sans grandes conséquences. Si ce report déclenche une réaction en chaîne, nous pourrions risquer de ne pas terminer le projet en temps et en heure, ce qui s'avérerait fatal pour le projet d'exploration sur Mars, impliquant des pertes d'argent conséquentes. Il est donc important pour nous de bien respecter le planning. Pour cela deux solutions s'offrent à nous : augmenter le nombre d'heures que nous passons sur l'artefact problématique, ou dans un cas plus urgent, rayer un certain nombre de fonctionnalités non essentielles prévues dans notre planning.

Une seconde menace pouvant affecter le bon déroulement du projet peut être reliée à l'indisponibilité d'un membre ou d'une partie de l'équipe. Plusieurs scénarios sont envisageables :

— Un ou plusieurs membres de l'équipe décide(ent) d'abandonner le projet quel qu'en soit le problème ou encore,

— Un ou plusieurs membres de l'équipe sont indisponibles pendant une période bien définie (Pour des raisons d'ordre familiales par exemple).

Pour faire remédier à ces situations imprévisibles, la meilleure solution qui s'offre à nous demeure la communication. Si un membre doit s'absenter, informer à l'avance permettra à l'équipe toute entière de refaire la planification des tâches en conséquence.

Un troisième risque, moins probable, serait que l'un des membres de l'équipe se blesse lors d'une manipulation avec un drone et ne puisse de ce fait plus continuer à travailler sur le projet. Conformément aux consignes de sécurité fournies avec les drones, nous nous devons donc de porter en tout temps une paire de lunettes protectrice pour éviter tout problème qui pourrait nous pénaliser par la suite.

Un quatrième risque provient d'un possible changement des requis, qui peut entre-autre être dû à l'insatisfaction du client. Pour contrer cette possibilité, nous essayer de rencontrer de manière hebdomadaire les coordonnateurs de l'agence spatiale (Lajoie, Pierre-Yves et Arseneault, Samuel) pour s'assurer que notre avancement correspond bien aux attentes. Cette pratique nous permet également d'être au courant le plus tôt possible d'un potentiel changement des requis.

Finalement, le bris de matériel est un risque en soi car il peut entraîner un retard dans le développement et la remise. Avec la situation actuelle de pandémie, toute livraison prend plus de temps que d'habitude. Aussi, le prêt de matériel entre équipe n'est pas envisageable car les consignes déconseillent fortement tout regroupement. Pour atténuer ce risque, nous allons redoubler de vigilance en manipulant le matériel fourni.

5.3 Tests

Les sections de prise de décisions du drone nécessitent des tests pour assurer le bon fonctionnement des drones dans des situations critiques. Il faut donc s'assurer que le micrologiciel de nos drones agit selon le comportement qu'on désire selon les paramètres décidés. Ainsi, la logique du drone qui reçoit les données lues des capteurs qui génère un comportement en conséquence doit être testée avec des valeurs prédéfinies. Ces testes nous permettront d'assurer un comportement prévisible du drone.

Le serveur sera testé en « boîte noire », en testant que les informations transitent correctement entre les drones, les clients, et la base de donnée. On pourra pour cela déterminer une liste de messages à envoyer (pulse, takeoff, sensordata...) et une liste de résultats (base de données mise à jour, message transmis au client, message transmis au drone) à envoyer et à vérifier en entrée et en sortie du serveur maître.

Concernant l'interface de contrôle, nous opterons pour des tests unitaires utilisant `Jasmine`, afin de tester tous nos composants. Nous testerons ainsi si les boutons fonctionnent (vérifier le bon envoi des messages via la *websocket*), et nous pourrons minimiser les sources de problème dans notre application (éliminer la possibilité que le client n'envoie pas de message et se concentrer sur le serveur et le firmware pour le débogage).

5.4 Gestion de configuration

Le code source de la solution sera organisé dans un dépôt GitLab, comprenant trois sous-dépôts :

- `drone`, comprenant tout le code `C++` du micrologiciel embarqué ainsi que les fichiers de configuration du simulateur
- `server`, comprenant tout le code Python du serveur maître qui s'occupera de gérer les drones, la base de donnée et les messages du client et d'effectuer un traitement sur les données reçues
- `dashboard`, comprenant l'application Angular qui s'occupera de fournir une interface web à l'opérateur pour le contrôle et l'affichage des données des drones

La documentation du code sera faite avec Doxygen, en générant un PDF par projet, et sera stocké dans les entrepôts associés. Quant à la documentation de la conception, ainsi que la documentation de la solution en générale, elles se trouveront toutes les deux à la racine du dépôt GitLab principal.

Conclusion

En définitive, nous proposons dans ce document une solution qui répond au besoin d'exploration soulevé par l'agence spatiale. En plus de respecter les requis, nous ajoutons une touche personnelle qui augmentera la robustesse et la fiabilité du système à développer. Avec nos années d'expériences dans le domaine, nous pouvons affirmer que le projet sera achevé dans les délais impartis, en incluant tous les tests et ajustements si nécessaires. Tous les membres de notre compagnie sont mobilisés pour la bonne réussite de ce projet et en attente de vos instructions pour le commencement des travaux.

Références

[1] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński et P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," dans *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2017, p. 37–42.

[2] "Flow deck v2." [En ligne]. Disponible : https://www.bitcraze.io/products/flow-deck-v2/

[3] "Multi-ranger deck." [En ligne]. Disponible : https://www.bitcraze.io/products/multi-ranger-deck/

[4] "Crazyradio pa." [En ligne]. Disponible : https://www.bitcraze.io/products/crazyradio-pa/

[5] "Argos." [En ligne]. Disponible : https://www.argos-sim.info/concepts.php

[6] Bitcraze, "bitcraze/crazyflie-firmware." [En ligne]. Disponible : https://github.com/bitcraze/crazyflie-firmware

[7] D. Peng, L. Cao et W. Xu, "Using json for data exchanging in web service applications," *Journal of Computational Information Systems*, vol. 7, n^o. 16, p. 5883–5890, 2011.

[8] "App layer." [En ligne]. Disponible : https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/userguides/app_layer/

[9] K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen et G. C. H. E. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, n^o. 35, 2019. [En ligne]. Disponible : https://robotics.sciencemag.org/content/4/35/eaaw9710

[10] "Technologies web pour développeurs." [En ligne]. Disponible : https://developer.mozilla.org/fr/docs/Web/API/WebSockets_API

[11] Bitcraze, "bitcraze/crazyflie-lib-python." [En ligne]. Disponible : https://github.com/bitcraze/crazyflie-lib-python

[12] ——, "Crazyflie-clients-python." [En ligne]. Disponible : https://github.com/bitcraze/crazyflie-clients-python

[13] G. Van Rossum *et al.*, "Python," 1991.

[14] "Welcome to tinydb !¶." [En ligne]. Disponible : https://tinydb.readthedocs.io/en/latest/

[15] "Docker overview," Mar 2021. [En ligne]. Disponible : https://docs.docker.com/get-started/overview/