POLYTECHNIQUE MONTRÉAL

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 : ***105*** :

Omar Azizi

Aymen-Alaeddine Zeghaida

Skander Soussou

Driss Benzekri

Persia Shahdi

Le 1 Octobre 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 de mettre en place un système d’exploration pour un essaim composé d’un nombre arbitraire de drones miniatures qui vont explorer une pièce et reporter ces informations à une interface opérateur basée sur le Web. À travers cette interface, un opérateur doit pouvoir donner des consignes aux drones tel que démarrer/arrêter le système et faire le suivi des opérations. Le projet est donc composé de trois parties:

D’abord une interface web à partir de laquelle les commandes concernant les missions seront envoyées vers le drône (qui les reçoit grâce au *Crazyradio*); cette UI sert aussi à visualiser les données récoltées par l’essaim et aussi à dresser des cartes de l'environnement explorées. Cette interface sera utilisée de même pour les simulations. Les commandes à réaliser sont de lancer la mission et la terminer (décollage des drônes et leur atterrissage), ainsi que de pouvoir mettre à jour le logiciel.

Ensuite, il y a le logiciel embarqué (crazyfirmware) qui roule sur le microcontrôleur des drônes. Les drônes communiquent aussi entre eux (peer-to-peer) à l’aide de communication radio 2.4GHz. Et enfin, le simulateur ARGoS servira essentiellement de test (penser “sandbox”) pour simuler un vol (mission) physique dans un environnement contrôlé.

En résumé, le produit demandé est un ensemble de logiciels (serveur, client, drone et simulateurs) devant communiquer entre eux pour réaliser les requis demandés. Nous espérons grâce à ce projet devenir les précurseurs de l’exploration spatiale afin d’orienter un robot tel que curiosity vers les endroits les plus intéressants. Pour ce faire nous aurons des choix de conceptions à faire, ceux-ci seront basés sur les forces des membres de notre équipe ainsi que des requis globaux du projet .

La remise du projet est décomposée entre 3 livrables:

- la Preliminary Design Review (PDR)
- la Critical Design Review (CDR)
- la Readiness Review (RR)

Ceux-ci seront plus détaillés à la section 1.3.

1.2 Hypothèse et contraintes (Q3.1)

Pour la réalisation de ce projet, nous pensons être confrontés à de multiples contraintes, nous citerons donc nos hypothèses quant à la réalisation de ce projet.

Contraintes de temps:

- L'agence requiert que le nombre d'heures consacrées par les membres sur le projet ne dépasse pas 630 heures de travail, sans quoi notre proposition pourrait être jugée non conforme et être refusée. Il faut donc s'assurer de respecter cette contrainte en attribuant un poids à chaque tâche qui nous renseignera sur le nombre d'heures requises.
- La date du livrable final avec toutes les fonctionnalités est due au 7 décembre 2021. On doit donc être très organisé, partitionner le projet en sprints avec des livrables et des réunions quotidiennes afin d'atteindre notre objectif et conséquemment démontrer notre sérieux et solidifier nos relations avec l'agence spatiale canadienne.

Contraintes matérielles:

- Le projet requiert l'utilisation de drones Bitcraze Crazyflie 2.1 munis uniquement d'un multiranger et d'un flow-deck.
- La communication entre les drones et la station au sol se fera uniquement grâce à la CrazyRadio PA.
- La station sol doit être un ordinateur munie d'une CrazyRadio qui enverra des commandes haut niveau aux drones et recevra en retour les logs.

Contraintes logicielles:

- L'utilisation de docker est obligatoire. Nous devons pouvoir exécuter la simulation et l'ensemble des autres modules (section 5.4) à l'aide de la commande "*docker-compose*". Nous devons donc nous familiariser avec le docker afin de rendre notre projet beaucoup plus accessible et simple d'usage.

- L’utilisation du simulateur ARGoS comme environnement de contrôle des drones simulés. L’opérateur de la station au sol doit pouvoir se connecter aux drones simulés via la même interface d’utilisateur que pour les drones physiques.

Contraintes de Sécurité:

- Il faut impérativement que chaque personne proche des drones en mouvement soit munie de lunettes de sécurité.
- Le drone doit explorer un environnement où il n’y a pas d’eau. Pour un environnement extérieur, il faut s’assurer que les conditions météo soient convenables. Pour un environnement à l'intérieur, il faut éviter des endroits munis de gicleurs.

1.3 Biens livrables du projet (Q4.1)

a) Preliminary Design Review

Le premier des livrables du projet est la Preliminary Design Review (PDR). Comme l’indique le nom du livrable, c’est une démonstration préliminaire (du design) qui comporte une simple démonstration de nos drones ainsi qu’une interaction avec notre interface Web afin de démontrer notre maîtrise des technologies utilisées.

Ce livrable contient ces deux requis fonctionnels :

- Une interface utilisateur qui comportera les boutons suivant :
    - Identifier
    - Connecter
    - 2 boutons : Lancer la mission et Terminer la mission.
- Chaque bouton permet d’envoyer une requête HTTP de type POST vers le serveur. Ce dernier va intercepter la requête et envoyer un paquet en utilisant le protocole CRTP (AppChannel) dans le cas du drone physique. De même pour Argos, sauf que la communication entre lui et le serveur sera établie à l'aide de TCP socket.
- Chaque drone devra 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.

- Pour ce qui est de Argos, 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

C'est donc un prototype minimal mais primordial car il posera les bases pour le reste de notre projet.

Cet artéfact sera remis le 1 octobre 2021.

b) Critical Design Review

Le second livrable est la Critical Design Review (CDR). Pour cette remise nous aurons la structure complète et presque finale du projet. Nous aurons à répondre aux commentaires formulés par l'agence lors de la Preliminary Design Review. Après quoi nous évaluerons les changements/précisions qui devront être ajoutés.

Les requis fonctionnels qui devront être présents en plus de ceux de la PDR sont:

- L'interface utilisateur devra montrer l'état des drones (attente, en mission…) avec une mise à jour de fréquence minimale de 1 Hz.
- L'interface utilisateur doit être visualisable sur plusieurs types d'appareils (PC, tablette, téléphone…) via réseau. Au moins deux appareils doivent pouvoir se connecter et avoir accès à la visualisation des données pendant une mission.
- Après le décollage, les drones (physiques et simulés) devront explorer l'environnement de façon autonome.
- Les drones devront éviter les obstacles détectés à l'aide de leurs capteurs.
- L'environnement virtuel pour les tests dans ARGoS doit pouvoir être généré aléatoirement.
- Il faut finalement proposer un prototype qui, grâce aux informations récoltées avec les senseurs, puisse générer une carte de l'environnement exploré et la sauvegarde dans une base de données.

Cet artéfact sera remis le 5 novembre 2021.

c) Readiness Review

Le dernier livrable est le Readiness Review (RR). Pour cette ultime remise, nous présenterons le système complet et fonctionnel. La documentation du projet sera mise-à-jour pour refléter ce qui a été accompli par notre équipe.

Nous devrons fournir les démos vidéos de l'ensemble des requis que nous aurons complétés, ainsi que le code nécessaire à l'exécution du projet.

Cet artéfact sera remis le 7 décembre 2021.

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Nous avons décidé de répartir les tâches selon le niveau de familiarité des cinq membres. Cependant, nous avons également essayé de faire en sorte que chacun des membres puissent développer ces points faibles en lui attribuant des tâches plus adaptées et réalisables dans un délai raisonnable.

Nous avons décidé de séparer notre équipe en deux sous-équipes. La première équipe s'occupera de la simulation et du code embarqué pour les drones, tandis que la seconde s'occupera de la plateforme Web. Nous avons également un des membres qui servira d'intermédiaire et de gestionnaire. Ce dernier qui a pour tâche principale d'implémenter le serveur devra avoir une bonne vue d'ensemble du projet.

Simulation/Embarqué : Skander, Aymen.

Plateforme Web : Omar, Driss, Persia,

Serveur: Omar

Ensuite, pour ce qui est de notre organisation en lien avec l'avancement du projet, nous nous réunissons plusieurs fois par semaine (en présentiel ou sur discord) afin de parler des avancées et des problèmes que chacun des membres a pu rencontrer. Ainsi, nous nous mettons à l'abri de tout retard possible. De plus, le fait d'avoir un membre qui travaille avec les groupes permet de toujours garder une vision globale du projet et des avancements des deux

côtés. En combinant cette idée avec des mini-remises (sprint), nous nous attendons à avoir une quantité de travail uniforme tout au long de la session et d'éviter de mauvaises surprises en fin de session.

Enfin, en ce qui concerne le rôle de chacun au sein de l'équipe, nous aurons Omar comme coordinateur de l'équipe. Il s'occupera de mesurer l'avancement du projet et de gérer notre temps de la façon la plus optimale. Nous pouvons donc dire que nous avons une organisation centralisée, malgré que la plupart des décisions soient prises en équipe.

2.2 Entente contractuelle (Q11.1)

L'entente contractuelle pour ce projet est de type contrat livraison clé en main à prix ferme.
Ce choix est basé sur la demande du client et du travail que nous allons fournir.

En effet, dans notre cas, nous fournirons un système fonctionnel avec des requis particulier à une date précise. Le paiement sera émis lors de la livraison et de l'acceptation par le client. Nous ne fournirons pas de conseils ou encore notre expertise pour une période de temps donnée. Nous pouvons donc éliminer l'option d'un contrat à terme. De plus,le type de contrat partage des économies ne convient pas à notre projet car malgré qu'il soit défini sur une période de temps fixe comme le contrat livraison clé en main, nous ne partagerons pas les économies générées par le projet avec l'agence.

Notre projet sera donc de type d'entente contractuelle contrat livraison clé en main à prix ferme.

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)

Le diagramme ci-dessous représente l'architecture générale de notre projet, nous avons décidé de cacher toute la complexité du projet pour obtenir une architecture simple.

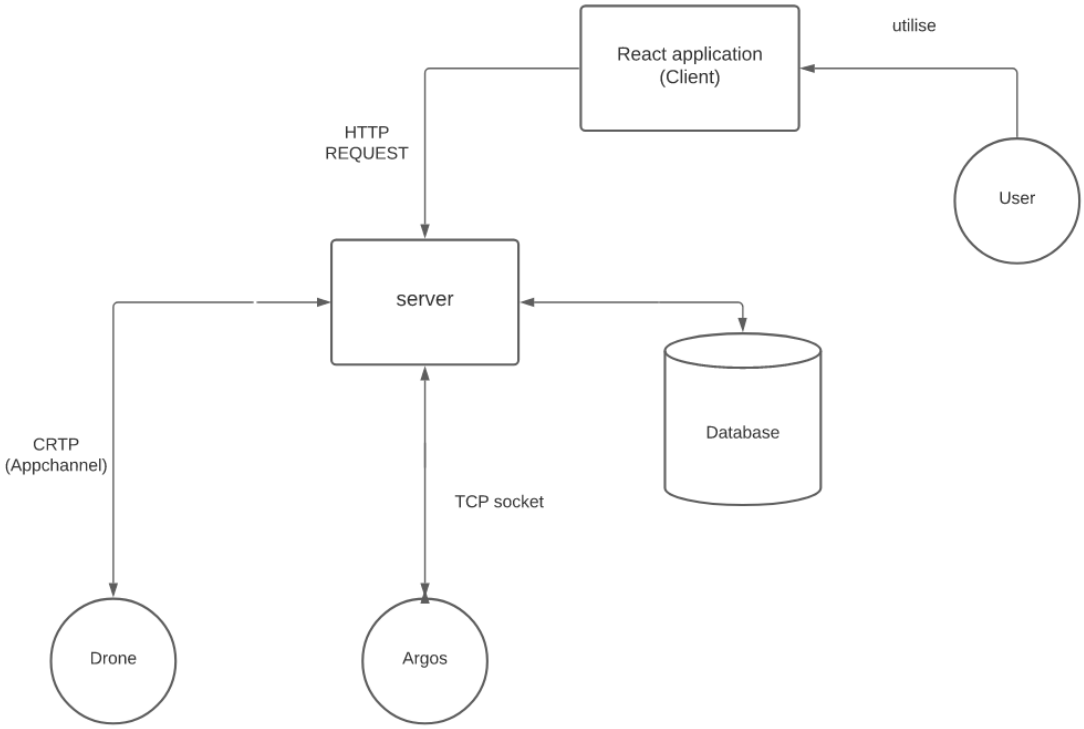


Fig 1 : Architecture logicielle générale

Nous avons opté pour React comme framework pour notre application web, étant donné sa simplicité et sa rapidité. En effet, React permet de créer une application web en la répartissant en de simples components. Il peut créer son propre DOM, appelé Virtual DOM pour améliorer ses performances et détecter les changement des variables **states** et **props**. L'utilisation des **functional components** ainsi que les **hooks** facilite la création de notre application.

Pour ce qui est du serveur (le backend), nous avons choisi d'utiliser Flask pour une raison simple, pour pouvoir importer des classes du module cflib-python. Il est vrai que nous pouvions opter pour Django, mais Flaks s'avère moins complexe.

Nous allons utiliser Firebase pour le stockage des informations nécessaire dans les bases de données que Firebase propose. Firebase DB permet de synchroniser les données et le client en temps réel grâce aux observables qu'il possède.

3.2 Station au sol (Q4.5)

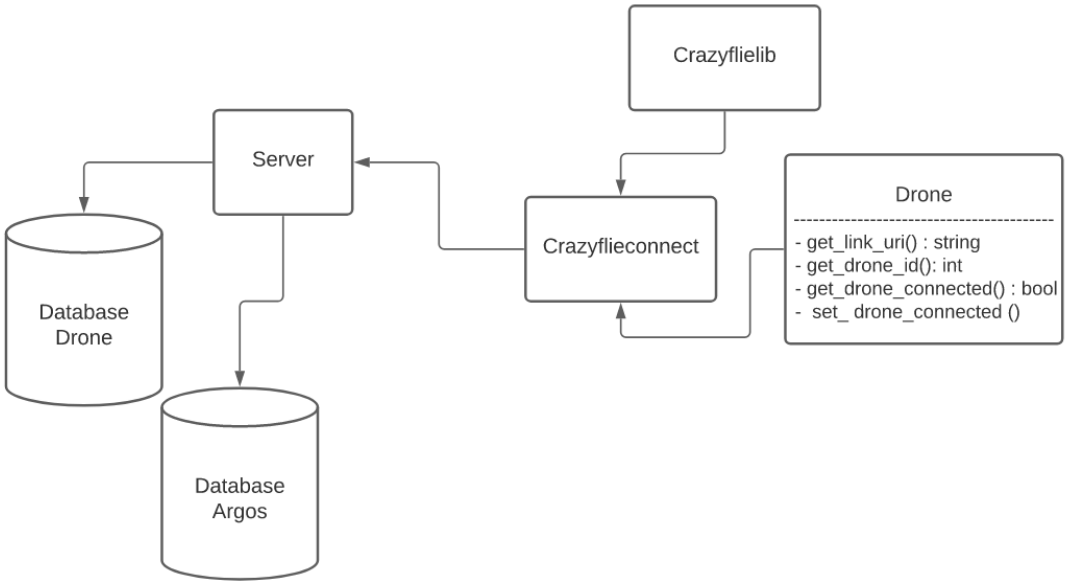


Fig 2 : Architecture station au sol

Notre architecture backend se base sur flask et firebase db. Le module Crazyflieconnect représente le plus grand module de notre architecture backend, c'est ce dernier qui tisse les liens entre le reste des modules, il utilise des librairies prises du Crazyflielib-python. On utilise aussi socketio du côté serveur pour implémenter la communication entre le serveur et argos. Le module server permet de lancer le serveur Flask, et reçoit les différentes requêtes POST et GET qui en les gérant appelleront les méthodes du crazyflieconnect nécessaire pour la simulation d'un drone physique/virtuel ou bien pour envoyer/récupérer des données de la database afin de lancer une ancienne simulation.

Le choix de deux bases de données sont pour séparer les missions d'exploration du drone physique et du drone argos, plutôt que de filtrer une seule base de données. Il y a aussi un avantage au niveau conceptuel, considérant que les explorations des missions physiques et simulées ne sont pas pareilles.

En ce qui a trait aux containers Docker, le client sera dans un container, et le serveur dans un autre; nous utilisons docker compose pour orchestrer ces deux containers.

Les raisons du choix de Flask et Firebase ont été citées plus haut.

3.3 Logiciel embarqué (Q4.5)

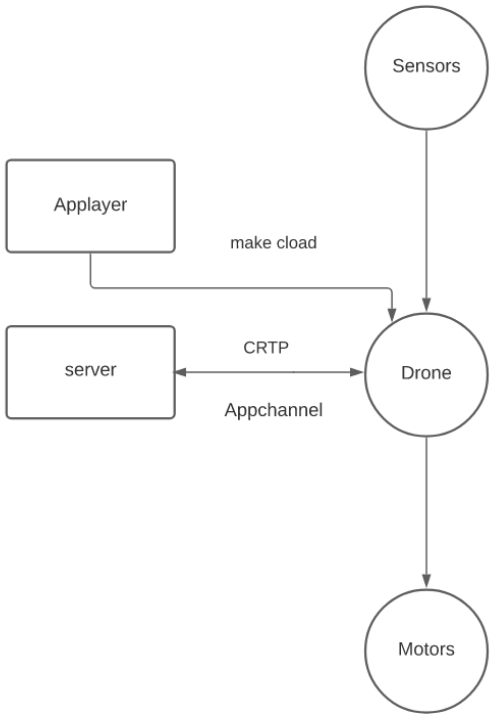


Fig 3 : Architecture du système embarqué avec le drone Crazyflie

Après avoir chargé, notre machine à état est au niveau du drone. Ce dernier devra attendre les commandes de haut niveau qui lui seront envoyées par le serveur, qui lui utilise la librairie Python afin d'envoyer des données avec le protocole CRTP. Bitcraze nous fournit une encapsulation de ce protocole avec Appchannel. Dès que le drone commence sa mission, on peut extraire les données telles que la position, la vitesse (x, y, z, vx, vy, vz), les angles (roll, pitch, yaw) et la batterie avec le module TOC (Table of contents) qui nous permets de récupérer, utiliser et enregistrer les données du drone. Ces données seront affichées au niveau client et utilisées par le Kalman filter state estimator ainsi que la logique qu'on implémenter afin de contrôler les moteurs. Pour contrôler les moteurs on envoie des setpoint. Pour les axes x et y nous utilisons les vitesses qu'on trouve beaucoup plus précises que les positions x et y. qui seront envoyées au commander. Pour l'axe des z on utilise les positions absolues dans la majorité des cas et la vitesse pour l'atterrissage. Finalement pour ce qui est du peer-to-peer, on compte utiliser l'API P2P du crazy-firmware pour établir la communication entre les drones.

Pour l'algorithme d'exploration nous comptons utiliser SGBA [1]. Avec cet algorithme chaque drone aura une direction favorite qu'il continuera à suivre jusqu'à rencontrer un obstacle. Chaque drone aura aussi un identifiant déterminé à l'aide de son adresse. Les identifiants des drones nous informent sur leurs priorités. Le drone avec la priorité la plus haute évitera uniquement les obstacles et ceux avec une plus basse priorité devront éviter les obstacles et les drones plus prioritaires. Pour cette remise nous avons utilisé WallfollowingWithAvoid dans le même répertoire git qui suit exactement la même logique que SGBA mais fonctionne uniquement sur deux drones.

Pour cette remise notre machine a était vraiment simpliste. On utilise les états Start_mission et Stop_Misson afin de modifier les booleans

3.4 Simulation (Q4.5)

ARGoS est un simulateur de robots multi-physiques, qui permet de modéliser des essaims de robots à grande-échelle. Au sein de cette application, le rôle du simulateur dans cette application est de valider par une représentation grâce au module Qt OpenGL.

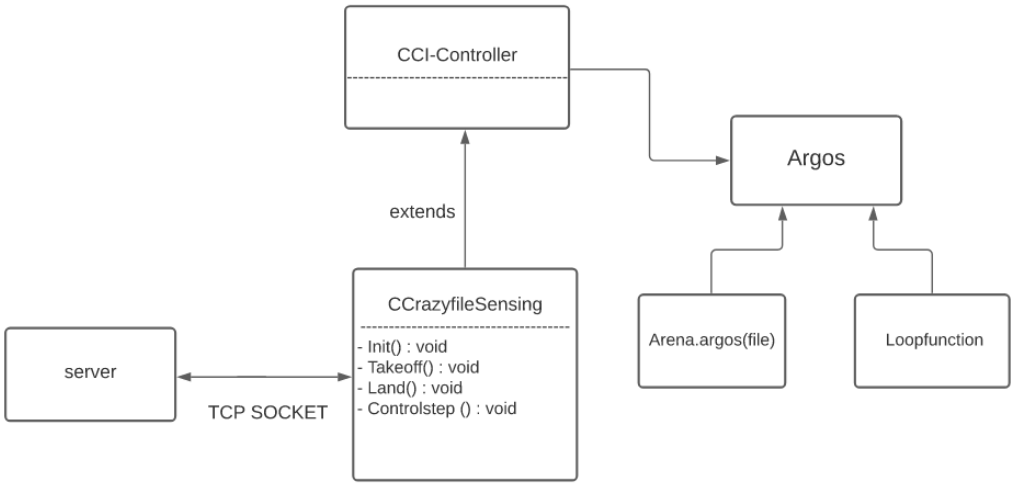


Fig 4 : Architecture logicielle de la solution ARGoS pour la simulation

Afin de modéliser l'application et par souci d'interopérabilité, l'exécution d'ARGoS sera cette fois-ci conteneurisée au moment du lancement de la simulation avec ***x11docker***, afin de bénéficier du GUI pour le simulateur. Le simulateur pourra communiquer avec le serveur ***Flask*** grâce à un socket TCP, dont l'API est déjà fournie par la librairie Socket-IO. Le contrôle du modèle Crazyflie se fera à travers le contrôleur **argos::CCrazyflieSensing**, qui hérite de l'interface **argos::CCI-Controller.** Ce dernier sera combiné avec le fichier de configuration XML **.argos** qui contient la configuration du parcours et la position de départ des drones simulés, et les fichiers de configuration **loopfunctions**.

Nous prévoyons d'implémenter pour l'exploration des drones simulés un algorithme rudimentaire de suivi de murs afin d'éviter les obstacles, avec un parcours conservateur qui sera de plus raffiné pour obtenir une meilleure couverture. Lorsque les drones sont à moins de 30% de leurs charges, ils retroussent leur chemin jusqu'à revenir à la base.

3.5 Interface utilisateur (Q4.6)

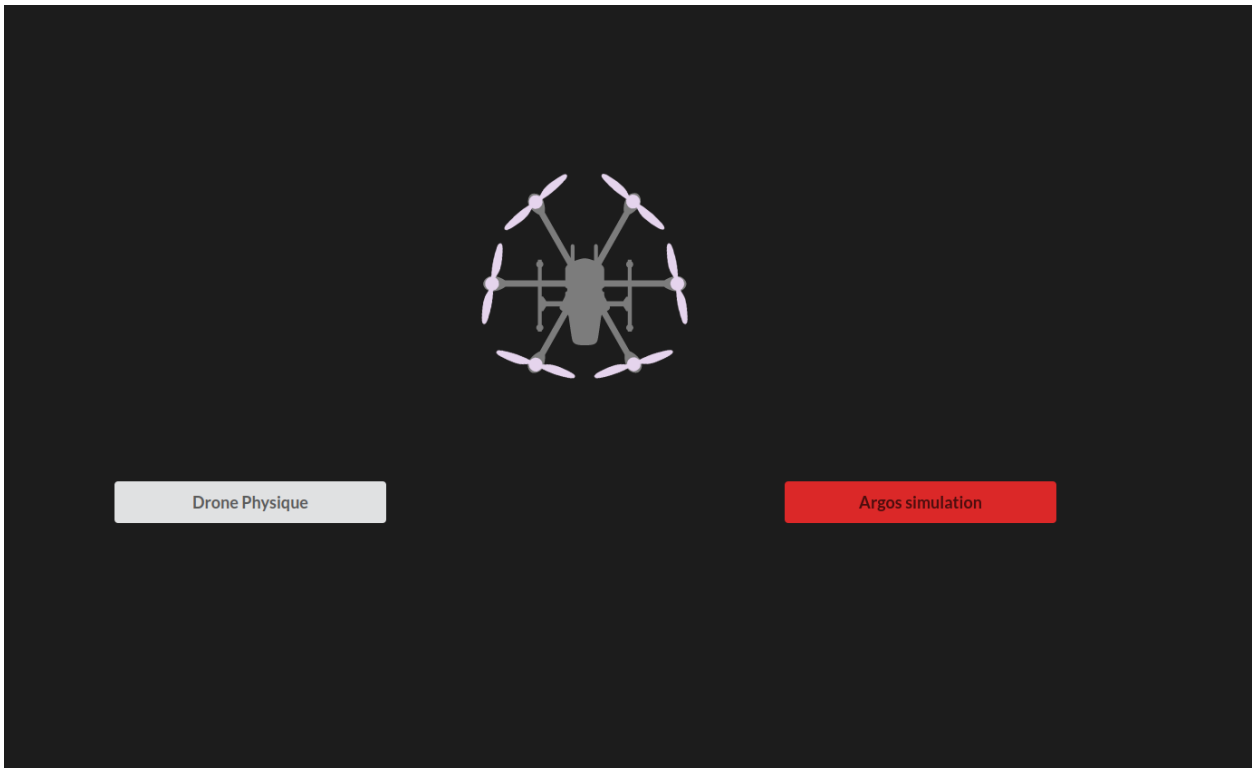


Fig. 5 : Aperçu de l’interface usager

Dès que l’utilisateur lance notre application, il fait face à un drone svg animé ainsi que deux boutons “Argos Simulation” et “Drone Physique”.

Notre interface utilisateur contient 3 grands modules :

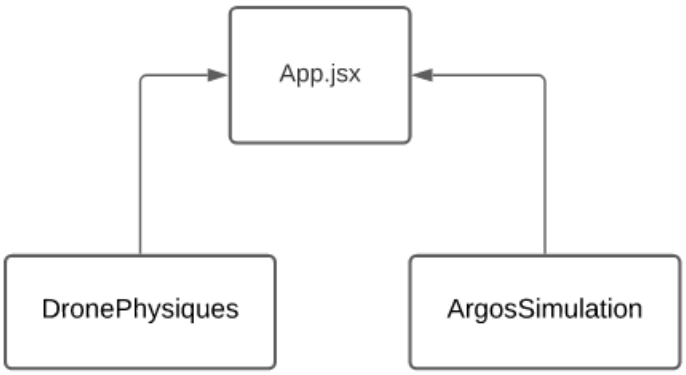


Fig. 6 : Architecture du Client

- Un module App.jsx qui contient les différentes routes de l'application.
- Un module DronePhysiques.jsx qui contient toutes les fonctionnalités dont le drones a besoin ainsi que l'état des drones.
- Un module ArgosSimulation.jsx qui contient toutes les fonctionnalités dont la simulation a besoin ainsi que l'état des drones.

Fig. 7 : Aperçu de l'interface du drone physique

Dès que l'utilisateur appuie sur le bouton "Drone Physique", il est accueilli avec une map qui permet de visualiser les déplacements des 2 drones en temps réel. Nous supposons que nos drones sont toujours disponibles pour démarrer une mission/ l'arrêter ou même faire un retour à la base. Une fois que nous sommes connectés aux drones, les deux carte blanche en haut à gauche de l'interface afficherons l'état des deux drones (en attente, en mission…) ainsi que le pourcentage de la batterie. Finalement, les deux boutons Lancer Logs/ Arrêter Logs permettent d'afficher les logs que nous recevons du drone (position x,y; batterie; range-left, range-right, range-top, range-back).

Fig. 8 : Aperçu de l'interface du drone physique

Fig. 9 : Aperçu de l'interface du drone physique

Pour ce qui est de Argos, dès que l'utilisateur pèse sur le bouton 'Argos simulation', la fenêtre d'argos s'ouvre automatiquement et une connexion socket est établie entre le serveur et argos. Deux boutons sont disponibles sur l'interface: "Lancer simulation et arrêter simulation". Ainsi que deux carte blanche pour afficher l'état des drones simulés (en attente, en mission, ...) et le pourcentage de leurs batteries.

3.6 Fonctionnement général (Q5.4)

Le lancement de l'application se fera grâce à une unique commande **Docker-compose up** qui servira à construire et à lancer les images dockers de tous les services (serveur, client, simulation...) qui composent l'application. En parallèle, un script bash servira à lancer la simulation Argos, se connectera par la suite grâce à son socket TCP.

Ce fonctionnement a pour but d'être efficace, minimaliste et aussi convivial que possible afin d'offrir à l'utilisateur une prise en main rapide. Docker-compose avec les autres scripts Bash serviront à cacher la complexité du reste des composants de l'application.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

Le prix du projet est basé sur un budget ventilé en fonction des lots de travail et des taux suivants :

- développeur-analyste : 130$/h
- coordonnateur de projet : 145$/h

Nous estimons la charge de travail à 11 heures par semaine pour chaque membre pour ce qui relève du développement du projet. Avec 11 semaines de travails on aura donc 11 * 11 = 121 heures par développeur.

$$Coût_{développeur-analyste} = 4 * taux\ horaire_{développeur-analyste} * nombre\ d'heure_{développeur-analyste}$$

Nous estimons donc le montant associé aux salaires des cinq développeurs analystes à :

62 920 $

À ça nous ajoutons 11 heures par semaine pour le coordinateur. Avec 11 semaines de travails on aura 11*11 = 121 heures

$$Coût_{coordonnateur} = taux\ horaire_{coordonnateur} * nombre\ d'heure_{coordonnateur}$$

Nous estimons donc le montant associé aux salaires des coordonnateurs à :

17 545 $

Nous avons donc un total de **80 465$** pour l'ensemble du projet et pour un total de 605 heures de travail.

4.2 Planification des tâches (Q11.2)

Le diagramme suivant indique l'allocation du temps prévu pour chaque tâche.

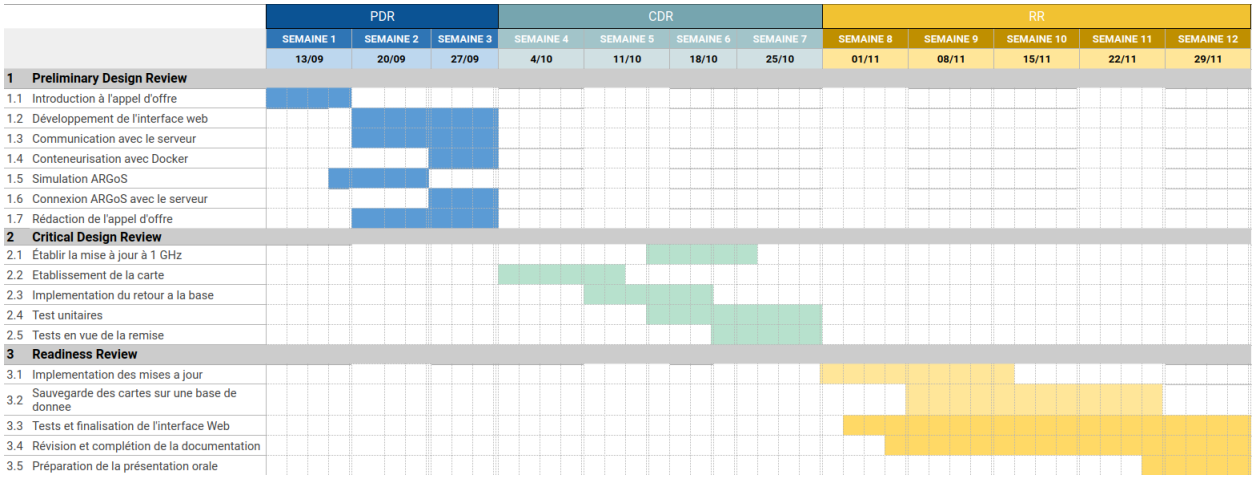
Livrable	Tâche	Coordonnateur **[h]**	Développeurs **[h]**	Total **[h]**
PDR	Familiarisation avec React et conception de l'interface web		10	10
	Connecter le serveur et le client	10		10
	Implémentation du serveur avec Flask	15		15
	Familiarisation avec le crazy		10	10

	firmware			
	Programmation des leds et implémentation d'un exemple start mission		20	20
	Maîtrise du simulateur Argos		20	20
	Dockerisation du projet		5	5
	Implémentation du stat mission par le serveur sur ARGoS		20	20
	Total			105
CDR	Etablir la mise à jour des logs a 1 Hz	10	15	25
	Etablissement de la carte	15	35	50
	Implémentation du retour à la base	15	25	40
	Complétion de la documentation		25	45
	Tests en vue de		25	45

	la remise			
	Implémentation de l’algorithme d'exploration	10	35	45
	Total			290
RR	Implémentation des mises à jour	10	30	40
	Sauvegarde des cartes sur une base de données	10	35	45
	Dockerization de l’application	5		5
	Tests et finalisation de l’interface Web	20	20	40
	Implémentation des explorations dans une pièce quelconque	0	40	40
	Etablissement communication P2P		30	30
	Total			200
Grand Total				605

4.3 Calendrier de projet (Q11.2)

Les dates cibles de terminaison de phases importantes de ce projet sont résumé dans le diagramme de Gantt suivant :



4.4 Ressources humaines du projet (Q11.2)

Notre équipe est composée de cinq membres ayant des atouts dans différents domaines ce qui nous permettra d'avoir un groupe complémentaire avec des spécialistes de chaque technologie utilisée. Ces cinq membres sont les suivants:

1) **Driss Benzekri** possède des expériences professionnelles en développement web ainsi qu'en développement dans les langages C et C++ . Il est particulièrement doué pour le développement frontend. Comme l'ensemble des membres de notre équipe, il est familier et à l'aise avec les logiciels de gestion de version tel que Git.
2) **Persia Shahdi** possède également des expériences en développement web et plus particulièrement l'aspect backend. Il a également une expérience professionnelle en entrepreneuriat et maîtrise bien la gestion de projet.
3) **Skander Soussou** possède des expériences professionnelles en C/C++ et Python. Ce dernier a une passion prononcée pour la robotique et s'est conséquemment proposé pour la partie embarquée du projet.
4) **Aymen Zeghaida** possède de l'expérience professionnelle en matière de développement d'applications sur des systèmes embarqués, et porte un intérêt poussé pour le développement en C++ et d'interfaces d'utilisateurs.

5) **Omar Azizi** possède de l'expérience avec React et les communications entre serveurs et appareils tierces et aussi en C et C++. Ce dernier se démarque par ses qualités inter-relationnelles.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Notre équipe s'engage à fournir un travail de qualité dans un format standardisé suivant les conventions de codages reconnues. L'ensemble des biens livrables seront révisés par un minimum de 2 membres n'ayant pas travaillé sur ces livrables. Ces derniers devront approuver le code et fournir des commentaires constructifs menant à l'amélioration globale des livrables. Cette méthodologie nous permettra d'assurer que le code est clair et compréhensible pour toute personne ayant les connaissances requises.

5.2 Gestion de risque (Q11.3)

Les principaux risques liés à ce projet sont liés à l'aspect technique du projet ainsi qu'aux différentes échéances.

On entend par risques liés à l'aspect technique tous risques qui mènerait à un mauvais fonctionnement de nos drones ou d'une de nos interfaces logicielles. Nous retrouvons donc les risques les plus triviaux tels qu'un bris des drônes ou de leurs composantes. Ces risques seront présents tout au long du projet et sont le plus souvent de nature accidentelle.

Cependant nous avons des pièces de rechange disponibles auprès de l'agence. Les risques liés aux bris du matériel sont donc de faibles importances. Pour ce qui est des risques liés au manque de temps, nous avons décidé de privilégier le plus grand nombre de requis parfaitement fonctionnels, plutôt que plusieurs requis débutés mais non complets. Par la même logique, nous complétons les requis qui nous semblent les plus abordables en premier, tout en gardant en tête le lien qu'il peut y avoir entre les requis.

Notre façon de faire se résume donc à classer les requis par ordre de difficultés selon nos préférences. S'il y a des fonctionnalités à sacrifier, ce sera en fonction du temps; nous mettons la priorité sur un produit qui répond à moins de requis mais qui est fonctionnel, plutôt qu'un produit qui tente de répondre à un plus grand nombre de requis mais avec une qualité moindre, ou encore de manière incomplète.

5.3 Tests (Q4.4)

Nous nous engageons à tester et valider les biens livrables avant la présentation d'un livrable.

- Chaque entrepôt git possède son propre banc de test
- La majorité des tests surtout les nécessaires seront accomplis
- Impossible de compiler s'il ya des erreurs de Lint pour la partie React.
- Plusieurs tests de couverture sur le client (React) avec le framework "Jest" et PyTest pour Flask.
- Pour Argos, nous avons prévu de tester plusieurs parcours générées aléatoirement.

Nous prévoyons aussi déployer une pipeline sur GitLab avec 3 étages (Build, Lint et Test) pour tester l'intégration des différentes branches au moment des merges.

5.4 Gestion de configuration (Q4)

Le système de contrôle de version utilisé est Git que nous utiliserons sur GitLab. Nous allons séparer l'entrepôt en plusieurs sous-modules indépendants, le tout, de la façon suivante.

1) INF3995-Main

Ce module sera l'entrepôt principal et contiendra les autres sous modules. Comme le projet est conteneurisé il suffit d'exécuter le fichier docker-compose pour rouler l'ensemble des conteneurs du projet. On peut donc démarrer l'entièreté de notre projet à partir de cet entrepôt. Ce module contiendra également un fichier README contenant les différentes instructions relatives au projet ainsi que les liens vers nos vidéos démos.

2) INF3995-simulation

Ce module contiendra l'ensemble du code concernant la simulation ARGoS. Il contiendra également un fichier README avec les instructions et différentes commandes nécessaires pour compiler et exécuter la simulation.

3) INF3995-Crazyflie-Firmware

Ce module contiendra principalement le code lié au contrôle de notre drône et un fichier Docker permettant d'obtenir les dépendances permettant de mettre le code du drone à jour.

4) INF3995-client

Ce module contiendra l'ensemble du code créé pour notre application Web. Il contiendra l'interface utilisateur permettant d'envoyer des commandes à notre Crazyflie. Ce module contiendra également un dossier contenant l'ensemble de nos tests unitaires pour le frontend..

5) INF3995-server

Ce module contiendra l'ensemble du code créé pour le serveur de notre application Web. Le backend sera construit en python à l'aide de Flask. Il contiendra également un dossier dédié à nos tests pour le backend.

Le projet utilisera aussi une pipeline pour automatiser les tests afin d'adhérer aux principes de l'intégration et du développement continue (CI/CD).

5.5 Déroulement du projet (Q2.5)

[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.]

Lors de la remise de la PDR, nous avons réussi à satisfaire les exigences liées à la documentation du projet. Cependant, nous avons complété un seul des deux requis pour la démo. La simulation n'avait pas été complétée par manque de temps. Le retard pour la PDR a été rattrapé par la suite et le comportement global attendu lors de la PDR a été accompli quelques jours plus tard. Malheureusement, nous avons encore accumulé du retard lors de la période des examens ce qui nous a obligé à passer plus de temps lors des derniers jours avant la remise et nos estimations quant aux tâches restantes étaient mauvaises. En effet, nous

pensions pouvoir rattraper ce retard et avoir tous les requis fonctionnels pour la remise ce qui n'est pas le cas. Nous devons donc rattraper ce retard au plus tôt pour ne pas se retrouver dans la même situation pour la remise finale. Nous prévoyons de faire plus de réunions et de rencontres chaque semaine pour pouvoir suivre l'avancement de plus prêt et ainsi éviter de passer trop de temps sur une seule et même tâche.

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

Les requis RF1, RF2, RF3, RF4 et RF5 sont fonctionnels.

Cependant, nous éprouvons par moment des problèmes lors de la simulation pour le requis RF2.

Référence:

[1] SGBA App layer. [En ligne]. Disponible: https://github.com/tudelft/SGBA_CF2_App_layer