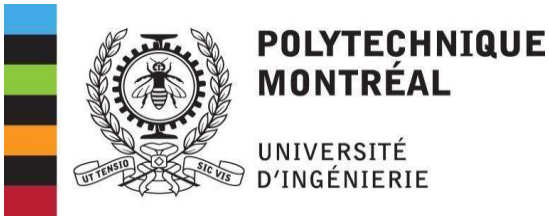


Département de génie informatique et génie logiciel

# INF3995

## Projet de conception d'un système informatique

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

***Conception d'un système aérien d'exploration***

Équipe No 101


| Nom et Prénom | Signature |
|---|---|
| Paul Marie Akoffodji | P.M |
| M. Ibrahim Majid | M.I.M |
| Manel Mokrani | M.M |
| Erika Fossouo | E.F |
| Patrick Cobanovic | P.C |


07 Décembre 2022

# Tables des Matières

ANNEXES **40**

# 1. Vue d'ensemble du projet

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

Le but final du projet est de concevoir un ensemble de logiciels opérationnels permettant à un essaim de drones miniatures d'explorer une pièce ou un compartiment de manière autonome et de rapporter des métriques de vol à l'opérateur au moyen d'une interface web. En effet, le drone doit répondre à bon nombre de critères et de requis d'ordre matériels, logiciels et fonctionnels. Ces derniers doivent être présentés ultérieurement à l'aide de vidéos démonstratives. L'utilisateur doit donc pouvoir interagir avec une interface utilisateur afin de lancer ou donner des instructions. Ces dernières peuvent être données aux drones physiques ainsi qu'aux drones simulés.

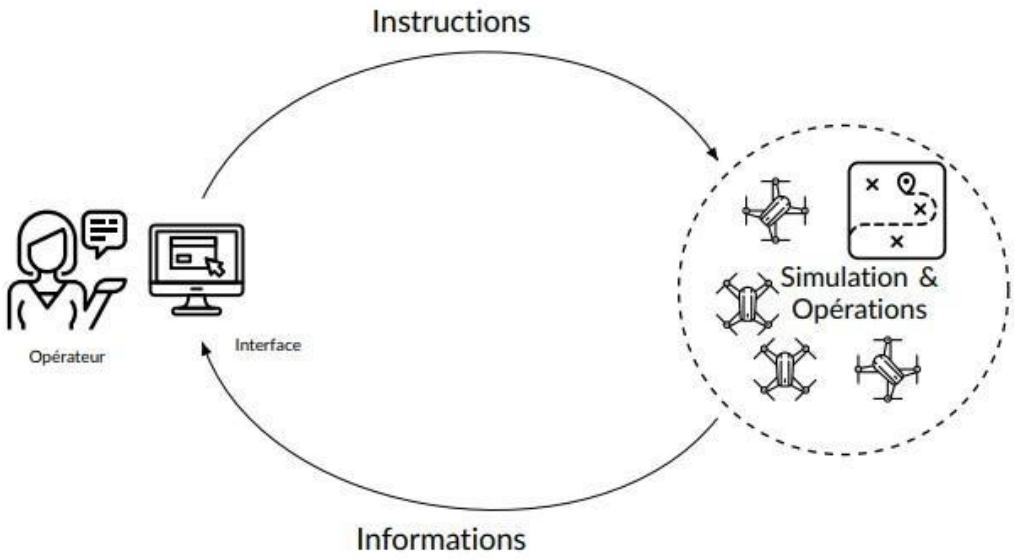


Figure 1 : Schéma de communication globale (Moodle INF3995, 2022)

En ce qui concerne les objectifs, la réalisation du projet nécessite la conception de trois grandes parties : la station au sol, la partie embarquée sur les drones physiques et la partie simulée avec Argos.

À long terme, le projet permettra d'offrir des options de solutions dans le cadre de l'exploration spatiale de terres inconnues, comme Mars, en se basant sur de nouvelles approches d'exploration, comme l'utilisation d'un essaim de drones pour optimiser l'exploration de ces territoires.

## *1.2 Hypothèses et contraintes (Q3.1)*

**1.2.1-** Dans le cadre du projet, plusieurs hypothèses seront faites :

- Les drones seront utilisés à des fins d'exploration dans un cadre académique et ne seront pas utilisés à d'autres fins.
- On pourra supposer que les drones voleront dans un espace fermé sécuritaire avec la présence de murs et d'obstacles.
- On assumera qu'au moins un drone de l'essaim est toujours en communication avec la station au sol.
- En trouvant la solution pour opérer sur deux drones il serait aisé de l'implémenter sur dix drones.

**1.2.2** - En termes de contraintes matérielles et techniques

- Les drones ne doivent pas décoller avec un niveau de batterie inférieur à 30%.
- Le retour à la base doit rapprocher les drones de leur position de départ pour qu'ils soient à moins de 1 m de celle-ci.
- L'environnement simulé doit avoir un minimum de 3 murs (sans compter les 4 murs extérieurs)
- Toutes les composantes logicielles, sauf celles embarquées sur le drone physique, doivent être conteneurisées avec Docker.
- Le seul moyen de communication entre la station au sol et les drones physiques doit être une Bit craze Crazyradio PA connectée à la station au sol.

- Il se pourrait que les drones physiques aient des problèmes matériels, comme des senseurs ou des moteurs qui fonctionnent mal, des hélices brisées, etc.

- Des problèmes matériels avec les machines virtuelles ou les systèmes d'exploitation

**1.2.3** -En termes de contraintes liées aux ressources humaines

- La période de relâche pour les membres de l'équipe qui est souvent synonyme de préparations pour les intras
- L'abandon du projet par un ou plusieurs membres de l'équipe.
- Les scénarios de maladies empêchant les membres de l'équipe de travailler sur leurs tâches.
- La fatigue accumulée au cours de la session susceptible de diminuer le rendement des membres de l'équipe
- Les TPs et les remises des autres cours qui pourraient empiéter sur le temps consacré au projet.
- La fin de session imminente qui coïncident avec les périodes de révision d'examens
- Le retard accumulé pour certaines tâches du fait d'incompréhensions

## *1.3 Biens livrables du projet (Q4.1)*

Tout au long du projet, nous allons livrer des livrables spécifiques à différents moments. Voici le récapitulatif de ces dates et une brève présentation du contenu de ces livrables.

| Date | Titre du livrable | Description du livrable |
|---|---|---|
| Vendredi 30 Septembre 2022 à 23h55 | (PDR) Preliminary Design Review | - Plan du projet<br>- Présentation de l'architecture globale<br>- Prototype préliminaire |
| Vendredi 04 Novembre 2022 à 23h55 | (CDR) Critical Design Review | - Prototype avancé<br>- Interface web évoluée avec la présence des commandes complémentaires<br>- Présence de la base de données conservant les métriques de vol<br>- Cartographie des surfaces explorées |
| Vendredi 07 Décembre 2022 à 23h55 | (RR) Readiness Review | - Implémentation de toutes les fonctionnalités<br>- Plan et résultats des tests<br>- Retour sur l'apprentissage continu |

# 2. Organisation du projet

## *2.1 Structure d'organisation (Q6.1)*

Pour les besoins du projet et au vu de la charge de travail nous avons décidé de conserver notre structure de départ en autres c'est-à-dire les binômes et d'avoir des personnes en renfort pour épauler les binômes ou de travailler individuellement sur des petits modules du projet. Cependant pour la remise du CDR nous avions travaillé sur les différents modules de manière individuelle . Il en résulte l'assignation de rôles suivante:

- Majid : Coordonnateur de projet

**Responsabilité principale** : Logiciel embarqué des drones, Serveur
**Responsabilités annexes**:

- ❖ Scrum Master
- ❖ Planification des rencontres et répartition des tâches

- Paul Marie : Analyste développeur

**Responsabilité principale**: Logiciel embarqué drones,
Serveur
**Responsabilités annexes**:

- ❖ firmware
- ❖ Rédaction artefacts de projet

- Manel : Analyste développeuse

**Responsabilité principale:** Simulation Argos
**Responsabilité annexe**: Docker

- Patrick : Analyste développeur

**Responsabilité principale** : Interface utilisateur
**Responsabilité annexe** : firmware

- Erika : Analyste développeuse

**Responsabilité principale**: Interface utilisateur
**Responsabilité annexe** : Simulation Argos

### *2.2 Entente contractuelle (Q11.1)*

Pour le compte du projet, nous avons finalement opté, après rétrospection, pour le type de contrat livraison clé en main (prix ferme). En effet, nous devons travailler sur le projet de A à Z et livrer le produit final. Les requis sont déjà établis et un changement de ces derniers ne peut se faire lors du développement. De plus, nous devons penser à notre architecture et établir un plan clair et précis de cette dernière. Les changements si nécessaires devraient être discutés entre les parties prenantes ce qui nous a poussé a opté pour ce type de contrat.

## 3. Description de la solution

### *3.1 Architecture logicielle générale (Q4.5)*

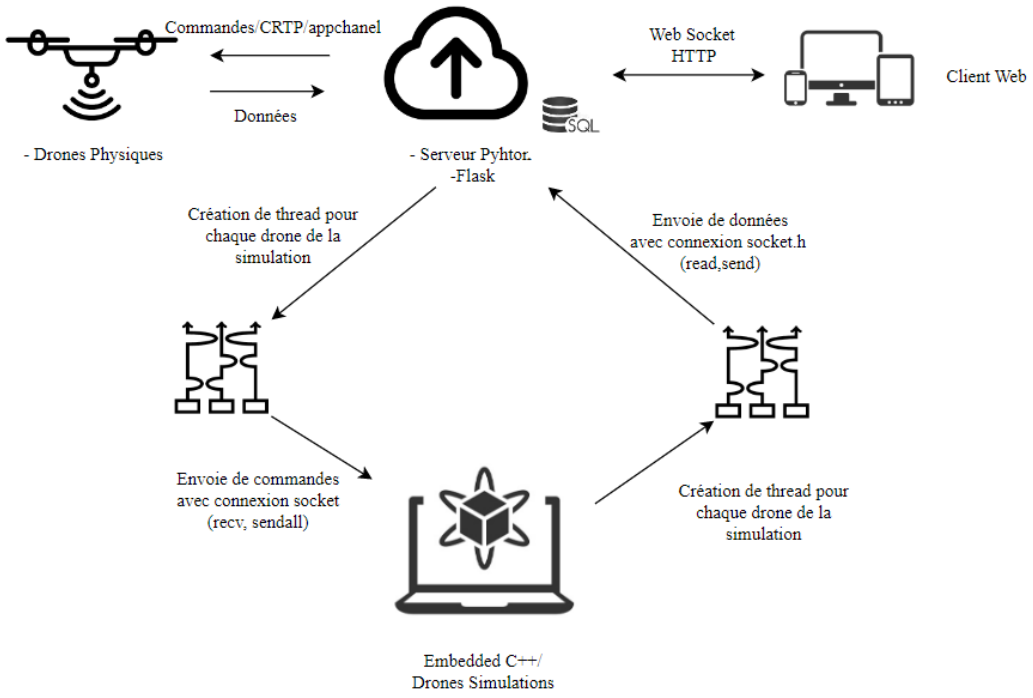


Figure 2 : Schéma Architecture logicielle générale (Paul Marie, 2022)

Ci-dessus le schéma de l'architecture globale de notre système en partant d'une vue générale. On aura a priori quatre grands modules:

- **Le client web** : représenté par l'opérateur qui à travers l'interface web donne les instructions de haut niveau aux drones et choisit le type de support pour le lancement de la mission (physique vs simulation). Pour l'implémentation de cette partie, nous avons choisi d'utiliser le framework Angular associé au typescript, car ils sont familiers pour tous les membres de l'équipe.
- **Le serveur backend python** : Il aura pour rôle de faire le lien entre les actions de l'opérateur via l'interface web (clique de boutons) et le système de communication des drones physiques et de la simulation. Ce qui permettra d'envoyer des commandes aux drones physiques et simulés (commencer mission, retour à la base, etc.) De plus, le serveur recueille les informations collectées par les drones physiques et les drones de simulation afin de les afficher dans l'interface web conformément au requis R.C.1, via la communication web socket-IO. Il sera aussi en mesure de conserver les données liées à l'historique des missions effectuées dans une base de données et dans un fichier json pour l'historique des logs de missions. Nous avons choisi d'utiliser le framework Flask pour notre serveur.
- **La station de drones de physique:** Cette station est composée de deux drones physiques, ces derniers communiquent des informations destinées au serveur web. Ils reçoivent également des instructions provenant du client web, par l'intermédiaire du serveur.
- **La station drones simulés :** Cette station est composée de deux à dix drones simulés. Chaque drone explore l'arena tout en évitant les obstacles qu'il pourrait rencontrer et envoie des données propres à ce dernier (état du drone, numéro de drone, position du drone en x y et z)

grâce à des sockets. Les drones reçoivent aussi des commandes de la station au sol.

## *3.2 Station au sol (Q4.5)*

Notre système (station au sol) est séparé(e) en trois modules distincts : le client, le serveur et le CrazyRadio PA. La station au sol a pour objectif d'assurer la communication avec les drones physiques, les drones simulés et l'utilisateur.
Le serveur redirige les événements provenant de l'utilisateur à travers l'interface utilisateur et y réagit, notamment grâce aux événements que nous avons insérés avec les sockets. De même, les drones physiques, via l'interface de la classe app-channel.h dans le firmware[1], pourront envoyer et recevoir des paquets en provenance du serveur. Le client dispose d'un service de Sockets qui contient une méthode 'sendMessage'injectée dans le component command.ts. Cette fonction envoie un événement particulier en direction du serveur dépendamment du mode choisi (physique/simulation). Une fois l'événement envoyé, il est capté dans le serveur. Dans le fichier app.py, les fonctions on-connect et on_initialize sont exécutées dès le lancement du serveur afin d'établir la connexion avec le serveur Flask Python et de connecter les drones physiques également. L'événement envoyé par le client est réceptionné dans le serveur. S'il est en direction des drones physiques, il fait appel aux fonctions de la classe Drone.py. Un paquet est envoyé grâce à la méthode 'appchannel.send_packet' vers les drones physiques avec comme information un code d'identification de la commande à réaliser qui sera récupéré et traité dans le code embarqué par la suite. Si l'évènement est en direction des drones de simulation, ils seront gérés de leurs bords. Nous ne ferons pas cas du détail de ce processus dans cette section. Le fichier Log.py contient une des méthodes permettant de recevoir les logs en provenance des drones physiques, ainsi que les drones simulés. Ces données sont ensuite stockées dans les fichiers fichier.json correspondant au mode en cours dans le serveur.

Dans le backend une base de données NO-SQL Mongo non relationnelle a été implémentée pour la sauvegarde des historiques de missions et des métriques de vol. Nous avons opté pour ce choix car les contraintes d'exigences en termes de sécurité des données étaient négligeables. Par le biais des requêtes HTTP, des informations sont envoyées et extraites du serveur Mongo afin d'être affichées dans l'interface utilisateur notamment à travers le service data-service.ts du côté client, de plus un contrôleur de routes a été implémenté dans le fichier app.py pour nos requêtes.

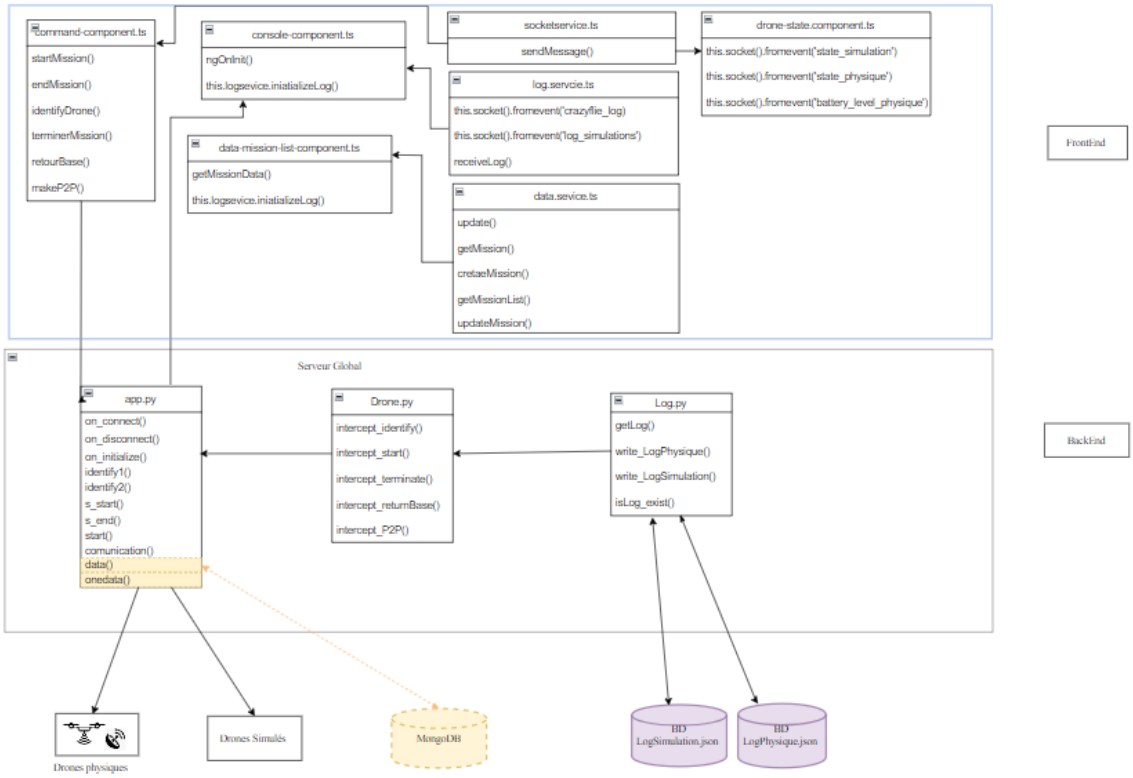


Figure 3 : Diagramme de classe de la station au sol (Paul Marie, 2022)

Le CrazyRadio PA permettra d'établir la communication radio entre les drones et la station au sol conformément au requis R.M.4 et permettra à ce dernier d'envoyer des informations et des paquets qui seront sauvegardés dans la base de données conformément au requis R.F.8

La station au sol sera également en mesure de conserver les logs liées aux missions effectuées deux fichiers json distincts également comme montré dans la figure grâce aux fonctions d'écriture write_LogSimulation() et write_LogPhysique(). De plus, les données liées aux cartes d'exploration élaborées par les drones devront être enregistrées sur la station au sol au sein de la base de données Mongo, cela répondra aux requis R.F.17 et R.F.18.

## *3.3 Logiciel embarqué (Q4.5)*

Le code embarqué ainsi que le code d'exploration du produit final seront implémentés sur les boards des drones en utilisant l'API de BitCraze. Afin de répondre aux requis R.L.1 et R.L.3. L'utilisation du «ranging deck» et du «Flow deck V2» sera primordiale dans cette partie de la structure logiciel du code embarqué, afin de permettre aux drones de détecter les objets ainsi que les obstacles de manière autonome grâce aux métriques offertes par les senseurs montés sur les drones physiques conformément aux requis R.M.3 et R.F.5.

Un système de machine à états pourrait également nous permettre à ce niveau de définir une approche d'exploration pour nos drones physiques. Une succession d'états et d'actions sera faite dépendamment de certaines situations dans lesquelles les drones se trouvent. Cela entraînera des transitions entre les états afin de permettre aux drones de varier leurs actions et d'effectuer des mouvements spécifiques dans la suite du projet tout en effectuant l'exploration.

De plus, nous allons essayer de nous baser sur le principe du "wall-following" afin de rendre les mouvements du drone autonomes conformément au requis R.F.4.

Toute cette logique sera donc mise sur les drones directement .

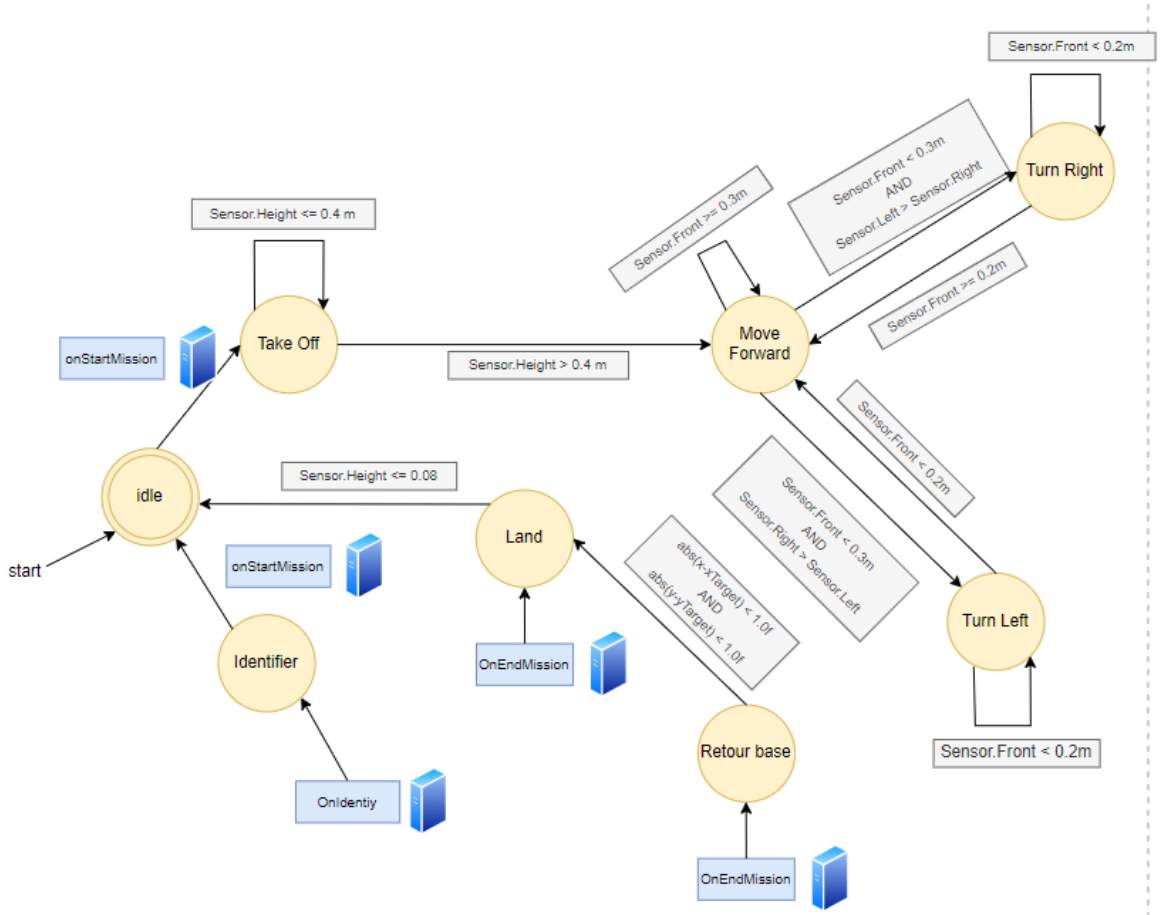


Figure 4 : Machine à états logiciel embarqué ( Majid ,2022)

Le schéma ci-dessus présente la machine à état régissant les mouvements du drones dans l'espace et modélisant le comportement attendu. Le schéma proposé au PDR ne comprenait pas les interactions avec le serveur et n'était pas au goût de la réalité.
Le senseur avant sera le moyen de détection principal d'obstacle du drône. Lorsqu'il détecte un obstacle, il va effectuer soit une rotation de 90 degré dans le sens antihoraire ou bien une rotation de 90 degré dans le sens horaire. L'idée est qu'il fasse face autant que possible au sens de son mouvement. La décision de virer à gauche ou à droite se prend selon que l'obstacle à gauche est plus éloigné que celui à droite ou vice versa.

Le serveur, au travers des paquets, peut forcer le drône à se mettre dans un état. À titre illustratif, si le drône est actuellement en train d'avancer (**MoveForward**) et qu'il reçoit un paquet "**Terminer la mission**", il amorce alors la phase d'atterrissage (changement d'état). Dans la version ajournée pour le RR, il répondra également à la commande "**Retour à la base**' où il va tenter de se diriger vers son point de départ.

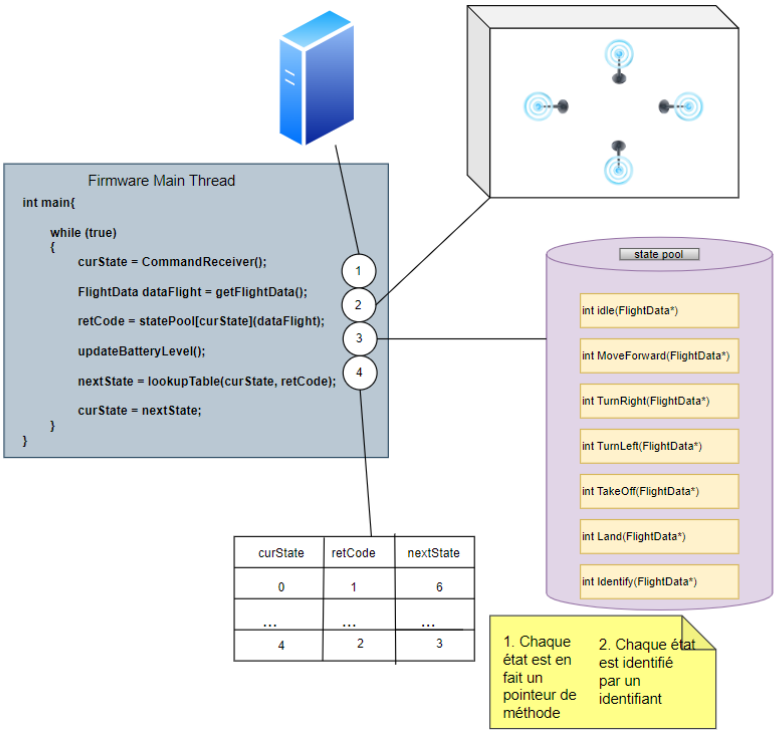


Figure 5 : Schéma logiciel embarqué (Majid, 2022)

La figure 4.2 est un schéma qui poursuit les points abordés à la section 4.1.

La transposition de machine à état à l'implémentation en C se fait de la façon suivante :

- Chaque état est un pointeur de méthode qui prend en paramètre les données de vol actuel (position, senseur) pour sortir un résultat;
- Les résultats possibles sont : REPEAT (rester dans l'état courant à la prochaine itération de la boucle de rétroaction) , DONE (passer au prochain état), FAIL (échec)
- etat_suivant = f(etat_courant, retCode)
- Le drone écoute et échange des messages avec le serveur à travers commandReceiver()
- Le drone va tester le voltage de sa batterie afin de savoir s'il doit retourner à la base

### *3.4 Simulation (Q4.5)*

Le rôle de la simulation est très important dans le projet. Il est primordial, car il permet de tester sur demande les fonctionnalités de la station au sol et de pouvoir tester conceptuellement les algorithmes de contrôle pour les drones avant de les déployer sur les drones physiques.

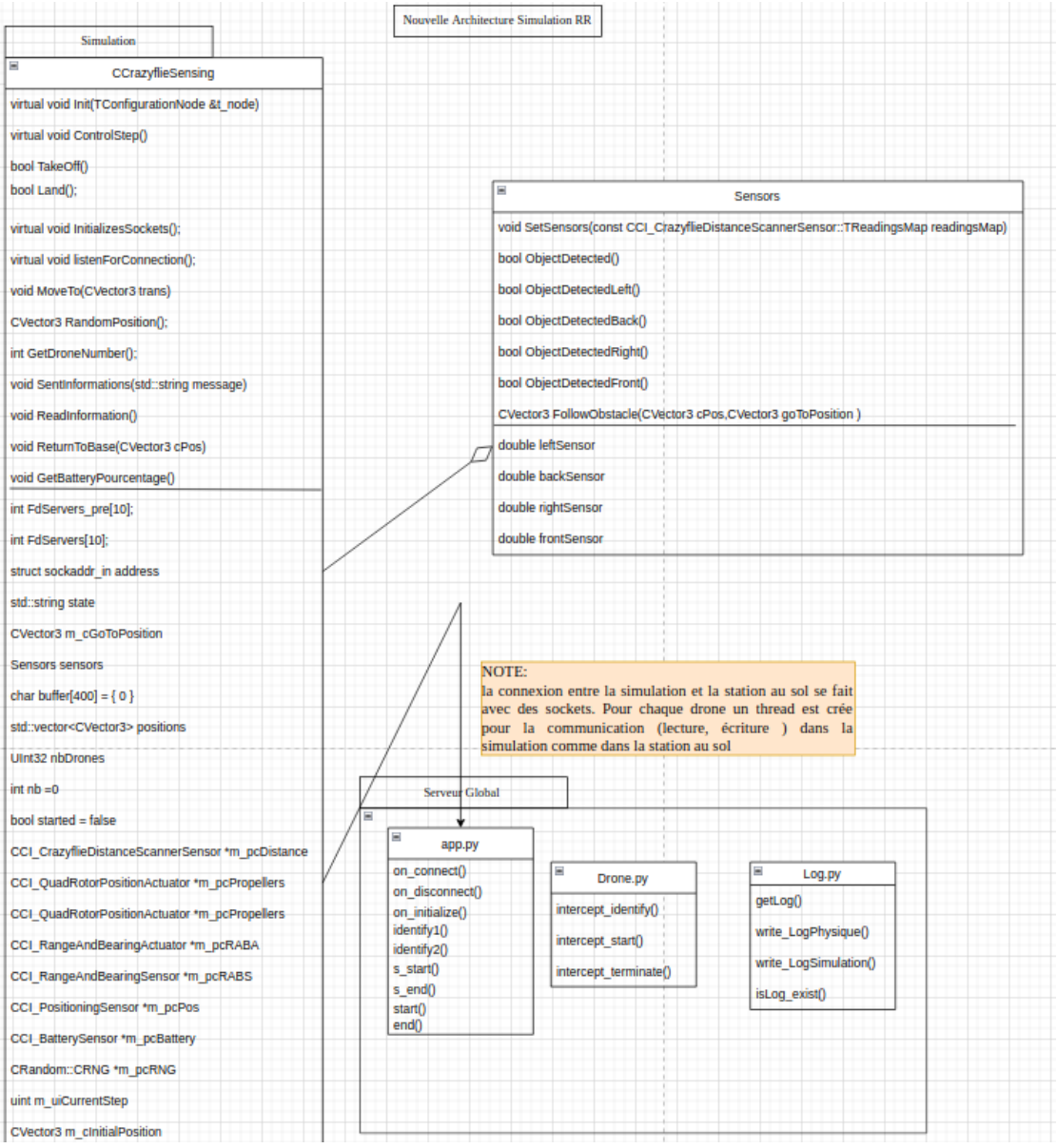


Figure 6 : Schéma Simulation ( Manel, 2022)

Les fichiers principalement modifiés dans la simulation sont : *crazyflie_sensing.h*, *crazyflie_sensing.ccp, sensors.h, sensors.cpp et crazyflie_sensing.argos.*

**Classe CCrazyflieSensing** : Cette classe comporte les fonctionnalités principales de la simulation.

*Méthode Init* : initialise plusieurs attributs de la classe, comme le tableau des descripteurs de fichier de socket de chaque drone, l'initialisation des sockets pour chaque drone, le nombre de steps, etc.

*Méthode ControlStep* : méthode appelée en boucle à chaque step qui comporte la logique principale de la simulation (la commination des états et des logs, le random walk, le décollage, l'atterrissage,etc)

*Méthode TakeOff* : permet de faire décoller les drones [R.F.2][R.F.4][R.F.5]

*Méthode Land* : permet de faire atterrir les drones [R.F.2][R.F.4][R.F.5][R.F.6][R.F.7]

*Méthode InitializesSockets* : créer un socket pour chaque drone puis un thread pour permettre la communication sans bloquer le programme principal [R.F.2][R.F.3][R.C.1][R.F.6][R.F.7][R.F.8][R.F.9][R.F.10] [R.F.17]

*Méthode listenForConnection* : permet d'attendre pour une connection du backend à la simulation (lors de l'envoi d'une commande) [R.F.2][R.F.3][R.C.1][R.F.6][R.F.7][R.F.8][R.F.9][R.F.10] [R.F.17]

*Méthode MoveTo* : permet de diriger les drones vers la position passée en argument [R.F.4][R.F.5]

*Méthode RandomPosition* : retourne un vecteur position avec un dx et un dy entre -1 et 1 [R.F.4][R.F.5]

*Méthode GetDroneNumber* : retourne le numéro du drone

*Méthode SentInformations* : permet d'envoyer les états et les données du drone [R.F.2][R.F.3][R.C.1]

*Méthode ReadInformation :* permet de lire les commandes qui arrivent de la station au sol

*Méthode ReturnToBase* : implémente l'algorithme du retour à la base (voir plus bas)

*Méthode GetBatteryPourcentage :* récupère la valeur la batterie, la modifie pour un pourcentage et retourne le pourcentage de la batterie

**Classe CCrazyflieSensing :** Cette classe gère tout ce qui est senseur des drones et algorithme du random walk.

*Méthode SetSensors* : cette méthode permet de récupérer les valeurs des senseurs et de les affecter au attributs correspondant [R.F.4][R.F.5]

*Méthode ObjectDetected* : cette méthode retourne True si n'importe lequel des senseurs détecte un obstacle. Elle retourne False sinon [R.F.4][R.F.5]

*Méthode ObjectDetectedLeft:* cette méthode retourne True si le senseur de gauche détecte un obstacle. False sinon [R.F.4][R.F.5]

*Méthode ObjectDetectedBack:* cette méthode retourne True si le senseur arrière détecte un obstacle. False sinon [R.F.4][R.F.5]

*Méthode ObjectDetectedRight:* cette méthode retourne True si le senseur de droite détecte un obstacle. False sinon [R.F.4][R.F.5]

*Méthode ObjectDetectedFront:* cette méthode retourne True si le senseur d'avant détecte un obstacle. False sinon [R.F.4][R.F.5]

*Méthode FollowObstacle:* cette méthode implémente l'algorithme d'évitement d'obstacles [R.F.4][R.F.5]

Toutes les fonctionnalités de la simulation sont fonctionnelles avec deux à dix drones. Le nombre de drones doit être spécifié dans le fichier .argos avant démarrer la simulation. Lors du démarrage de la simulation, l'attribue *nb_drones* est récupéré du fichier *experiments/crazyflie_sensing.argos* et stocké dans une variable. La station au sol récupère le nombre de drones de la simulation. Pour ce faire, une première connexion est faite avec la simulation via socket pour demander le nombre de drones. Le nombre de drones, qui a déjà été récupéré et initialisé dans la simulation, sera alors envoyé via socket vers la station au sol.

Puisque la vue de la page simulation de notre application web s'adapte en fonction du nombre de drones nous faisons un socket.emit pour pouvoir récupérer le nombre de drones dans nos différents services et components.

L'utilisation de thread a été choisie pour permettre la communication avec des sockets entre la simulation et le backend. En effet, puisque les sockets sont bloquants, il nous faut un système pour pouvoir attendre les requêtes du serveur sans bloquer la simulation.

Pour ce qui est de l'algorithme de random walk pour la simulation il est comme suit.

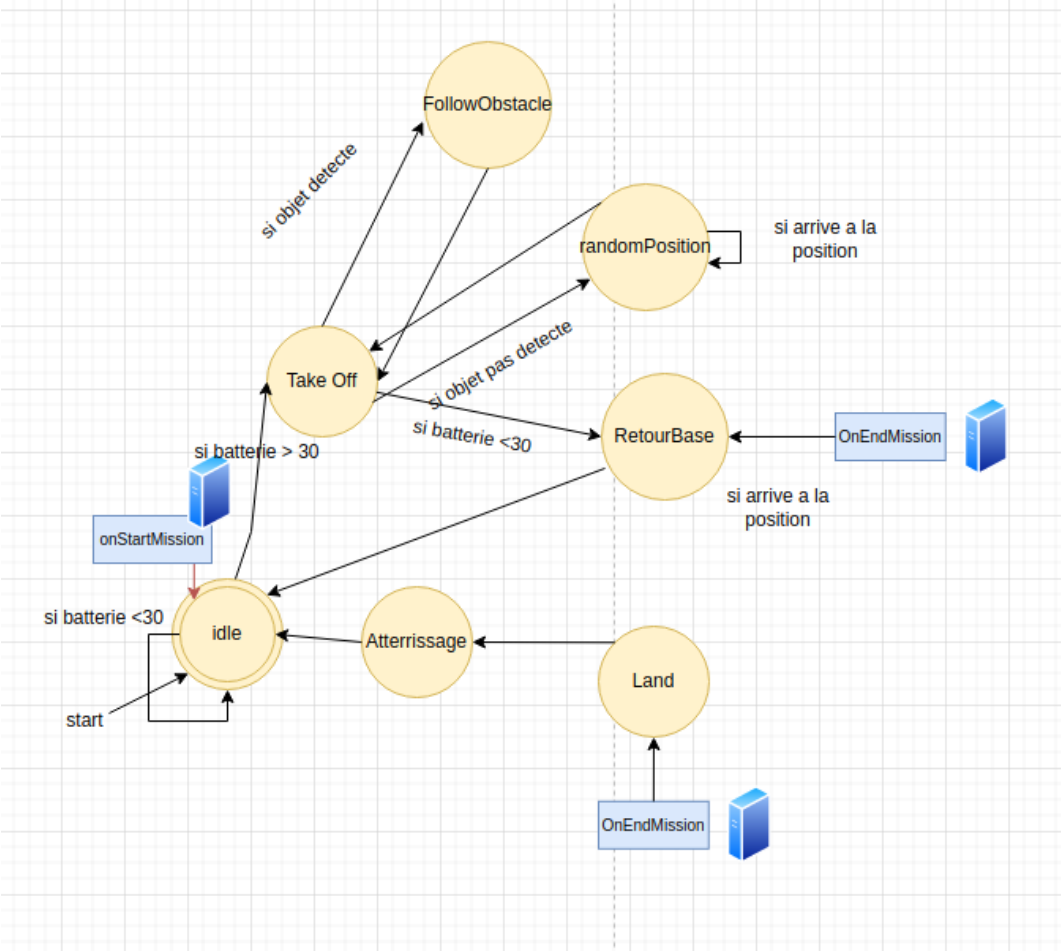


Figure 7 : Machine à état simulation (Manel, 2022)

Pour le retour à la base un vecteur est créé pour stocker toutes les positions où le drone s'est rendu. Une fois le button Retour à la base appuyé ou lorsque le niveaux de batterie devient inférieur à 30 lors d'un vol, le retour à la base est déhanché. Le vecteur de positions est alors lu à partir de la fin et le drone se dirige vers la position lue. Le drone retourne donc sur ses pas et refait son parcours à l'envers jusqu'à ce qu'il arrive à sa position de départ pour enfin se poser.

## *3.5 Interface utilisateur (Q4.6)*

Pour l'architecture relative à l'interface utilisateur, nous avons opté pour l'utilisation du framework Angular assez bien documenté et facile à prendre en main. Il sera utilisé afin de produire l'interface web pour le client de notre application. Nous avons opté pour ce framework, car comme évoqué précédemment, la plupart des membres de l'équipe sont assez familiers avec ce dernier, car ils l'ont utilisé tout au long du projet intégrateur 2.

L'interface comportera les boutons correspondants aux différentes commandes de haut niveau à envoyer aux drones notamment ‘'Lancer la Mission'', "Identifier", "Retour à la base", "Terminer Mission' et “Peer to peer”'. Dépendamment de l'option choisie dans l'interface, on peut envoyer ces commandes aux drones physiques ou aux drones simulés. Le choix du mode (physique ou simulation) sera déterminé par l'utilisateur à l'aide d'un menu où il aura alors le choix entre les deux modes. Cela permettra de répondre aux requis R.F.1, R.F.2 et R.F.6.

Dans cette interface, on aura accès à l'état des drones déployés (en mission, accident ..). En plus de leur état, on pourra voir l'identifiant des drones ainsi que leur niveau de batterie. Cela permettra de compléter les requis R.F.3 et R.F.13. De plus, il sera également possible de voir une console de débogage avec les logs de la mission dès lors que cette dernière est lancée. Ces logs comprenant toutes les informations de débogage nécessaires au développement seront

disponibles en tout temps via une fenêtre dynamique dans l'interface utilisateur afin de répondre au requis R.C.1.

Enfin, sur la même page, on retrouvera une carte en deux dimensions qui représente la carte d'exploration des drones, physiques ou simulés selon le mode choisi. Cette dernière permet d'observer en temps réel la position du drone dans son environnement mais aussi d'observer l'environnement lui-même (notamment avec la représentation des obstacles rencontrés par le drone lors de son exploration). Un menu permettra de choisir la carte du drone que l'on souhaite observer et il sera évidemment possible de choisir de passer d'une carte à l'autre pendant la mission. Ce dernier élément de l'interface, sur cette page du moins, permet de répondre au requis R.F.8.

À partir de la page principale, on pourra, en choisissant la database dans le menu, nous diriger vers la page contenant toutes les informations concernant les missions antérieures. Tout d'abord, on peut voir une liste des missions ou on peut observer plusieurs attributs en lien avec chaque mission. On peut voir le temps auquel la mission a débuté, sa durée, le nombre de drones qui ont été utilisés ainsi que la distance parcourue par les drones. De plus, on a la possibilité de trier les missions selon tous les attributs. Il y a aussi deux boutons pour chacune de ces missions. Le premier était supposé être un bouton pour afficher la map de la mission et le deuxième bouton est un bouton delete qui enlève la mission dans la base de donnée. En dessous de cela, on pourra aussi voir une liste des missions antérieures effectuées avec la date et l'heure de la mission. Lorsqu'on clique sur une de ces dates, il y a une liste déroulante des logs de cette mission si jamais on doit faire du débogage. Cela permettrait de répondre aux requis R.F.17 et R.F.18.

L'image ci-dessous représente ce à quoi les deux pages de l'interface utilisateur devrait ressembler sur un ordinateur roulant le client.

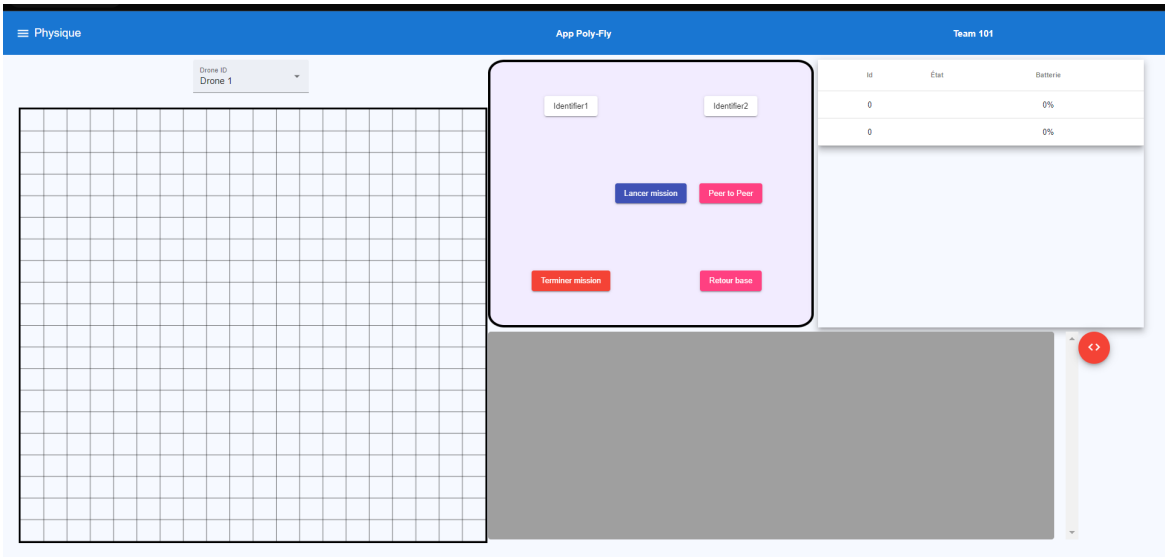


Figure 8 : Schéma page principale Utilisateur (Équipe 101, 2022)

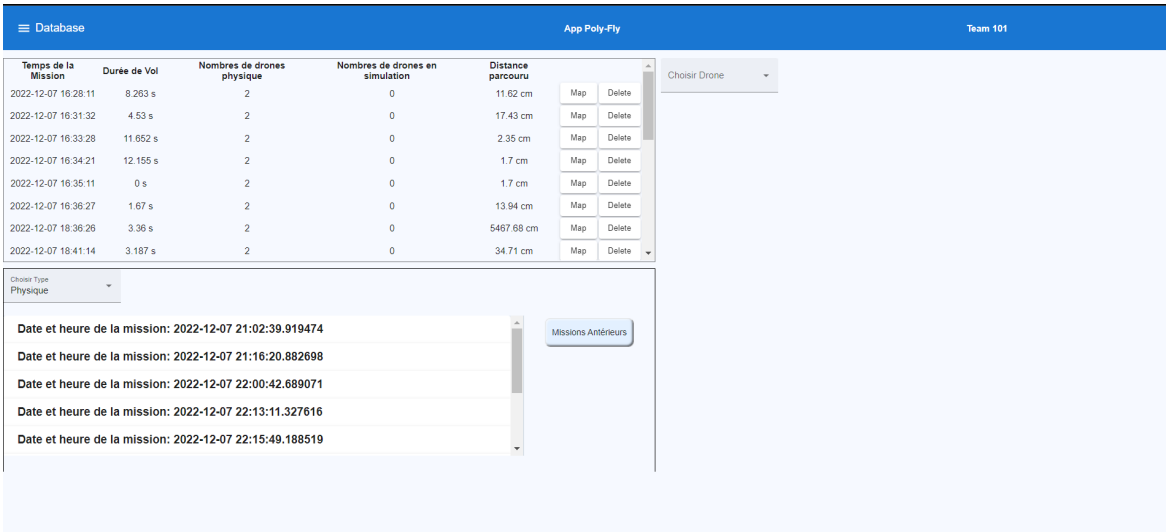


Figure 9 : Schéma page database Utilisateur (Équipe 101, 2022)

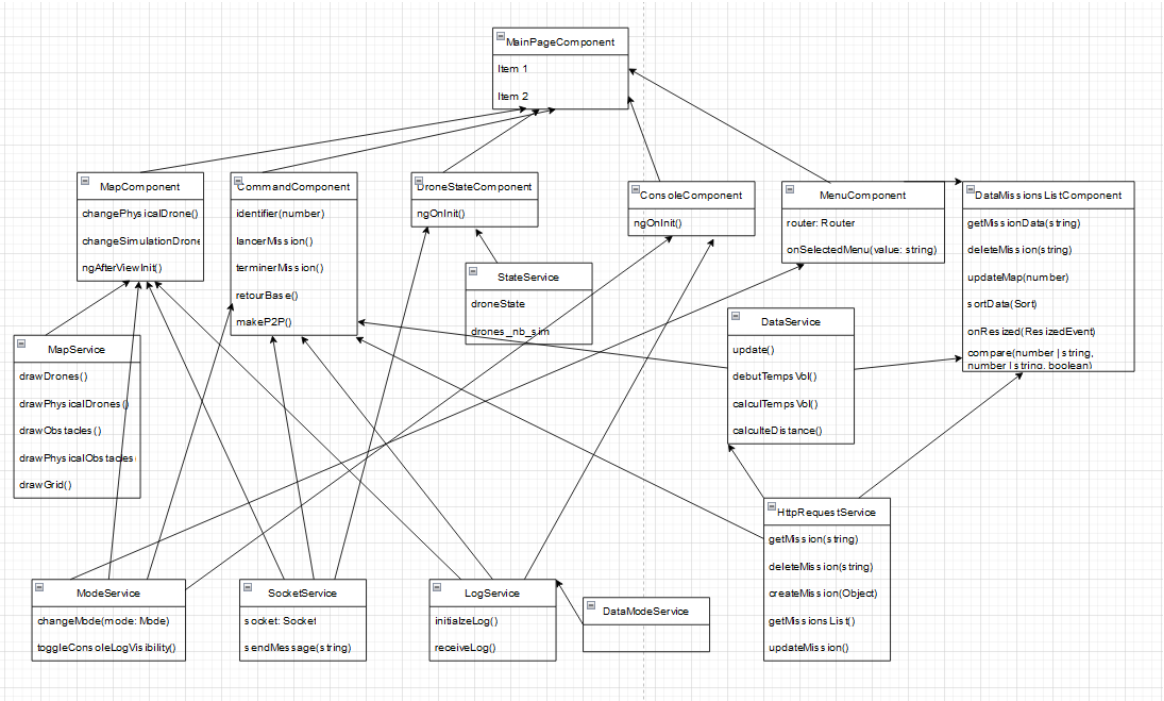


Figure 10 : Diagramme de classe de l'interface utilisateur (Erika, 2022)

## *3.6 Fonctionnement général (Q5.4)*

Afin de lancer notre système, idéalement nous aurions voulu la commande '**'sudo docker up -d** '' permettra de rouler directement tout le projet. Le client web sera accessible à travers le port 80 et le serveur quant à lui sera accessible sur le port 5000. L'utilisation d'un laptop équipé d'Ubuntu serait aussi recommandée afin de pouvoir tester et rouler le projet sans problème. Cependant nous avons pour la remise pu faire fonctionner les dockerfiles de nos modules de façon individuelle.

Lancer les composantes individuellement :

*Lancer le client :*

1 - Se déplacer dans le dossier client *cd client*

2 - lancer Angual avec *ng serve*

*Lancer le serveur :*

1 - Se déplacer dans le dossier serveur *cd serveur*

2 - Lancer Flask avec *flask run*

*Lancer la simulation :*

1 - Se déplacer dans le dossier serveur *cd simulation*

2 - Build l'image docker de la simulation docker *build -t simulation:latest*

3 - Run un container avec la nouvelle image

```docker run -it \
   --env="DISPLAY" \
   --env="QT_X11_NO_MITSHM=1" \
   --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" \
   --publish 8080-8090:8080-8090
   simulation:latest\
```

NOTE : si une erreur de display s'affiche il suffit d'exécuter sur un terminal

*xhost +local:root*

NOTE2 : pour changer le nombre de drones dans la simulation, il faut modifier le fichier .argos à deux endroits <params nb_drones = ""> et la place habituelle des nombre de drones.

# 4. Processus de gestion

## *4.1 Estimations des coûts du projet (Q11.1)*

En supposant que chaque membre du groupe passe en moyenne six heures lors des laboratoires et cinq heures en moyenne hors des séances de laboratoire sur le projet on pourrait aboutir à l'estimation des coûts ci-dessous sur une période de trois mois soit douze semaines :

- Développeur-analyste : 130$/h
- Coordonnateur de projet : 145$/h

| Membre d'équipe | Salaire hebdomadaire | Salaire total |
|---|---|---|
| 145$/h | 145$/h * 11h = 1595$ | 19 140$ |
| 130$/h | 130$/h * 11h = 1430$ | 17 160$ |
| Salaire Finale Équipe : 19140$ + (17160$ * 4) = 87 780$ | | |

Après rétrospection nous avons pu nous conformer à cette estimation jsuqu'à la fin du projet. Et nous n'avons pas de changements importants par rapport à cette section.

## *4.2 Planification des tâches (Q11.2)*

L'avancement des tâches va se faire avec l'outil de gestion de projet Gitlab. Chaque requis choisi sera documenté sous forme de "issues" et aura un poids synonyme du niveau de difficulté possible. Grâce à la plateforme, on sera en mesure d'assigner les issues puis de fixer des dates d'échéances. Le projet aura trois principaux milestones assimilables à des sprints correspondant aux échéances de remises des trois livrables. Le premier "milestone" (sprint 1) servira à répondre aux requis du PDR et certaines fonctionnalités de base en autres. Les prochains milestones (Sprint 2 et Sprint 3) seront entre autres alloués à la réalisation des requis restants pour la remise des deux derniers livrables : CDR et RR. La répartition des jalons reste arbitraire et pourrait changer éventuellement en fonction de l'avancement du projet et des contraintes qui se présenteront à nous.

| Tâches | Jalon | Temps alloué | Personne responsable |
|---|---|---|---|
| R.F.1 | | 04 semaines | Majid /Paul Marie |

| | | | |
|---|---|---|---|
| R.F.2 | Sprint 1 | 04 semaines | Majid,Manel,Paul Marie, Erika |
| R.F.3 | | 04 semaines | Majid |
| R.F.4 | | 04 semaines | Majid/Paul Marie |
| R.F.5 | Sprint 2 | 04 semaines | Manel,Majid |
| R.F.10 | | 04 semaines | Team Server |
| R.C.1 | | 04 semaines | Majid, Paul Marie, Patrick |
| R.C.3 | | 04 semaines | Manel |
| R.Q.1 | | 04 Semaines | Toute l'équipe |
| R.Q.2 | | 04 Semaines | Toute L'équipe:Tests.pdf |
| R.F.6 | Sprint 3 | 04 semaines | Majid/Manel |
| R.F.7 | | 04 semaines | Majid/Manel/Paul Marie |
| R.F.8 | | 04 semaines | Erika/Majid |
| R.F.9 | | 03 semaines | Erika |
| R.F.12 | | 03 semaines | À venir |
| R.F.11 | | 03 semaines | Erika/Majid/Manel |
| R.F.13 | | 03 semaines | Majid |
| R.F.17 | | 04 semaines | Patrick/Paul Marie |
| R.F.18 | | 03 semaines | Patrick/PaulMarie |
| R.F.19 | | 04 semaines | Majid |

| | | | |
|---|---|---|---|
| R.C.2 | | 03 semaines | Majid/Manel |
| R.C.4 | | 03 semaines | Toute l'équipe |
| R.C.5 | | 03 semaines | Manel |

### *4.3 Calendrier de projet (Q11.2)*

Nous pourrons présenter le calendrier du projet à travers le diagramme de Gantt. Ce diagramme nous permettra d'avoir en premier un aperçu global du projet d'une part, il renseigne également sur l'état d'avancement des tâches, permet d'avoir un aperçu de l'allocation des ressources pour chaque tâche, et d'avoir une meilleure approche de gestion de projet. Dans le diagramme ci-dessous, les losanges bleus font référence aux dates cibles de remises du PDR, CDR et RR. Les jalons en vert, bleu et rouge représentent les tranches de temps allouées à la réalisation des requis pour le compte des sprints 1, 2 et 3. Suite aux PDR cet aménagement des tâches à du changer pour des raisons de retard au niveau de la planification il en résulte donc cette nouvelle planification des tâches.

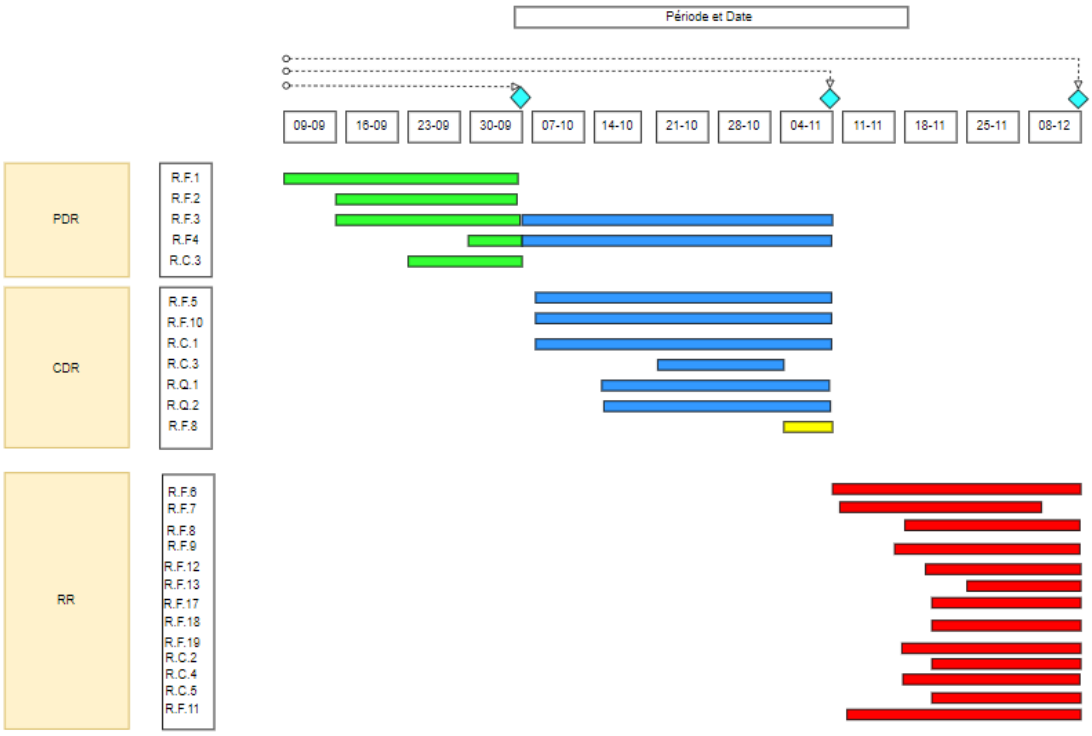


Figure 11 : Schéma Diagramme de Gantt (Paul Marie , 2022)

## *4.4 Ressources humaines du projet (Q11.2)*

L'équipe est composée de cinq membres possédant chacune des compétences techniques et relationnelles acquises à Polytechnique et aussi grâce à leur expérience en stage. On comptera au sein de l'équipe :

**Paul Marie Akoffodji**
Paul Marie en est à sa troisième année de baccalauréat en génie informatique. Il a de l'expérience avec C/C++. Il a aussi de l'expérience en réalisation d'artéfact de conception logiciel grâce à son stage effectué en équipe de projet avec Giro.

**Ibrahim M. Djido Abdoul Majid**
Majid en est également à sa troisième année en génie informatique, il a de l'expérience en programmation orientée objet C/C++. Il a également travaillé

avec Polystar où il s'est familiarisé avec les systèmes de contrôles embarqués notamment.

**Manel Mokrani**
Manel en est également à sa troisième année en génie informatique, elle a de l'expérience avec C/C++. De plus elle a de l'expérience avec la programmation graphique cela pourrait être un atout exploitable au cours du projet.

**Erika Fossouo**
Erika en est également en troisième année de génie informatique est a de l'expérience en C/C++. De plus, elle a des connaissances avec le langage python appliqué au backend.

**Patrick Cobanovic**
Patrick est en quatrième année de génie informatique, il a de l'expérience en C/C++ et Angular. De plus, il a également des connaissances en réseaux informatiques plus spécifiquement en architectures technologiques de transmissions et en base de données.

# 5. Suivi de projet et contrôle

## *5.1 Contrôle de la qualité (Q4)*

Tout au long du projet, nos livrables seront soumis à un processus de révision continu aussi bien au niveau du code que sur les produits livrables. Pour le code python nous allons utiliser la norme de style "PEP 8 – Style Guide for Python Code" [2] afin de nous assurer que nos noms de classes, de fonctions , de variables soient conformes aux standards de programmation. Pour le code typescript dans le client et pour le Framework Angular, nous allons opter pour la norme ECMAScript afin de rencontrer les standards de programmation. Cela nous permettra de répondre au requis R.Q.1. Pour le code embarqué, nous allons adopter Doxygen [4] comme convention pour le formatage et la

présentation. Chaque module du client et du serveur sera vérifié et possédera des tests unitaires précis décrivant le protocole de validation de ce dernier. Quant à l'interface web, il devra répondre aux dix règles heuristiques de l'interface usager web. L'intégration de chaque fonctionnalité sur la branche principale devra passer par une merge request avec, de manière générale, deux approvers et autant de reviewers que disponible. Enfin, tous les livrables seront revus en groupe avant soumission afin de garantir la soumission d'un travail élaboré.

## *5.2 Gestion de risque (Q11.3)*

*Au* cours du projet, plusieurs situations ou problèmes d'ordre technique, matériel, ou de gestion peuvent arriver compromettant ainsi la réalisation de ce dernier dans de bonnes conditions. Le tableau ci-dessous résume certains problèmes que nous pourrions rencontrer ainsi que quelques approches de solutions. L'échelle du niveau d'impact est basé sur l'estimation du risque associé à chaque problèmes et leurs répercussions. Plus la valeur d'impact sera élevée plus le risque associé à ce dernier l'ai également.

| Problèmes | Niveau d'impact | Approches de Solutions |
|---|---|---|
| - Le manque de familiarité des membres de l'équipe avec les nouvelles technologies (Agros, Docker,Crazyflie, Client) selon les versions du système d'exploitation | 4 | - Procéder à un briefing en groupe des problèmes de logiciels rencontrés<br>- Se familiariser avec la documentation de ces nouvelles technologies et lien pertinents |
| - Manque de Cohésion d'équipe et de communication sur l'avancement des tâches entre les membres | 4 | - Instaurer un climat de coopération dans l'équipe<br>- Promouvoir la saine émulation des membres de l'équipe |
| - Bris matériels au niveau des drones physiques lors des tests en volière | 3 | - Veiller à utiliser le drone physique uniquement à l'intérieur de la volière<br>- Manipuler le drone avec tact et précision |
| -Bris physiques sur les opérateurs lors des tests avec les drones en volière | 3 | - Utiliser l'équipement de sécurité fourni par l'entreprise contractante (lunettes de sécurité,etc) |

| | | |
|---|---|---|
| -Mauvais choix d'architecture ou de patron de conception logiciel pour le projet rendant difficile l'implémentation des fonctionnalités spécifiques | 3 | - Choisir les bons patrons de conception, éviter les anti patrons<br>- Faciliter la réutilisabilité du code |
| -Mauvaise gestion du temps alloué à certaines tâches et retard dans les remises de fonctionnalités | 4 | - Ne pas sous-estimer la complexité d'une fonctionnalité<br>- Analyser les contours et besoins que l'implémentation d'une fonctionnalité nécessite<br>- Donner des rétroactions périodiques sur l'avancement des tâches |

### *5.3 Tests (Q4.4)*

Pour chaque sous-système seront réalisés.finalement à la place des tests unitaires ils sont décrits en détail dans le fichier Tests.pdf

- Pour le matériel, on pourrait procéder à une vérification visuelle des drones physiques hebdomadairement, afin de voir si toutes les pièces sont en norme afin de s'assurer qu'il n'y a pas de pièces défectueuses susceptibles de compromettre le vol de ces derniers.

- On pourrait tester également le fonctionnement des boutons 'Identifier Drone', 'Lancer Mission', 'Terminer mission', 'Retour à la base' et 'Mise à jour' dans l'interface afin de s'assurer que nos drones sont identifiables.
- On a également pu tester l'envoi des données en continu de nos drones concernant leurs états ainsi que quelques métriques de vol depuis la station au sol aussi bien en simulation que sur les drones physiques.

## *5.4 Gestion de configuration (Q4)*

Pour ce qui est de la gestion de configuration, notre code sera disponible sur notre repos GitLab sur la branche dev. Nous comptons séparer le projet en quatre gros modules: un dossier pour le web client, un dossier pour le serveur, un dossier pour la simulation avec Argos et un dernier dossier pour le firmware embarqué pour nos drones physiques. Des commentaires seront mis, si nécessaire, dans le code également afin de renseigner sur certaines fonctions et fichiers. Des fichiers de README.md seront également ajoutés à la racine du projet afin de donner des informations sur la compilation des fichiers et ainsi que les commandes Docker pour rouler les conteneurs.

## *5.5 Déroulement du projet (Q2.5)*

Au cours du déroulement du projet, pour la remise du CDR nous pourrions affirmer que nous avons travaillé de façon ardue suite aux commentaires reçus après le PDR. Nous avons aussi changé la dynamique d'équipe afin de responsabiliser toute l'équipe et équilibrer la charge de travail. Au niveau des requis, nous avions validé complètement certains requis comme l'identification des drones, le lancement ou encore l'arrêt d'une mission. D'autres requis étaient validés à un niveau assez avancé comme l'exploration et l'évitement des obstacles, que ce soit en simulation ou avec les drones physiques. La plupart de nos tests fonctionnels passaient.

Nous avons malheureusement fait fasse à certains imprévus notamment le jour de la remise avec les drones physiques. Les decks de nos deux drones n'étaient pas conformes et fonctionnaient de manière erratique. Cependant nous avons sauvé des vidéos de nos fonctionnalités qui étaient correctes mais pas tournées directement à la volière. En ce qui concerne les fonctionnalités non implémentées

comme la carte d'exploration ou les fonctionnalités moins abouties comme l'exploration des drones finalement, c'est dû au temps passé sur un nombre restreint de fonctionnalités qui ne fonctionne plus correctement. Le temps passé à essayer de réparer ces fonctionnalités quand même essentielles a fini par nous manquer pour les autres.

Suite à la remise du PDR, nous avions revu notre calendrier de projet de la section 4.3 parce qu'il y a eu plusieurs imprévus que nous avions vu venir également dont notamment l'arrivée des intras qui a bousculé un peu l'avancement des tâches et provoqué un peu de retard. Cependant l'équipe s'est plus ou moins rattrapée en donnant le maximum sur les requis fonctionnels à rendre. L'effort alloué a été conséquent afin de réaliser le maximum de requis. Les problèmes d'ordre matériel et technique nous ont beaucoup ralenti et pénalisé indirectement et nous déplorons le manque de support parfois des chargés bien qu'ils essaient de faire le maximum.

Pour le RR nous sommes partis avec de nouvelles bases. Nous avons pu faire assez de requis mais certains requis n'ont pas pu être testés totalement pour des causes d'ordre matériel avec les drones d'une part et par manque de temps d'autre part. Certaines fonctionnalités sur les drones physiques ont été implémentées, d'autres pas finalement terminées, mais tous les requis ont été implémentés en partie. La fin de session cumulant avec la remise de projet puis les présentations nous ont fait travailler de façon ardue ces derniers jours. Mais nous rendons un travail assez satisfaisant au vu de l'effort investi pour le compte de la remise finale.

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

## *6.1 Requis fonctionnels*

R.F.1: Les drones physiques répondent correctement à la commande "identifier" lancée à partir de l'interface utilisateur. Les DELs des drones clignotent à la réception de cette commande.

R.F.2: Les commandes "Lancer mission" et "Terminer mission" sont disponibles sur l'interface utilisateur pour les deux modes et marchent bien sur les deux drones physiques comme simulés celà est démontré également dans les vidéos.

R.F.3: Un tableau contenant des informations sur les drones est disponible dans l'interface utilisateur pour les deux modes et est mis à jour à chaque 1 Hz minimum.

- simulation: Lorsque l'état d'un drone change, le tableau est mis à jour à la seconde près.
- physique: Lorsque l'état d'un drone change, le tableau est mis à jour à la seconde près.

R.F.4: Les drones doivent explorer l'environnement de façon autonome.

- Dans la simulation, les deux drones se lèvent et se déplacent de manière autonome nous avions une vidéo pour les deux drones et pour dix drones le fonctionnement est effectif

- Pour les drones physiques, nous avions une vidéo de l'exploration mais elle n'est pas tournée à la volière. Nous n'avions pas les deux drones donc malheureusement il n'y a qu'un drone dans la vidéo. Cependant, le code est fonctionnel.

R.F.5: Les drones doivent éviter les obstacles présents dans l'environnement.

- Pour la simulation, on a les drones qui évitent les obstacles assez bien.
- Pour les drones physiques,on a une vidéo d'un contournement d'obstacles avec un drone, tournée avant que nos drones aient des soucis.

R.F.8: Un espace est disponible sur l'interface utilisateur pour y incorporer les informations par rapport aux positions des obstacles.

- L'affichage de ces informations se fait bien dans la simulation peut importe le nombre de drones
- L'affichages des informations se fait mais n'est pas parfait à cause de problèmes matériel avec les drones

R.F.9 : Nous arrivons à voir les drones en continu sur la map.

- Marche parfaitement en simulation
- Ne marche pas très bien en physique à cause de problèmes matériel avec les drones
-

R.F.10: L'interface utilisateur doit être disponible comme service Web et visualisable sur plusieurs types d'appareils (PC, tablette, téléphone) via réseau. Nous avons une vidéo qui montre justement ce comportement sur un téléphone. Il est même possible de partir une mission à partir du téléphone.

- Nous pouvons voir notre application web sur un téléphone mobile et voir les logs en temps réel ainsi que les logs des missions précédentes. En plus des obstacles et des drones qui se dessinent.

R.F.12: La position des drones est déterminée automatiquement par la station au sol.

- Ce requis n'a pas été implémenté par manque de temps

R.F.13: Pendant la mission, ou pas, retourner le drone. Le tableau qui affiche l'état des drones affichera alors 'accident' pour le drone concerné.

R.F.17: On peut retrouver une base de donnée sur une page lorsqu'on choisit l'option Database dans le menu de la page principale. Elle contient les données et peut être triée par différents attributs. Il y a aussi un composant dans lequel on peut trouver tous les logs d'une mission lorsqu'on clique sur celle-ci.

- Les tests pour ce requis ont tous étés positifs. Nous pouvons créer une mission et la stocker dans une base de données. Le calcul du temps de vol, du nombre de drones et de la distance se font très bien.
- Nous pouvons aussi supprimer des missions qui est un plus

R.F.18: Aller dans l'onglet 'database' puis sélectionner une mission. La carte d'exploration du premier drone de la mission sera alors alors affichée. On peut choisir d'afficher celle d'un autre drone avec le menu déroulant.

- Le stockage des maps n'a malheureusement pas pu être implémenter au complet et n'est pas fonctionnel

R.F.19: Cliquer sur le bouton 'Peer to peer' sur l'interface utilisateur. On s'attend à ce que la DEL du drone le plus proche de la station au sol soit verte tandis que l'autre soit en rouge.

## *6.2 Requis de conception*

R.C.1: Les drones en simulation affichent correctement les logs pour le débogage en continu lors d'une mission. On peut retrouver l'état, l'id, la position et le data des capteurs dans chaque log. De plus, les logs sont sauvegardés pour être affichés comme missions antérieures dans le futur. Il y a également les logs pour les drones physiques qui fonctionnent très bien de même que l'historique.

R.C.2: Le logiciel complet de la station au sol doit se lancer avec une seule commande. Il faut lancer la commande docker compose.

R.C.3: L'environnement virtuel pour la simulation apparaît avec au moins trois murs, différents des quatre murs externes, placés aléatoirement à chaque recompilation de Argos l'environnement est aléatoire.

R.C.4: L'interface utilisateur doit être facile d'utilisation et lisible. Lorsqu'on lance l'application, l'utilisation est plutôt claire, les boutons accessibles et compréhensibles, les éléments tous visibles à une taille adéquate.

R.C.5: Pour changer le nombre de drones dans la simulation, il faut modifier le fichier .argos à deux endroits <params nb_drones = ""> et la place habituelle des nombre de drones. Sur l'interface, tout est mis à jour : le tableau qui affiche l'état des drones, les logs, la carte.

### *6.3 Requis de qualité*

R.Q.1: Pour le requis de qualité nous avons utilisé les standards de programmation python PEP-8 pour formater notre code. Pour le code embarqué, nous avons utilisé le standard de programmation de Barr ainsi que pour nos fichiers .h et .c. Pour le code typescript dans le client, nous avons utilisé le google TypeScript Guide.[2][5][6]

R.Q.2: Tous les composants du système peuvent être testés en suivant les étapes expliquées dans le fichier "Tests.pdf" et en comparant le comportement obtenu avec le comportement attendu qui est également détaillé dans le même fichier.

## 7. Travaux futures et Recommandations (Q3.2)

Malgré le fait que nous soyons satisfait du travail général accompli, il existe toutefois des éléments manquants à notre production. On peut citer le requis au choix RF12, stipulant que l'orientation des drones doit être spécifiable dans l'interface. La gestion et la priorisation des tâches les plus critiques ne nous ont pas permis de nous pencher sur ce requis assez longtemps. Il nous a été difficile de travailler avec les drones physiques dû à leur faible fiabilité (très fragile) et de l'imprécision relative de leurs senseurs (filtre Kalman pour la position absolue par exemple).

Concernant les ouvertures, sachant que notre drone est capable d'explorer l'environnement, de cartographier une zone méconnue et de revenir à son point de départ, nous voyons ici un domaine d'application pertinent tel que le secourisme. Il sera possible pour une équipe de secours d'envoyer nos drones

cartographier une zone où la visibilité est réduite afin d'intervenir plus efficacement et avec moins de risques.

## 8. Apprentissage Continu (Q12)

### Majid :

lacunes identifiées : Difficulté à gérer le projet (aider et guider tout le monde) et gérer ses propres tâches.

méthode de remédiation : Délégation de tâches.

amélioration : Le réseau de travail de l'équipe est décentralisé entre les différentes sous-équipes (simulation, drone physique, client) et je faisais le pont entre les différents composants. Il aurait éventuellement fallu travailler en 100% décentralisé, sans avoir de médiateur assigné.

### Patrick :

lacunes identifiées : Difficulté au début du projet à se retrouver et communiquer avec l'équipe donc du retard a été accumulé au niveau du partage de connaissances.

méthode de remédiation : Discussion avec Majid et Paul Marie pour améliorer la communication et être sur la bonne piste.

amélioration : Faire un review (en terme de message sur discord) à chaque jour sur ce qui a été fait et qui doit être fait pour ne pas perdre de temps et améliorer la communication.

### Paul-Marie :

lacunes identifiées : Écriture d'une grande quantité de code mort. J'échafaude le projet pour valider mon implémentation mais je n'enlève pas les outils de débogage par la suite.

méthode de remédiation : Utilisation d'un formateur de code strict.

amélioration : Concentration et focus lorsque je développe du code.

**Erika :**

lacunes identifiées : Demander de l'aide en cas de blocage plutôt que d'accumuler du retard sur l'échéance d'un backlog.

méthode de remédiation : Utilisation du serveur discord et du salon bug en particulier.

amélioration : Discuter avec les concepteurs des parties qui me posent un problème pour trouver une solution commune.

**Manel :**

lacunes identifiées : Procrastination.

méthode de remédiation : Mise en place d'échéancier strict de la part de l'équipe.

amélioration : Bien évaluer les tâches avant de me les assigner.

# 9. Conclusion (Q3.6)

En conclusion, ce projet intégrateur fut un défi majeur pour toute l'équipe en termes d'apprentissage, de coordination et d'expérience humaine. Nous nous sommes heurtés à bons nombres d'obstacles de toutes sortes, certains parfois prévus d'autres totalement imprévus. Le manque de support et l'encadrement parfois faisait défaillance quand bien même nous sommes en fin de troisième année de Génie informatique. Celà a été une contrainte supplémentaire à la réalisation effective du projet. Mais ne manquant pas de ressources et de potentiels au sein de l'équipe nous avons pu surmonter ces obstacles. Le manque d'organisation au sein de l'équipe à certains moments nous a pénalisés bien qu'on avançait de manière individuelle sur nos tâches. Nous avons essayé au maximum de rester à jour et organisé par rapport à la planification faite en amont pour être sûr de ne pas avoir de retard à l'approche de la remise et de faire une remise en bonne et due forme. Mais malheureusement nous avons pas pu attendre totalement l'objectif mais nous demeurons satisfaits du travail réalisé car nous venons de très loin comparativement à d'autres équipes. Et celà fut vraiment la plus belle victoire que nous ayons eu dans ce projet. Ne jamais abandonner et toujours donner le maximum de soi même si on est pas les meilleurs on peut travailler pour.rien n'est impossible avec un peu de volonté.

## 10. Références (Q3.2)


- [1] Bitcraze. *Getting started with the Crazyflie 2.X.* (Septembre, 2022). Tiré de : *https://www.bitcraze.io/documentation/tutorials/getting-started-withcrazyflie-2-x/*
- [2] Python. *PEP Index.*(Septembre, 2022). Tiré de : *https://peps.python.org/pep-0008/*

- [3] Blog.Net. L'essentiel de la syntaxe Typescript.(Septembre, 2022). Tiré de : *https://cdiese.fr/syntaxe-typescript-en-10-min/* [y]

- [4] Doxygen. *Generate documentation from source code*. (Septembre, 2022) .Tiré de : *https://doxygen.nl/*
- [5] BarrGroup. barr Group Standard Tiré de : https://barrgroup.com/sites/default/files/barr_c_coding_standard_2018.pdf
- [6]TypeScriptGuide. (Novembre, 2022). Tiré de : https://google.github.io/styleguide/tsguide.html


## ANNEXES