Département de génie informatique et de génie logiciel

INF3995

Projet de conception d'un système informatique

Conception d'un système aérien minimal pour exploration

Rapport final de projet

Équipe No. *203*

Samba B. Bal
Mathurin Chritin
Hubert Grootenboer
Issam E. Maghni
Eya-Tom Augustin Sangam

Avril 2020

Table des matières

1. Objectifs du projet

Dans cette section, rappeler l'objectif commercial et ce que le système créé doit satisfaire comme exigences.

…

Dans le but de répondre à son besoin de stimuler l'industrie et de maximiser l'impact de ses investissements, l'agence spatiale canadienne désir investir dans un projet permettant d'envoyer des robots munis de capteurs minimaux pour les besoins d'explorations dans l'espace. Si le projet se montre viable, il permettra de donner un nouveau souffle au secteur et offrira de nouvelles perspectives.

Le système d'exploration présenté ici regroupe deux composantes fondamentales. Un essaim de drone est envoyé pour la reconnaissance du site et transmettra par la suite les informations récoltées. Cela permettra à un robot plus équipé d'explorer les zones les plus intéressantes.

Le système permet en outre la gestion à distance des drones. Un opérateur utilise donc un fureteur web pour contrôler le système et visualiser les données générées par les drones.

2. Description du système

En ce qui concerne le système mis en place, il est composé de deux parties : la station au sol et la partie embarquée. La figure ci-dessous montre le système vu de haut niveau.

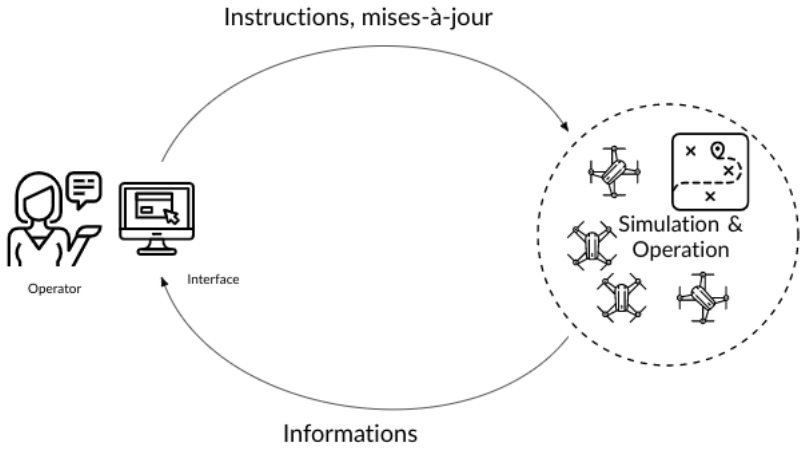


Figure 1 - Système conçu

— La station au sol : il s'agit d'un ordinateur avec une interface web qui communique 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. Cette dernière regroupe le serveur maitre ainsi que les différents clients. Toute la gestion se fait à travers une interface web.

— La partie embarquée : il s'agit des drones (réels ou simulés) et du logiciel qui roule sur leur microcontrôleur de bord. Les drones communiquent entre eux et avec la station à travers un canal de communication à 2.4GHz.

2.1 Logiciel embarqué (Q4.5)

Architecture, librairies utilisées et structure du code. Aussi, ce qui a changé par rapport aux versions proposés dans la CDR et pourquoi.

…

Le changement principal depuis le CDR concerne l'algorithme d'exploration. Au CDR, et dans la simulation, on utilisait un algorithme basique d'évitement d'obstacle, qui faisait plus ou moins rebondir les drones contre les murs. Cela nous a permis de comprendre le comportement du drone dans ARGoS : nous avons approché un par un les concepts essentiels (avancer, reculer, tourner, s'arrêter…) mais aussi appris à récupérer les données des lasers et d'initier un protocole de communication avec la station au sol.

Notre intention de départ était d'utiliser exactement le même code de logique tant pour le drone réel que pour la simulation sur ARGoS. Nous avons créé une couche de haut niveau afin d'abstraire les commandes. Cette approche fonctionna parfaitement pour les drones réels, mais pas pour la simulation, parce qu'il y a des différences fondamentales entre la manière de piloter les drones réels et celle de piloter les drones simulés sur ARGoS.

Sur le simulateur, le déplacement des drones se fait par la position, c'est-à-dire qu'on demande au drone de se rendre à une position absolue ou relative et c'est à lui de se débrouiller pour se rendre à cette position en le laissant gérer la physique du simulateur. Dans la réalité, il est impossible d'utiliser cette approche, qui repose entièrement sur le fait que le drone connaît sa position absolue (x,y,z) en tout temps et en tous points. Même en utilisant le « FlowDeck » [1], il est difficile pour le drone réel de connaître sa position, à cause des nombreuses sources d'erreur qui apparaissent dans la réalité (texture du sol, luminosité, erreurs d'estimations de la caméra…). Le déplacement des drones réels se fait donc par application d'une vitesse : on indique au drone d'avancer à une certaine vitesse dans une certaine direction.

Nous avons bien tenté d'implémenter un déplacement par vitesse dans ARGoS, qui le supporte pour tous les autres robots inclus avec ce simulateur, mais sans succès pour les crazyflies. Après avoir envoyé un message au développeur d'ARGoS, nous avons appris (trop tardivement pour que nous puissions changer avant la remise), que nos essais infructueux étaient dus à un bogue corrigé dans la prochaine version d'ARGoS, sortie le 18 avril 2021. C'est pourquoi, suite à nos recherches sur le sujet, nous sommes tombés sur un article de *Science Robotics* s'intitulant « *Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment* » [2].

L'article décrit un algorithme d'exploration d'environnement inconnu qui s'avère s'appliquer très bien à notre contexte, c'est pourquoi nous nous sommes basés dessus pour le logiciel embarqué du drone réel.

ARGoS a donc joué un rôle important dans notre compréhension des concepts d'exploration et de pilotage des drones. Il nous a également servi à tester la communication avec le serveur maître, ainsi qu'à développer la carte de l'interface utilisateur à partir de données crédibles (simulant les données réelles). En revanche, l'algorithme d'exploration du simulateur n'est pas strictement identique à celui implémenté sur les drones réels, il s'agit plutôt d'une version évoluée de l'algorithme du CDR, basé sur une machine à états de la forme suivante :

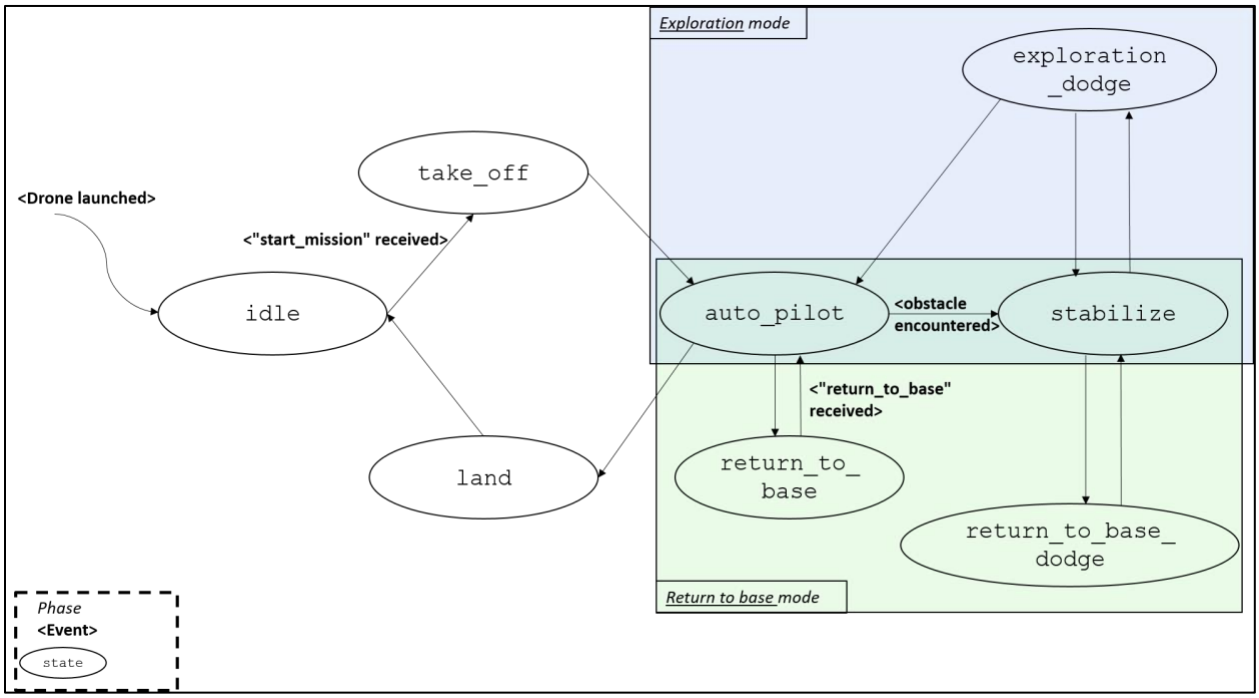


Figure 2 - Machine à états des drones du simulateur

Les drones ont deux modes de fonctionnement qui changent essentiellement leur manière d'éviter les obstacles (leurs états « **_dodge* »). En mode « exploration », les drones tournent sur eux-mêmes jusqu'à ce qu'il n'y ait plus d'obstacle immédiat devant eux, puis continuent d'avancer tout droit (« *exploration_dodge »)*. En mode « return to base », face à un obstacle, le drone se déplace latéralement en restant toujours orienté vers son point de décollage, dans le but d'y retourner et d'y atterrir (« *return_to_base_dodge »*).

Sur les drones réels, en nous basant sur l'article [2], nous avons modifié le code source de références pour le faire répondre à nos requis. Les drones utilisent l'algorithme SGBA (Swarm Gradient Bug Algorithm).

On s'est inspiré de l'implémentation de référence pour SGBA. Par contre, nous avons ajouté par-dessus le système de construction CMake. Aussi, nous avons porté et modifié l'implémentation pour qu'elle soit écrite en C++. Cela nous permet d'avoir un code qui

peut s'exécuter, du moins dans le côté technique, sur ARGoS et le drone embarqué. Cela nous a aussi permis d'utiliser les fonctions fournis par la librarie standard de C++, qui n'existe pas en C.

2.2 La station au sol (Q4.5)

Architecture, librairies utilisées et structure du code. Aussi, ce qui a changé par rapport aux versions proposés dans la CDR et pourquoi.

…

La station au sol est au cœur de notre système. Elle permet de faire la gestion des drones mais également de visualiser les résultats des missions d'explorations. De manière générale, l'architecture n'a pas évolué depuis la remise du rapport du CDR. La figure ci-dessous montre un schéma de son architecture.

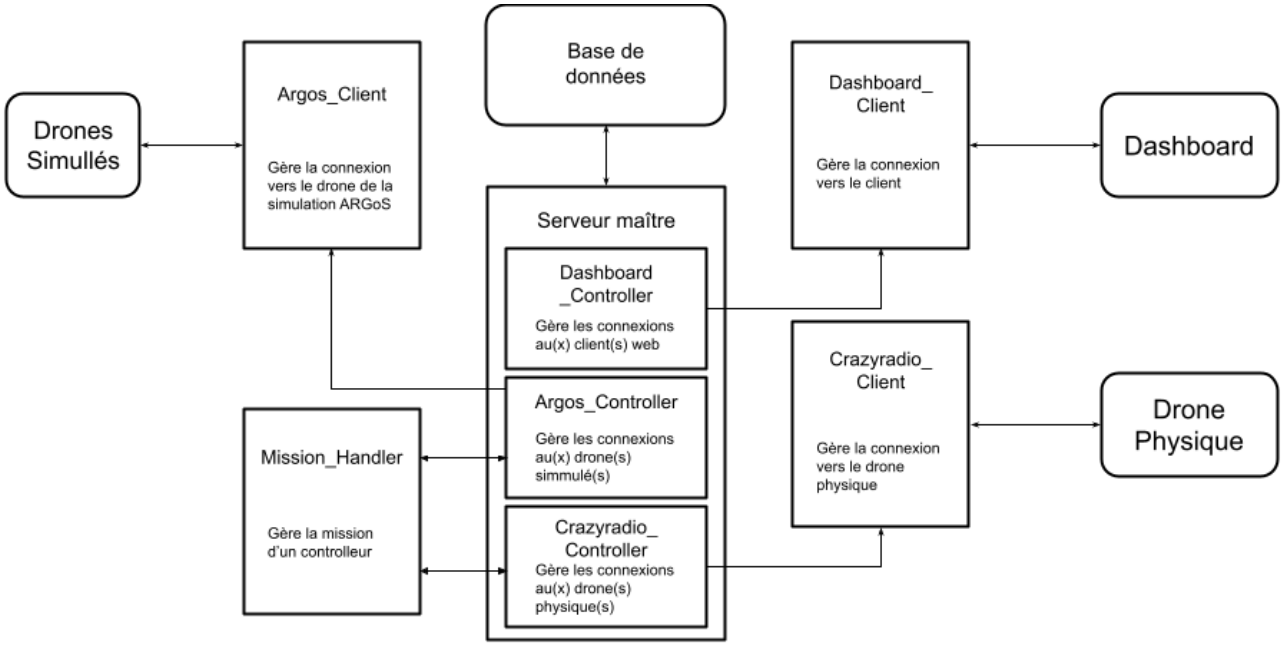


Figure 3 - Architecture globale du serveur maître

Notre Station au sol est restée fidèle au prototype qui a été présenter lors du CDR. Le seul changement majeur qui est à souligner est le fait qu'il est maintenant possible pour l'utilisateur de flasher le Crazyflie directement depuis l'interface. En effet, la version du CDR imposait aux utilisateurs de mettre les Crazyflie en mode bootloader avant de lancer le serveur. Ce dernier réalise ensuite le flash de manière automatique. Désormais, l'utilisateur peut s'en passer et décider à quel moment il veut flasher le drone. Mais mieux encore, il n'a plus besoin de placer le Crazyflie en mode bootloader pour le flasher.

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 pour interagir avec le serveur tandis que les drones physiques utilisent l'antenne et les drones de simulation utilisent un socket 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.

Depuis le CDR, la communication vers Argos a légèrement changé : nous n'utilisons plus qu'un seul client pour transmettre des informations pour des raisons de conflits de thread dans la simulation.

Le serveur maître s'occupe de faire le lien entre ces trois serveurs, et de sauvegarder les informations utiles dans la base de données. Il est implémenté en utilisant le langage Python pour deux principales raisons :

- Les librairies de contrôle à distance des drones à savoir `crazyflie-lib-python` [3] et `crazyflie-clients-python` [4] de `Bitcraze` [5] 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.

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

Pour gérer les connexions au dashboard, nous avons utilisé la librairie `flask-threaded-socket` qui permet de mettre en place des connexions `websocket` facilement. Pour la connexion à la simulation, nous utilisons la librairie de socket de python. Elle nous permet d'établir une connexion TCP entre la simulation et le contrôleur et d'échanger des octets de données. Enfin, pour la communication avec les drones physique, nous utilisons les librairies de crazyflie.

Depuis la CDR, nous avons ajouté un gestionnaire de mission : les contrôleurs d'ARGoS [6] et de Crazyflie peuvent l'instancier au moment où le dashboard commence une mission. Ce gestionnaire reçoit les positions, l'orientation, et les données des capteurs des drones puis les traitent pour avoir des points représentant les obstacles détectés. De plus il doit transformer les positions des drones pour s'adapter au système d'axe du dashboard. En effet, la simulation, les drones réels et le dashboard possèdent tous des référentiels différents. Nous avons ajouté la possibilité de simplifier le nombre de points à afficher dans la carte en supprimant les points trop proches les uns des autres. Pour ce faire nous utilisons la librairie `kdtree`, nous permettant de chercher le point le

plus proche de manière très efficace, améliorant ainsi les performances de la carte de manière drastique.

2.3 L'interface de contrôle (Q4.6)

Principales parties du code (modules, classes) et librairies utilisées. Interfaces graphiques.

…

Nous avons mis un accent particulier sur l'interface de contrôle pour répondre aux exigences des utilisateurs. Ce dernier représente un élément essentiel dans le système car il permet de réaliser la gestion des drones mais également l'analyse des résultats d'exploitation. Elle est représentée par une application Web. Nous l'appelons communément le Dashboard.

L'interface utilisateur 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 connectés au serveur en indiquant leur position, leur vitesse ou leur niveau de batterie. La seconde permet de voir les missions actuelles ou précédentes ainsi que la carte de la zone explorée par les drones lors des missions. La dernière dispose d'un éditeur de code permettant de modifier le code que les drones exécutent en pleine mission.

Cette interface Web est basée sur les technologies Web récentes HTML5, CSS3, JavaScript (utilisant le framework Angular). Outres ces libraires standard nous avons utilisé :

- La librairie CodeMiror (https://www.npmjs.com/package/codemirror) : pour obtenir un éditeur de code moderne.
- La librairie SimplebarAngular (https://www.npmjs.com/package/simplebar-angular) : pour obtenir de belles barres de défilement.
- La librairie Marked (https://www.npmjs.com/package/marked) : pour transformer un texte md en code HTML.
- La librairie svg-pan-zoom (https://www.npmjs.com/package/svg-pan-zoom) : pour transformer notre Svg en Canvas virtuel qu'on peut agrandir et rétrécir à notre goût.

En termes de code, le Dashboard est constitué de trois éléments principaux : des pages, des composants et un routeur. Un composant est un regroupement cohérent d'éléments visuels. Un composant peut en contenir d'autres. Une page représente une vue complète et cohérente du fureteur. Une page peut contenir plusieurs composants et éléments visuels. Le routeur quant à lui, choisi quelle page afficher selon ce qui est demandé par l'utilisateur.

Le diagramme suivant montre l'ensemble des composants et des pages du Dashboard.

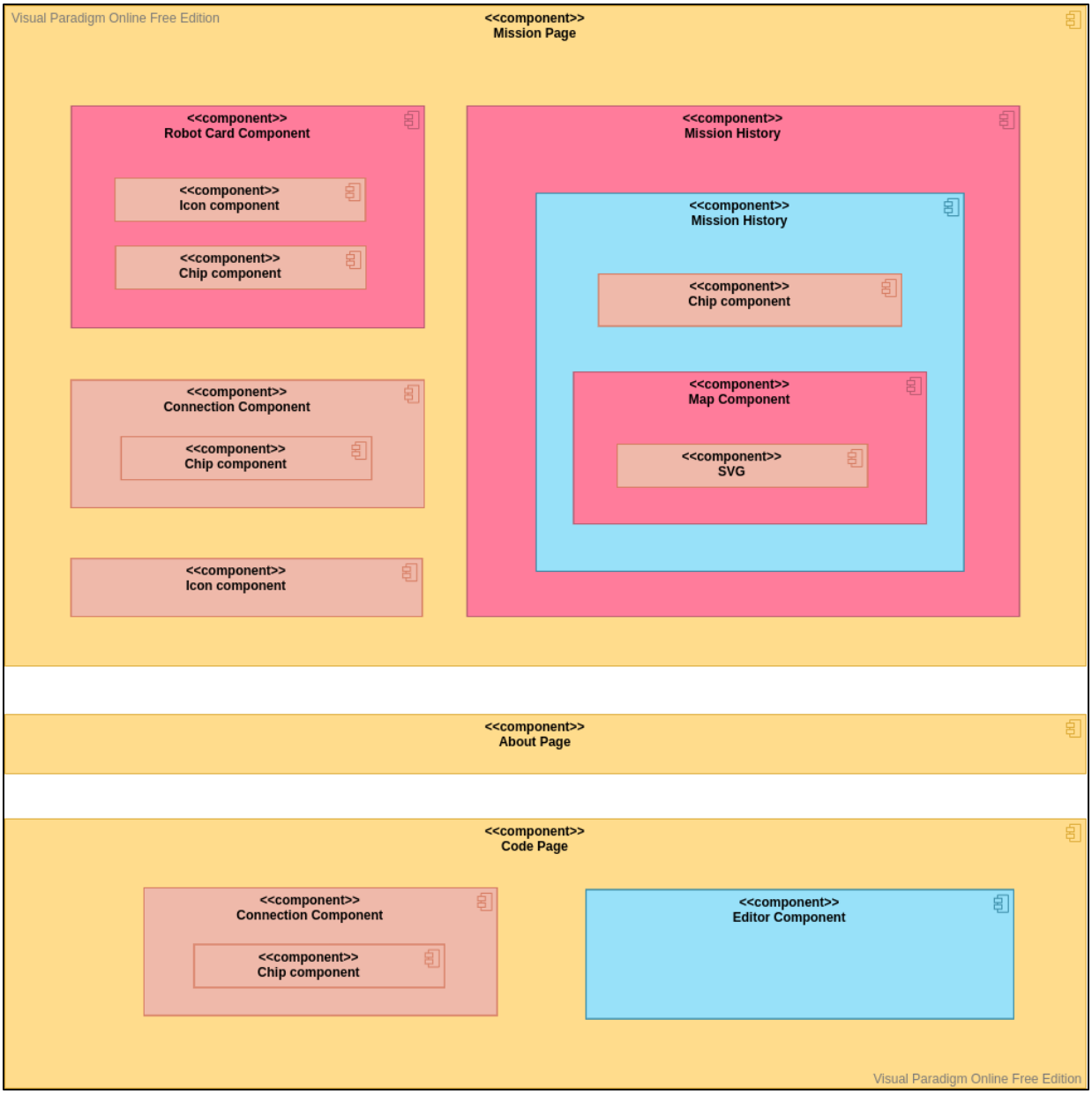


Figure 4 – conception du dashboard

2.4 Fonctionnement général (Q5.4)

Comment arriver à faire fonctionner votre système à partir du code développé. Interface usager, téléchargement, initialisation et démarrage, etc…
Aussi, ajouter les détails pour que le correcteur puisse faire fonctionner votre système

…

Durant tout notre processus de développement, la facilité d'utilisation de notre solution était au cœur du débat. Cela nous a poussé à faire certains choix qui ont concernés aussi bien les dépôts git que la façon de lancer le système.

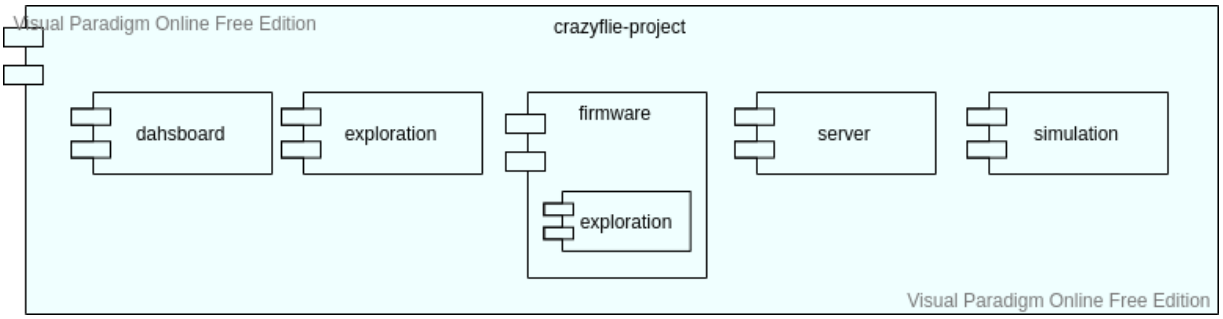


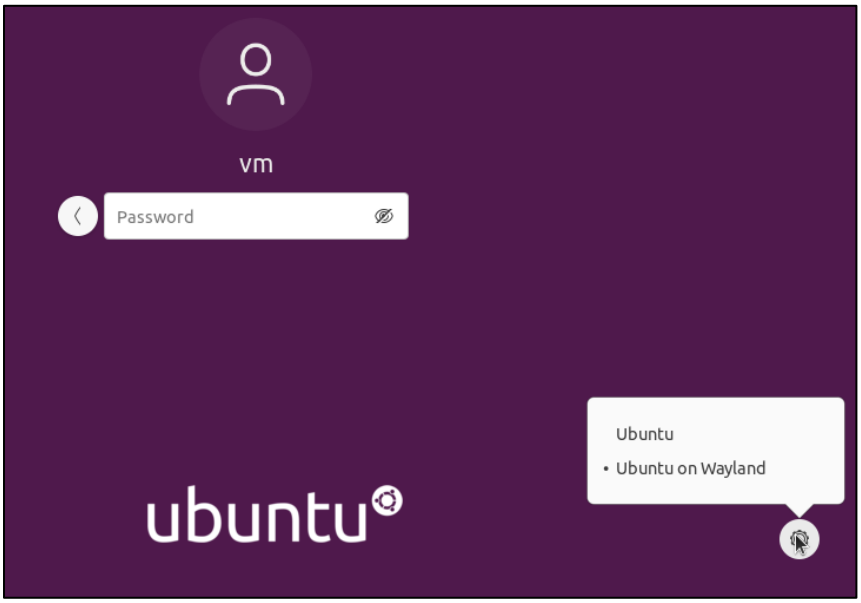
Figure 5 - organisation du dépôt gitlab

Avant toute chose, on suppose que Docker [7] et Docker-Compose sont installés et configurés pour ne pas nécessiter les permissions root. Ceci se fait en s'ajoutant au groupe système « docker ».

La difficulté que nous avons rencontrée concerne le lancement d'ARGoS (comportant une interface graphique) dans un conteneur (plus précisément dans un docker-compose). L'exécution d'un logiciel avec interface graphique depuis Docker nécessite la configuration de X11. Cette configuration est cependant laborieuse, nécessite l'installation de diverses dépendances et comporte beaucoup d'aléas.

Notre approche consiste à informer la librairie graphique d'ARGoS, Qt, d'utiliser Wayland comme back-end au lieu de X11, et tout simplement exécuter depuis un environnement Wayland. Deux approches sont possibles pour y parvenir. Voici les instructions pour Ubuntu:

Option A : Choisir Wayland dans l'interface de connexion si possible



.

Figure 6 - connexion à une session utilisateur Wayland sur Ubuntu

Option B : Installer et exécuter Weston, un compositeur graphique léger utilisant Wayland.

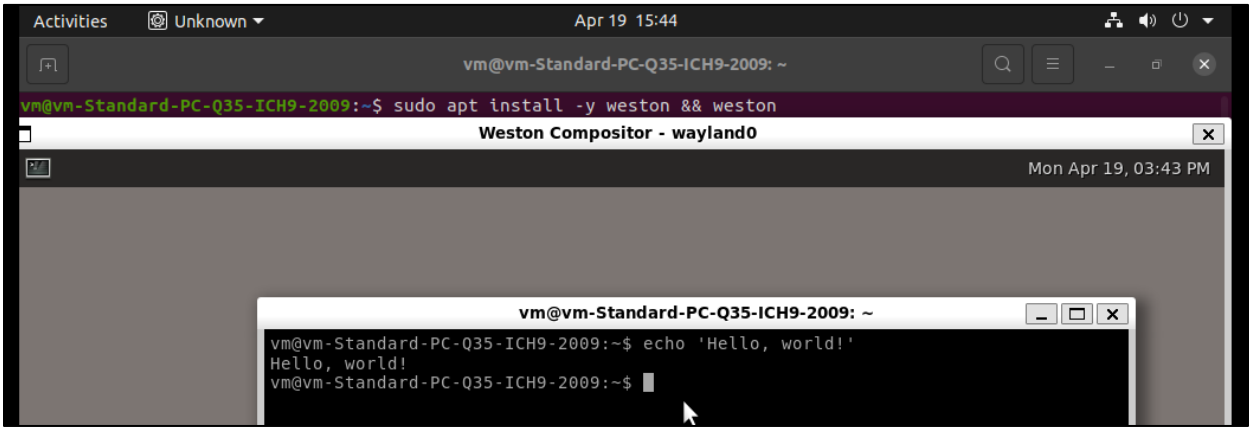


Figure 7 - installation et lancement de weston sur ubuntu.

Si vous avez suivi l'option B (installer weston), tapez les commandes suivantes dans le terminal à l'intérieur de weston. Si vous avez choisi l'option A, tapez les commandes suivantes comme vous le feriez normalement.

```
$ git clone --recurse-submodules git@gitlab.com:polytechnique-montr-
al/inf3995/20211/equipe-203/crazyflie-project.git
$ cd crazyflie-project && ./launch.sh
```

Après que le système a démarré, ouvrez votre fureteur à l'adresse http://localhost:4200.

Note : il y a un bogue sur Firefox avec les requêtes CORS qui empêche le client angular à communiquer avec le serveur python. Il faut utiliser Chrome ou Chromium.

NB: Pour que les drones puissent être retrouvé, il faut s'assurer d'utiliser le channel 56. Vous pouvez vérifier cela directement dans le cfclient.

Les adresses des deux drones doivent être 0xE7E7E7E701 et 0xE7E7E7E702 voire plus. Un maximum de 9 drones est permis soit jusqu'à 0xE7E7E7E709.

Une vidéo démontrant le fonctionnement de tout le système est disponible dans le dépôt gitlab au chemin /rr/Video_de_presentation.mp4.

Enfin, vous avez accès à un éditeur de code qui permet de compiler et flasher le drone avec le code visible dans l'éditeur. Une fois votre code rédigé et validé, pressez le bouton « compile and flash », ce qui va compiler et envoyer le code au drone. Vous n'avez pas à toucher aux drones à aucun moment, tout se passe sans intervention humaine sur les robots : c'est là le principal avantage.

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

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

…

Dans la globalité, nous estimons que le projet a été très bien réussi. Le tableau ci-dessous présente les différents requis (initiaux) et des commentaires sur leur réalisation.

Tableau 1 : Bilan des requis

Requis	Détails	Commentaires
R.F.1	Une fois la commande de départ reçue, les drones doivent être capables de parcourir le périmètre d'une pièce de max 100 m2 en absence d'autres commandes de la station au sol.	Fonctionne
R.F.2	Le système doit être conçu pour fonctionner idéalement avec un	Fonctionne

	nombre arbitraire de drones, ne tenant pas compte du système de communication avec la station au sol, mais jamais moins que deux (2).	
R.F.3	Les drones doivent éviter les obstacles sur leur chemin, incluant les autres drones.	Fonctionne mais n'est pas suffisamment testé dans les cas extrêmes. On n'avait que deux drones et ils n'étaient pas souvent chez une même personne.
R.F.4	L'essaim de drones doit répondre au minimum aux commandes suivantes, disponibles sur l'interface utilisateur : Take-off / Start mission, Land / End mission, Software update, Return to base	Fonctionne
R.F.5	L'interface utilisateur doit montrer les informations suivantes, mises à jour avec une fréquence minimale de 1 Hz : 1. Nombre des drones 2. État des drones (standby, in-mission, crashed) 3. Vitesse courante des drones 4. Niveau de batterie des drones 5. Carte générée durant l'exploration 6. [Optionnel] Position des drones 7. [OPTIONNEL] Éditeur pour le code du drone	Fonctionne, même les requis optionnels.
R.F.6	L'algorithme d'exploration de l'environnement n'a pas de contraintes. Cela peut être une séquence de mouvements aléatoires (random walk), mais la carte générée par les drones doit rassembler l'environnement exploré. La précision de la carte n'est pas sujette de requis.	Fonctionne
R.F.7	L'intégration des mesures des différents drones est faite par la station au sol en supposant que les	Fonctionne. Le "Maximum Likelihood Estimation" n'a pas été implémentée.

	positions et orientations initiales des drones sont connues ou [OPTIONNEL] en résolvant un problème d'optimisation permettant d'inférer la carte la plus plausible selon les données recueillies (Maximum Likelihood Estimation).	Pour pallier ce problème, l'utilisateur a l'opportunité de renter les coordonnées absolues des drones avant de lancer la mission.
R.C.1	Le système doit suivre un processus de conception qui valide l'approche en simulation avec le simulateur ARGoS avant l'expérimentation sur le matériel.	Fonctionne
R.C.2	Chaque composant du logiciel doit avoir un test unitaire correspondant.	Nous n'avons malheureusement fait que des tests d'intégrations tout au long du projet.
R.C.3	Le système de développement logiciel doit avoir un ou plusieurs tests de régression qui valident chaque addition ou retrait de code	
R.C.4	Le système doit être testable dans un environnement virtuel ARGoS avec une seule commande sur un terminal Linux (p.ex. avec docker-compose)	C'est le cas
R.C.5	L'environnement virtuel pour les tests dans ARGoS doit pouvoir être généré aléatoirement	Fonctionne (option de lancement 0)
R.C.6	La simulation de la mise à jour du logiciel avec ARGoS n'est pas nécessaire	Elle n'a pas été implémentée
R.L.1	Les drones doivent être programmés en utilisant l'API de BitCraze. Cependant, il est possible d'utiliser une machine virtuelle basé sur l'API à bord du drone (p.ex. la machine virtuelle de Buzz, BVM).	Nous avons réaménagé le système de compilation initial pour y ajouter du C++. Le C++ est également le langage utilisé dans l'éditeur de code de l'interface de contrôle.
R.L.2	Les drones doivent communiquer en utilisant l'API pour la communication P2P de BitCraze. L'utilisation d'un serveur comme relais des messages entre les drones est interdite.	Les drones communiquent entre eux via P2P.

R.L.3	La distance entre les drones et la station au sol (pour le retour à la base) doit être estimée à travers la puissance de signal (RSSI) reçue par le Crazyradio PA	Il était impossible pour nous de réaliser ce requis car nous ne disposons que d'un seul Crazyradio PA (voir R.M.2). Même si nous en avions un deuxième, il nous fallait le déposer tout au milieu de la salle connecté par un long fil USB à l'ordinateur (Ce dont nous ne disposons pas). Pour contourner ce problème, nous avons remplacé l'angle entre le RSSI et le drone par celui entre le drone et son point de départ entré par l'utilisateur dans l'interface de contrôle.
R.L.4	L'interface de contrôle doit être disponible comme service Web et visualisable sur plusieurs appareils (PC, tablette, téléphone...) via réseau	C'est le cas. L'application fonctionne bien avec les tablettes comme les ordinateurs.
R.L.5	L'opérateur du système doit avoir un moyen de vérifier que le système de collecte de données opère correctement et que les données elles-mêmes sont manipulées correctement (logs). Il n'y aura pas de format précis imposé pour fournir les informations jugées pertinentes à l'opérateur. Cependant, le format devra être standardisé.	Les logs sont récupérés depuis la fenêtre de terminal qui a lancé le docker-compose et sont disponibles depuis le dashboard pour témoigner du bon fonctionnement du système.
R.L.6	Les drones envoient les mesures du « ranging deck » durant l'exploration à la station au sol avec une fréquence minimale de 1 Hz	C'est le cas

4. 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.

…

De manière générale, le projet s'est bien passé pour notre équipe. Du point de vue interne, la cohésion était au rendez-vous. Nous avons pu livrer le système qui nous a été commandé en respectant les critères de qualité que nous nous sommes fixés mais aussi les exigences techniques et fonctionnels imposé par le client.

Le projet nous a confronté à des aspects de gestion de projet que nous n'avions pas eu l'occasion de rencontrer auparavant. L'équipe en ressort plus mature et est satisfaite de la qualité de la solution proposée : l'interface utilisateur est très réussie et regroupe de manière esthétique les fonctionnalités indispensables. Le logiciel embarqué est le fruit de beaucoup d'heures de travail afin de pouvoir être implémenté à l'aide de CMake et C++, ce qui nous a facilité la tâche et a permis de concevoir un système solide à l'aide des ressources externes que nous avons consultées.

Une des choses que nous n'avons pas pu réaliser est la vidéo de présentation prévue initialement dans l'appel d'offre pour démontrer le bon fonctionnement du système. Sa réalisation prévoyait tout de même une quinzaine d'heures dans le planning, que nous avons utilisées plutôt pour préparer la démonstration à présenter à l'Agence Spatiale.

5. Travaux futurs et recommandations (Q3.5)

Qu'est-ce qui reste à compléter sur votre système ?

…

En ce qui concerne notre système, nous pouvons aisément dire qu'il répond aux exigences. Il s'agit d'une preuve de fonctionnement de ce projet soutenu par l'agence spatiale. Il pourrait représenter le début vers une nouvelle ère de l'exploration spatiale. Pour l'instant, il s'agit d'un prototype. Nous avons quelques idées d'améliorations.

D'abord, nous savons que la solution est robuste et fiable. De ce fait, nous proposons d'investir dans le but de tester notre solution avec un nombre de drones plus important. Mieux, il faudrait même augmenter la limite de 9 drones qui nous est actuellement imposée.

Ensuite, le requis fonctionnelle 7 proposé en option de résoudre le problème d'optimisation avec le "*Maximum likelihood Estimation"* n'a pas été implémenté par manque de temps. Nous aimerions nous y pencher sérieusement, car cela renforcerait la crédibilité des données affichées par l'interface.

Concernant ARGoS, nous aimerions utiliser un système de contrôle basé sur des vitesses comme mentionné au point 2.1. Les corrections ajoutées dans la nouvelle version d'ARGoS devraient nous faciliter la tâche, nous n'avons à l'heure où nous écrivons ce rapport pas encore eu le temps de nous y pencher. Enfin, bien que notre

solution fonctionne et génère des cartes qui ressemblent aux environnements explorés, nous trouvons qu'il y a une place pour l'amélioration de notre algorithme, qui présente parfois quelques lacunes pour les obstacles cylindriques isolés.

6. Apprentissage continu (Q12)

Un paragraphe par membre (identifié en début de paragraphe) de l'équipe qui doit aborder chacun de ces aspects de façon personnelle:

1. *Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*
2. *Méthodes prises pour y remédier.*
3. *Identifier comment cet aspect aurait pu être amélioré.*

…

Augustin

Deux choses ont marqué ce projet selon moi. La diversité des technologies utilisées et la notion de gestion de projet. Il fallait apprendre beaucoup de langages, outils et de concepts nouveaux.

Du point de vue gestion de projet, il fallait pouvoir planifier toute la session à l'avance et essayer de suivre cette planification jusqu'au bout. Cela était nouveau pour moi car jusqu'ici, dans la plupart du temps, tout était planifié dans les grandes lignes (sprints, livrables, etc) par les professeurs responsables. En résumé, cette grosse autonomie était nouvelle pour moi. Le point que j'aurai aimé le plus améliorer durant le volet gestion de projet est mon habilité à mieux prédire la durée des tâches, que je sous-estimais un peu trop. J'ai rarement fait des estimations justes et elles m'ont souvent fait croire que j'avais plus de temps que prévu. Il m'est cependant dur d'identifier une méthode absolue pour mieux faire la planification, car c'est un savoir qui s'améliore avec le temps.

Du point de vue code, ma plus grosse difficulté était le débogage. Il n'était pas toujours le plus évident. Sur ARGoS par exemple, il n'est pas possible de faire un débogage pas à pas. Chose à laquelle j'étais très habitué avant. Pour remédier à ce problème je faisais beaucoup d'affichage console. Pour améliorer ce point, le mieux aurait été de faire beaucoup de tests unitaires durant le projet et s'assurer de ne pas avoir à déboguer des bogues triviaux.

Hubert

En commençant le projet, je n'avais que des compétences limitées dans le domaine du réseau. Je savais comment cela fonctionnait, mais je n'avais aucune expérience concrète. Pour pallier cela, j'ai été chargé de faire la majorité du serveur central à notre solution. Je me suis aussi pas mal renseigné en ligne pour approfondir mes connaissances et identifier quelles technologies conviendraient le mieux à notre projet. Étant donné que je n'étais pas expert dans ce domaine, il m'était difficile de prédire le temps qu'allait me prendre une tâche. Bien que j'aie le sentiment de ne pas m'être trompé trop de fois, je pense que je peux tout de même m'améliorer. Par contre, je ne pense pas qu'il y ait de vraie manière de s'améliorer sur ce point, mis à part acquérir plus

d'expérience. Enfin, comme beaucoup d'étudiants qui prennent ce cours je pense, j'ai eu du mal à réaliser la partie gestion de projet de notre réalisation. Prédire à l'avance la majorité du travail à réaliser quand on ne connait pas les technologies, le matériel ou l'importance de chaque requis du projet n'était pas une chose familière. Pour résoudre cela je prévoyais des petites tâches chaque semaine, plus facile à prédire. Seulement, ces petites tâches participaient à la réalisation d'une plus grande mais que je ne trouvais toujours pas suffisamment précise. Je pense que le plus grand levier d'amélioration dans ce domaine est de passer plus de temps à définir des tâches de moyenne envergure mais tout en étant précises.

Issam

La nouveauté des tâches que je devais accomplir constituait une source de difficulté. Par exemple, la programmation multitâche en C++ était un élément nouveau pour moi. Du fait de ne pas m'être bien renseigné là-dessus et plongé directement dans le code, j'ai dû écrire trois ou quatre itérations pour arriver à un résultat stable, c'est-à-dire maintenir une connexion permanente avec le serveur. J'ai dès lors vite compris que ma méthode de travail était sous-optimal. J'ai alors rectifié le tir avec prochaine tâche à accomplir. En effet, je n'avais jamais utilisé la technologie Docker et ne réalisait pas tout le potentiel qu'offrait qu'elle offrait. J'ai alors consacré une journée entière à me documenter sur les fonctionnalités et son utilisation depuis la console et la librairie Docker SDK. L'écriture du code se déroula avec très peu d'imprévu et j'ai réussi à faire fonctionner le tout en un temps raisonnable. J'ai donc appris qu'à vouloir parfois économiser du temps en sautant des étapes, on en perd plus. Cette leçon me servira dans mes futurs projets, académiques ou professionnels.

Mathurin

La nouveauté du projet de 3è année pour moi était l'obligation de suivre une réelle gestion de projet : la planification de la session au début du projet était pour moi l'étape la plus difficile. Il était compliqué d'estimer les temps qu'on allait allouer sur des sujets dont nous ne savions pas grand-chose (exemple : la communication entre les drones et la station au sol, ou le simulateur ARGoS). En nous renseignant suffisamment et surtout en commençant à travailler sur certaines technologies avant même la réponse à l'appel d'offre, pour pouvoir appréhender ce qui était possible ou pas (surtout dans ARGoS dont je me suis beaucoup occupé tout au long du projet), j'ai pu avoir une meilleure idée des quantités de temps nécessaires pour les différentes tâches identifiées, mais cela m'aura consommé beaucoup de ressources en début de session. La difficulté résidait également beaucoup dans la documentation du simulateur, qui n'est pas très exhaustive. Ces aspects difficiles ne peuvent pas vraiment être améliorés puisqu'ils existeront dans tous les projets ; et au contraire, je suis plutôt satisfait d'y avoir été confronté dans le cadre d'un projet intégrateur avant de m'insérer sur le marché du travail.

Samba

Le projet intégrateur de 3e année a été un défi sur tous les plans. Le sujet proposé représenté un challenge du fait de son caractère innovateur. Par ailleurs, l'introduction des règles de gestion de projet a été une nouveauté à laquelle je n'étais pas préparé. Les cours fournis par le professeur ont servi d'initiation. Mais, il a fallu les remises périodiques pour mieux saisir l'intérêt de la chose et monter en compétence en ce qui concerne la planification. Je pense qu'il pourrait être intéressant d'ajouter en deuxième année un cours de gestion de projet de base. Ce dernier pourrait s'inscrire dans une logique de suite du cours INF1040 introduction à l'ingénierie informatique dispensé en première année.

7. Conclusion (Q3.6)

Par rapport aux hypothèses et à la vision que vous aviez du système lors du dépôt de la réponse à l'appel d'offre, que concluez-vous de votre démarche de conception maintenant que le système est complété ?

…

Lors de notre réponse à l'appel d'offre, nous avions émis des hypothèses en plus d'accepter des contraintes. Tout au long de notre développement, elles ont été les lignes directrices de nos actions. Nous avons ainsi pu développer un système robuste, fiable et respectant les exigences. Nous avions opté pour un développement guidé par les requis et les remises qui étaient prévues d'avance. Cette façon de faire a bien fonctionné car nous avons pu compléter toutes les tâches. Mieux, nous avons pu ajouter quelques touches personnelles comme nous l'avions prévu au moment de notre réponse à l'appel d'offre. Il est également important de mentionner que nous avons été confrontés à la plupart des risques et des problèmes que nous avions identifiés dans l'appel d'offre, ce qui nous a permis de mieux réagir sur le moment. Maintenant que notre système est complet, nous sommes très satisfaits de nos travaux et de notre produit final.

8. Références

Références utilisées autres que celles utilisées par le professeur durant le cours. Un site web utilisé pour la programmation (cadre de travail, outils, librairies) devrait être listé.

…

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

[2] 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, no. 35, 2019. [En ligne]. Disponible : https://robotics.sciencemag.org/content/4/35/eaaw9710

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

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

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

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

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

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

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

[10] D. Peng, L. Cao et W. Xu, "Using json for data exchanging in web service applications," Journal of Computational Information Systems, vol. 7, no. 16, p. 5883–5890, 2011.

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

[12] 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.

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

[14] G. Van Rossum et al., "Python," 1991. "Welcome to tinydb !¶." [En ligne]. Disponible : https://tinydb.readthedocs.io/en/latest/