POLYTECHNIQUE MONTRÉAL

Rapport Projet Intégrateur 3 : Conception d'une démonstration de vol en formation de drones Crazyflie

Auteur *Matricule*

Marie-Ève Lamer 1898763

Présenté à :

David Saussié

Pour le cours :

AER3900

Génie aérospatial

2 mai 2020

Table des matières

Table des figures

Liste des tableaux

Liste d'acronymes

cf	Crazyflie
LPS	Loco Positionning system
TWR	Two-Way Ranging
TdoA	Time Difference of Arrival

1 Résumé

Dans le contexte de recherche au laboratoire de robotique mobile et des systèmes autonomes à Polytechnique, le professeur David Saussié s'intéresse à mettre sur pied une démonstration de vol en formation de drones Crazyflie.

Le Crazyflie a été utilisé en parallèle avec le système de positionnement LPS (Loco Positioning System) afin de commander le drone dans une séquence de vol autonome. Le développement de ce programme s'est fait dans le langage Python à l'aide de la librairie "cflib" développée par Bitcraze. La séquence de commande a d'abord été conçue pour le vol d'un drone, puis a été ajustée pour permettre l'envoi de commandes à plusieurs Crazyflies en parallèle.

Pour ce projet, 4 ancres du LPS ainsi que deux drones ont été utilisés pour les tests. Il a été possible d'accomplir un vol en formation d'une séquence de vol stationnaire avec les deux drones. La séquence n'a pas été davantage complexifiée dû à l'espace restreint accessible pour les tests ainsi que la précision faible du système de positionnement avec seulement 4 ancres.

2 Abstract

At Polytechnique Montreal's Mobile Robotics and Autonomous Systems Laboratory, Professor David Saussié is interested in developing a flight demonstration of a swarm of Crazyflie drones. This quadcopter was designed for indoor flight, and is suited for both amateur and advanced drone research. Bitcraze, the Crazyflie Designer, has developed a public Python library to facilitate the drone's development and use of the different systems. It is also possible for members of the community to contribute to those libraries.

The crazyflie in parallel with the LPS (Loco positioning system) were used to create an autonomous flight sequence with a drone swarm. The programming language that was used is Python with the library "cflib" developed by Bitcraze. The fight sequence was first created for one Crazyflie, and then adjusted to allow to command multiple Crazyflies in parallel.

For this project, four LPS anchors and two Crazyflies were used for the demonstration tests. It was possible to program an autonomous flight of a hover sequence with two drones. Due to the low precision of the LPS, as there were only four anchors, and the restrained space available, the sequence was limited to a hover flight.

3 Introduction

Le Crazyflie est un quadricoptère conçu pour des vols d'intérieurs et est accessible autant aux amateurs qu'à des fins de recherche et développement poussés. En effet, la compagnie Bitcraze, le fabricant du Crazyflie, a développé des librairies publiques facilitant l'utilisation des différents systèmes ainsi que le contrôle du drone. Il est également possible pour la communauté de contribuer à ces librairies open source.

Son accessibilité et ses caractéristiques en font un drone idéal pour concevoir une démonstration d'essaim. Cette démarche vise à étudier le contrôle du drone Crazyflie à l'aide du système de positionnement LPS puis de concevoir un programme python permettant le vol autonome de plusieurs drones en simultané.

Ce rapport fait d'abord état de la logistique et des objectifs du projet. Il présente ensuite les différentes informations identifiées sur le Crazyflie, sur le système de position Loco ainsi que sur les interfaces de développement utilisées. La méthodologie suivie pour cette étude est détaillée puis les résultats des différents tests sont présentés. Finalement, la structure du programme final sera expliquée ainsi qu'une analyse des difficultés rencontrées.

4 Définition du projet

4.1 Objectifs

Les objectifs initialement définis avec le directeur de projet David Saussié sont :

1. Développer un programme permettant de commander une flotte de Crazyflies
2. Intégrer au programme une séquence permettant de faire une démonstration de vol autonome
3. Développer une interface usager permettant à des utilisateurs non familiers avec Python de commander la flotte
4. Faire un vol de démonstration avec la flotte de Crazyflies

Certaines modifications et limitations ont été apportées au plan initial dû à la crise du COVID19.

1. Comme l'accès au laboratoire était impossible, l'équipement et l'espace de vol ont été limités. Ainsi, la séquence de démonstration a été fixée à un vol stationnaire avec deux Crazyflies
2. Le développement de l'interface usager a été abandonné

4.2 Échéancier et livrables

L'échéancier initialement fourni a également été retardé dû aux changements suite aux mesures de la COVID19. Le tableau 1 montre les dates prévues et réelles des livrables identifiés au début du projet.

Livrable	Date prévue	Date de complétion
Fiche détaillée	24 janvier	22 janvier
Programme de commande pour un drone	15 mars	1er avril
Programme final pour flotte de drones	5 avril	20 avril
Rapport de projet	17 avril	30 avril
Présentation orale	17 avril	30 avril

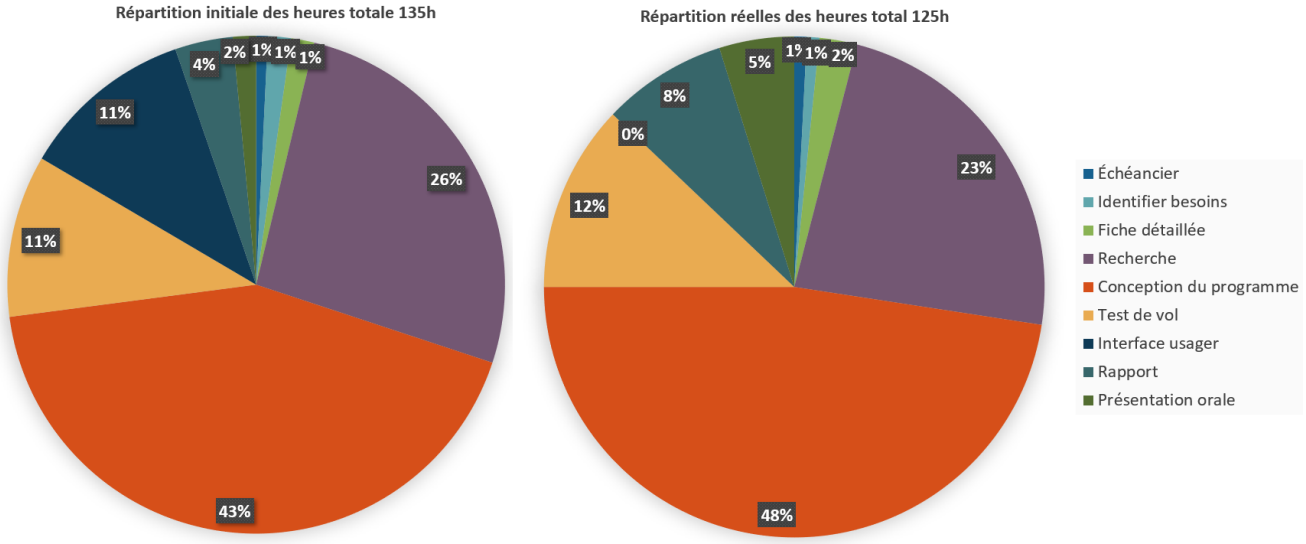
Tableau 1: Tableaux des dates de livrables

Le programme de commande pour un drone a été retardé dû à une compétition de société

technique à l'étranger. Le retour de compétition a forcé la quarantaine ce qui a restreint l'accès au laboratoire et donc au système de positionnement. Pour pallier cette problématique, le directeur de projet a proposé de prêter l'équipement nécessaire à l'atteinte des objectifs pour la durée du projet. Les ajustements ont créé au final un retard de 2 semaines sur la complétion du projet.

La figure suivante compare la répartition des heures estimées sur le projet ainsi que la répartition réelle des heures accordées à chaque aspect. Le temps total prévu du projet était de 135 heures.

FIGURE 1: Répartition des heures du projet sur les différents aspects



La répartition du temps sur les différents aspects est assez représentative des estimations. On note que ce sont la recherche ainsi que la programmation de la séquence qui représente près de 75% des heures allouées. Dans l'ensemble le projet représente environ 120h de travail sur les 135h initialement prévues.

5 Présentation du Crazyflie

Le Crazyflie est un drone de type quadricoptère. Comme il pèse environ 27g et que sa taille représente une surface de 9x9cm (moteur à moteur), il est conçu principalement pour les vols d'intérieur. Avec uniquement ses composantes de base, il est possible de le piloter à partir d'une application mobile "Crazyflie". Si l'on utilise ensuite la Crazyradio, il est possible de le contrôler avec une manette ou d'un ordinateur par python. Également, plusieurs libraires et fonctionnalités ont été développées et sont accessibles au public sur le site de **Bitcraze** ainsi que **Github**. Ainsi, le Crazyflie a l'avantage d'être autant accessible pour le loisir que pour le développement et la recherche.

FIGURE 2: Crazyflie 2.1 [1]

Les pièces nécessaires au vol du Crazyflie avec le système de positionnement sont présentées dans le tableau suivant (sans tenir compte des pièces de rechange) :

Pièces	**Quantité**
Crazyflie Control Board	1
250mAh LiPo battery	1
DC coreless motor	4
Motor mounts	4
Foam battery pad	1
CCW propellers	2
CW propellers	2
Loco positionning deck	1
Crazyradio PA 2.4 GHz usb dongle	1
Loco positonning node	4 à 8

Tableau 2: Liste des pièces

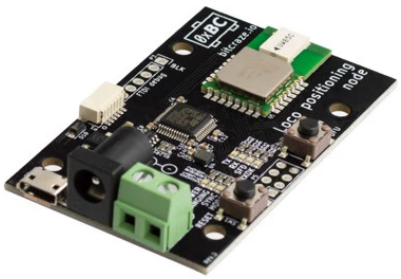
6 Système de positionnement Loco

Sans système de position, le Crazyflie utilise son accéléromètre et son gyroscope afin d'estimer ses mouvements relatifs. Il n'a toutefois pas de notion de sa position dans l'espace. Également, la précision sans positionnement ne permet pas un vol stable. Ainsi, pour ce projet, on utilisera le système de positionnement LPS afin de communiquer au drone sa position.

Pour ce faire, le Crazyflie doit être équipé du Loco Positioning Deck. Ensuite, il faut un minimum de 4 ancres (Loco Positioning Node) afin de pouvoir calculer sa position dans un espace 3D. Toutefois, la précision augmente lorsqu'on augmente le nombre d'ancres comme cela crée de la redondance. Pour ce projet, les circonstances de la COVID19 ont limité l'expérimentation à 4 ancres.

Le LPS a l'avantage d'être un système de positionnement relativement peu dispendieux comparé à ce qui se trouve sur le marché. Également, dans le contexte où l'on souhaite faire des démonstrations de vol possiblement hors du laboratoire de recherche, le système est facilement déplaçable. En effet,il suffit d'amener les ancres, de pouvoir les alimenter et de modifier les positions des ancres dans le repère.

(a) Loco positioning deck [6]



(b) Loco positioning node [6] (ancre)

6.1 Modes du LPS

Le système LPS peut opérer en 3 différents modes : TWR (Two Way Ranging), TdoA2 (Time Difference of Arrival 2) et TdoA3 (Time Difference of Arrival 3).

TWR : Dans ce cas, la balise sur le deck du Crazyflie envoie une requête à toutes les ancres en séquence ce qui lui permet de calculer sa distance par rapport à l'ancre. Comme dans ce mode le Crazyflie doit constamment envoyer l'information aux ancres, cela ne permet de voler qu'un seul Crazyflie à la fois. Ce mode a été configuré pour fonctionner avec idéalement 6 ancres pour apporter de la redondance et un maximum de 8 ancres. [3]

TdoA2 : Contrairement au TWR, le mode TdoA2 permet la communication avec plusieurs drones en simultanée. Cette fois, ce sont les ancres qui envoient des paquets à la balise du Crazyflie en séquence. Avec ces signaux, le Crazyflie enregistre la différence du temps d'arrivée de deux signaux et calcule ainsi sa distance par rapport à ces deux ancres. Comme TWR, ce mode a été configuré pour fonctionner avec un minimum de 4 ancres et un maximum de 8. La particularité de ce mode est qu'il est basé sur une ancre maîtresse (ID0) qui synchronise la séquence de transmission des 8 ancres. C'est pourquoi un maximum est imposé. [5] Cette communication a l'avantage d'empêcher la perte d'information si deux paquets arrivent au Crazyflie au même moment. Toutefois, cela amène également le désavantage d'être dépendant de l'ancre maître comme si celle-ci cesse de fonctionner le système arrête la transmission.

TdoA3 : Le mode TdoA3 est très semblable au TdoA2. Il est basé sur le même principe, mais celui-ci a été développé dans le but de pouvoir intégrer plus de 8 ancres au système. Cela permet ainsi d'agrandir l'aire de vol. En effet, comme le Crazyflie doit être relativement prêt des ancres afin de capter les paquets, 8 ancres limitent l'étendue du vol. Afin de contrer le problème du maximum d'ancres, ce mode opère sur une transmission aléatoire des ancres plutôt que synchronisées. Ainsi, la transmission est indépendante pour toutes les ancres. [5]

6.2 Configuration du LPS

Plusieurs étapes de configuraiton sont nécessaires avant l'utilisation du LPS. Il faut d'abord mettre à jour le firmware des ancres. Ensuite, on doit leur imposer un mode d'opération et leur assigner un ID de 0 à 8 (ou plus dépendamment du mode utilisé). Pour ce faire, il faut télécharger la plus récente version du firmware sur le site de Github [4]. Bitcraze a mis sur pied un outil permettant de faire la mise à jour des ancres. Cet outil fait partie du package téléchargé avec le firmware. On peut ensuite suivre les instructions sur le site suivant afin d'utiliser l'outil lps à partir de Linux : https ://github.com/bitcraze/lps-toolsrunning-instruction.

Lorsque le firmware est mis à jour, il faut porter attention à s'assurer qu'une des ancres possède l'ID0 si l'on souhaite utiliser le mode TdoA2. S'il n'y a pas d'ancre ID0, le système ne fonctionnera pas dans ce mode.

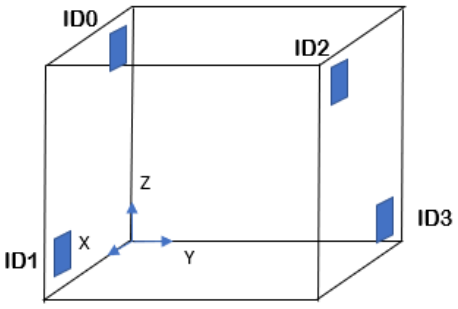
Une fois les ancres configurées, il faut les disposer dans l'espace de vol de sorte qu'elles respectent certaines règles de base. Ainsi, les ancres doivent :

— Délimiter le volume de l'air de vol

— Être placée à au moins 15cm d'une surface (mur, plancher, plafond)

— Être positionnée le plus symétriquement possible dans l'espace

La figure suivante illustre la configuration à 4 ancres utilisée pour ce projet ainsi que la configuration recommandée par Bitcraze pour 6 à 8 ancres.



Coordonnées des ancres: (x, y, z)

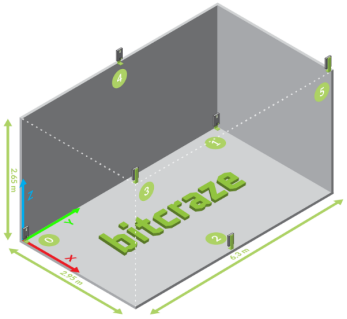
ID0: (0.44, 0.19, 1.74)

ID1: (2.40, 0.14, 0.40)

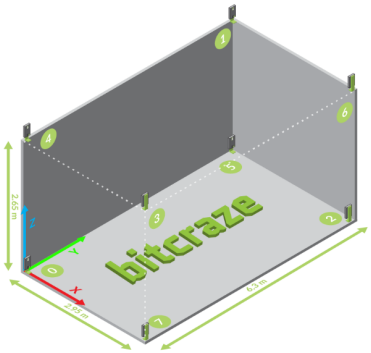
ID2: (2.43, 3.30, 1.67)

ID3: (0.26, 3.32, 0.26)

(a) Disposition du LPS dans le cadre de ce projet



(b) Disposition recommandée pour 6 ancres [2]



(c) Disposition recommandée pour 8 ancres [2]

Lorsqu'elles sont positionnées, il faut donner aux ancres leur position dans l'espace. Cela se fait à l'aide du cfclient (présenté à la section 7.1) ou à partir de Linux directement. Dans ce cas, le cfclient a été utilisé afin d'imposer les coordonnées aux ancres.

7 Interfaces utilisées

7.1 cfclient

Bitcraze a développé une interface appelée le cfclient permettant de se connecter au Crazyflie et d'y récolter ses informations de vol, de performance et de position. C'est à l'aide du cfclient qu'il est également possible de piloter le Crazyflie avec une manette de type "Gamepad" (le Crazyflie ne supporte pas à la base les manettes de type transmetteur). Dans ce projet, le cfclient n'a pas été utilisé pour voler le crazyflie, mais plutôt afin de configurer certains paramètres et valider la connexion. Dans le cadre de ce projet, l'installation du cfclient sur Windows a créé beaucoup de complications. Il a donc été décidé d'installer la machine virtuelle configurée par Bitcraze qui utilisait Linux comme système d'opération. Ainsi, le cfclient a entre autres permis de :

— Vérifier la connexion de la radio au Crazyflie

— Faire l'installation du nouveau firmware sur le Crazyflie

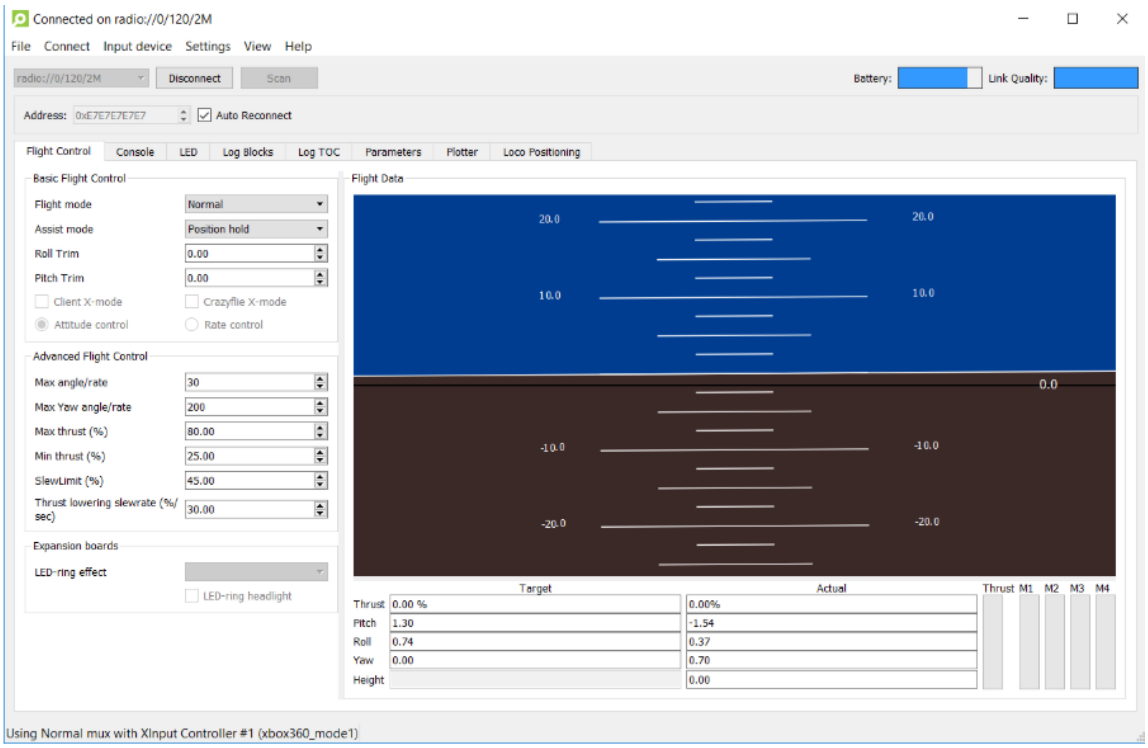
— Modifier l'adresse des Crazyflies pour permettre le vol à plusieurs drones

— Configurer le mode et la position des ancres du LPS

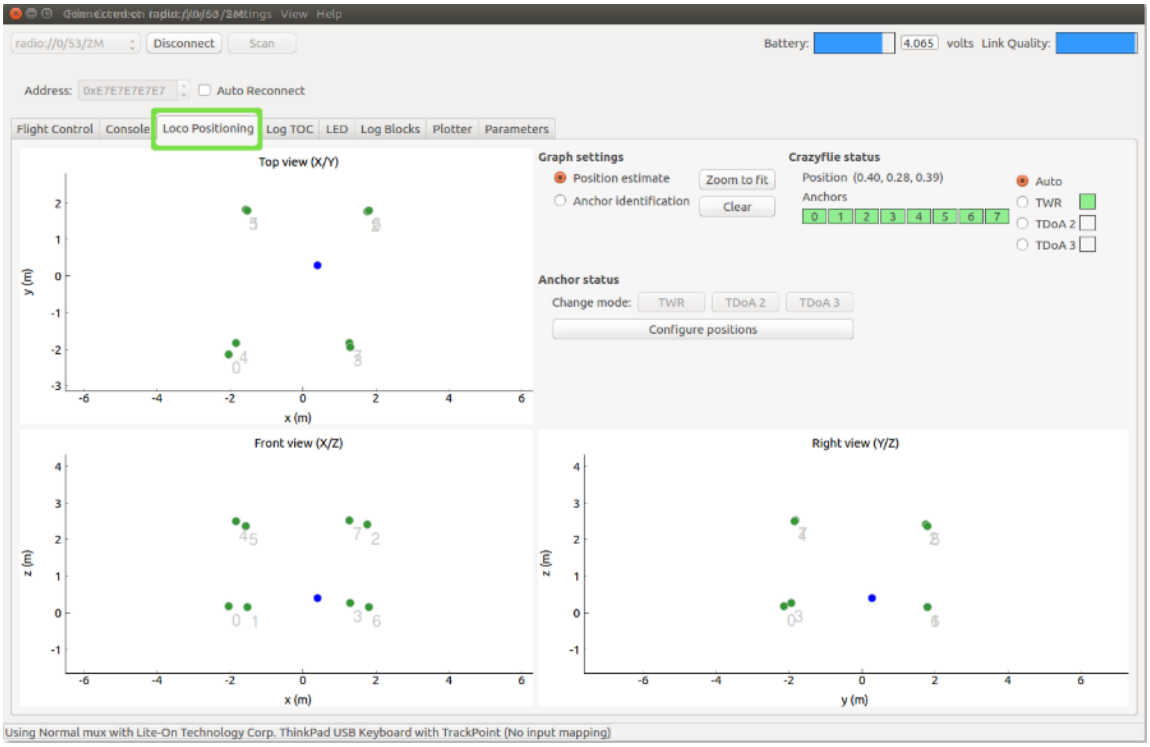
— Vérifier la communication entre le LPS et le Crazyflie

— Vérifier la charge de la batterie du Crazyflie

— Vérifier la position donnée par le LPS



(a) cfclient flight tab [2]



(b) cfclient loco tab [2]

7.2 Python

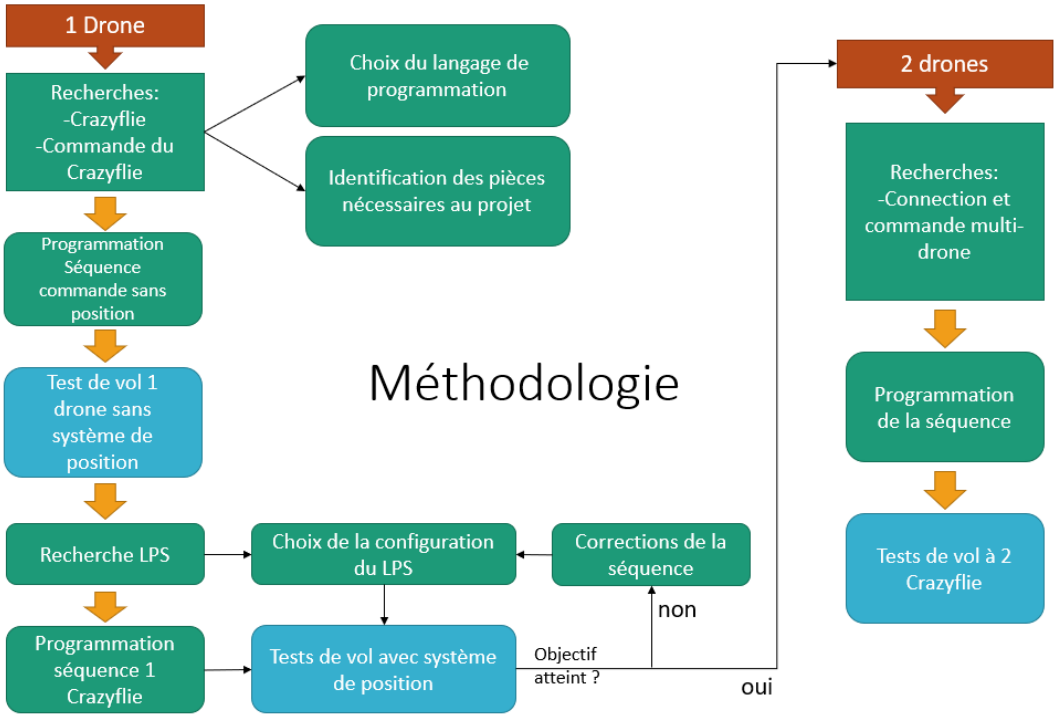
Les aspects de développement et de programmation de la séquence de vol ont été plutôt supportés par Python pour ce projet. La principale raison qui justifie ce choix est la présence de la librairie Python "cflib" développée par la communauté Bitcraze qui contient les packages nécessaires à la connexion au Crazyflie, à l'utilisation du système de positionnement ainsi qu'au vol à plusieurs drones. Comme programmer l'entièreté des fonctions permettant la communication radio et le soutien du système de positionnement serait trop poussé pour le cadre de ce projet, cette librairie publique permet de bâtir à partir de protocoles déjà fonctionnels.

Les deux packages utilisés de cette libraire sont "crtp" qui permet la connexion aux drones, puis "Crazyflie" qui permet la communication de plusieurs types de données avec celui-ci.

8 Méthodologie

La méthodologie de travail a été organisée en deux principales phases. La première consistait à bâtir un programme de commande à un drone. Cette stratégie permet de bien connaître les fonctionnalités du drone et d'avoir une séquence de vol autonome fonctionnelle avant l'intégration de plusieurs drones. Ainsi, la seconde étape consiste à ajuster ce programme à une communication multidrone. La figure suivante présente un résumé de la méthodologie de travail.

FIGURE 6: Résumé de la méthodologie de travail



8.1 Recherches Crazyflie et LPS

La première étape du processus vise à acquérir une compréhension du fonctionnement et des systèmes du Crazyflie. Cette étape a permis d'identifier les informations suivantes :

1. Les composantes du Crazyflie et du LPS ainsi que leur fonction
2. Les interfaces de développement possible
3. Les modes d'opération du LPS
4. Les librairies Python permettant le contrôle du Crazyflie

Les deux principales sources d'information sur le Crazyflie sont **Bitcraze** ainsi que **Github**.

8.2 Séquence sans positionnement

Avant les premiers tests, le Crazyflie ainsi que la PA Radio ont nécessité quelques préparations. Il faut s'assurer des étapes suivantes :

— Mise à jour du firmware du Crazyflie

— Installation du Driver de la Radio en libusb sur l'ordinateur

Le Crazyflie peut exécuter des commandes sans utilisation d'un système de positionnement. Il utilise alors son accéléromètre interne pour exécuter des mouvements relatifs. Les premiers tests ont donc été des commandes simples sans le LPS. Ce test a suivi le protocole suivant :

1. Recherche des Crazyflie disponibles par la CrazyRadio
2. Connexion au Crazflie
3. Envoi d'une commande de déplacements

Ce premier test permet de valider la communication de la radio au Crazyflie ainsi que la réponse de ce dernier à l'envoi de commande.

8.3 Intégration du LPS

Une fois la communication radio-drone établie, l'étape suivante consiste à intégrer le système de positionnement. L'outil LPS ainsi que le Cfclient ont permis la configuration des ancres et validé la communication entre le Crazyflie et le LPS. Les étapes nécessaires à la configuration du LPS sont :

— Mettre à jour le firmware sur toutes les ancres

— Assigner un identifiant aux ancres entre ID0 et ID8

— Configurer le mode d'opération des ancres à TdoA2

— Positionner les ancres dans l'air de vol

— Donner aux ancres leurs coordonnées dans un repère posé

Une fois ces étapes complétées, le système peut être utilisé afin de faire voler le Crazyflie. Cela amène au prochain test, qui est un test de position sur un Crazyflie immobile. Le programme permettant ce test tient la structure suivante :

1. Connexion à l'adresse du Crazyflie
2. Recherche des ancres disponibles
3. Connexion aux ancres
4. Commande au Crazyflie de calculer sa position
5. Enregistrement de la position dans le temps

Ce test permet d'analyser les différents modes du LPS et d'évaluer la précision de chacun. Il permet également de tester la communication du Crazyflie au système de positionnement.

Une fois l'intégration du système accompli, il est possible de programme la séquence de vol stationnaire pour un drone. Le test suivant est donc un vol stationnaire à un drone avec le système de positionnement. Ce test s'est déroulé comme suit :

1. Connexion à l'adresse du Crazyflie
2. Connexion aux ancres
3. Commande au Crazyflie de calculer sa position
4. Enregistrement de la position dans le temps
5. Envoi d'une position désirée

Ce test permet d'étudier la réponse du crazyflie à une commande position et d'évaluer la précision du LPS lorsque le drone est en mouvement.

8.4 Séquence multidrone

Une fois qu'une séquence de vol avec un drone a été programmée et que le système de positionnement est fonctionnel, on cherche à transférer cette séquence à plusieurs drones. L'objectif est de pouvoir utiliser la même séquence à envoyer à plusieurs drones en parallèle. Les points importants à déterminer pour le vol à drones multiples sont :

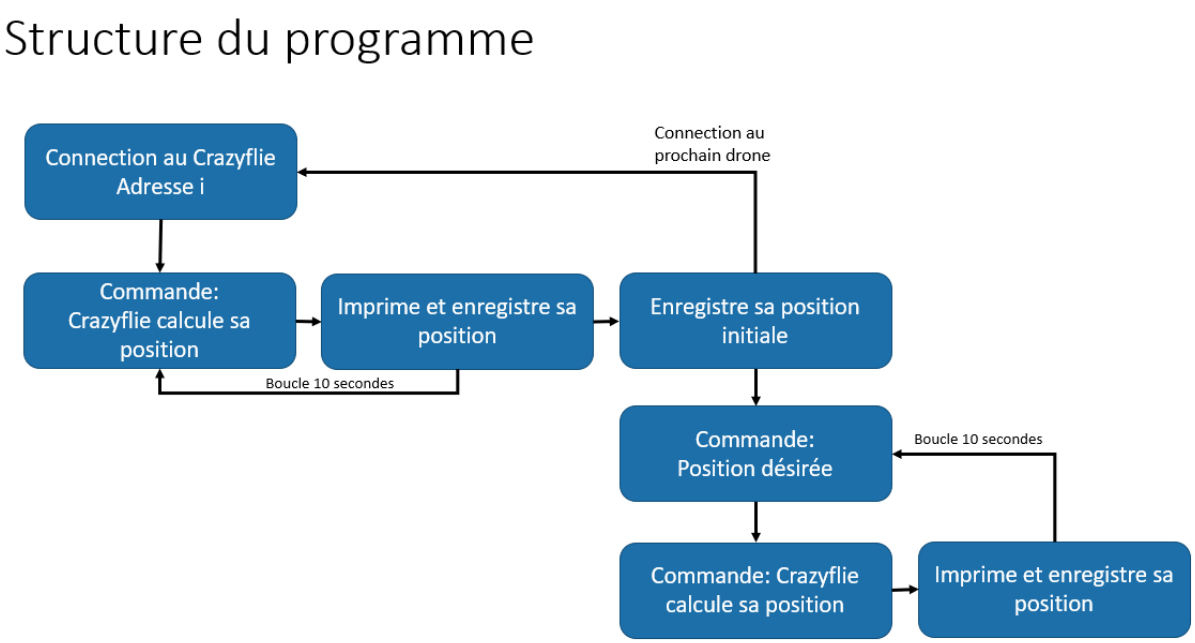
— Paramètres communs et non-communs aux drones dans la séquence

— Fonction permettant l'envoi en parallèle ou en séquentiel de commandes

9 Structure du programme

Cette section vise à décrire les différentes fonctions utilisées dans le programme de la séquence de vol ainsi que la structure générale. La figure suivante illustre un résumé de la stratégie du programme. Le code complet se trouve en annexe.

FIGURE 7: Structure du programme "Main swarm"



9.1 Connexion aux Crazyflie

Les adresses des deux Crazyflie ont été modifiées à

- radio ://0/80/2M/E7E7E7E701
- radio ://0/80/2M/E7E7E7E702

Les étapes pour la connexion sont les suivantes :

1. Initialiser les drivers à l'aide du module **crtp** dans la librairie **cflib**. Cela permet d'initialiser la PA Radio
2. La fonction **cflib.crpt.scan_interfaces()** permet d'identifier les adresses des drones disponibles si ceux-ci sont allumés. Cette étape n'est pas nécessaire si l'on connait déjà les adresses.
3. Créer un vecteur URIS avec toutes les adresses des Crazyflies

4. Pour se connecter à un seul drone :
   with SyncCrazyflie(link_uri, cf=Crazyflie(rw_cache='./cache')) as scf :
5. pour se connecter à plusieurs drones :
   factory= CachedCfFactory(rw_cache='./cache'
   with Swarm(URIS, factory=factory) as swarm :

9.2 Calcul de position initiale

Comme les réponses de position des Crazyflie ne sont pas à intervalles réguliers, il n'était pas possible de séparer les positions enregistrées à chaque Crazyflie en envoyant la commande en simultanée. C'est pourquoi la position initiale est déterminée comme suit :

1. Connexion à un drone
2. La fonction **wait_for_position_estimator()** estime la position chaque 100ms et l'enregistre dans un historique. La fonction continue jusqu'à convergence de la position. Le seuil de convergence est posé ici lorsque $(max_{x,y,z} - max_{x,y,z}) < 0.001$
3. Lorsque la convergence est atteinte, la fonction **reset_estimator()** réinitialise le filtre Kalman qui permettra ensuite d'obtenir les données de position du Crazyflie.
4. La fonction **start_position_printing()** crée une nouvelle variable "kalman.state" pour chaque coordonnée dans un registre. Elle utilise un callback qui active la fonction **position_callback()** lorsque les données de position sont reçues.
5. La fonction **position_callback()** enregistre les coordonnées dans des variables globales et imprime la position instantanée pour une période de 10 secondes
6. Le lien est fermé puis la position initiale est calculée comme la moyenne des coordonnées enregistrées dans la boucle de 10 secondes
7. Les étapes précédentes sont répétées avec chaque Crazyflie utilisé et leurs positions initiales sont enregistrées dans un vecteur

Les requêtes de position sont envoyées à chaque 100ms, ce qui permet d'avoir beaucoup de données de position pour calculer la position initiale. Le choix d'utiliser la moyenne sur une période de 10 secondes a été fait dans le but d'éviter des erreurs importantes lors d'une mauvaise communication momentanée qui enregistrerait une position aberrante.

9.3 Commande multidrone

Afin de pouvoir envoyer des commandes simultanément à plusieurs drones, il faut regrouper les adresses dans une liste et les paramètres nécessaires à la séquence dans une seconde liste. Le programme suit la séquence suivante pour envoyer des commandes au Crazyflie :

1. Calcul de la position initiale de chaque drone
2. Définition des paramètres de la séquence de la forme :
   params_hover = {uri_cf1 : [Position_hover_cf1, position_initiale_cf1], uri_cf2 : [Position_hover_cf2, position_initiale_cf2]}
3. Connexion à tous les Crazyflie
4. Utilisation du module Swarm pour calculer la position pendant 5 secondes :
   swarm.parallel(reset_estimator)
   swarm.parallel(start_position_printing)
5. La fonction **run_sequence_hover** exécute la commande de vol stationnaire

Avec le système de positionnement c'est le module Commander du package Crazyflie qui permet d'envoyer des positions désirées au drone. La fonction qui exécute la séquence de vol stationnaire nécessite une liste avec les déplacements $(\Delta X, \Delta Y, \Delta Z)$. Il peut prendre plusieurs groupes de déplacements désirés. Elle ne tient présentement pas compte des déplacements désirés en x et y, mais ces entrées ont été conservées dans le but de développements futurs. La fonction exécute comme suit :

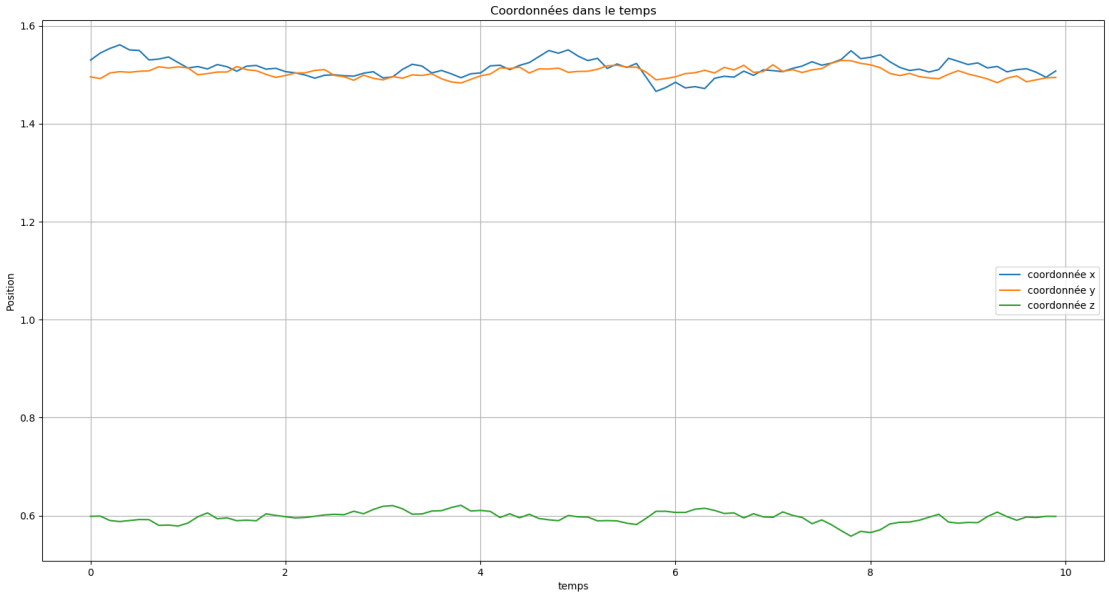
1. Pour chaque hauteur de vol désirée, la fonction **commander.send_hover_setpoint(vx, vy, yarate, zdistance)** envoie la position en z à partir de la position initiale et du déplacement désiré pour une période de 10 secondes
2. Pour l'atterrissage, la fonction **send_velocity_world_setpoint(vx, vy, vz, yawrate)** permet de commander une vitesse de descente qui dépend de la hauteur de la dernière position de vol stationnaire.

10 Résultats

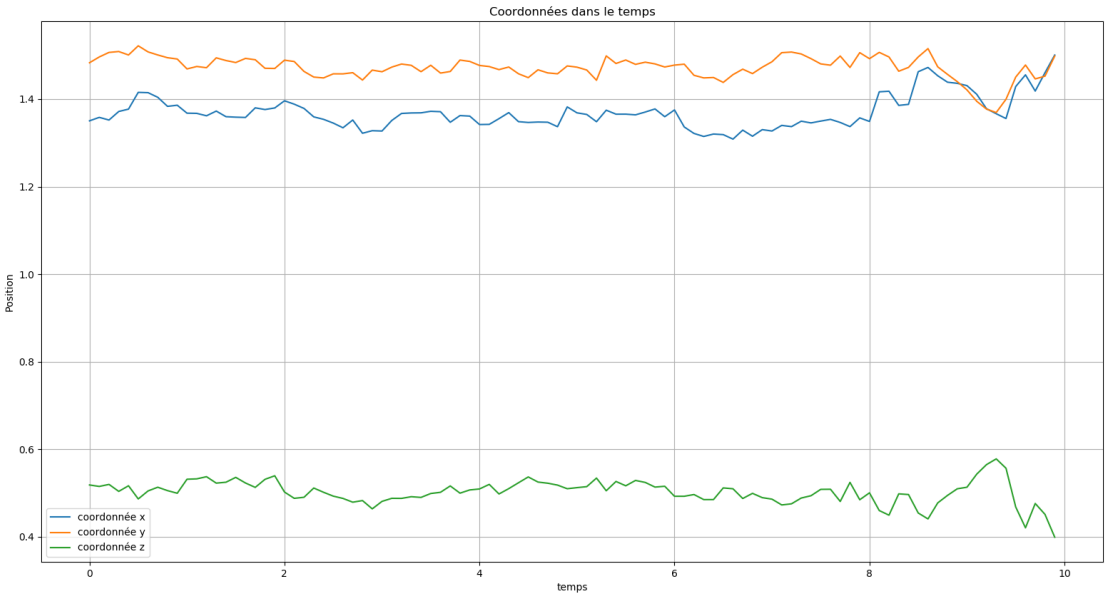
10.1 Séquence immobile à un Crazyflie

Le premier test effectué afin d'évaluer la précision du système de positionnement a été simplement un test immobile demandant au Crazyflie de calculer sa position. Le test a été effectué avec 4 ancres disposées tel que présenté précédemment. Celles-ci ont été configurées successivement dans les trois modes d'opération afin d'évaluer la précision de chacun. Plusieurs tests ont été effectués afin de valider la réponse des trois modes. Les 3 figures suivantes montrent les résultats d'un des tests sur une séquence de 10 secondes sans décollage.

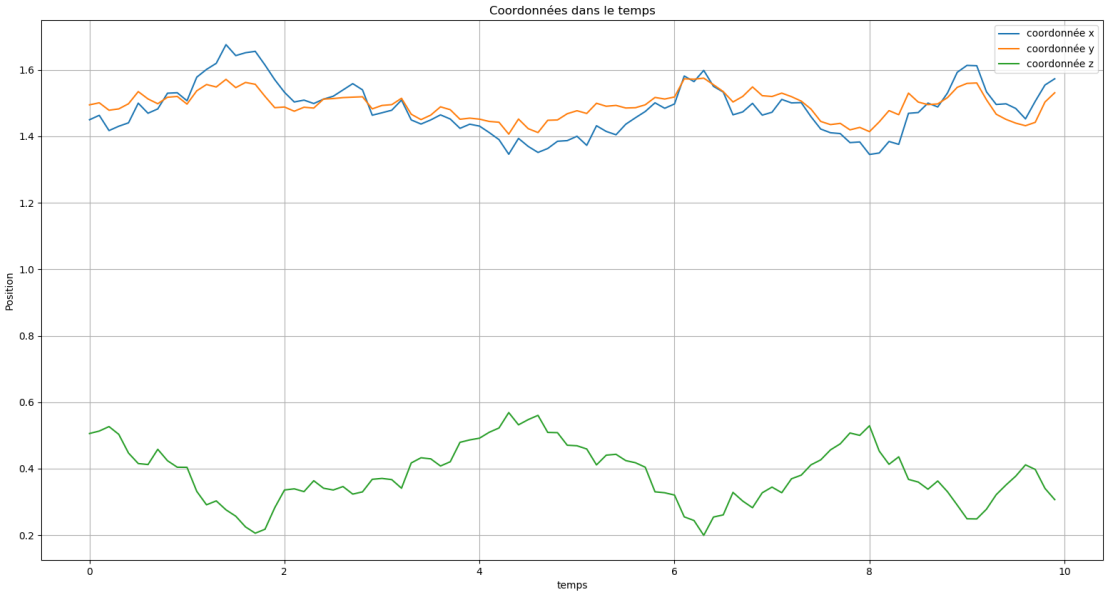
FIGURE 8: Test de position pour un Crazyflie immobile



(a) TWR



(b) TdoA2



(c) TdoA3

On peut observer des 3 graphiques que le mode le plus précis est le TWR, avec une variation de moins de 10cm. Le mode TdoA2 montre une précision de +/- 20cm. Pour un petit aire de vol, une variation de cet ordre rend le vol instable et ne permet pas de faire des manoeuvres complexes. Toutefois cette précision peut être améliorée en augmentant le nombre d'ancres puisque cela permet de créer de la redondance dans le système. Finalement, le mode TdoA3 montre une mauvaise précision de l'ordre de 40cm. Tout comme TdoA2, augmenter le nombre d'ancres permettrait d'améliorer l'exactitude des données

de position. Ces tests ont permis de confirmer les avantages et désavantages de chacun des modes.

	Avantages	Désavantages
TWR	-Meilleure précision	-Permet un seul Crazyflie -Maximum de 8 ancres
TdoA2	-Meilleure précision que TdoA3 -Permet vol de plusieurs drones	-Maximum de 8 ancres
TdoA3	-Possibilité de plus 8 ancres -Permet vol de plusieurs drones	-Mauvaise précision à 4 ancres

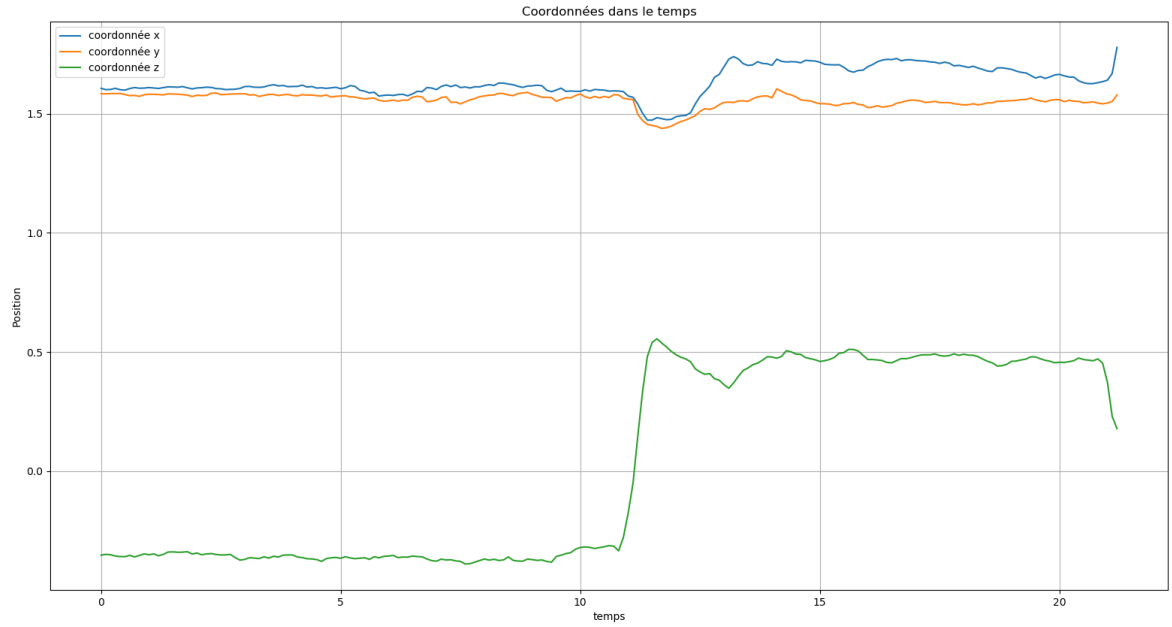
Tableau 3: Choix du mode du LPS

Le mode TWR a été éliminé puisqu'il ne permet pas le vol à plusieurs drones. Également Comme l'aire de vol pour ce projet était restreint et que seulement 4 ancres ont été utilisées, l'avantage du TdoA3 n'était pas applicable. C'est donc le mode **TdoA2** qui a été choisi.

10.2 Séquence stationnaire un Crazyflie

Une fois le mode des ancres configurées, la séquence de vol stationnaire a d'abord été conçue pour un drone. La figure suivante montre les résultats des coordonnées d'un test d'une séquence de vol stationnaire à 0.5m pendant 10 secondes. Sa position initiale réelle dans le repère imposé était de (1.4, 1.45, 0.0)

FIGURE 10: Séquence de vol stationnaire



Les points forts des résultats de ce test sont :

- Stabilité du vol
- Hauteur désirée atteinte et maintenue

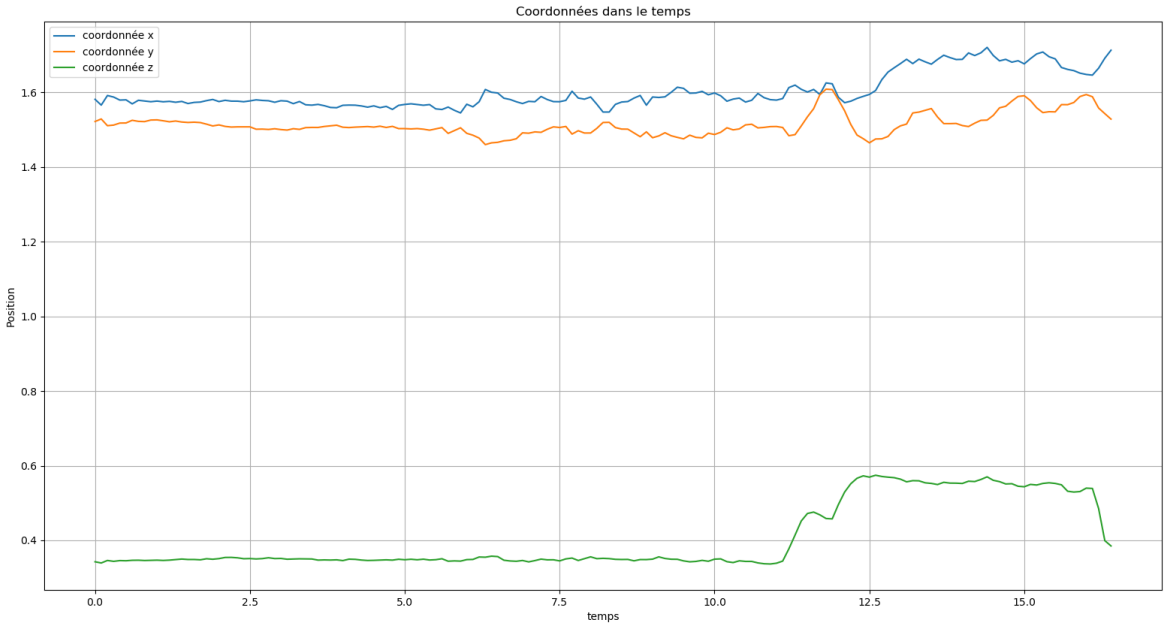
Les points faibles sont :

- Position absolue erronée
- légère dérive en x et y au décollage et à la fin de la séquence

Suite à plusieurs tests, il a été identifié que la position absolue initiale perçue par le Crazyflie est imprécise, particulièrement pour les coordonnées en z (ici environ -0.3m). Il a été testé de modifier la hauteur initiale du Crazyflie afin de vérifier si la position en z était influencée par la réflexion sur le sol. Toutefois, même surélevé, la position perçue montrait une imprécision de l'ordre de 10 à 30cm en z.

Afin d'évaluer la différence de cette précision avec un autre drone, le même test a été conduit sur un second Crazyflie :

FIGURE 11: Séquence de vol stationnaire



On remarque que la position absolue perçue par un second Crazyflie est différente et toujours erronée (tendance sur plusieurs tests). Il y a donc une influence du Loco Deck ainsi que du drone sur l'imprécision. Comme il n'a pas été possible d'ajuster la précision du LPS sur la position absolue avec davantage d'ancres, la séquence a été programmée de sorte d'imposer la position initiale comme la position perçue plutôt que comme la position initiale réelle. La séquence commande ainsi un déplacement plutôt qu'une position désirée dans l'espace.

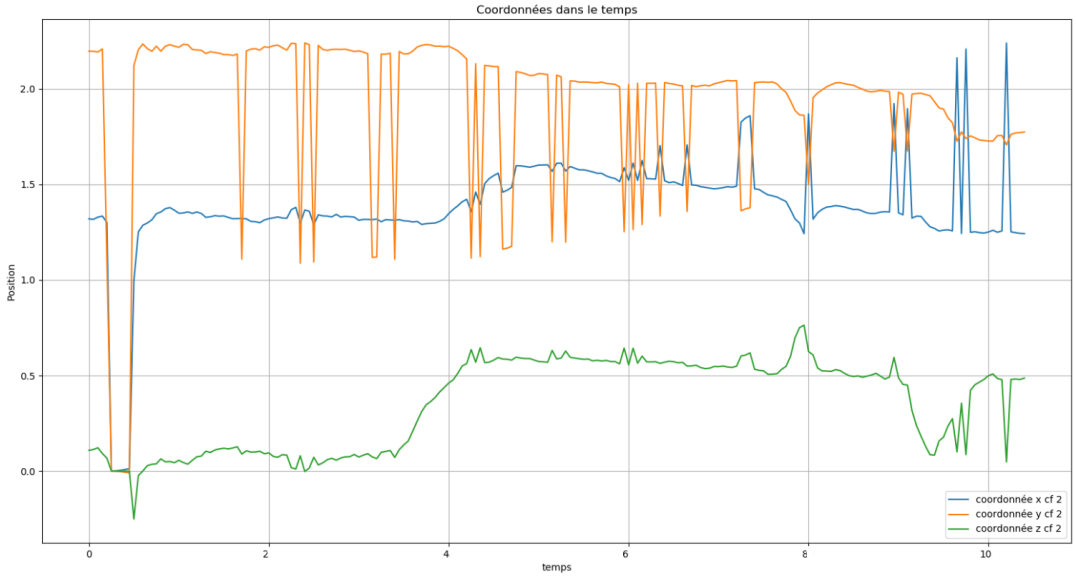
Au niveau de la qualité du vol stationnaire, la variation de la position perçue était faible, ce qui permettait tout de même un vol stable pour les deux drones. Le problème d'imprécision est donc orienté surtout sur la position absolue et non sur la variation de position perçue.

10.3 Vol stationnaire à deux Crazyflie

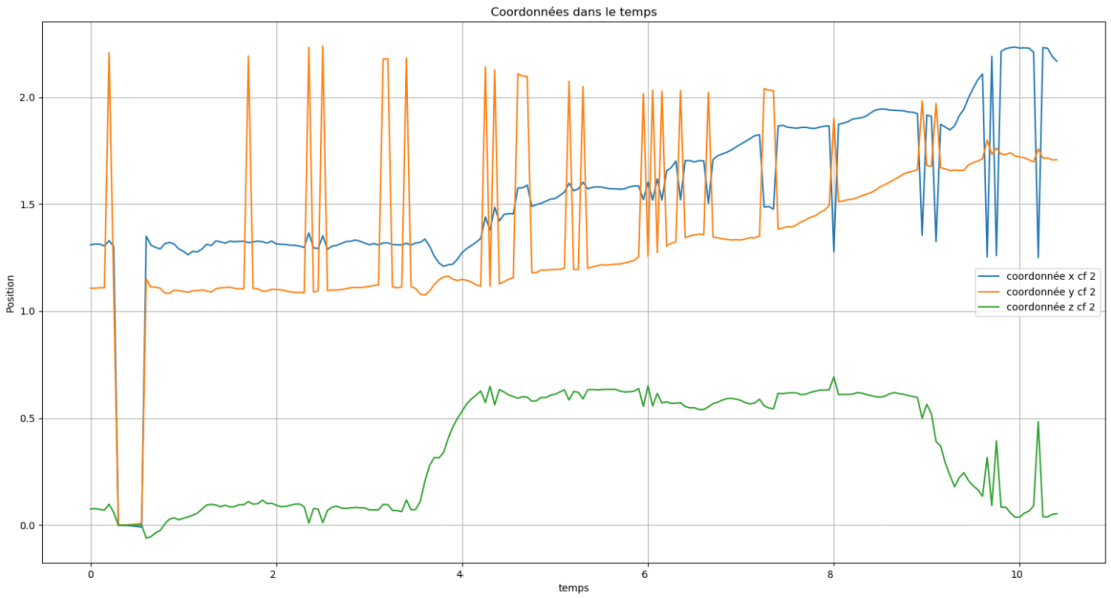
Le programme utilisé pour la séquence de vol à un drone a ensuite pu être ajusté pour la communication à plusieurs drones. C'est la fonction **Swarm** dans la librairie cflib qui a été choisie pour envoyer la séquence de vol en parallèle aux 2 Crazyflie. Pour que la radio puisse capter plusieurs Crazyflie, il a d'abord fallu modifier leurs adresses des afin qu'elles soient différentes.

Dans la prochaine section, la fonction qui permet la commande multidrone "Swarm" sera présentée plus en détail. Il est toutefois intéressant dans cette section de discuter des positions enregistrées par les drones lors de la démonstration de vol. Les deux figures suivantes montrent la position enregistrée pour chacun de drone dans un test de vol stationnaire à deux drones. Dans ce test, les Crazyflies sont commandés de décoller après 3 secondes, monter de 0.5m puis de rester à leur position pendant 5 secondes.

FIGURE 12: Test de vol stationnaire à deux Crazyflies

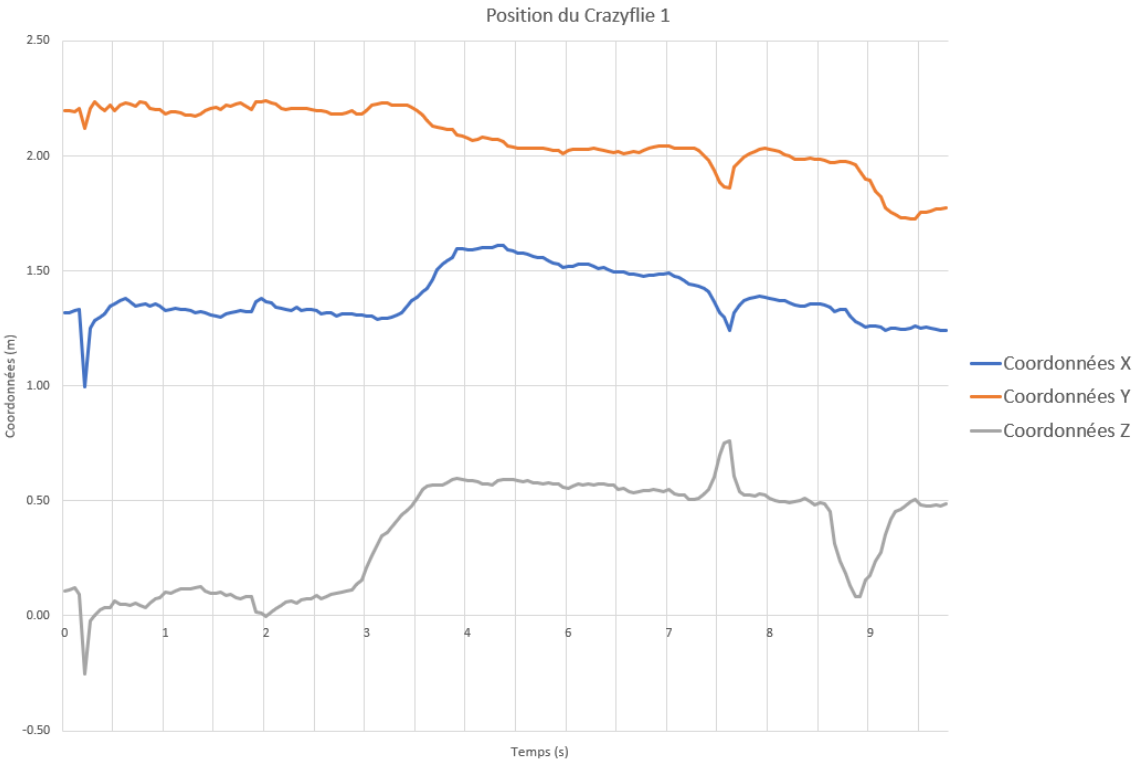


(a) Coordonnées enregistrées du Crazyflie 1

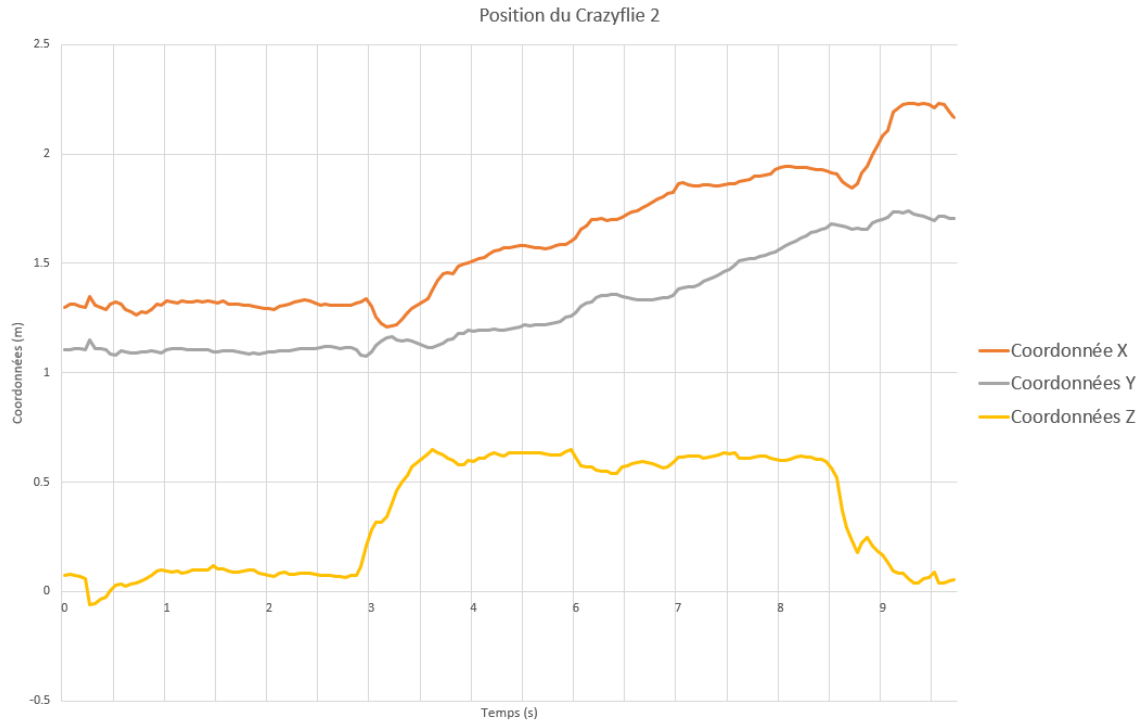


(b) Coordonnées enregistrées du Crazyflie 2

On observe que la position enregistrée oscille énormément. Cela est dû au fait que la réponse de position qu'envoient les Crazyflies n'est pas à un intervalle régulier même si la commande de requête de position est envoyée à chaque 100ms. Ainsi, le Crazyflie 01 pourrait envoyer trois données de position pour une du Crazyflie 02. Il n'a donc pas été possible d'enregistrer les positions de chacun des drones automatiquement. Suite à un classement à la main des données de position, voici le résultat du vol stationnaire.



(a) Coordonnées classées du Crazyflie 1



(b) Coordonnées classées du Crazyflie 1

On observe qu'il persiste de légères oscillations de 0.1m, mais que la position en z est bien maintenue par les deux Crazyflies. La problématique dans ce test est plutôt dans la dérive

en x et y. Ceci est le cas pour les deux drones, mais plus prononcé pour le Crazyflie 2 qui dérive de 1m de sa position initiale. Cela est dû au fait que la séquence utilise la fonction **commander.send_hover_setpoint(vx, vy, yawrate,zdist**). La séquence utilise le module Commander pour envoyer au Crazyflie des vitesses en x et y, un taux de lacet de 0 rad/s puis une hauteur à atteindre basée sur un déplacement désiré. Dans ce cas, comme la position x et y n'est pas fixée par une commande, le drone ne tente pas de maintenir sa position initiale x et y. Il va ainsi dériver à une vitesse constante qui, avec les imprécisions du drone, n'est pas nulle dans ce cas.

11 Analyse

11.1 Difficultés rencontrées

Position avec LPS Les principales difficultés rencontrées ont été reliées à la précision du système de positionnement. La configuration initiale des ancres a pris beaucoup plus de temps que prévu et a également été ajustée à plusieurs reprises afin d'améliorer la précision. Notamment, les ancres ont été disposées en configuration diagonale plutôt que symétrique, puis elles ont été éloignées des murs et du sol pour réduire la réflexion sur les surfaces.

Ensuite, l'inexactitude de la position absolue a amené à plusieurs modifications sur la stratégie des commandes de la séquence. Plutôt que d'être basée sur une position désirée dans le repère, les commandes ont été orientées vers un déplacement désiré. Ainsi, la différence de précision sur la position initiale entre les drones avait moins d'effet sur le résultat de l'action du drone. Le manque de précision du système de position avec 4 ancres avait tout de même un impact sur la stabilité générale du vol stationnaire.

Log de données à multiples drones La deuxième difficulté rencontrée a été la séparation des données reçues des deux Crazyflie. Comme la réponse de position ne se faisait pas à un intervalle régulier, les positions enregistrées n'étaient pas séquentielle d'un drone puis de l'autre. Cela rendait aléatoire l'assignation des données. Ainsi, pour des fins de visualisations des résultats, les données de positions de la séquence à deux drones ont été triées à la main. Toutefois, pour de longues séquences avec beaucoup de données, cela deviendrait non viable.

Dérive dans les axes x et y Les difficultés rencontrées avec la position absolue a poussé certains changements de la séquence. Initialement, la commande envoyée au Crazyflie était une position désirée par la fonction **set_position_setpoint(x, y, z,yawrate)**. Comme la position absolue était très différente de la position réelle, la position désirée ne permettait pas de faire simplement lever le drone. La séquence a donc été modifiée pour imposer seulement une distance en z. Toutefois, la fonction **send_hover_setpoint(vx, vy, yawrate, zdist** ne permet pas d'imposer un contrôle sur la position x et y. Comme l'implémentation du calcul de la position initiale de chaque drone a été ajoutée à la fin

du projet, il ne restait pas assez de temps pour remodifier la séquence avec une structure axée sur une position désirée (x,y,z).

11.2 Améliorations proposées et développements possibles

Les difficultés rencontrées ainsi que les résultats ont soulevé des aspects qui pourraient être améliorés ou davantage poussés. On note les points suivants :

- Augmenter le nombre d'ancres utilisées afin d'évaluer son influence sur la précision de la position
- Tester une séquence de vol à plus de deux drones afin d'évaluer l'influence sur la communication et la précision
- Investiguer une fonction de commande multidrone permettant une communication inter drones
- Complexifier la séquence de vol stationnaire
- Investiguer les autres possibilités de systèmes de positionnement plus précis, notamment le VICON
- Structurer les commandes sur des positions désirées qui contraignent en (x,y,z) plutôt que des déplacements en z désirés

12 Conclusion

En conclusion, l'objectif principal du projet de programmer une démonstration de vol en formation a été atteint. Il a été possible d'envoyer des commandes à deux drones en simultané dans une séquence de vol stationnaire. Le système de position Loco a été intégré au vol avec 4 ancres.

La méthodologie suivie a permis de concevoir une base solide de commande à un drone qui a pu être ajusté à une communication multidrones. Cela a également permis d'évaluer la précision du système et des Crazyflie sur la position pour différents scénarios. Il a été identifié que la précision du LPS à 4 ancres est faible quant à la position absolue perçue par le Crazyflie, mais permet tout de même un vol stationnaire relativement stable.

Ainsi, il serait intéressant d'évaluer dans un projet futur l'impact d'augmenter le nombre d'ancres et de Crazyflies utilisés ainsi que de tester une séquence de vol autonome plus complexe qu'un vol stationnaire si la précision obtenue le permet.

Cette étude ne couvre qu'en surface les possibilités de développement et de recherche avec le drone Crazyflie. Ce drone open source permet d'étudier et de modifier son contrôleur ou d'intégrer différents systèmes de positionnement tels VICON ou Lighthouse. Il y a donc beaucoup de développement de contrôle autonome possible avec ce quadricoptère.

ANNEXES

A Code Python du programme développé

File - C:\Users\Marie-Eve\Google Drive\Université\3e année\Projet 3\main_swarm_remise.py

```
 1 import math
 2 import time
 3 import matplotlib.pyplot as plt
 4 import numpy as np
 5
 6 import cflib.crtp
 7 from cflib.crazyflie import Crazyflie
 8 from cflib.crazyflie.log import LogConfig
 9 from cflib.crazyflie.syncCrazyflie import SyncCrazyflie
10 from cflib.crazyflie.syncLogger import SyncLogger
11 from cflib.crazyflie.swarm import CachedCfFactory
12 from cflib.crazyflie.swarm import Swarm
13
14
15 #nombre de drones
16 nb_cf=2
17
18 # URI to the Crazyflie to connect to
19 uri_1 = 'radio://0/80/2M/E7E7E7E701'
20 uri_2='radio://0/80/2M/E7E7E7E702'
21
22 uris={uri_1, uri_2}
23
24 def wait_for_position_estimator(scf):
25     print('Waiting for estimator to find position...')
26
27     log_config = LogConfig(name='Kalman Variance',
   period_in_ms=100)
28     log_config.add_variable('kalman.varPX', 'float')
29     log_config.add_variable('kalman.varPY', 'float')
30     log_config.add_variable('kalman.varPZ', 'float')
31
32     var_y_history = [1000] * 10
33     var_x_history = [1000] * 10
34     var_z_history = [1000] * 10
35
36     threshold = 0.001
37
38     with SyncLogger(scf, log_config) as logger:
39         for log_entry in logger:
40             data = log_entry[1]
41
42             var_x_history.append(data['kalman.varPX'])
43             var_x_history.pop(0)
44             var_y_history.append(data['kalman.varPY'])
45             var_y_history.pop(0)
```

```
46              var_z_history.append(data['kalman.varPZ'])
47              var_z_history.pop(0)
48
49              min_x = min(var_x_history)
50              max_x = max(var_x_history)
51              min_y = min(var_y_history)
52              max_y = max(var_y_history)
53              min_z = min(var_z_history)
54              max_z = max(var_z_history)
55
56
57              if (max_x - min_x) < threshold and (
58                      max_y - min_y) < threshold and (
59                      max_z - min_z) < threshold:
60                  break
61
62
63 def reset_estimator(scf):
64     cf = scf.cf
65     cf.param.set_value('kalman.resetEstimation', '1')
66     time.sleep(0.1)
67     cf.param.set_value('kalman.resetEstimation', '0')
68
69     wait_for_position_estimator(cf)
70
71 def flight_data_callback(timestamp, data, logconf):
72     global roll
73     global pitch
74     global yaw
75     global thrust
76
77     roll_d=data['stabilizer.roll']
78     pitch_d = data['stabilizer.pitch']
79     yaw_d = data['stabilizer.yaw']
80     thrust_d=data['stabilizer.thrust']
81
82     roll.append(roll_d)
83     pitch.append(pitch_d)
84     yaw.append(yaw_d)
85     thrust.append(thrust_d)
86
87     print('pos:_({},_{},_{})'.format(roll_d,pitch_d,yaw_d,
   thrust_d))
88
89 def print_flight_data(scf):
90     lg_stab = LogConfig(name='Stabilizer', period_in_ms=100)
```

```
 91      lg_stab.add_variable('stabilizer.roll', 'float')
 92      lg_stab.add_variable('stabilizer.pitch', 'float')
 93      lg_stab.add_variable('stabilizer.yaw', 'float')
 94      lg_stab.add_variable('stabilizer.thrust', 'float')
 95
 96      cf = Crazyflie(rw_cache='./cache')
 97
 98      scf.cf.log.add_config(lg_stab)
 99      lg_stab.data_received_cb.add_callback(
    flight_data_callback)
100      lg_stab.start()
101
102  def position_callback(timestamp, data, logconf):
103      global x_pos
104      global y_pos
105      global z_pos
106
107
108      x=data['kalman.stateX']
109      y = data['kalman.stateY']
110      z = data['kalman.stateZ']
111
112      x_pos=np.append(x_pos,x)
113      y_pos = np.append(y_pos, y)
114      z_pos = np.append(z_pos, z)
115      print('pos:_({},_{},_{})'.format(x,y,z))
116
117  def start_position_printing(scf):
118      log_conf=LogConfig(name='Position', period_in_ms=100)
119      log_conf.add_variable('kalman.stateX','float')
120      log_conf.add_variable('kalman.stateY', 'float')
121      log_conf.add_variable('kalman.stateZ', 'float')
122
123      scf.cf.log.add_config(log_conf)
124      log_conf.data_received_cb.add_callback(position_callback
    )
125      log_conf.start()
126
127  def landing(cf,last_pos):
128      print("Landing!")
129      time.sleep(1)
130      vz=0.2
131      cf.commander.send_velocity_world_setpoint(0.0, 0.0, -vz
    , 0.0)
132
133      if (last_pos[2]*10/vz)<50:
```

```
134         for i in range(50):
135             cf.commander.send_hover_setpoint(0.0, 0.0, 0.0,
    0)
136             time.sleep(0.1)
137     else:
138         for i in range(last_pos[2]*10/vz):
139             cf.commander.send_hover_setpoint(0.0, 0.0, 0.0,
    0)
140             time.sleep(0.1)
141 
142 def take0ff(cf,pos_init):
143     print("Take off")
144     time.sleep(1)
145     take_off_time=1
146     take_off_height=0.5
147     vz = take_off_height/take_off_time
148 
149     for i in range(int(take_off_time/0.1)):
150         cf.commander.send_velocity_world_setpoint(0.0, 0.0,
    vz, 0.0)
151         time.sleep(0.1)
152 
153 def run_sequence_hover(scf, hover_pos,pos_initiale):
154     print("starting hover sequence")
155     cf = scf.cf
156 
157     #Hover sequence
158     for position in hover_pos:
159         print('Setting position {}'.format(position))
160         time.sleep(2)
161 
162         for i in range(100):
163             cf.commander.send_hover_setpoint(0.0, 0.0, 0.0,
    pos_initiale[2]+position[2])
164             time.sleep(0.1)
165 
166     #Landing
167     last_pos=(pos_initiale[0], pos_initiale[1], pos_initiale
    [2]+hover_pos[-1][2])
168     land_time=1
169     vzl=last_pos[2]/land_time
170     time.sleep(0.5)
171 
172     for i in range(int(land_time / 0.1)):
173         cf.commander.send_velocity_world_setpoint(0.0, 0.0
    , -vzl, 0.0)
```

```
174         time.sleep(0.1)
175
176     cf.commander.send_setpoint(0, 0, 0, 0)
177     time.sleep(0.5)
178
179     cf.commander.send_stop_setpoint()
180     time.sleep(0.5)
181
182
183 if __name__ == '__main__':
184     cflib.crtp.init_drivers(enable_debug_driver=False)
185
186     x_pos=[]
187     y_pos=[]
188     z_pos=[]
189     roll=[]
190     pitch=[]
191     yaw=[]
192     thrust=[]
193     position_initiale=[]
194
195     #Calcul de position intiale
196     for i in uris:
197         with SyncCrazyflie(i, cf=Crazyflie(rw_cache='./cache
    ')) as scf:
198             reset_estimator(scf)
199             start_position_printing(scf)
200             time.sleep(6)
201             scf.close_link()
202             initial_x = np.mean(x_pos)
203             initial_y = np.mean(y_pos)
204             initial_z = np.mean(z_pos)
205             position_initiale.append((initial_x,initial_y,
    initial_z))
206             x_pos = []
207             y_pos = []
208             z_pos = []
209
210     print('got initial position')
211     time.sleep(2)
212
213     # Paramètre séquence vol stationnaire
214     hover_sequence = [(0.0, 0.0, 0.5)]
215     params_hover = {uri_1: [hover_sequence,
    position_initiale[0]], uri_2: [hover_sequence,
    position_initiale[1]]}
```

```
216
217     #Run Sequence
218     factory= CachedCfFactory(rw_cache='./cache')
219     with Swarm(uris, factory=factory) as swarm:
220
221         #get position
222         swarm.parallel(reset_estimator)
223         time.sleep(1)
224
225         #flight data printing
226         print_flight_data(scf)
227
228         #position printing
229         swarm.parallel(start_position_printing)
230         time.sleep(5)
231
232         #Hover sequence
233         swarm.parallel(run_sequence_hover, args_dict=
    params_hover)
234         time.sleep(1)
235         swarm.close_links()
236
237     #Visualisation de la position
238     #Arranger positions des nb_cf drones
239     x_fin = np.zeros((nb_cf,int(len(x_pos)/nb_cf)))
240     y_fin = np.zeros((nb_cf,int(len(y_pos)/nb_cf)))
241     z_fin = np.zeros((nb_cf,int(len(z_pos)/nb_cf)))
242
243     for i in range(0,nb_cf):
244         x_fin[i,:]=x_pos[i::2]
245         y_fin[i, :] = y_pos[i::2]
246         z_fin[i, :] = z_pos[i::2]
247
248     print("link closed")
249     time.sleep(1)
250
251     #Print position
252     for j in range(0,nb_cf):
253         a = np.arange(0, ((len(x_fin[0])) / 2), 0.5)
254         plt.plot(a, x_fin[j,:], label="coordonnée x cf %d"
    %(i+1))
255         plt.plot(a, y_fin[j,:], label="coordonnée y cf %d"
    %(i+1))
256         plt.plot(a, z_fin[j,:], label="coordonnée z cf %d"
    %(i+1))
257
```

```
258         plt.xlabel('temps')
259         plt.ylabel('Position')
260         plt.title('Coordonnées dans le temps')
261         plt.legend()
262         plt.grid()
263         plt.show()
264
265
```

Références

[1] Bitcraze. *Crazyflie 2.1*. URL : https://www.bitcraze.io/products/crazyflie-2-1/. (accessed : 24.04.2020).

[2] Bitcraze. *Getting started with the Loco Positioning system*. URL : https://www.bitcraze.io/documentation/tutorials/getting-started-with-loco-positioning-system/. (accessed : 24.04.2020).

[3] Bitcraze. *Loco Positionning System*. URL : https://www.bitcraze.io/products/loco-positioning-system/. (accessed : 24.04.2020).

[4] GitHub. *lps node firmware*. URL : https://github.com/bitcraze/lps-node-firmware/releases. (accessed : 25/04/2020).

[5] Bitcraze Store. *Positioning*. URL : https://store.bitcraze.io/collections/positioning. (accessed : 24.04.2020).

[6] Bitcraze Wikibitcraze. *TDoA2 VS TDoA3*. URL : https://wiki.bitcraze.io/doc:lps:tdoa:tdoa2-vs-tdoa3. (accessed : 24.04.2020).