Université de Lille

POLYTECH LILLE

04/05/2020

Rapport du Projet IMA3-4 pour le semestre 8

Crazyflie & Projet Godot

P21 : « A la découverte des commandes des drones »

Kadir Tekin, Jawad Soulaimani, Selim Bensalem

ENCADRANTS : M. REDON ET M. NAKRACHI

Remerciements

Nous voulons dans un premier temps remercier notre encadrant de projet M. Nakrachi, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter notre réflexion.

Nous remercions également M. Redon pour ses conseils, et aussi pour nous avoir accordé un nouveau projet. Grâce à son nouveau projet, nous avons pu mener à bien notre travail pendant cette période de confinement.

Enfin, nous adressons nos sincères remerciements à tous les professeurs, intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé nos réflexions et ont accepté de nous rencontrer et de répondre à nos questions durant nos recherches.

À tous ces intervenants, nous présentons nos remerciements, notre respect et notre gratitude.

Sommaires

Introduction

Au cours de ce rapport nous allons retracer notre travail au cours du semestre 8. A cause du COVID19, nous avons dû interrompre notre travail sur le Crazyflie pour passer à un projet accessible avec le matériel du bord. Nous allons donc diviser ce rapport en deux parties : une partie sur le Crazyflie et une autre sur le projet Godot.

Tout d'abord, rappelons ce qui a été fait les semestres derniers avec le Crazyflie :

- semestre 6 : explications du fonctionnement d'un quadricoptère avec des schémas publiés sur le wiki. En fin de semestre nous avons reçu le drone, ce qui nous a permis de faire une maquette pour illustrer notre étude théorique.
- semestre 7 : programmation d'algorithmes python permettant d'effectuer un vol autonome d'un point A à un point B tout en évitant des obstacles. Ce code est interprété par l'ordinateur et envoie des commandes au drone au cours du temps (mode station de contrôle).

Nos objectifs initiaux étant atteints, nous avons défini une nouvelle orientation pour notre projet.
Lors de notre soutenance à la fin du S7, on nous a conseillé de creuser un peu plus dans l'architecture matérielle et logicielle du drone afin de mieux comprendre son fonctionnement et éviter l'utilisation d'un PC et de lib python pour faire voler le drone.

De plus nous allons mettre en place un réseau de capteur nous permettant de localiser la position de notre drone par rapport à un repère fixe cartésien (x,y,z).

D'un autre côté pendant le confinement, on nous a confié un nouveau projet. Le but est de simuler un drone sur Godot, avec un moteur physique (i.e. avec les équations dynamiques régissant le drone), et de le réguler avec Matlab. Les deux programmes communiqueront entre eux par UDP.

1. Crazyflie

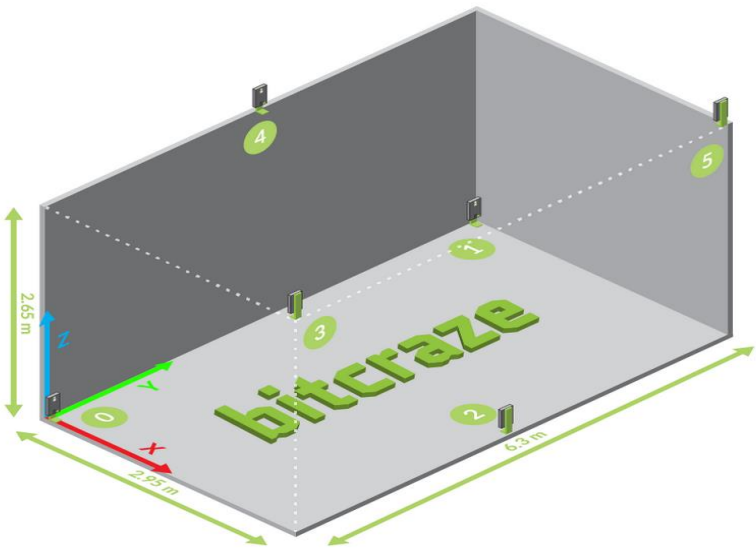
1.1. Réseau de capteurs

Comme expliqué durant le semestre 7, nous n'avons durant ce semestre pas travaillé avec la position relative du drone (le drone se repère par rapport à sa position précédente), mais avec sa position absolue (par rapport à un repère fixe dans l'espace). Afin d'avoir la position relative du drone, nous avons utilisé le Loco Positioning System. Ce système est composé de deux types de capteurs :

-Le loco positioning Deck est le capteur directement fixé sur le drone.

-Le loco positioning Node, ou "ancrage", au nombre de 6, sont répartis dans la pièce.

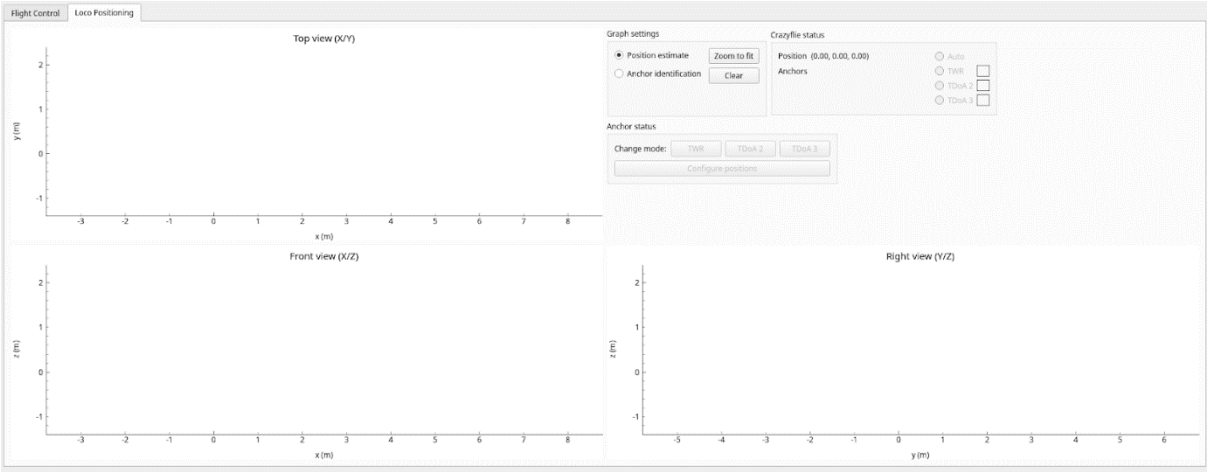
Nous avons réparti les ancrages dans la pièce selon la configuration suivante :



carte de l'emplacement des 6 capteurs

Chaque ancrage possède un numéro précis. Nous attribuons un numéro entre 1 et 6 à chaque node en le flashant à l'aide du logiciel LPS node firmware. Chaque capteur est alimenté par une batterie externe. L'ensemble capteur et batterie externe est placé dans une boîte en bois conçue à la découpe laser et placée dans la pièce.

Les coordonnées des capteurs dans la pièce sont visualisées sur le logiciel Crazyflie Client, qui nous permet d'observer la pièce à travers un repère fixe X,Y,Z de ce type :

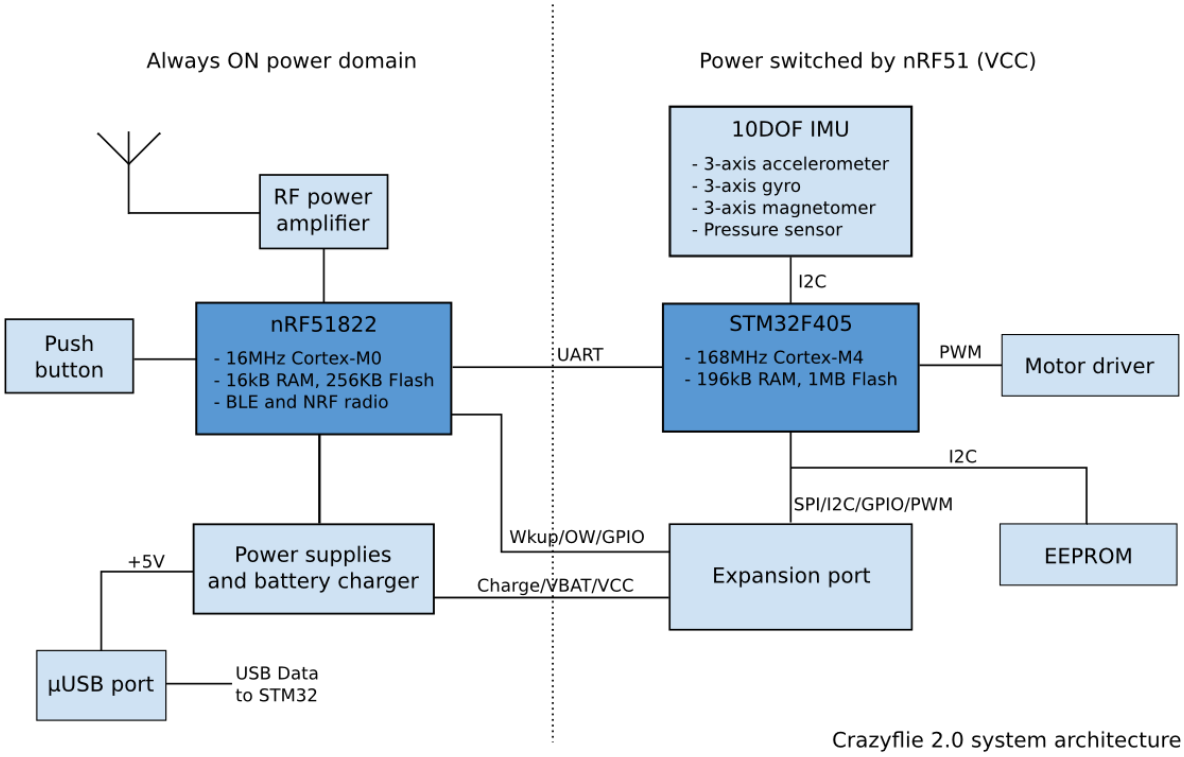


capture d'écran du logiciel Crazyflie Client dans lequel on reçoit la position du Crazyflie au cours du temps

Nous avions prévu de comparer les coordonnées des capteurs obtenues sur ce logiciel à celles réelles dans la pièce (origine du repère identique à celui utilisé par le Crazyflie Client) afin de déterminer la précision des capteurs, mais dû au COVID 19 (fermeture de l'école) et l'impossibilité d'avoir accès à notre matériel, nous n'avons pas pu réaliser ces mesures.

1.2. Architecture logicielle et matérielle

Avant de commencer à explorer le firmware du drone pour essayer d'implémenter une tâche autonome, il nous semble important de faire rapidement le point sur l'architecture matérielle et logicielle du drone.



Architecture matérielle du Crazyflie 2.0

Le Crazyflie est constitué des composants suivants :

- STM32F405: Cortex-M4, 168 MHz, 192 kB SRAM, 1 MB flash
- nRF51822: Cortex-M0, 32 MHz, 16 kB SRAM, 128 kB flash
- MPU-9250: centrale à inertie 9 axes (IMU)
- LPS25H: capteur de pression
- 8 kB EEPROM.
- port micro USB
- ports d'expansion : I2C, UART, SPI, GPIO

Et plus précisément :

Le nRF51 contient un Cortex-M0, 16kB de RAM et 256kB de Flash, qui lui permettent de gérer la communication radio et la gestion d'alimentation du drone. Cependant il n'est pas assez puissant pour faire tourner les algorithmes de contrôle de vol du drone.

Le STM32F405 est composé d'un Cortex-M4, de 196kB de RAM et de 1MB de Flash. Le cortex M4 lui permet de faire des calculs puissants, d'exécuter des algorithmes pour contrôler son vol.

Le nRF51 permet de communiquer avec le drone par Bluetooth et par CRTP (Crazy RealTime Protocol). Le Bluetooth sert à contrôler le drone via l'application smartphone développée par le constructeur. Le CRTP, Crazy RealTime Protocol, est un protocole de communication qui permet de communiquer entre une antenne, la CrazyRadio PA, et le drone, en 2,4GHz. La CrazyRadio PA est une antenne que l'on branche en USB sur notre PC.

Enfin, terminons par le rôle des deux microcontrôleurs. Le nRF51 et le STM32 communiquent entre eux par UART. Le nRF51 agit en tant qu'esclave et le STM32 en tant que maître.

Le nRF51 gère :

- le bouton (physique) ON/OFF
- l'alimentation du système (STM32, capteurs et cartes d'extensions)
- le chargement de la batterie
- communication radio et Bluetooth (protocoles Bluetooth et CRTP)
- détecter, installer les cartes d'extension

Le STM32 gère toutes les autres tâches du drone, par exemple :

- contrôle et régulation des moteurs
- contrôleur de vol
- mesures (lecture des valeurs des capteurs)
- Les tâches implémentées dans le firmware

Enfin, au niveau logiciel, le firmware du drone est open source, il est accessible sur le git du constructeur. Le firmware est constitué d'un ensemble de tâches utilisant FreeRTOS. Toutes les tâches du drone sont temps-réels et implémentées sous formes de tâches FreeRTOS.

1.3. Premier pas en vol autonome

1.3.1. Les limites du système

Pour visionner la valeur des capteurs, nous sommes obligés d'utiliser le logiciel Crazyflie Client. Nous ne pouvons pas avoir de retour sur les capteurs une fois que le drone est en vol. En effet, pour que le drone décolle, il ne faut qu'aucun autre programme ne communique avec lui. Comme le code est interprété sur notre ordinateur puis envoyé sur le drone, il ne faut pas que le logiciel communique avec le drone au même moment, car seul un programme à la fois peut envoyer des paquets au drone. Donc si le logiciel envoie des paquets, on ne peut pas exécuter notre code python et vice versa.

Le problème est que l'on n'arrive pas à implémenter un code python afin d'avoir les coordonnées en x,y,z pendant le vol, on est dépendant du logiciel pour cela. Le drone envoie seulement des données brutes qui sont traitées par filtre de Kalman. Nous n'avons pas réussi à ce jour à tirer des informations exploitables sur ces données bruts. Une piste pourrait être de se concentrer davantage sur ces données afin d'en tirer quelque chose.

D'un autre côté on ne peut pas exécuter de code depuis le logiciel Crazyflie Client, impossible donc avec ce logiciel d'évaluer les valeurs de retour de ces capteurs tout en exécutant du code.

1.3.2. Problèmes rencontrés

Problème au sol

Au repos, lorsque l'on étudiait les valeurs de retour des capteurs, ils indiquaient une valeur négative en Z ou très élevé (supérieur à 3 mètres). Nous avons étudié la position des capteurs et nous nous sommes rendu compte que les capteurs posés au sol étaient situés légèrement au-dessus du drone. Pour résoudre ce problème nous avons décidé de surélever le drone en le posant sur le carton afin qu'il soit situé au-dessus des capteurs lorsqu'il est posé sur le sol.

Problèmes en vol

Afin de tester nos capteurs en vols, nous avons essayé d'exécuter un code Python.

En cours de vols, notre drone est incapable de suivre les coordonnés qui lui sont imposées. Dès qu'il décolle il se crash. Nous avons dû l'attacher afin de réduire les dégâts. D'après les limites que nous avons décrites plus haut, il nous est impossible de savoir les coordonnées que le drone a lues avant de se crasher. La seule chose que l'on peut savoir est la valeur des coordonnées que l'on a donnée comme consigne à notre drone.

Afin de résoudre ce problème, nous avons essayé de diminuer la précision qui était de 1mm initialement à 1 cm. Ce que nous appelons précision est l'écart relatif entre la position que l'on veut

atteindre et la position où l'on est. En réduisant la précision, nous avons espéré avoir un mouvement plus souple. Malheureusement, nous n'avons pas obtenu de résultats fructueux avec cette méthode.

Une seconde méthode, que nous n'avons pas pu tester, consiste à changer les algorithmes de calculs de position.

1.3.3. Fonctionnement d'une tâche de vol autonome sous FreeRTOS

La programmation de tâche autonome n'est pas permise par le constructeur. Nous avons accès au firmware, mais il sert surtout à développer des drivers pour intégrer de nouvelle carte d'extension pour le drone, comme par exemple : https://www.bitcraze.io/products/flow-deck-v2/

Cependant, comme le drone est composé de mémoire flash, avec un firmware open source et une architecture matérielle connue, il est envisageable de programmer des tâches FreeRTOS. Comme le constructeur ne le recommande pas, il n'a pas documenté son wiki sur la façon d'implémenter ses propres tâches temps-réelles. Nous nous sommes documentés sur une thèse de master écrite par Mr Luke Beumont Barrett.

Il est possible de programmer des tâches temps-réelles en écrivant notre code FreeRTOS dans le répertoire Crazyflie-firmware/src/modules/src du firmware du Crazyflie. Ensuite il faudra déclarer notre tâche dans le config.h qui se trouve dans crazyflie-firmware/src/config.

https://github.com/bitcraze/crazyflie-firmware/tree/master/src/modules/src
https://github.com/bitcraze/crazyflie-firmware/blob/master/src/config/config.h

Nous avons seulement eu le temps de lire le mémoire et d'implémenter l'un des exemples de code fourni dans ce mémoire. Nous n'avons pas pu aller plus loin en raison de la fermeture des écoles dû au COVID19.

2. Projet Godot

2.1. Communication UDP

Nous avons mis en place au cours de ce projet une communication UDP afin de faire communiquer Matlab et Godot. Au cours d'une simulation sur Godot, le drone envoie en continue des données sur le drone (positions, valeurs des moments des perturbations) à un serveur Matlab en UDP. Ces données sont utilisées par Matlab pour réaliser différentes tâches (simulations des moteurs, régulation). Les résultats issus de ces tâches sont envoyés par un client Matlab à Godot. La communication UDP est donc bouclée et s'effectue en continue. Les données entre Client et Serveur UDP s'envoient sous forme de chaîne d'ASCII. Le serveur récupère ces données en ASCII et les convertit en type de données souhaitées (int, float, chaine de caractère etc. …)

En fonction de la tâche réalisée sur Matlab, les données envoyées par le client Godot sont différentes. Pour la simulation des moteurs sur Simulink, la trame UDP envoyée par le client Godot est la suivante:

- le mot "Start" qui indique le début d'une trame.
- Vitesse angulaire phi
- Vitesse angulaire psi
- Vitesse angulaire thêta
- Vitesse de translation sur z

Le serveur Matlab lui envoie :

- le mot "Start" qui indique le début d'une trame.
- Vitesse du moteur 1
- Vitesse du moteur 2
- Vitesse du moteur 3
- Vitesse du moteur 4
- le mot "quit" qui indique la fin d'une trame.

Pour la régulation sur Matlab, le client Godot envoie les valeurs des moments du vent, tandis que le serveur Godot lui envoie la même trame que pour la simulation des moteurs, à savoir la vitesse des quatre moteurs.

2.2. Godot

2.2.1. Mise en place de l'environnement graphique

Avant d'implémenter les équations, nous avons mis en place un environnement 3D qui nous permettra de modéliser le déplacement du drone au cours du temps.

Nous avons utilisé plusieurs scènes pour pouvoir simuler le drone.

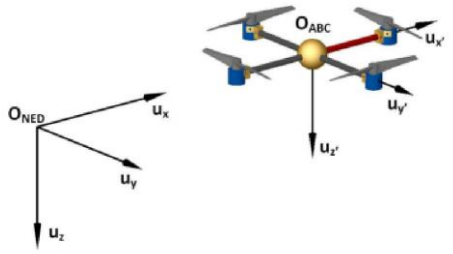
Tout d'abord, une scène de type KinematicBody (pour les déplacements 3D), à l'intérieur : un drone importé depuis internet (format .obj), et un contour (CollisionShape) qui nous permettra d'interagir avec d'autres solides.

Ensuite, une scène de type StaticBody, à l'intérieur : un pavé de couleur verte qui simule le sol.

Enfin, une scène principale de type Spatial (environnement 3D), à l'intérieur, on instancie nos scènes "le_drone" et "le_sol" en tant qu'enfant de notre scène principale. On définit aussi 3 caméras : une caméra principale, et 2 caméras secondaires qui seront affichées en bas à droite lors de la simulation.

2.2.2. Equations du modèle dynamique

Pour modéliser le drone, nous travaillerons dans 2 repères, un fixe O_{ned} (earth frame), et un mobile O_{abc} (body frame)



Dans ces deux repères, on définit 2 vecteurs :

$\begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T$ qui contient la **position** linéaire et angulaire du drone dans le **repère fixe**.

$\begin{bmatrix} u & v & w & p & q & r \end{bmatrix}^T$ qui contient la **vitesse** linéaire et angulaire du drone dans le **repère mobile.**

En entrée, nous prenons les vitesses des 4 moteurs qui nous permettront de calculer les forces et moments générés par ces vitesses.

avec

f_t : force de poussée verticale
$\tau_{x,y,z}$: les moments selon x,y,z
l : distance moteur – centre du drone
d : facteur de trainée (drag coefficient)
b : facteur de poussée (thrust coefficient)
Ω_n : la vitesse du moteur

$$\begin{cases} f_t = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ \tau_x = bl(\Omega_3^2 - \Omega_1^2) \\ \tau_y = bl(\Omega_4^2 - \Omega_2^2) \\ \tau_z = d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{cases} \tag{2.16}$$

Pour les valeurs de d, l, b de l'équation (2.16) nous prendrons celles du Crazyflie obtenues pages 12 et 13 de ce papier : https://arxiv.org/pdf/1608.05786.pdf

De même pour les éléments de la matrice d'inertie, diagonale, car on considère que notre drone a deux plans de symétrie :

$$\mathbf{I} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \in \mathbb{R}^{3\times 3}.$$

Enfin après avoir défini nos repères, nos constantes et nos entrées, on modélise le modèle linéaire du drone :

$$\begin{cases} \dot{\phi} = p \\ \dot{\theta} = q \\ \dot{\psi} = r \\ \dot{p} = \frac{\tau_x + \tau_{wx}}{I_x} \\ \dot{q} = \frac{\tau_y + \tau_{wy}}{I_y} \\ \dot{r} = \frac{\tau_z + \tau_{wz}}{I_z} \\ \dot{u} = -g\theta + \frac{f_{wx}}{m} \\ \dot{v} = g\phi + \frac{f_{wy}}{m} \\ \dot{w} = \frac{f_{wz} - f_t}{m} \\ \dot{x} = u \\ \dot{y} = v \\ \dot{z} = w \end{cases} \tag{2.36}$$

Avec les perturbations du vent définies par :

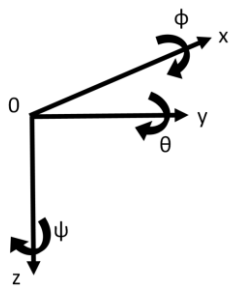
$$\begin{bmatrix} f_{wx} & f_{wy} & f_{wz} & \tau_{wx} & \tau_{wy} & \tau_{wz} \end{bmatrix}^T \in \mathbb{R}^6$$

2.2.3. Résultats

Après avoir implémenté l'équation 2.36 en se basant sur 2.16, 2.33, on procède à quelques tests afin de vérifier si le modèle est correct et nous permet bien de simuler les 6 degrés de liberté du drone. Pour ce faire, nous nous baserons sur le travail que nous avons effectué au semestre 6. En effet nous avions défini les 6 degrés de liberté du drone en fonction de la vitesse de ses 4 moteurs.

Ici, nous avons relevé les vitesses angulaires et linéaires dans le repère fixe en fonction des vitesses des 4 moteurs (= V_{mn} dans notre code, = Ω_n dans les équations) afin d 'observer si oui ou non on obtient nos 6 mouvements.

Pour rappel notre repère fixe est le suivant :



Pour les vitesses maximales du moteur, nous avons pris celles du Crazyflie, qui peuvent aller jusqu'à 6160 rad/s. Les moteurs 1 et 3 tournent dans le sens horaire et sont situés de part et d'autre du quadricoptère et les moteurs 2 et 4 dans le sens antihoraire (cf. travail du semestre 6). Toutes les mesures seront faites avec un vent nul.

Vitesse maximale en translation verticale – axe Z						
V_{mn} (rad/s)	$\dot{x}$ (m/s)	$\dot{y}$ (m/s)	$\dot{z}$ (m/s)	$\dot{\phi}$ (rad/s)	$\dot{\theta}$ (rad/s)	$\dot{\psi}$ (rad/s)
$\Omega_1 = 6160$ $\Omega_2 = 6160$ $\Omega_3 = 6160$ $\Omega_4 = 6160$	0	0	-1.47	0	0	0

Dans un premier temps nous calculons la vitesse maximale théorique de translation horizontale selon l'axe Z négatif. Z est négatif car le drone se déplace vers le haut, alors que l'axe Z est orienté vers le bas.

Ainsi, selon notre modèle, le Crazyflie peut aller au maximum à 1,47 m/s vers le haut lorsque les moteurs tournent à leur maximum (ce qui n'arrive pas car la batterie LiPo 240mAh ne tiendrait pas, il faudrait 4.2V et 1A sur chaque moteur pour atteindre la vitesse max théorique).

Maintenant, tentons de trouver la vitesse minimale qu'il faut donner aux moteurs pour que le drone commence à décoller. Nous considèrerons que le drone décolle lorsqu'il atteint une vitesse de -1cm/s selon Z. Notre modèle ne fonctionne qu'avec des petits angles en ϕ et en θ (simplification cos et sin) et il néglige l'aspiration du drone par le sol lorsque celui est très proche du drone. En résolvant :

$\dot{z} = (f_{wz}-f_t)/m = -0.01$, on trouve $\Omega_{1=}$ $\Omega_{2=}$ $\Omega_{3=}$ $\Omega_{4=}$507rad/s

avec $f_{wz} = 0$ (vent négligé dans cette partie)

Ainsi, notre drone décolle lorsque ses 4 moteurs tournent à 507rad/s.

Les translations selon les axes X et Y sont en fait une combinaison de mouvement (tangage/pitch) et de variations des vitesses des 4 moteurs. Nous ne nous intéresserons pas ici à ces mouvements car nous étudions seulement la rotation du drone selon les axes X,Y,Z et non ses déplacements linéaires selon ces axes.

Etudions maintenant les rotations du drone selon la vitesse des moteurs.

Rotation – axe Z						
V_{mn} (rad/s)	$\dot{x}$ (m/s)	$\dot{y}$ (m/s)	$\dot{z}$ (m/s)	$\dot{\phi}$ (rad/s)	$\dot{\theta}$ (rad/s)	$\dot{\psi}$ (rad/s)
$\Omega_1 = 3000$ $\Omega_2 = 2900$ $\Omega_3 = 3000$ $\Omega_4 = 2900$	0	0	-0.34	0	0	-0.43

En faisant tourner les moteurs 2 et 4 moins rapidement que les 1 et 3, on engendre une rotation d'axe Z de -0.43 rad/s, soit environ 25°/s. De plus les 4 moteurs tournant à ~3000rad/s, ils entrainent un déplacement selon Z de 34cm/s.

Rotation – axe X						
V_{mn} (rad/s)	$\dot{x}$ (m/s)	$\dot{y}$ (m/s)	$\dot{z}$ (m/s)	$\dot{\phi}$ (rad/s)	$\dot{\theta}$ (rad/s)	$\dot{\psi}$ (rad/s)
$\Omega_1 = 2980$ $\Omega_2 = 2980$ $\Omega_3 = 3000$ $\Omega_4 = 3000$	0	0	-0.35	2.73	0.1	0

En faisant tourner légèrement moins rapidement les moteurs 1 et 2, on engendre une forte rotation d'axe X de 2.73 rad/s, soit environ 156°/s. Cette forte vitesse angulaire s'explique par les équations définissant le moment en X des moteurs du drone (cf. équation 2.16). De plus les 4 moteurs tournant à ~3000rad/s, ils entrainent un déplacement selon Z de 35cm/s.

Rotation – axe Y						
V_{mn} (rad/s)	$\dot{x}$ (m/s)	$\dot{y}$ (m/s)	$\dot{z}$ (m/s)	$\dot{\phi}$ (rad/s)	$\dot{\theta}$ (rad/s)	$\dot{\psi}$ (rad/s)
$\Omega_1 = 3000$ $\Omega_2 = 2900$ $\Omega_3 = 3000$ $\Omega_4 = 3100$	0	0	-0.35	0	1.07	0

En faisant tourner les moteurs 2 et 4 moins rapidement que les 1 et 3, on engendre une rotation d'axe Y de 1.07 rad/s, soit environ 61°/s. De plus les 4 moteurs tournant à ~3000rad/s, ils entrainent un déplacement selon Z de 35cm/s.

En nous basons sur notre étude du semestre 6, on peut dire que notre simulation Godot est en adéquation avec notre étude théorique qui définissait les mouvements du drone en fonction de la variation de vitesse sur ses quatre moteurs.

2.3. Matlab – Simulink

2.3.1. Simulation des moteurs avec Matlab & Simulink

Le but est de pouvoir simuler les moteurs sur le logiciel Matlab/Simulink pour ainsi faire de la régulation et communiquer les vitesses de moteur à Godot.

Equations des moteurs

Dans cette partie, nous allons créer les moteurs du drone dans un modèle schématique avec Simulink. Pour ce faire, nous allons avoir besoin des équations en 2.16 et 2.36 qui nous permettront d'exprimer la vitesse des moteurs en fonction des vitesses angulaire et la vitesse de translation sur z du drone (ces données seront transmisse par Godot par la suite).

$$\Omega_1 = \frac{\sqrt{\frac{f_t}{b}} - \sqrt{\frac{T_z}{d}}}{2} - \Omega_3$$

$$\Omega_2 = \frac{\sqrt{\frac{T_z}{b}} + \sqrt{\frac{f_t}{d}}}{2} - \Omega_4$$

$$\Omega_3 = \frac{\sqrt{\frac{T_x}{bl}} + \frac{\sqrt{\frac{f_t}{b}} - \sqrt{\frac{T_z}{d}}}{2}}{2}$$

$$\Omega_4 = \frac{\sqrt{\frac{T_y}{bl}} + \frac{\sqrt{\frac{T_z}{b}} + \sqrt{\frac{f_t}{d}}}{2}}{2}$$

$$T_x = \dot{p} \times I_x - T_{wx}$$

$$T_y = \dot{q} \times I_y - T_{wy}$$

$$T_z = \dot{r} \times I_z - T_{wz}$$

$$f_t = -(\dot{w} \times m) + f_{wz}$$

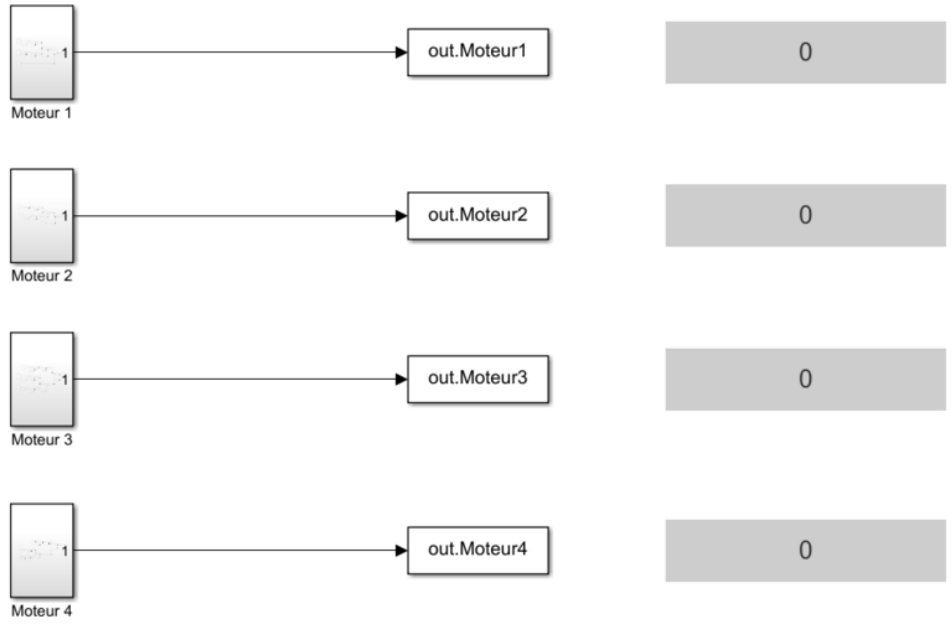
Avec

- $\dot{p}$, $\dot{q}$ et $\dot{r}$ les vitesses angulaires du drone
- $\dot{w}$ la vitesse de translation sur z

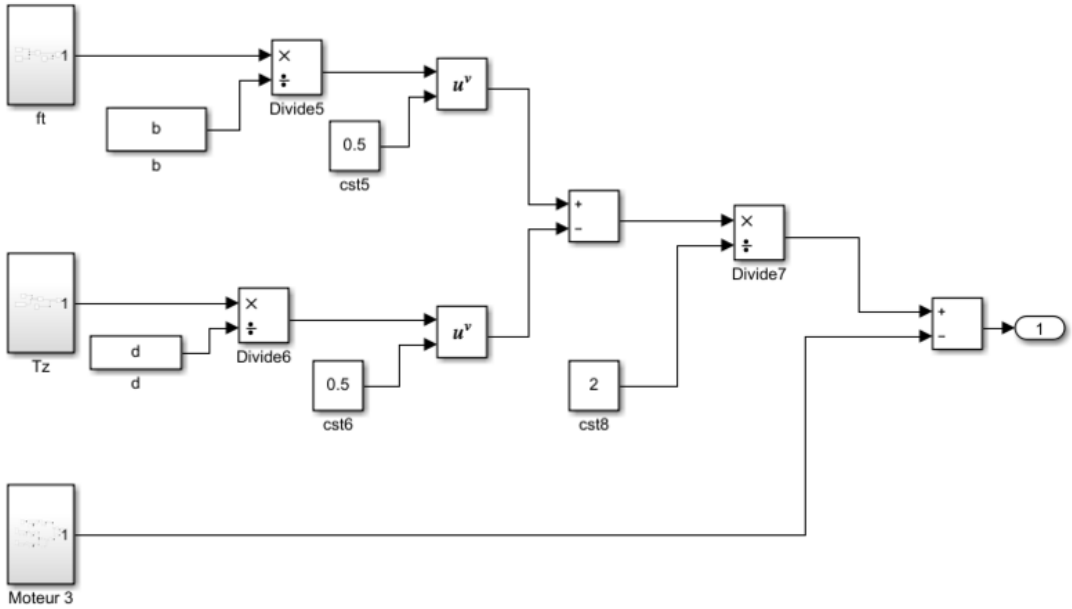
Ces équations permettront d'établir un schéma sur Simulink pour pouvoir simuler les moteurs.

Schéma Simulink

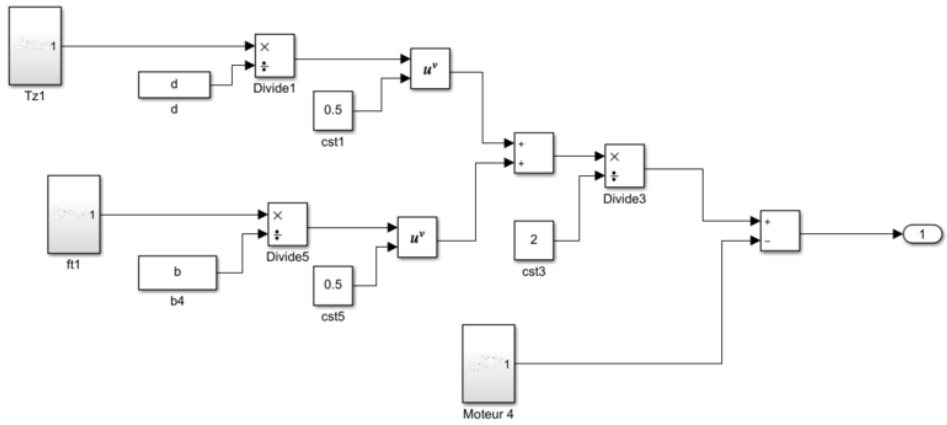
Voici le modèle global Simulink :



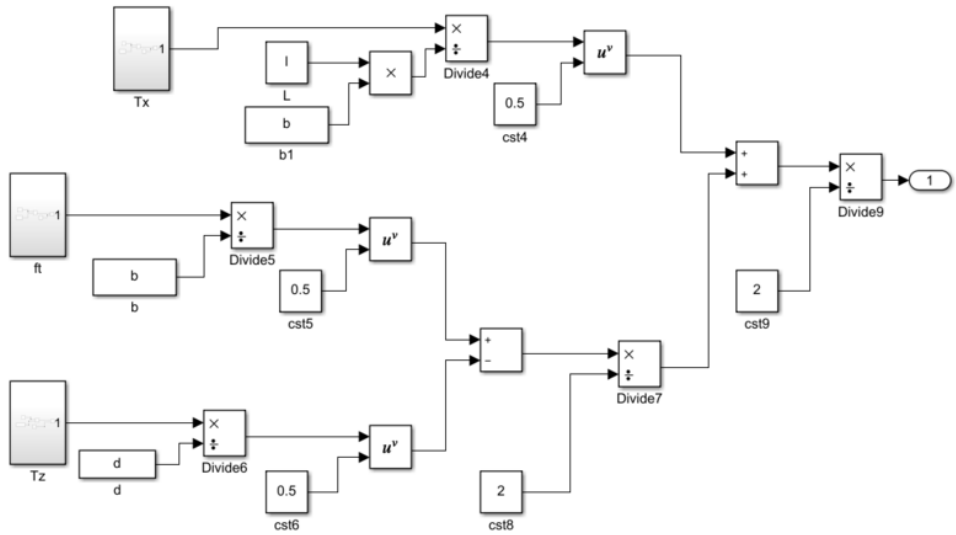
Moteur 1 :



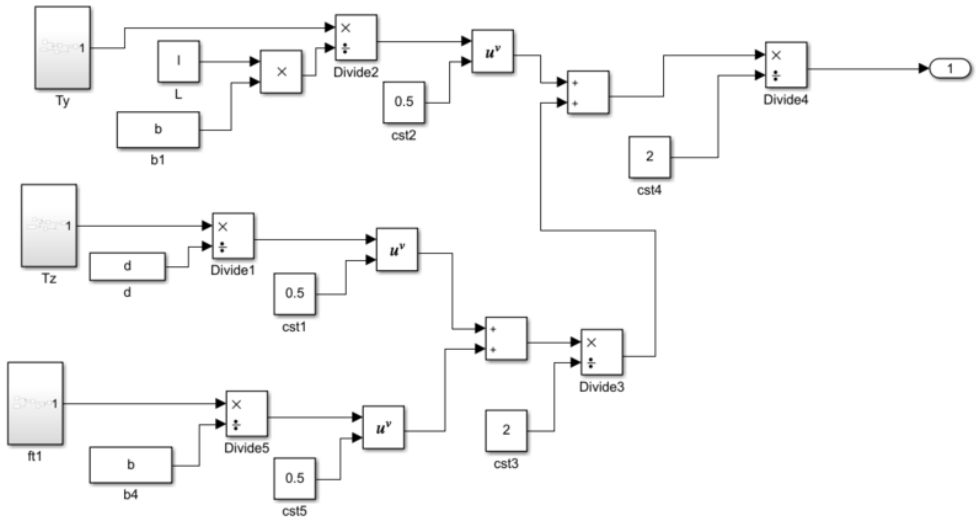
Moteur 2 :



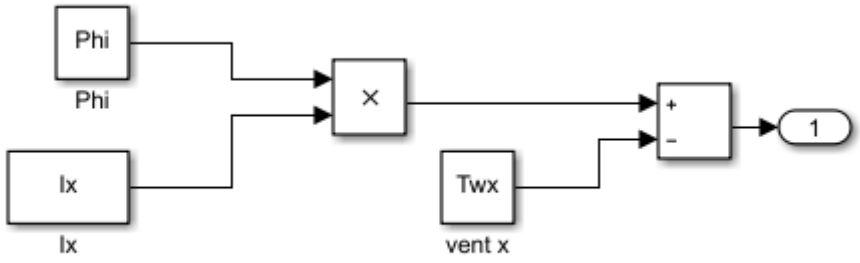
Moteur 3 :



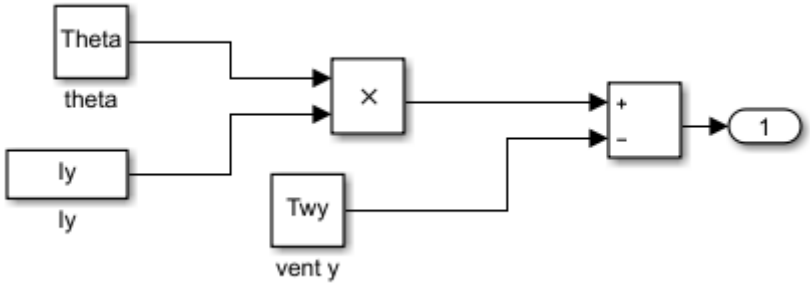
Moteur 4 :



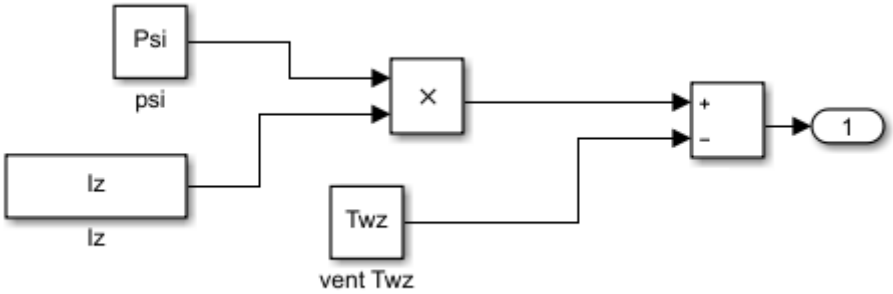
Moment en X généré par la vitesse des 4 moteurs, Tx :



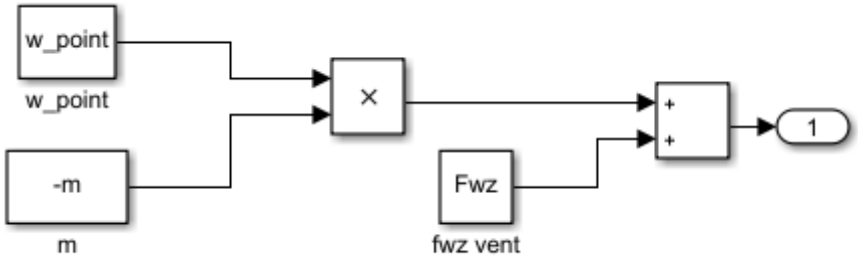
Moment en Y généré par la vitesse des 4 moteurs, Ty :



Moment en Z généré par la vitesse des 4 moteurs, Tz :



Force de poussée verticale, Ft :



Exploitation des résultats

Maintenant, nous allons faire plusieurs tests pour examiner et comparer les vitesses des moteurs avec celles simulées dans Godot pour voir si notre modèle est fonctionnel.

La première expérimentation qu'on va effectuer commencera avec une vitesse maximale de translation sur z de 1,47 m/s.

Vitesse maximale en translation verticale sur -axe Z				
Vmoteur (rad/s)	$\dot{\theta}$ (rad/s)	$\dot{\phi}$ (rad/s)	$\dot{\psi}$ (rad/s)	$\dot{w}$ (m/s)
Ω_1 = 6156	0	0	0	-1,47
Ω_2 = 6156				
Ω_3 = 6156				
Ω_4 = 6156				

On remarque qu'on est très proche du résultat sur Godot qui était théoriquement de 6160 rad/s pour chaque moteur avec les mêmes configurations.

Pour la deuxième expérimentation, nous allons faire une rotation sur l'axe Y de 1,07 rad/s (61 °/s).

Rotation -axe Y				
Vmoteur (rad/s)	$\dot{\theta}$ (rad/s)	$\dot{\phi}$ (rad/s)	$\dot{\psi}$ (rad/s)	$\dot{w}$ (m/s)
Ω_1 = 2960	0	1,07	0	-0,34
Ω_2 = 1867				
Ω_3 = 2417				
Ω_4 = 4054				

On remarque que les moteurs ont la même configuration que sur Godot lors du même essai mais les vitesses de moteurs sont différentes, on suppose que cela est dû au fait que les moteurs ne soient pas régulés avec un PID et donc sont amenés à donner des valeurs de vitesses avec une erreur plus ou moins grande.

2.3.2. Régulation avec Matlab

Nous avons ensuite réalisé un script Matlab pour effectuer la régulation du drone. Pour cela, nous partons des équations 2.36 et 2.16. Nous sommes partis du principe suivant : pour réguler le drone, il faudrait dans un premier temps annuler le moment du vent avec une bonne valeur des vitesses des moteurs. Pour cela nous résolvons pour chaque axe :

moment_moteur = - moment_vent, en fonction de Vm1,Vm2,Vm3,Vm4 (vitesses des moteurs)-

Pour rappel voici les équations des moments en x,y,z générés par la vitesse de rotation des moteurs :

$$\tau_x = bl(\Omega_3^2 - \Omega_1^2)$$
$$\tau_y = bl(\Omega_4^2 - \Omega_2^2)$$
$$\tau_z = d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2)$$

Et, ici, la relation entre le moment des moteurs (Tx, Ty, Tz), le moment du vent (Twx, Twy, Twz), et la vitesse de rotation selon x,y,z ($\dot{\theta}$, $\dot{\psi}$, $\dot{\phi}$) :

$$\dot{\phi} = p$$
$$\dot{\theta} = q$$
$$\dot{\psi} = r$$
$$\dot{p} = \frac{\tau_x + \tau_{wx}}{I_x}$$
$$\dot{q} = \frac{\tau_y + \tau_{wy}}{I_y}$$
$$\dot{r} = \frac{\tau_z + \tau_{wz}}{I_z}$$

Pour simplifier les calculs dans un premier temps, on prend en compte seulement les moments générés par le vent. On part du principe que le drone est immobile, et qu'il est soumis à un vent constant, qui le fait tourner selon un ou plusieurs axes. L'idée est de trouver une valeur de Vm1, Vm2, Vm3, Vm4 pour annuler le moment généré par le vent.

On obtient le système à 3 équations suivant :

$$\text{solve}\left(\left\{\begin{array}{l}(v3^2 - v1^2)\cdot 1.32\cdot 10^{-5} = vx \\ (v4^2 - v2^2)\cdot 0.56\cdot 1.32\cdot 10^{-5} = vy \\ (v2^2 + v4^2 - v1^2 - v3^2)\cdot 1.385\cdot 10^{-6} = vz\end{array}\right\}, \{v1, v2, v3, v4\}\right)$$

$$v1 = 0.003953\cdot\sqrt{-2.42375e9\cdot\left(vx + 1.78571\cdot\left(vy + 5.33718\cdot\left(vz - 0.000003\cdot c5^2\right)\right)\right)} \text{ and } v2 = 0.065795$$

vx, vy, vz correspondent aux moments du vent (avec le signe moins, vx=-T_wx). Nous avons résolu ce système à l'aide d'un calculateur formel XCAS qui nous a donné 8 solutions, valables dans un certain domaine de validité. Chaque solution nous fournit 4 valeurs de vitesses qui compenseront les moments du vent. Nous ne conservons qu'une solution dont le domaine de validité correspond avec les valeurs des vitesses max et min des moteurs du drone.

Nous implémentons la solution suivante dans Matlab :

```
v1=0.003953*(−2.42375*10^9*(vx+1.78571*(vy+5.33718*(vz−0.000003*v4^2))))^(1/2);

v2=0.065795*(−3.125*10^7*(vy−0.000007*v4^2))^(1/2);

v3=194.625*(vx−1.78571*(vy+5.33718*(vz−0.000003*v4^2)))^(1/2);

v4=max(340.0000001,378.0000001*sqrt(vy));
```

Conditions de validité :

```
−2.9*10^5*(v4^2−62455*(vy+5.4*vz)) <= vx <= 2.9*10^5*(v4^2−62455*(vy+5.4*vz))

vz <= min(3*10^6*(v4^2+34975*(vx−1.78571*vy)), 3*10^6*(v4^2−34975*(vx+1.78571*vy)))

and v4−max(0.,378*sqrt(vy)) >= 0 and vy >= 0

vy < 0.000007*v4^2

v4 >= 378*sqrt(vy) avec v4max=925 on obtient vymax=5.9
```

Les trois premières vitesses des moteurs dépendent des perturbations mais également de la vitesse du 4ème moteur. Cette vitesse a été fixée par son domaine de validité, qui dépend uniquement de la perturbation du vent selon l'axe y. Ainsi, les valeurs des vitesses calculées permettent aux moteurs du drone de se réguler et de rester stable.

Conclusion

Au cours du semestre 8, nous avons mis en place un réseau de capteur pour localiser le Crazyflie. Nous nous sommes penchés sur son architecture matérielle et logicielle, afin d'explorer les possibilités qui s'offraient à nous pour un effectuer vol autonome. Nous avons ainsi essayé d'implémenter une tâche FreeRTOS. Malheureusement face aux nombreux problèmes que nous avons décrits dans ce rapport et surtout à cause du confinement nous n'avons pas pu finir ce projet.

Ensuite, nous sommes passés à un autre projet. Dans le peu de temps qui nous restait avant la restitution finale, nous avons réussi à implémenter le modèle dynamique d'un quadricoptère dans Godot. Nous avons établi une communication UDP en boucle avec Matlab, puis commencé à réguler le drone.

En conclusion, ce projet d'un an et demi nous a fait gagner en autonomie. Nous avons appris à nous documenter, à nous poser des questions, et à tenter d'y répondre. Même si nous n'avons pas pu tout finir, c'est sur une note positive que nous terminons ce projet qui nous a permis de développer de nombreuses compétences.

Sources

Crazyflie's firmware
https://github.com/bitcraze/crazyflie-firmware

Design of a Trajectory Tracking Controller for a Nanoquadcopter. Technical report, Mobile Robotics and Autonomous Systems Laboratory
Luis C., & Le Ny, J., Polytechnique Montreal (August, 2016)

System Identification of the Crazyflie 2.0 Nano Quadrocopter
Julian Fôrster, ETH Zürich (2015)

Quadrotor control: modeling, nonlinear control design, and simulation
Francesco Sabatino, KTH Electrical Engineering, Stockholm Sweden (June 2015)

UWB Radiolocation Technology: Applications in Relative Positioning Algorithms for Autonomous Aerial Vehicles
Luke Beumont Barret, Murdoch University, Perth Australia (2018)