Projet Njörd

Topographie d'une zone
par communication avec une équipe de drones

AIGREAUL Clément - HENRIO Jordan - PHAM Chitin

20 mars 2015

Introduction

Nous vivons dans un monde où la technologie a atteint un point permettant de “donner vie” à des objets. Si bien qu'ils peuvent prendre des décisions, apprendre et communiquer. Une telle avancée permet des milliers d'applications aussi bien pour le divertissement, la domotique, l'industrie, ou encore l'assistance.

Parfois l'Homme peut être amené à devoir exécuter certaines missions sur des terrains dont il ne possède pas une connaissance exacte (comme par exemple une ville ayant subi une catastrophe naturelle). Ainsi un des problèmes majeurs des agents de sécurité est de connaître exactement la situation afin de mettre en place une stratégie d'approche. Des pertes humaines ne sont pas un risque à prendre, alors un intermédiaire devient nécessaire.

Grâce à l'avancée de la robotique, de l'intelligence artificielle et de nos moyens de communication sans fil, la création d'une équipe de robots permettant l'analyse d'un lieu peut être d'une très grande utilité dans ce genre de situation. Des pertes matérielles ne sont pas aussi grave que des pertes humaines.

C'est pourquoi nous avons choisi, dans le cadre de notre projet de fin d'études, de créer une équipe de drones volants qui communiquent ensemble par l'implémentation d'un serveur central qui reçoit des informations de la part des drones et qui dessine la topographie de la zone analysée. Ce rapport a pour but de présenter le développement et les choix technologiques de ce projet. Il explique les calculs réalisés pour le choix des composants, les erreurs que nous avons faites et présente les résultats obtenus pendant les phases de tests.

Table des matières

Chapitre 1

Serveur

Le but de ce chapitre est de présenter le serveur mis en place. Dans un premier temps nous expliquons le modèle de réseau utilisé pour la communication entre les drones et le serveur, et le fonctionnement interne du serveur. Dans une deuxième partie nous abordons, la manière dont nous avons implémenté cela.

1.1 Principe de fonctionnement

1.1.1 Modèle du réseau

Pour réaliser notre application nous avons choisi de mettre en place un modèle de communication basé sur un réseau étoilé. Le serveur représente le noyau du modèle et les branches sont représentées par les drones (voir Figure 1.1). Ainsi l'objectif du serveur est de dessiner la topographie de la zone étudiée à l'aide des données récoltées par l'équipe de drones. Bien que les drones se doivent d'être indépendants, il est important de toujours en garder le contrôle. Ainsi, le serveur a aussi la possibilité d'envoyer des ordres aux drones suivant certaines situations.

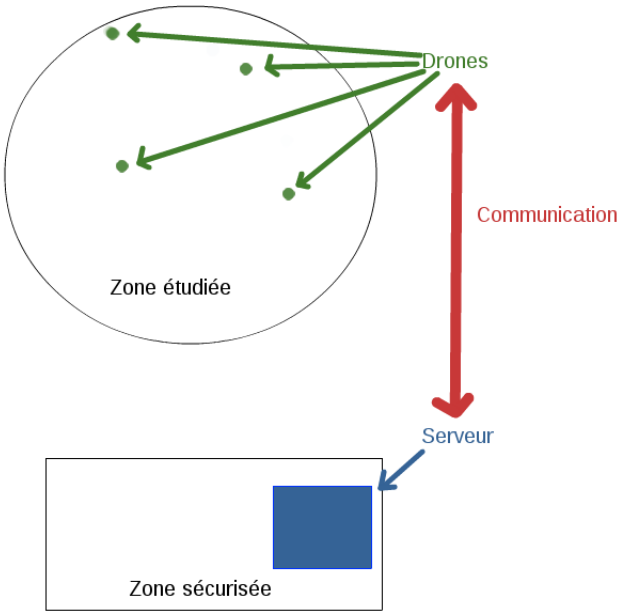

FIGURE 1.1 – Schéma du réseau

Nous avons choisi ce modèle car il répond à nos besoins et qu'il est plus simple à mettre en place qu'un réseau maillé par exemple. Chaque drone peut communiquer avec le serveur mais pas les autres drones. Et le serveur connait l'ensemble des drones utilisés dans l'application et peut donc communiquer avec eux. Aussi, si l'un des drones venait à tomber en panne durant une analyse, cela n'empêcherait pas le bon déroulement de l'application puisque le serveur pourrait continuer de communiquer avec le reste des drones, contrairement à une topologie en anneau par exemple (qui au passage augmenterait la latence de réception des données).

1.1.2 Fonctionnement interne du serveur

En ce qui concerne le fonctionnement interne du serveur nous avons pensé qu'il serait plus judicieux de le découper en plusieurs entités, où chacune d'elles serait associée à une tâche particulière. Ainsi, nous avons effectué le découpage suivant :

— Communication
— Sauvegarde/Chargement des données
— Topographie/Décision d'ordre

Le serveur est donc constitué de trois tâches qui tournent en concurrence à "l'infinie". Enfait, si nous avions fait une seule et unique tâche nous risquerions de perdre des données. En effet, le temps que le serveur traite les informations qu'il a reçu, les drones continuent de lui envoyer des données. Mais si le serveur n'écoute pas à ce moment là, alors ces données seraient perdues.

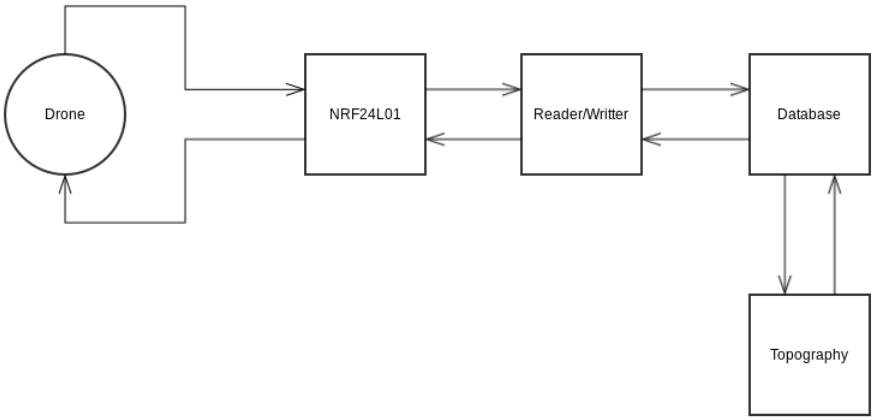

FIGURE 1.2 – Transitions des données

La première tâche, *Communication* (nommée *NRF24L01* sur la Figure 1.2, en référence au composant utilisé pour la communication), se charge de lire les messages envoyés par les différents drones et de les réécrire sur le port série du serveur. En effet, une partie du serveur est constitué d'un montage à base d'*Arduino* et c'est notre seul moyen de faire passer les données du microcontrôleur (voir section 2.1.1) vers le serveur. Cette même tâche se charge dans un deuxième temps de lire le port série, afin de vérifier s'il y a des ordres à envoyer aux drones. Si c'est le cas elle se charge de les communiquer aux drones concernés.

La deuxième tâche, *Sauvegarde/Chargement des données* (nommée *Reader/Writer* sur la Figure 1.2), lit le port série du serveur et enregistre chaque message dans une base de données. Dans un deuxième temps elle accède au contenu de la base de données, afin de vérifier si des ordres doivent être envoyés. Si c'est le cas elle les écrit sur le port série afin de les transmettre à la première tâche.

La troisième et dernière tâche, *Topographie/Décision d'ordre* (nommée *Topography* sur la Figure 1.2), récupère les entrées de la base de données (qui se comporte comme une pile) et les insère dans une matrice. Cette matrice représente la zone étudiée en vue de dessus, avec chacune de ses composantes représentant l'altitude aux cordonnées correspondantes aux indices de la composante. Ainsi, si $M_{1,1} = 150$ (avec M la matrice représentant la zone) cela signifie qu'au point $(1, 1)$ de la zone un drone a mesuré une altitude

de 150 centimètres. À chaque fois qu'une donnée est inséré dans la matrice, cette tâche se charge de dessiner le nouveau contenu à l'écran. Ce qui permet à l'utilisateur de suivre en direct l'évolution de l'analyse, et ce de manière graphique. Dans un deuxième temps cette tâche analyse si un ordre doit être envoyé. Cela dépend de plusieurs paramètres. Par exemple, si une zone de la carte n'a pas encore été dévoilée le serveur peut demander à un drone de s'y rendre pour y récolter des informations. Ou si un drone n'a plus beaucoup de batterie, le serveur peut lui demander de revenir dans la zone sécurisée afin que l'utilisateur le recharge. Si cette troisième tâche détermine qu'un ordre doit être envoyé, alors elle l'écrit dans la base de données afin de le faire remonter à la seconde tâche.

Maintenant que nous en savons un peu plus sur le principe de fonctionnement, nous pouvons présenter la manière dont nous avons implémenté cela.

1.2 Implémentation

La première tâche est constituée d'une partie matérielle et logicielle. La partie matérielle est un simple montage formé par une carte Arduino et un transmetteur/récepteur radio (voir Figure 1.3).

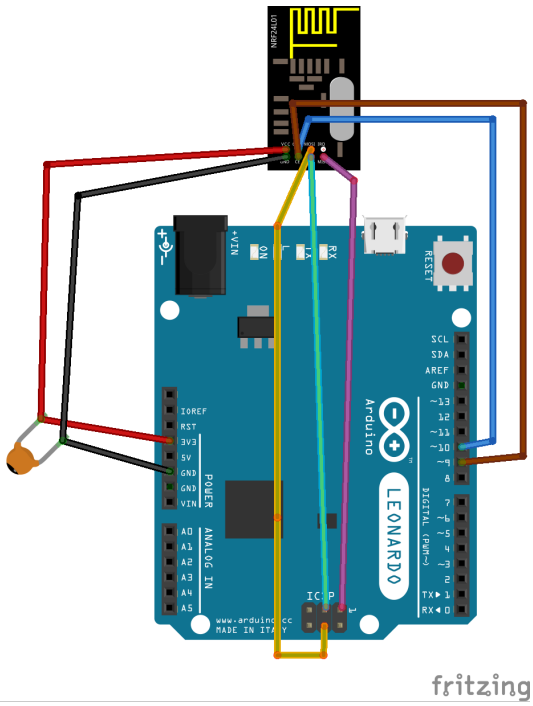

FIGURE 1.3 – Montage matériel du serveur

Puisque cette tâche utilise un montage à base d'Arduino, la partie logicielle de cette tâche est codée dans le langage d'Arduino. Ce programme se contente de parcourir l'ensemble des adresses des drones connus et de lire les messages reçus. Un message est représenté par un tableau de la forme indiquée par la Figure 1.2.

Avec :

— d : identifiant du drone
— x : position en abscisse
— y : position en ordonnée
— z : altitude du drone

d	x	y	z	$s1$	$s2$	$s3$	$s4$	$s5$	$s6$	e

FIGURE 1.4 – Représentation d'un message de drone

d	x	y	z	e

FIGURE 1.5 – Représentation d'un ordre

— $s1$ à $s6$: valeurs des capteurs
— e : code d'état

La tâche se charge alors de recopier le contenu des tableaux, sur le port série afin de les communiquer à la deuxième tâche. Une fois l'ensemble des messages recopiés, elle lit le port série afin de voir si des ordres doivent être envoyés. Si c'est le cas, elle forme un tableau (représenté par la Figure 1.2) et le transmet au drone concerné.

Avec :
— d : identifiant du drone destinataire
— x : position cible en abscisse
— y : position cible en ordonnée
— z : altitude cible du drone
— e : code supplémentaire

Que ce soit pour les messages ou pour les ordres, on peut remarquer qu'une valeur e est présente dans la dernière case du tableau. Cette valeur additionnelle permet de communiquer une information supplémentaire dans un message. Par exemple, on peut imaginer qu'un drone envoie $e = 5$ ce qui pourrait signifier "Je n'ai plus de batterie". Dans ce cas le serveur sait interpréter le cas où $e = 5$ et construirait un ordre en conséquence.

La deuxième tâche, quant à elle, est implémentée en *Python*[1]. Nous avons choisi ce langage car nous n'avons pas besoin d'un langage puissant comme le *C/C++* et il est moins gourmand que *Java*. De plus, il possède une communauté énorme. Ainsi, il existe des bibliothèques pour tout et n'importe quoi. Le langage étant très simple de base, avec toutes ces bibliothèques il devient sûrement le langage le plus simple pour déployer de petites applications. Par exemple, il existe une bibliothèque pour accéder au port série de la machine, avec des fonctions très simples d'utilisation.

Quand la deuxième tâche récupère un message elle doit l'insérer dans une base de données. Notre application ne nécessite pas la puissance du *SQL*. En effet, tout ce que nous voulons faire ces récupérer tout le contenu, entrée par entrée. Ainsi, nous n'avons pas besoin de faire des requêtes avec des entrées classée dans un certain ordre, ou alors des requêtes avec des conditions, etc. C'est pour cette raison que nous avons choisi d'utiliser une base de données *Redis*[2]. C'est une base de données basée sur des ensembles "clé-valeur". Ainsi, dans notre application la zone étudiée (la clé) sera associée à une pile de message (la valeur). Un autre point fort de notre choix d'implémentation est qu'il existe une bibliothèque pour la manipulation de Redis avec Python.

La troisième tâche est elle aussi implémentée en Python. Dans la section 1.1.2 nous avons expliqué que nous représentons la zone étudiée par une matrice. Nous partons du principe que nous ne connaissons pas la taille de la zone. Ainsi, notre matrice doit être constamment redimensionnée. Il existe encore une bibliothèque très utile en Python, nommée *Numpy*[3]. C'est une bibliothèque scientifique qui propose divers fonctions implémentant des algorithmes connus pour le calculs scientifique, mais dans notre cas ce qui nous intéresse c'est les structures de données qu'elle propose. En effet, les matrices Numpy sont très puissantes car elles disposent de divers méthodes pour leur manipulation et notamment des méthodes de redimensionnement dynamique. Aussi, cette tâche doit représenter graphiquement le contenu de la matrice. Numpy est très souvent couplé à une seconde bibliothèque nommée *MatPlotLib*[4]. Celle-ci permet de dessiner des graphiques à l'écran. Elle fonctionne très bien avec Numpy puisque ses méthodes s'attendent déjà à recevoir des objets venant de cette dernière.

Maintenant que nous avons présenté le fonctionnement du serveur nous allons pouvoir vous présenté un exemple de résultat.

1.3 Résultat

La Figure1.6 est une capture d'écran d'une topographie que nous avons obtenue par le biais du serveur.

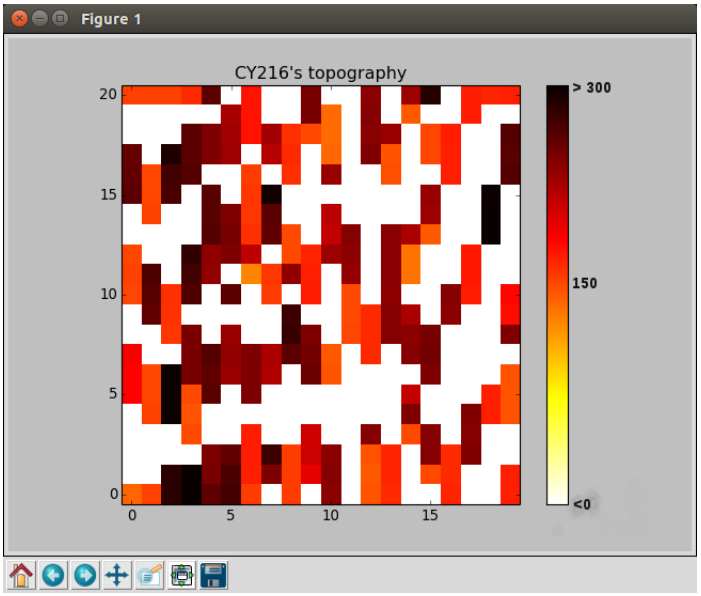

FIGURE 1.6 – Exemple de topographie

On peut y voir un certain nombre d'éléments. Le plus visible est le graphique lui-même. Il représente la zone étudiée en vue de dessus. Ainsi à chaque coordonnée de la zone est associée une certaine hauteur. Le degré de cette hauteur est représenté par une couleur plus ou moins foncé. Plus cette couleur est foncée plus le point est élevé. Sur la droite du graphique se trouve sa légende. Elle indique la valeur en centimètres du degré de hauteur. On remarque donc que pour des points blancs cela représente le sol, et que pour des points noirs cela représente une hauteur de trois mètres. Cela est dû au fait que les capteurs ultrasons que nous utilisons pour les mesures ont une distance de mesure maximum. Mais le serveur lui peut fonctionner avec n'importe quelle valeur.

Il faut noter que nous ne possédions pas de drone fonctionnel pour faire les essais du serveur. Ainsi l'exemple de topographie montré par la Figure1.6 ne représente rien de spécial. Nous avons fournit des coordonnées aléatoires pour chaque point et nous avons fait varier les mesures du capteur en plaçant notre main devant et en jouant avec la distance.

Dans tous les cas, lors de cet essais la réception et la représentation des messages se sont avérées fonctionnelles. Ainsi, si le serveur était utilisé avec un drone qui fournit sa véritable position et les valeurs renvoyées par ses capteurs, le serveur fournirait toujours les résultats attendus.

1.4 Envoi d'ordres

Toutefois, le serveur n'est pas encore totalement fonctionnel. Il reste l'envoi d'ordre à implémenter. La troisième tâche peut déterminer si une partie de la carte contient une zone encore non découverte. Ainsi, elle parcourt l'ensemble des drones et vérifie s'il existe un drone pour lequel un ordre n'a pas été donné (c'est-à-dire s'il existe un drone qui n'est pas en cours de déplacement vers une position cible). Si c'est le cas elle insère un ordre dans une deuxième pile Redis. La deuxième tâche lit le contenu de cette seconde pile. Si elle n'est pas vide elle réécrit l'ordre sur le port série du serveur. Cependant, nous n'arrivons pas encore à lire et écrire simultanément sur le port série, du côté Arduino. Donc nous ne sommes pas encore capable d'envoyer les ordres aux drones.

De plus notre serveur ne détermine le besoin d'envoyer un ordre seulement dans le cas d'une partie encore non dévoilée de la carte. Mais il devrait être capable de déterminer des ordres en fonction des différents états du drone. Par exemple dans le cas où le drone n'a plus de batterie, pour lui demander de revenir à la "base". Pour faire une liste exhaustive de tous les états nécessitant un ordre, il nous faudrait faire des essais à l'aide d'un drone fonctionnel, ce que nous n'avons pas réussi à obtenir. Mais nous aborderons ce problème dans la suite de ce rapport.

Chapitre 2

Premier drone

Étant donné notre cursus, nous ne savions pas réellement comment procéder pour monter un drone. Nous avons commencé par nous renseigner sur ce qui se fait en matière de drones. Il faut savoir qu'il en existe de plusieurs types, des petits, des grands, des appareils avec un vol "agressif" (rapide et agile), d'autres avec un vol optimisé pour la prise de vues, certains avec un vol dit "hybride", etc.

Pour l'application que nous voulons faire nous avons plutôt besoin d'un drone avec un vol hybride. Le but est de dessiner la topographie d'une zone, pour cela nous utilisons un capteur ultrason qui mesure la distance entre le drone et ce qu'il y a en dessous. Nous n'avons donc pas besoin d'un vol assez lent pour faire des prises de vues, mais il ne faut pas que le drone soit trop rapide afin de prendre le plus d'informations possible. De plus, pour des questions pratiques, il faut que le drone ait assez d'autonomie pour ne pas demander que sa batterie soit rechargée pendant une session d'analyse.

Parmi les drones déjà existants, le drone *Crazyflie* de chez *Bitcraze*[5] nous a intéressés par sa petite taille (voir Figure 2.1), par le fait qu'il soit libre et qu'on puisse facilement acheter tous les composants séparément. Nous avons donc décidé de baser notre modèle sur ce petit drone. Plus concrètement, le nôtre a une taille similaire au Crazyflie et dispose de la même batterie, des mêmes moteurs et des mêmes pales mais le reste des composants sont différents. Nous ne voulions pas passer directement par un Crazyflie, car un de nos objectifs est de construire le robot d'A à Z. De plus ce modèle n'est pas tout à fait adapté à ce que nous voulons faire, car il est trop léger et est alors trop véloce. Aussi il ne possède pas de capteur pour récolter des informations sur la zone survolée. Donc dans tous les cas nous aurions été obligés de le modifier.

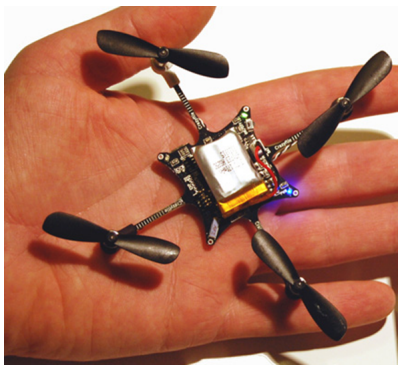

FIGURE 2.1 – Crazyflie

2.1 Composants

L'objectif de cette section est de présenter les différents composants que nous avons utilisés pour construire ce premier drone.

2.1.1 Microcontrôleur

Un microcontrôleur est un *system on chip*, c'est-à-dire que c'est une petite carte électronique proposant un certain nombre de fonctionnalités et pouvant fonctionner seule. C'est souvent un micro-processeur monté sur une carte avec plusieurs broches sur lesquelles l'utilisateur peut y connecter d'autres composants. Ainsi il peut réaliser un montage complexe qu'il pourra utiliser en programmant le microprocesseur. Au début de notre année scolaire, nous avons eu un séminaire pour nous apprendre le langage *Arduino*. Nous avons pu découvrir un langage vraiment simple à prendre en main lorsque l'on a déjà quelques bases en programmation avec des langages comme le *C*, ou le *C++*. Nous nous sommes donc tournés vers les technologies proposées par *Arduino* pour le choix du microcontrôleur. Finalement, nous avons opté pour une *Arduino Pro Mini*. Le principal intérêt de cette carte se trouve dans sa petite taille et sa légèreté, 18 sur 33 millimètres pour 2 grammes. De plus elle propose un nombre suffisant de broches pour l'ensemble de notre montage.

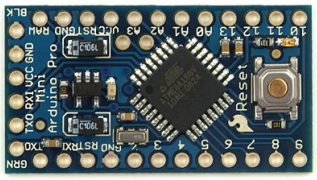

FIGURE 2.2 – Arduino Pro Mini

Une fois le microcontrôleur en main, il nous faut ajouter d'autres composants nécessaire à la mise en place d'un drone. En branchant ces composants dans les trous sur l'extérieur de la carte nous pourrons les contrôler de manière logicielle. Le microprocesseur, lui, se chargera de les faire fonctionner physiquement.

2.1.2 Equilibrage et localisation

Il est important de disposer d'une technologie pour assister le drone à se stabiliser. Pour cela, ce dernier doit connaître "en permanence" son angle d'inclinaison. Nous avons donc besoin d'un gyroscope. Le large catalogue de modules *Arduino* propose une pièce qui fournit un gyroscope et un accéléromètre, le *MPU-6050*. Ce module a une taille et un poids similaires à ceux du microcontrôleur, 25.5 sur 15.2 millimètres pour 1.5 grammes.

Pendant un certain moment nous nous sommes demandé comment nous pourrions connaître la position de notre drone dans l'espace. Naturellement nous avons pensé au GPS, mais la précision de ces technologies (pour rester dans un budget abordable) n'est clairement pas assez précise. Bien entendu, dans le cadre où le drone devrait faire son analyse en extérieur sur une zone assez grande, les GPS sont intéressants. Mais notre drone devra simplement analyser une salle de classe, alors une précision "au mètre près" est beaucoup trop large. Nous aurions plutôt besoin d'une précision de l'ordre du décimètre. Une autre solution a été évoquée, créer notre propre système de localisation, en créant une triangulation à l'aide d'un réseau d'antennes. Toutefois, pour des raisons de coûts et de poids, cette solution ne nous semble pas viable sur notre drone. Finalement une connaissance, nous a conseillé de travailler avec un accéléromètre. Un accéléromètre est un module permettant de mesurer l'accélération linéaire d'un système. En connaissant l'accélération de notre drone, il sera alors possible de déterminer sa vitesse et donc, ses déplacements dans l'espace. Nous avons donc choisi de nous tourner vers cette solution.

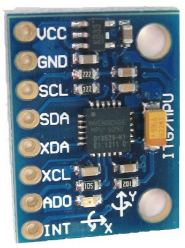

FIGURE 2.3 – MPU-6050

2.1.3 Communication avec le serveur

Afin que le drone puisse communiquer avec le serveur nous avons dans un premier temps pensé à utiliser des modules *XBee*, qui utilisent le protocole de communication sans fil, défini par le standard *IEEE 802.15.4*. Les modules *XBee* étant relativement chers (23 €), nous nous sommes penchés vers une autre technologie. Nous avons finalement opté pour un module de transmission 2.4GHz. Comme pour les autres composants que nous avons choisis, il est bon marché (0.8 €) et d'une taille de 15 sur 29 millimètres pour 2 grammes.

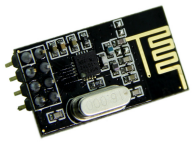

FIGURE 2.4 – Module de communication sans fil NRF24L01

2.1.4 Contrôle moteur

La plupart des drones implémentent un Electronic Speed Controler (ESC) pour chaque moteur. Ces composants servent à contrôler la vitesse du moteur ainsi que son sens de rotation. Il faut savoir qu'un ESC vaut environ 16 €. Ce qui fait 64 €pour un quadcopter. Outre le prix important de ces modules, leur poids (25 grammes/module) aussi nous oblige à nous orienter vers une autre solution. Nous avons donc pensé à créer notre propre ESC à l'aide d'un *MOFSET*, de composants basiques (condensateurs, résistances,...) et l'*Arduino*.

2.2 Circuit

Pour pouvoir assembler tous ces composants il faut réaliser le circuit imprimé. Pour cela nous avons utilisé le logiciel Fritzing. C'est un logiciel libre qui permet de réaliser des schémas électroniques ainsi que le PCB associé. Ce logiciel propose un large catalogue de base mais il est aussi possible de créer ses propres composants. Aussi, un grand nombre de personnes utilisent ce logiciel, ce qui permet dans la plupart des cas de ne pas avoir à créer de composants puisqu'il y a de grandes chances que des utilisateurs les aient déjà dessinés.

La figure 2.5 représente le schéma électronique de notre drone et la figure 2.6 le PCB associé.

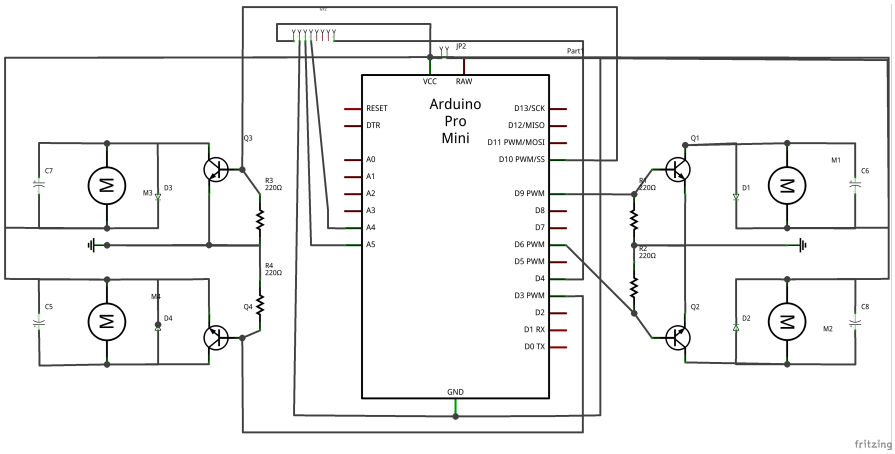

FIGURE 2.5 – Schéma du premier drone

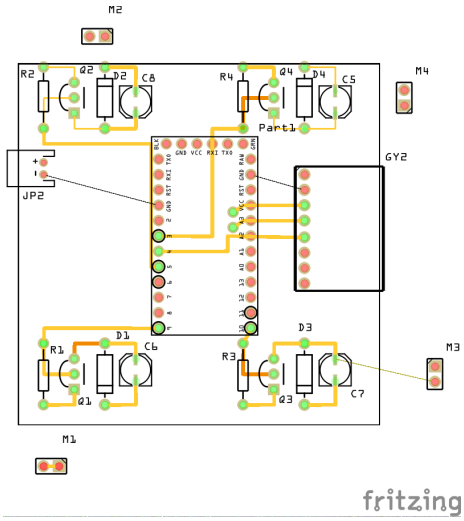

FIGURE 2.6 – PCB du premier drone

Nous avons fait le choix de ne pas utiliser de châssis. Ces composants sont assez compliqués à choisir, car il faut choisir entre un matériau léger mais coûteux et un matériau "lourd" et bon marché. Donc nous avons pensé que le circuit pourrait lui-même faire office de châssis. Bien entendu, ce choix est risqué car le drone est plus fragile. Mais notre application n'est pas vraiment "à risque" dans le sens où notre drone n'est pas censé faire des acrobaties et que nous comptions mettre en place des sécurités pour qu'il évite les obstacles de son environnement.

Avec les technologies choisis, on s'en sort pour environ 23 €de composants pour un drone. Le kit complet du crazyflie coûte à peu près 120 €. Nous comptions sur le fait que notre drone soit plus lourd (environ 35 grammes) que le *Crazyflie* (19 grammes) pour se ramener à un vol un peu moins agressif, mais cela n'a pas fonctionné. Aussi la batterie utilisée n'est pas intéressante en terme d'autonomie et fournit une tension de 3.3 Volts. Cependant les capteurs ultrason que nous comptions utiliser fonctionnent en 5 volts. Il nous a donc fallu changer aussi de batterie, ce qui nous a obligé à revoir beaucoup de choses. Car le type de batterie qu'il nous faut est plus lourd, ce qui implique de changer les moteurs. Ceci nous entraîne à la création d'un deuxième drone.

2.2.1 Fabrication

Pour ce qui est de la fabrication des appareils nous sommes passés par le fablab[6] de l'université de Cergy-Pontoise. Cet atelier se trouve à Gennevilliers (92), il propose un grand nombre d'outils mis à disposition gratuitement. La seule contrepartie est de donner de son temps à la vie du laboratoire. Toutefois pour utiliser certaines machines comme les imprimantes 3D (de qualité), les découpeuses lazer, les fraiseuses, etc... il est nécessaire de suivre une formation. Les gérants étant conscients que nous sommes soumis à des deadlines, nous ont proposé de faire les choses à notre place. En échange on devait simplement faire des formations sur l'utilisation du logiciel Fritzing et sur la fabrication d'un drone.

2.2.2 Premiers essais

Chapitre 3

Analyse

Chapitre 4

Deuxième drone

Chapitre 5

Conclusion

Annexe A

Suivi de projet

A.1 Gestionnaire de version

Pour ce projet nous avons choisi de former un groupe de trois personnes. Travailler en équipe a ses avantages, notamment pour le partage de tâches. Mais ceci peut entraîner des problèmes de version entre les travaux de chacun. Pour palier ce problème nous avons choisi d'utiliser le gestionnaire de version Git. Cet outil est vraiment pratique, puisqu'il permet de travailler à distance, de mettre à jour le code de chacun des membres, d'avoir un suivi de chaque implémentation, de faire des versions tests (sans toucher à la version fonctionnelle) ou encore de revenir à des versions précédentes du projet. Aussi cet outil nous permet de donner une visibilité à notre travail. Ainsi, si une personne compte faire un projet semblable au nôtre il pourra consulter, reprendre, modifier, améliorer... ce que nous avons fait. Notre répertoire Git est accéssible depuis cette adresse[7].

A.2 Communication

Dès le début du projet nous avons pensé qu'il serait intéressant de tenir un blog[8] pour communiquer notre progression. Lorsque l'on se lance dans un projet de cette envergure il est toujours utile de trouver des ressources sur Internet. Cela peut donner des idées et résoudre des problématiques que l'on rencontre. Afin de pouvoir communiquer avec le plus de monde possible, il est rédigé dans trois langues : anglais, français et japonais. Il présente l'avancée du projet, nos choix de développement, quelques notions de physique et les résultats de nos essais afin de guider les lecteur qui veulent créer une application similaire. Nous avons souhaité présenter le processus de construction d'un drone, sous une forme simple, en expliquant comment les choses fonctionnent.

Annexe B

Fonctionnement d'un gyroscope

Table des figures

Bibliographie

[1] Site Internet de Python. https ://www.python.org/.
[2] Site Internet de Redis. http ://redis.io/.
[3] Site Internet de Numpy. http ://www.numpy.org/.
[4] Site Internet de MatPlotLib. http ://matplotlib.org/.
[5] Site Internet de Bitcraze. http ://www.bitcraze.se/.
[6] Site Internet du Faclab. http ://www.faclab.org/.
[7] Réperoire Git du projet Njord. https ://github.com/NjordProject.
[8] Blog du projet Njord. http ://njordproject.github.io/.