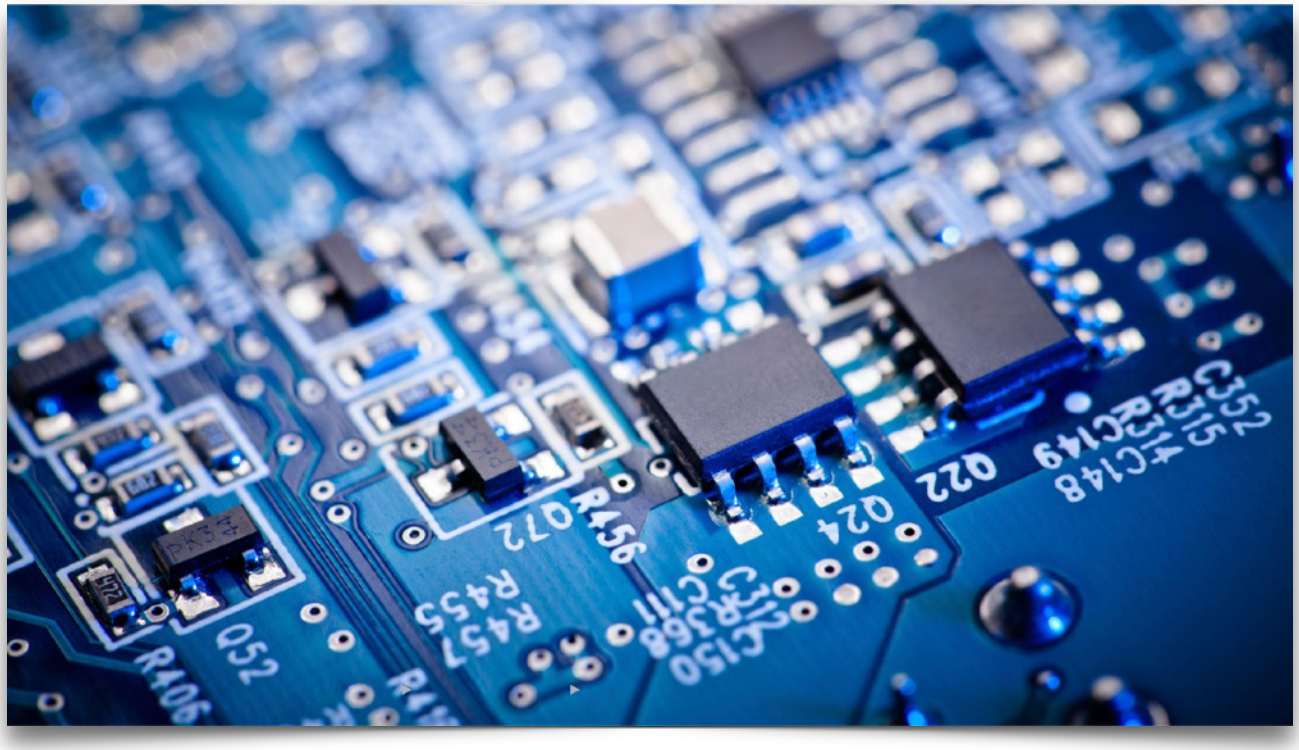
UNIVERSITÉ TOULOUSE III PAUL SABATIER

Rapport de projet

Unité d'enseignement : Bureau d'études
Licence 3 EEA-REL

Contrôle à distance d'un drone et mesure d'altitude



Binôme :
COLLE Rémi (N° étudiant : 21607362)
EL ALI Mohamed (N° étudiant : 21406893)

Encadré par :
Mr PERISSE Thierry

Avril 2017

Sommaire

Abstract

This document reports our project made in the third year of EEA-REL. Our project was about to make flying our drone on a vertical line. The goal was to make a two-way communication : the first one was commanding our drone's engines, then the drone had to send back datas about height and inertial measurements.

The idea was to make an electronic module that will be embedded into our drone. This one will receive instructions coming from another electronic module, the remote control. It's made of a joystick that will provide commands directly to the engines, an LCD screen that will receive and will display datas about height and inertial measurements.

In order to make this project, we needed know-how on digital electronic and analog electronic. We needed also to rack our brains on several Arduino issues because it wasn't a well-known programming langage. Each unit of our electronic cards were tested theoretically and were compared to the practice.

At the end of this report, we made a part about suggestions we may provide for a next version or if we had to take it up another time.

Introduction

Dans le cadre de l'unité d'enseignement « Bureau d'études » de la Licence 3 EEA-REL, chaque binôme doit choisir un sujet comme projet de 50h. Notre binôme a choisi de se pencher sur le sujet suivant :

Contrôle à distance d'un drone et mesure d'altitude

Les différents critères qui nous ont poussés à choisir ce sujet sont, dans un premier temps l'attirance que nous avons tous deux pour les drones. D'autre part, le fait que nous n'ayons jamais utilisé de carte Arduino a renforcé notre envie de découvrir cette technologie.

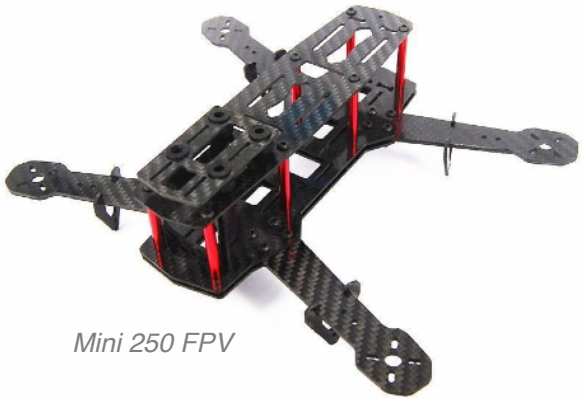
Notre rapport se divisera en quatre parties. Dans un premier temps, nous vous présenterons rapidement le drone et ses caractéristiques. Ensuite, nous vous parlerons des différentes parties techniques et de leur prise en main. Les deux parties qui suivent traiterons respectivement les parties drone et télécommandes. Pour finir, les difficultés que nous avons rencontré ainsi que les suggestions et améliorations possibles seront exprimées.

Partie 1 : Présentation du drone

1.1 - Les caractéristiques du drone

La structure du drone utilisée dans le projet est un Mini 250 FPV Quadcopter. Nous disposons de 4 moteurs Brushless et de 4 régulateurs de vitesse.

Voici la fiche technique de notre ensemble drone-moteurs-régulateurs :



Mini 250 FPV

STRUCTURE : MINI 250 FPV QUADCOPTER

MATIÈRE DOMINANTE : FIBRE DE CARBONE

POIDS : 180G

4 MOTEURS BRUSHLESS MT2204
4 RÉGULATEURS DE VITESSE 12A ESC
BATTERIE LIPO 3S - 1300MAH

1.2 - Le cahier des charges

Le cahier des charges imposé est le suivant :

1. **Réalisation d'un drone**
2. **Se familiariser avec les signaux de commande du drone**
3. **Etudier une solution de remplacement de la télécommande et du récepteur du drone par le matériel suivant : module Xbee, microcontrôleur PIC, Arduino…**
4. **Rendre la liaison bidirectionnelle en intégrant un capteur d'altitude sur le drone, éventuellement une caméra et d'autres capteurs**
5. **Faire un bilan énergétique du drone**

1.3 - Les signaux de commande du drone

Le bloc régulateur de vitesse ainsi que le bloc moteur sont ceux qui commanderont le drone. Grâce à un signal PWM en entrée du régulateur de vitesse, celui-ci contrôlera la vitesse de rotation de notre moteur en fonction de la commande appliquée en entrée.

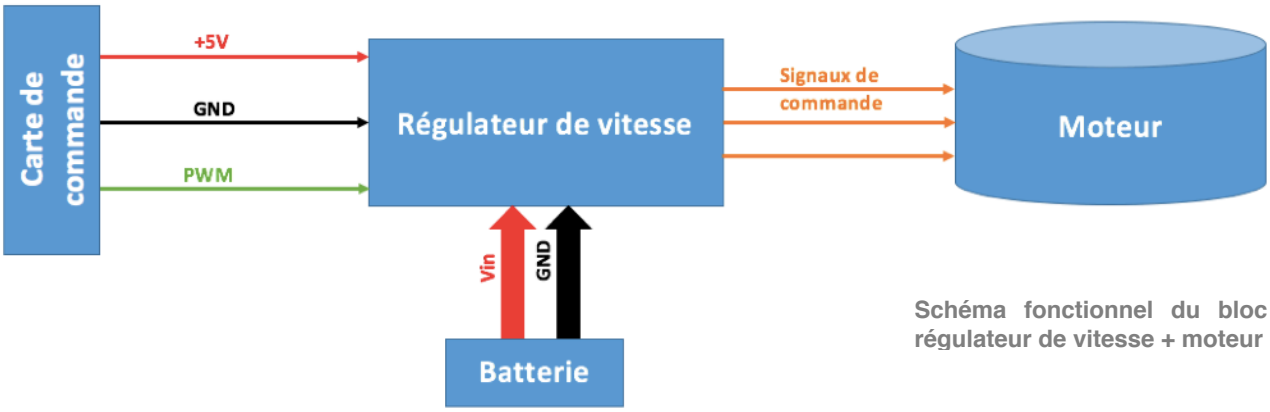
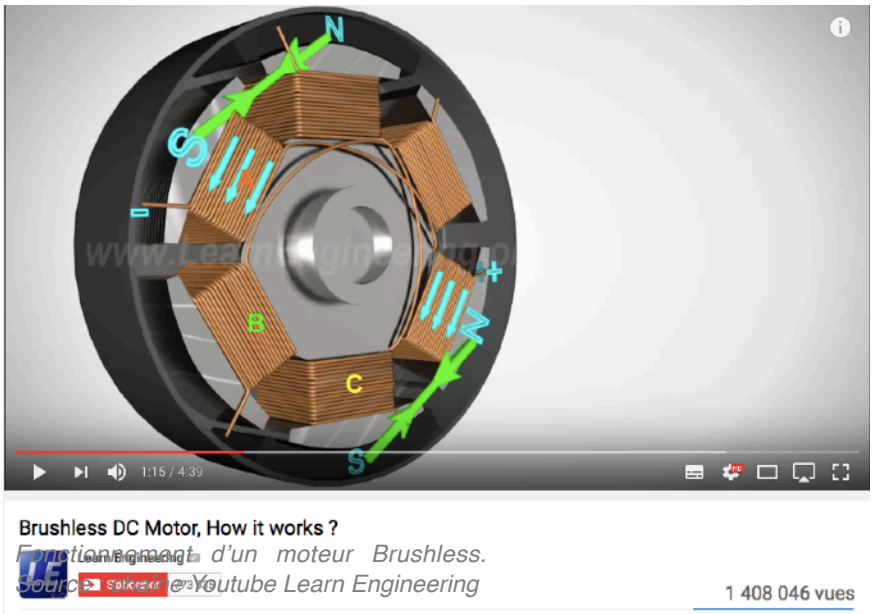


Schéma fonctionnel du bloc régulateur de vitesse + moteur

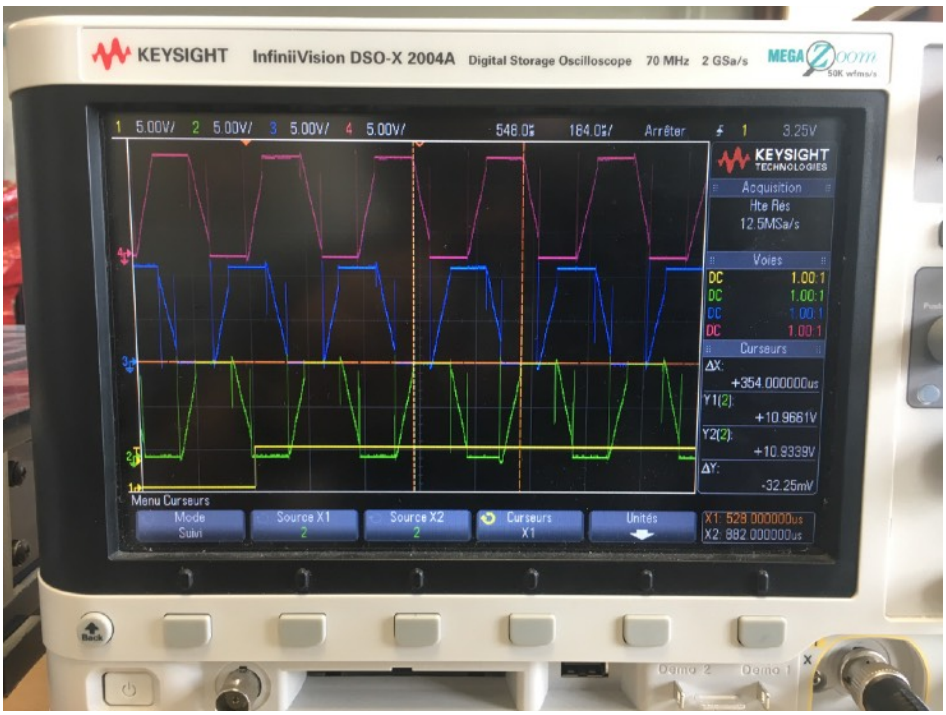
Moteur brushless… Comment ça marche ?

Le moteur Brushless est composé d'un stator et d'un rotor. Le rotor est un aiment permanent. Le stator, lorsque l'on lui applique une tension continue sur l'un de ses enroulements (spires), deviendra un électroaimant. Les pôles opposés du stator et du rotor vont s'attirer, donc le rotor va faire une rotation… Pour entrainer le moteur, il faudra faire varier l'alimentation des différents enroulements !



Fonctionnement d'un moteur Brushless.
Source : Youtube Learn Engineering

Sur ce schéma, l'enroulement A (en orange) est alimenté en tension continue, ce qui crée un champ électromagnétique sur l'enroulement A. On observe en bleu les pôles de l'aimant permanent et l'aimant électromagnétique. Les pôles opposés s'attirent, ce qui entraîne un mouvement du rotor. Sur la figure de la page suivante, on peut observer l'alternance de ces 3 signaux DC.



Visualisation sur oscilloscope des 3 signaux en sortie du régulateur de vitesse commandé depuis un Joystick

Régulateur de vitesse… Comment ça marche ?

Chaque moteur sera relié à son régulateur de vitesse. Le but de celui-ci est d'alimenter le moteur en tension continue, tour à tour, sur chacun de ses 3 enroulements (spires). En entrée du régulateur de vitesse, outre les tensions de la batterie et de la carte de commande toutes les deux embarquées, on retrouve un signal PWM. Ce dernier est celui qui commandera indirectement la vitesse du drone, en faisant varier l'alimentation des enroulements du moteur en tension DC. Sur l'image ci-dessus, nous observons les 3 sorties de notre régulateur de vitesse. On observe bien le décalage temporel entre chacune des tensions délivrées aux différents enroulements du moteur.

1.4 - Le drone en vol

En fonction de la vitesse de rotation des 4 moteurs et de leur sens de rotation, le drone en vol peut se déplacer de plusieurs manières. En voici les principales :

Vol stationnaire

Les 4 moteurs tournent à la même vitesse. Notons que les moteurs, opposés deux à deux, ont des sens de rotation différents[1].

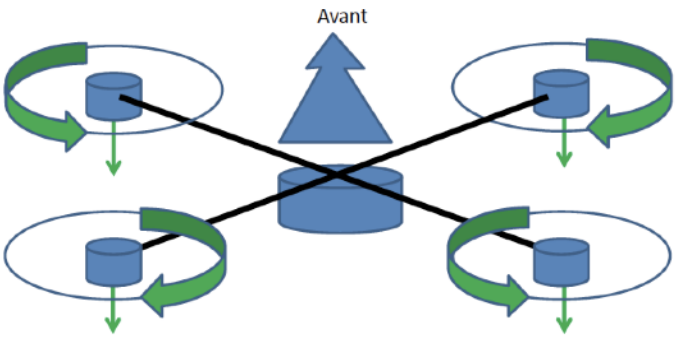


Schéma fonctionnel des moteurs en vol stationnaire

Tangage : avancer/reculer

Pour avancer (respectivement reculer) on ralentit les moteurs avants (respectivement arrières). Le drone va alors tanguer et sera porté vers le côté où les moteurs sont les moins rapide.[1]

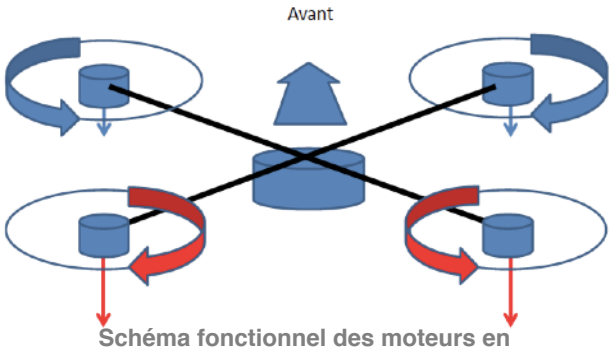


Schéma fonctionnel des moteurs en tangage (vers l'avant dans ce cas)

Roulis : droite/gauche

Pour aller à droite (respectivement gauche), on ralentit les moteurs de droite (respectivement gauche)[1].

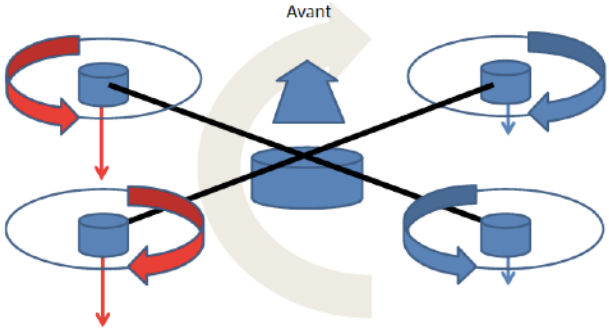


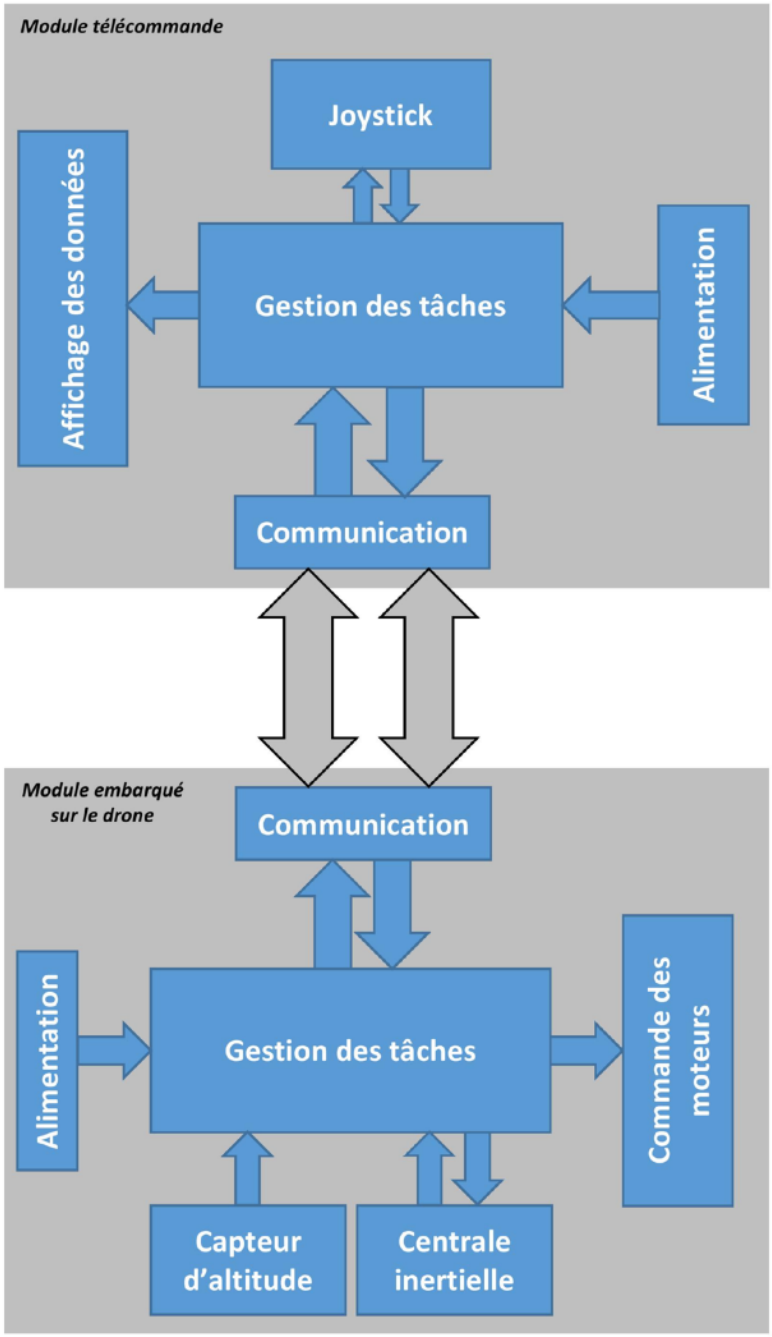
Schéma fonctionnel des moteurs en roulis (vers la droite dans ce cas)

Dans notre projet, nous nous concentrerons uniquement sur l'axe vertical. En effet, notre rôle sera de faire décoller le drone depuis sa position de repos, lui faire effectuer un vol stationnaire et ensuite le faire redescendre.

1.5 - Le schéma fonctionnel du drone et sa commande

La présentation du drone étant faite, nous avons ciblé les principaux axes de notre projet. Conformément à notre cahier des charges, le schéma fonctionnel de notre projet est exposé à la page suivante.

La structure du drone nous a été fournie déjà faite. Dans ce projet, nous nous chargerons de piloter les moteurs du drone avec des signaux de commande. Nous rendrons la liaison bidirectionnelle entre la commande et le drone, en embarquant sur ce dernier, une centrale inertielle et un capteur d'altitude. Enfin, nous créerons un module télécommande qui permettra de communiquer avec le drone, ce sera une solution de remplacement de l'HyperTerminal du PC.



Le module télécommande enverra les signaux de commande des 4 moteurs, contrôlant donc la vitesse de rotation des ces derniers à l'aide d'un Joystick. Il sera alimenté par une batterie de 9V que l'on trouve aisément sur le marché. Le module embarqué sur le drone recevra, à l'aide de la communication choisie, les signaux de commande provenant de la télécommande. Il enverra les données provenant de ses deux capteurs : altitude et centrale inertielle. Ces 2 données seront affichées sur un écran LCD se trouvant dans le module télécommande. Le drone sera également alimenté par une batterie de 9V, elle même déjà fournie avec son chargeur.

A présent, nous détaillons de façon technique les deux blocs : module télécommande et module drone.

Partie 2 : Les pré-requis

2.1 - Arduino : choix et prise en main

Arduino ou PIC ?

Dans le cadre de ce projet, nous avons rapidement choisi une Arduino.
Pour utiliser un PIC, il faudrait concevoir la carte électronique nous même, y rajouter des composants : condensateurs, quartz, régulateurs de tension… Une majeure partie de notre temps se serait alors tourné vers la conception de la carte électronique, son routage et son tirage. Nous nous sommes donc posé la question de savoir si une meilleure solution serait envisageable.

Arduino est une carte électronique programmable et son logiciel de programmation (IDE) est gratuit. En plus de son faible coût, cette carte permet de connecter des *Shields* qui sont conçus pour être aisément embarqués sur Arduino. Nul besoin de programmateur contrairement au PIC. La force d'Arduino se trouve sur internet, où des milliers de programmes sont disponibles dans diverses applications.

Choix de l'Arduino Nano

Parmi toutes les cartes de la gamme Arduino, nous avons rapidement sélectionné deux cartes : la NANO et la MINI. Les autres ont été rapidement oubliées du fait de leur poids et de leurs caractéristiques bien trop évoluées (mémoire vive, stockage…) pour notre projet (comparatif des Arduino en annexe). Les deux cartes citées ont exactement les mêmes caractéristiques en tous points. La seule différence se joue au niveau de la taille. En effet, la carte Arduino MINI plus petite, donc plus légère que l'Arduino Nano (parfait pour la faire embarquer sur l'aéronef). En définitive, la MINI est la plus adaptée à notre projet, mais, la NANO étant d'ores et déjà disponible, nous avons décidé de ne pas racheter le composant pour seulement quelques millimètres.

Prise en main de l'Arduino Nano

Dans un second temps, nous nous sommes initiés au langage spécifique d'Arduino à travers différentes manipulations que nous allons vous présenter ci-dessous.

AFFICHAGE D'UN MESSAGE SUR LE MONITEUR SÉRIE

Nous avons décidé de faire ce test dans l'optique d'utiliser plus tard la communication série du module Xbee et de la carte Arduino. Lors de la réalisation, nous avons utilisé la fonction **Serial** qui est une fonction déjà présente sur Arduino et conçue pour la communication série.

```
int affichageFait=0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("\nCommunication initialisée");
}

void loop() {
  // put your main code here, to run repeatedly:
  if (affichageFait==0) {
    Serial.println("Hello World");
    affichageFait=1;
    }
}
```

Code Arduino pour l'affichage d'un message

Cette première manipulation nous a permis de nous familiariser avec l'affichage d'Arduino. Ici, on utilise **Serial.println** pour afficher une seule fois « Hello World » sur le moniteur série et revenir à la ligne. Nous savons donc comment transmettre des informations sous forme de phrases ou de données.

CLIGNOTEMENT D'UNE LED
Lors de cette seconde manipulation, nous voulons faire clignoter une LED à intervalles réguliers.

```
const int LED = 12;

void setup() {
  // put your setup code here, to run once:
 pinMode(LED, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

  digitalWrite(LED, HIGH);
  delay(500);
  digitalWrite(LED, LOW);
  delay(500);
}
```

Code Arduino pour faire clignoter une LED

Pour ce faire, nous déclarons sur quel port la LED sera câblée (ici le port 12), nous définissons ensuite cette LED comme une sortie et précisons dans le loop que nous la voulons allumée pendant 500 millisecondes (état haut) et éteinte pendant 500 autres millisecondes (état bas). Ici, nous avons utilisé les sorties numériques et compris leurs fonctionnements, nous permettant de les utiliser pour le capteur d'altitude ou le PWM.

Ces essais nous ont permis de comprendre comment faire passer diverses informations à travers la carte. Ils nous permettrons aussi de faire les petits programmes utiles, par exemple, pour renvoyer les informations du capteur d'altitude sur l'écran, le tout, afin de pouvoir l'associer bloc à bloc pour finir par les deux blocs généraux « télécommande » et « drone ».

2.2 - Prise en main de la communication Xbee

Le protocole Zigbee utilisé à travers les modules Xbee permet de communiquer par ondes radio. Cette communication est plutôt simple à manipuler, et, étant donné le temps assez cours imparti il nous faut un mode de communication rapide à mettre en place. L'objectif est de permettre le contrôle du drone via Xbee.

Ces modules sont capables de transmettre les informations jusqu'à 1600m en extérieur et 100m en intérieur. De plus, ce type de module est suffisamment léger pour pouvoir être embarqué sur notre machine.

Dans le cadre de notre projet, chacun de nos modules (télécommande et drone) auront un Xbee accompagné de leur Arduino respectif.

Comment câbler un module Xbee sur la carte Arduino ?

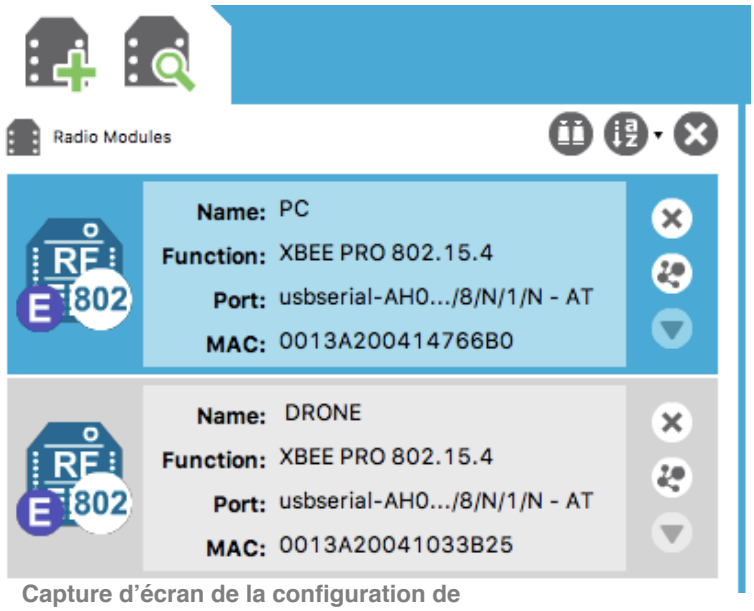
Arduino et Xbee communiquent entre eux par communication série. Pour cela, on dispose de 2 broches :

- TX : Transmission
- RX : Réception

La broche TX de l'Arduino sera connectée à la broche RX du Xbee. Suivant cette connexion, nous aurons un envoi d'information de l'Arduino vers le Xbee. L'inverse est réalisé pour la réception d'informations sur Arduino depuis le Xbee : le RX de l'Arduino est connecté au TX du Xbee.

Jumelage des 2 modules Xbee

Dans un premier temps, il faut jumeler les deux Xbee. Le logiciel XCTU, gratuit, est un HyperTerminal de communication série. Son grand avantage est que l'utilisateur pourra paramétrer ses modules Xbee en fonction de son utilisation. Les 2 modules Xbee doivent être de la même série (Pro dans notre cas) et doivent avoir les mêmes paramètres de configuration. Par défaut, nous choisissons :



Capture d'écran de la configuration de
2 Xbee sur le logiciel XCTU

Vitesse : 9600 Baud
Données : 8 bits
Pas de bit de parité
1 bit de stop
Pas de contrôle de flux

Xbee Pro

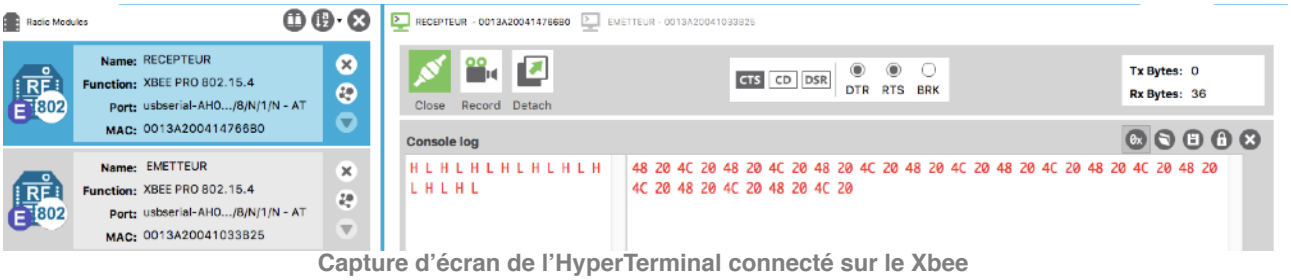
Pour rendre la lecture plus facile, nous donnerons des noms à nos modules Xbee. Prenons un exemple ici :

- DRONE : Module Xbee se trouvant dans le module embarqué dans le drone
- PC : Module Xbee relié au PC via USB

A noter que la configuration des deux modules se fait en connectant les Xbee à l'HyperTerminal via USB. Une fois configuré, le module Xbee appelé DRONE pourra être embarqué sur le drone.

Premiers tests

Après avoir jumeler les modules suivant la méthode décrite précédemment, nous allons pouvoir vérifier la communication. On demandera tout simplement à l'Arduino connecté au Xbee émetteur, d'envoyer une séquence 'H' et 'L'. Sur l'HyperTerminal, nous devrions réussir à lire cette séquence reçue par notre second Xbee récepteur câblé à l'ordinateur via USB [2].

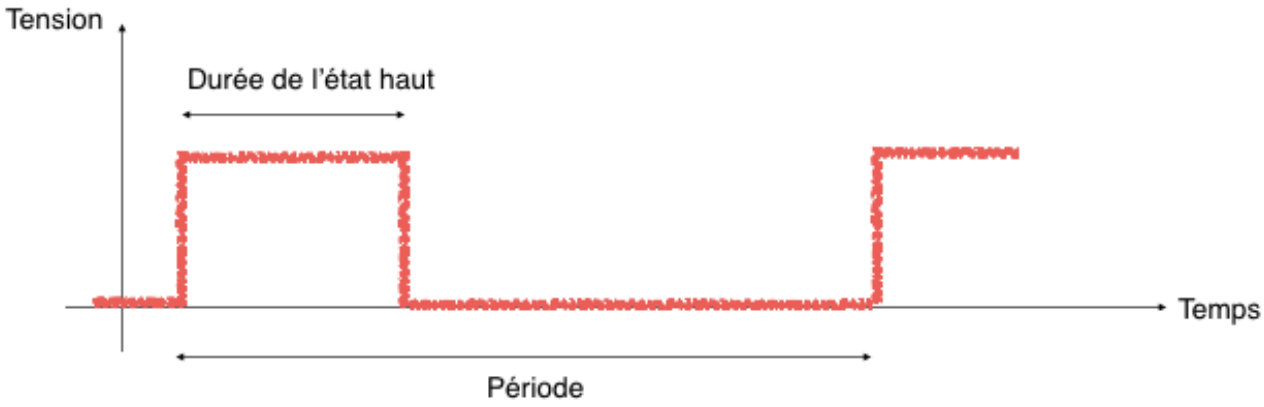


Capture d'écran de l'HyperTerminal connecté sur le Xbee
Récepteur : en rouge les données reçues

2.3 - Notions de PWM

Le PWM est un signal numérique. Ce signal carré possède une pseudo-période et deux états : état haut et état bas. La caractéristique principale de ce signal est que nous pouvons modifier la largeur de son état haut. Pour ce faire, il suffit de modifier le rapport cyclique alpha qui représente la durée de l'état haut (entre 0% et 100%).

La modification du rapport cyclique influe sur la tension moyenne produite par le signal.
Lorsque alpha vaut 0%, la tension moyenne est nulle. En revanche, si alpha vaut 50% la tension moyenne sera égale à la valeur de tension max/2. Pour finir, si alpha vaut 100% alors, la tension moyenne est égale à la tension maximale.

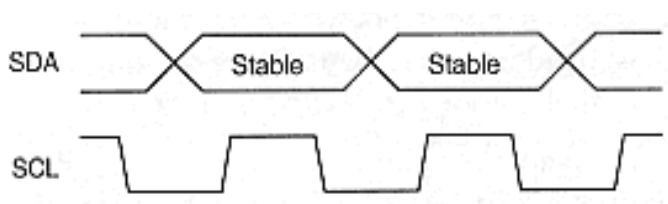


Le signal PWM sera utilisé dans les deux applications : commande des moteurs et capteur d'altitude.

2.4 - Introduction au Bus I2C

La centrale inertielle utilisée dans notre projet se base sur le bus I2C, en communication série. Pour mieux comprendre, voici les points les plus importants [3]:

- C'est un bus bifilaire qui utilise une ligne de données (SDA) et une ligne d'horloge (SCL)
- On peut échanger les données dans les deux sens
- Le bus est multi-maîtres
- Chaque abonné du bus possède une adresse codée sur 7 bits
- Un acquittement est généré pour chaque octet de donnée transféré



Chronogramme d'une transmission par bus I2C

La figure ci contre résume le principe fondamental d'une transmission I2C. L'émetteur doit positionner son paquet de données (SDA) avant le front montant de l'horloge (SCL). En effet, lorsque l'horloge est à son état haut, le paquet de données est transféré[3].

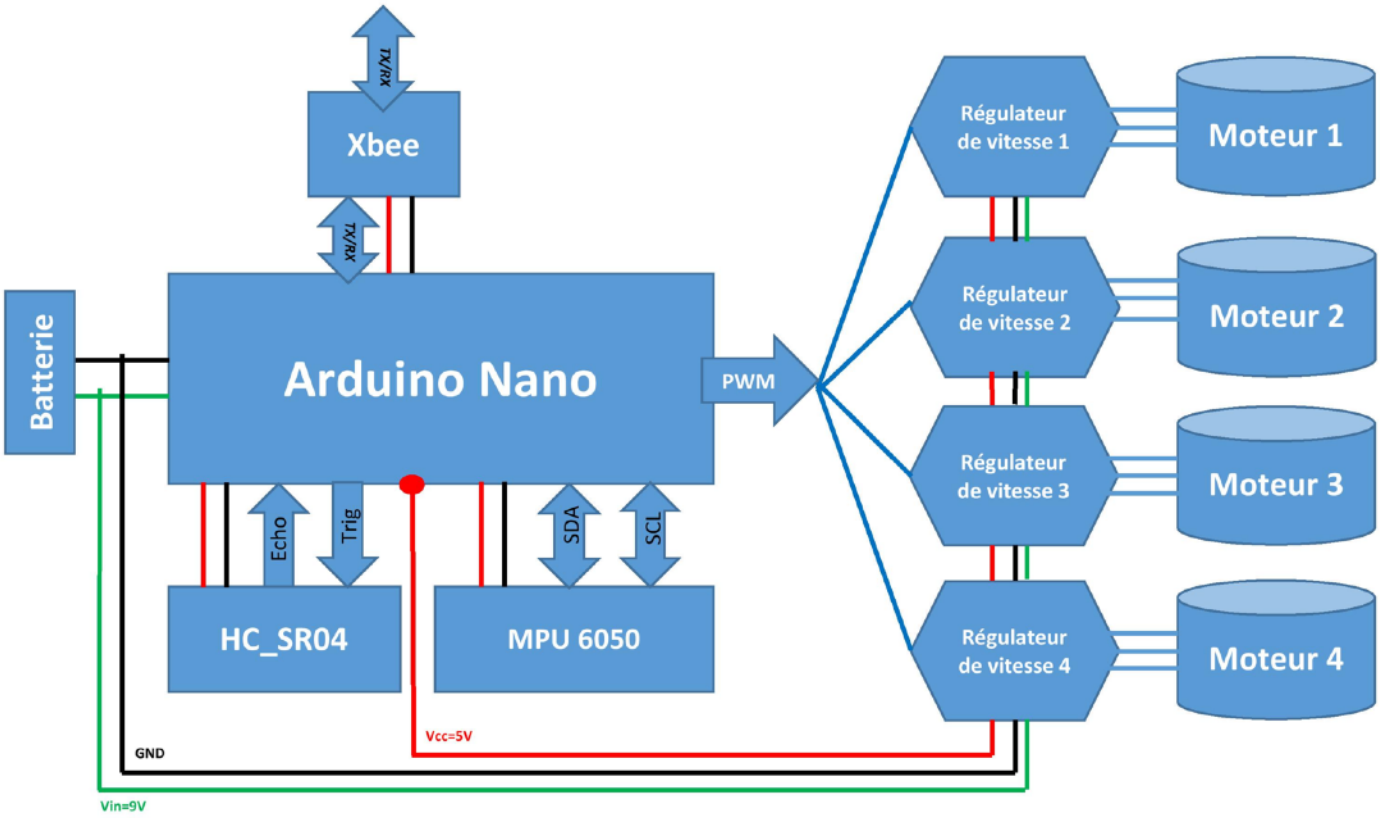
Comme c'est une communication série, c'est à dire que plusieurs paquets sont envoyés les uns après les autres, il faut bien distinguer ces différents paquets. On a donc deux éléments [3]:

- Condition de départ : réalisée au front descendant de SDA et au niveau haut de SCL
- Condition d'arrêt : réalisée au front montant de SDA et au niveau haut de SCL

Ce type de bus sera utilisé pour la centrale inertielle et l'écran LCD.

Partie 2 : Module embarqué sur le drone

2.1 - Schéma fonctionnel TECHNIQUE du module embarqué

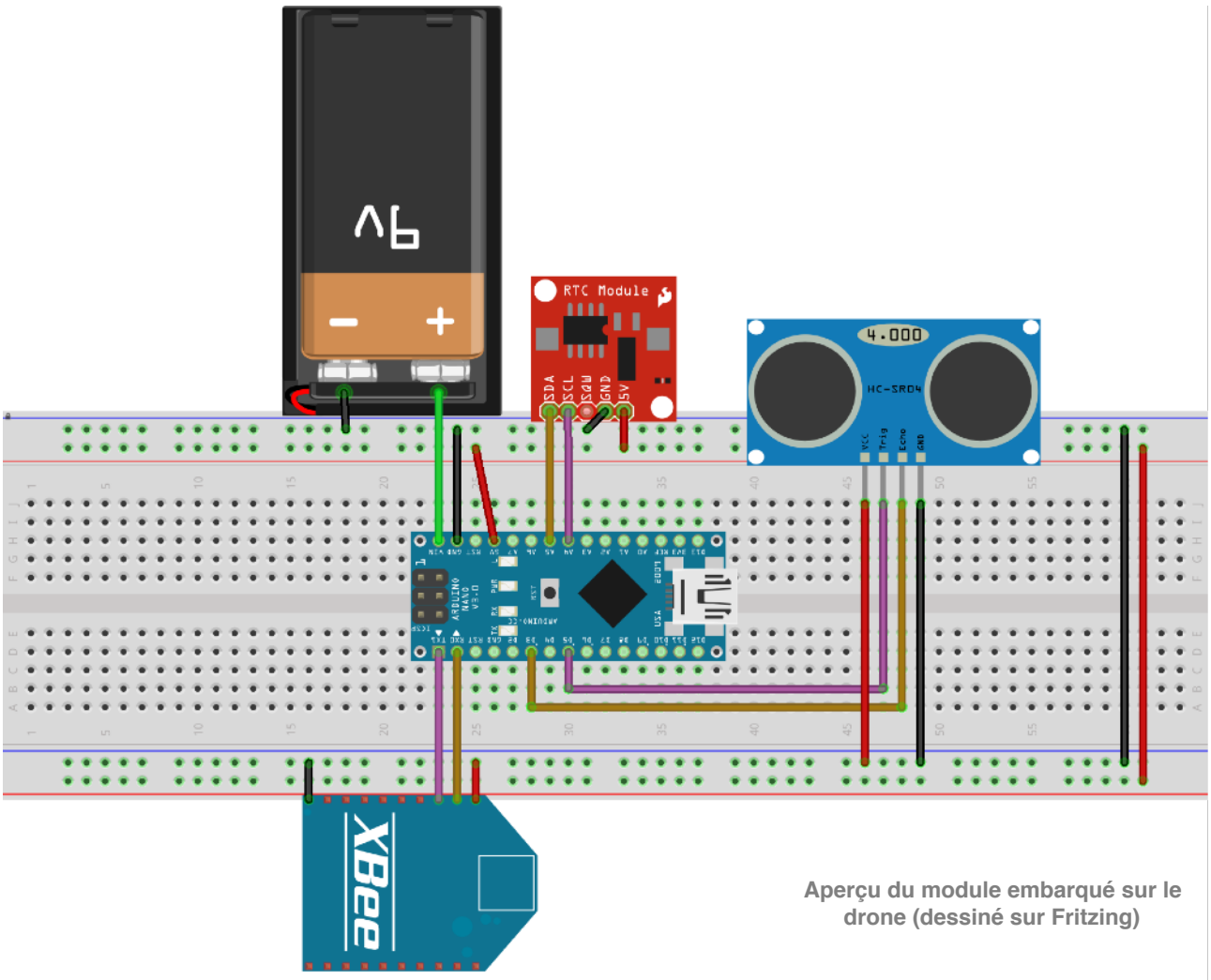


Une batterie rechargeable de 9V alimentera la carte Arduino Nano ainsi que les 4 régulateurs de vitesse. Nous retrouvons un capteur d'altitude HC_SR04, une centrale inertielle MPU6050, ainsi qu'un module de communication Xbee.

2.2 - Gestion des tâches : Arduino Nano

L'Arduino Nano semblait être l'une des meilleures cartes ayant comme compromis poids/ performance. En effet, le poids était une contrainte pour notre module embarqué sur le drone. Plus la charge est lourde, plus on a besoin de puissance et indirectement cela aurait un impact sur l'autonomie de la batterie.

Les 9V de la batterie du drone alimenteront la carte Arduino Nano. Le potentiel positif de la batterie est relié à l'entrée Vin de la Nano. Cette entrée possède un régulateur de tension qui convertira une tension dans la fourchette [7; 12]V en une tension Vcc de 5V. Le Vcc sera utilisé pour alimenter les capteurs et le module Xbee. La masse de tout le circuit sera donc celle de la batterie, reliée aussi au GND de l'Arduino Nano. Entre autre, les *pins* qui nous intéressent seront les entrées/sorties numériques, les sorties analogiques, et les sorties PWM ainsi que le TX/RX de la communication série.



Aperçu du module embarqué sur le drone (dessiné sur Fritzing)

Ci-dessus, un schéma de la platine électronique qui sera embarquée sur le drone. Nous mettrons dessus le capteur d'altitude HC_SR04, la centrale inertielle MPU6050, le module Xbee, ainsi que la batterie de 9V. A noter que cet Arduino recevra la consigne de la télécommande (état haut d'un PWM) et fabriquera lui même un signal PWM qui sera disponible sur sa sortie numérique D6.

2.3 - Signaux de commande du moteur

En entrée du régulateur de vitesse, on injecte un signal PWM. La durée de l'état haut du PWM fera tourner le moteur correspondant, plus ou moins rapidement. Les sorties PWM de notre Arduino Nano sont à une fréquence fixe de 62.500Hz. Nous chercherons à régler notre fréquence PWM aux alentours de 64Hz, cette valeur étant imposée par la datasheet de notre régulateur de vitesse.

Réglage de la fréquence d'une période

Pour une impulsion, on règle la période de notre signal pulse à 16.35ms (61Hz). Pour cela, il est possible de diviser la fréquence PWM par une des valeurs entières données par le constructeur. Celle qui nous intéresse est la division par 1024, elle permettra d'obtenir une fréquence de 61Hz, la plus proche de la fréquence imposée de 64Hz. Dans ce cas, la valeur de notre diviseur, appelée *prescale*, sera de 1024. Dans le tableau du constructeur, voici la ligne de code à rentrer dans le programme :

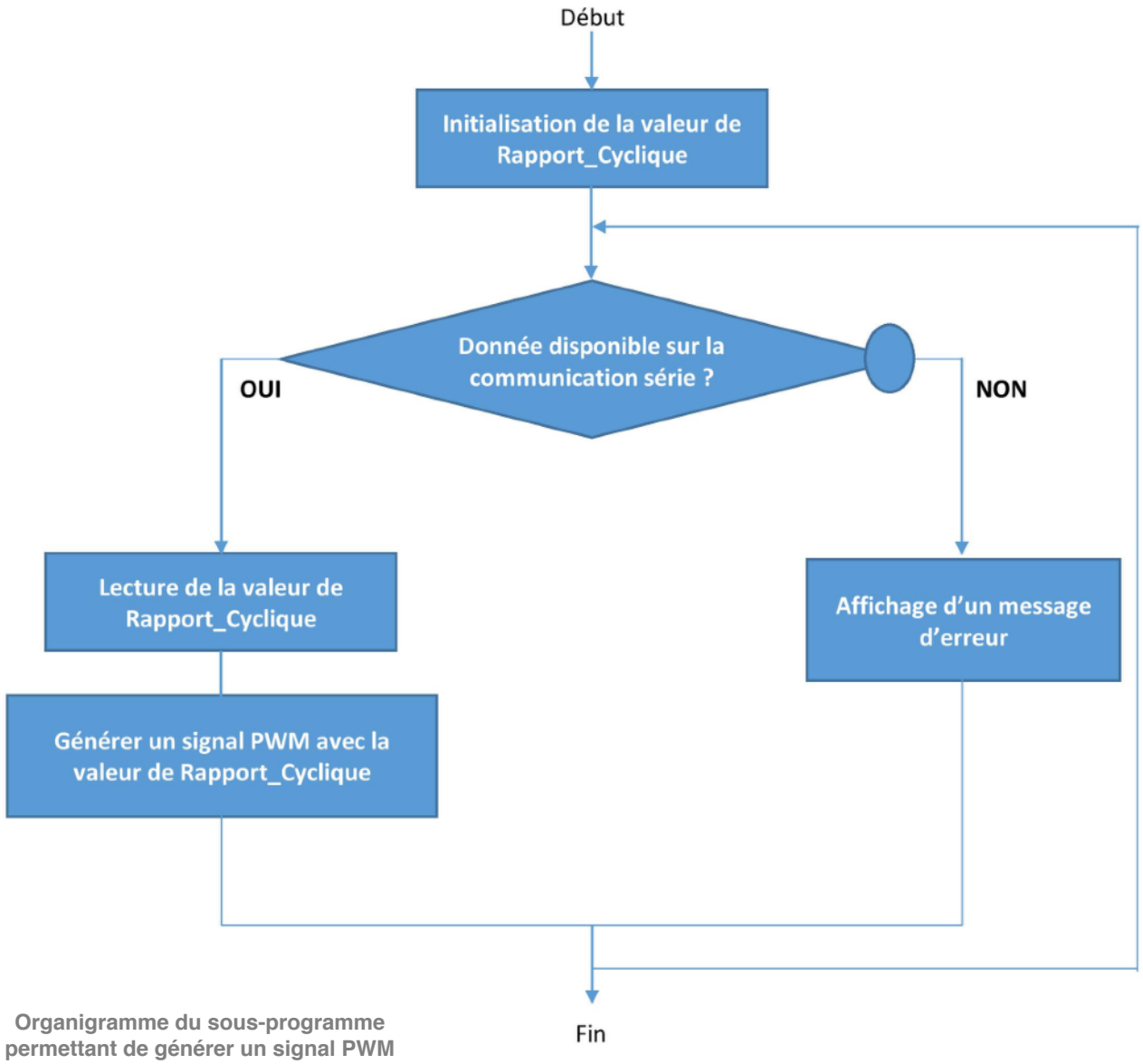
```
TCCR0B = TCCR0B & B11111000 | B00000101;
```

Réglage de la valeur du rapport cyclique

Le signal PWM étant injecté en entrée du régulateur de vitesse, la durée de l'état haut est le paramètre qui contrôlera la vitesse de rotation du moteur en question. La durée de l'état haut du PWM devra appartenir à l'intervalle [1.2 ; 2]ms imposé par la datasheet du régulateur de vitesse.

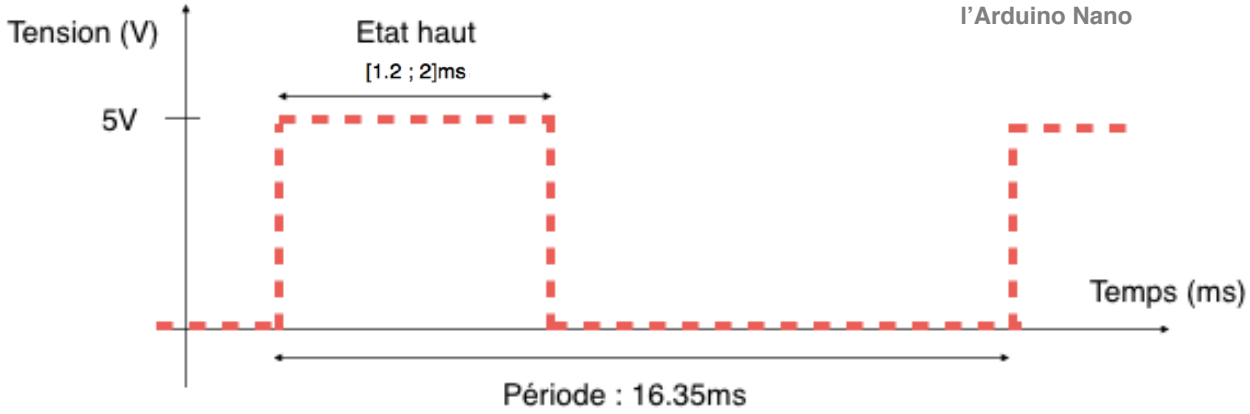
Le rapport cyclique est le paramètre principal que le module télécommande (par l'intermédiaire du joystick) devra envoyer à notre module embarqué sur le drone. L'Arduino qui contrôle ce dernier, recevra uniquement la valeur du rapport cyclique et se chargera de créer le signal PWM correspondant, sur un pin en sortie.

Pour le passage sur le banc de test, on utilisera un joystick délivrant [0 ; 3.5]V connecté à la carte Arduino. Il permettra donc à l'état haut de transiter dans l'intervalle [1.2 ; 2] ms de l'état haut.



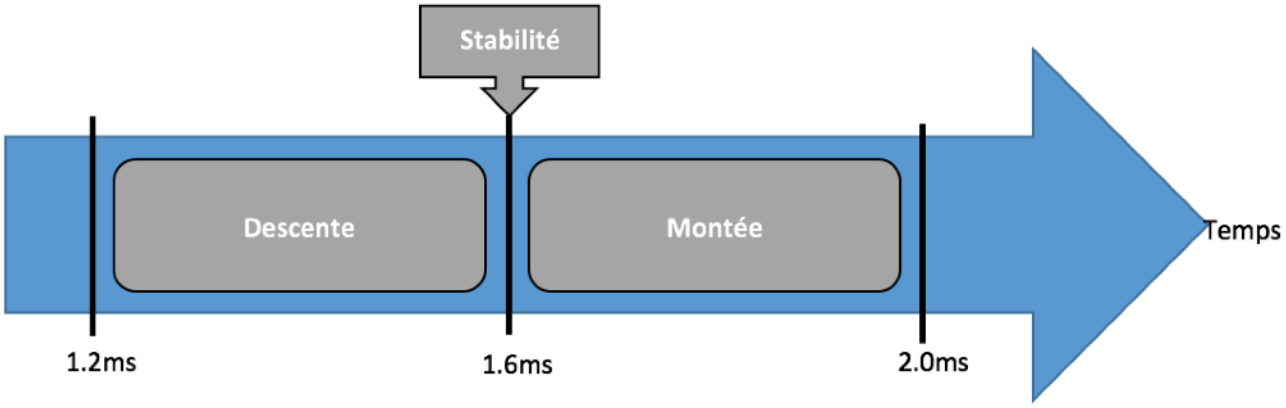
Organigramme du sous-programme permettant de générer un signal PWM pour contrôler le moteur

Allure du signal PWM en sortie D6 de l'Arduino Nano



Selon la durée de l'état haut, on aura 3 évolutions sur le drone :

- Valeur de référence (= 1.6ms) : Stabilité du drone
- Supérieur à la valeur de référence (> 1.6ms) : Montée du drone
- Inférieur à la valeur de référence (< 1.6ms) : Descente du drone



Incidence de la durée de l'état haut du signal PWM sur le drone

Ce signal sera disponible sur le pin numérique D6 de l'Arduino Nano, que l'on a configuré comme sortie. On connectera donc D6 à l'entrée du régulateur de vitesse.

2.4 - Capteur d'altitude : HC_SR04

Le cahier des charges nous imposait d'embarquer un capteur afin de rendre la liaison bidirectionnelle. Nous avons donc choisi d'intégrer un capteur d'altitude.
Le capteur est alimenté en +5V. Nous envoyons sur son entrée TRIG un signal PWM délivré par le GBF, avec l'état haut (5V) à au moins 10μs pour avoir le temps d'être détecté. Le capteur émet une série d'impulsions ultrasoniques et l'obstacle les lui renvoie. Le capteur renvoie alors un signal PWM sur sa sortie ECHO. C'est la durée de l'état haut du signal ECHO qui nous permettra d'obtenir la distance du capteur par rapport à l'objet, par la formule suivante (tirée du datasheet) [4]:

HC_SR04

(Durée de l'état haut en μs) / 58 = Distance en cm

Ci-dessous, nous retrouvons l'organigramme de notre algorithme embarqué sur l'Arduino. Notons que TRIG est une broche numérique déclarée en sortie, ECHO une broche numérique déclarée en entrée. Le programme Arduino se trouve en annexe de ce rapport.

Organigramme du sous-programme permettant de traiter les données du capteur d'altitude

Petit test avec un obstacle à 10cm environ

Pour ce test, nous avons câblé l'Arduino Nano au capteur HC_SR04. L'Arduino envoie un PWM avec l'état haut (+5V) à environ 30μs sur l'entrée TRIG et relié à la voie n°1 de l'oscilloscope.

Visualisation sur l'oscilloscope des signaux TRIG et ECHO du capteur d'altitude

Nous récupérons la sortie ECHO sur la voie n°2 de l'oscilloscope. Nous observons l'information renvoyée sur la sortie ECHO après chaque impulsion sur l'entrée TRIG. A l'aide des fonctionnalités de l'oscilloscope, nous mesurons la durée de l'état haut du signal de sortie, qui est à 600μs environ. Faisons le calcul pour vérifier cela : 600μs / 58 = 10.3cm, le résultat est cohérent avec l'obstacle du test.

2.5 - Centrale inertielle : MPU6050

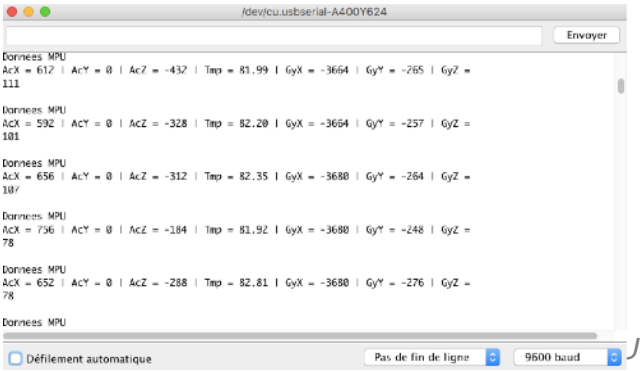
Cette centrale inertielle est un module 3 axes (x, y, z) comportant un capteur d'accélération couplé à un gyroscope. Dans le cadre de notre projet, cette centrale inertielle servira à nous renvoyer des données brutes concernant la position angulaire (gyroscope) et l'accélération linéaire (accéléromètre). Ainsi, le but final serait de faire de notre drone un système bouclé, utilisant les données MPU afin de corriger la stabilité du drone.

MPU6050

Concernant notre projet, par manque de temps et à cause de la difficulté, nous nous sommes limité à **l'acquisition des données brutes**.

Récupération des données brutes

Pour pouvoir faire communiquer Arduino Nano et MPU6050, il suffit de relier les pins SDA et SCL de la centrale inertielle, à deux entrées analogiques au choix, sur notre Arduino. Une adresse est donc affectée à la centrale inertielle. Le programme Arduino traitant la réception des données provenant de la centrale inertielle, sera mis en annexe de ce rapport.



Pour tester notre centrale inertielle, nous relions cette dernière à l'Arduino Nano. Le programme que nous rentrerons dans la Nano récupèrera les données de la MPU et écrira sur son port série. Nous observerons, dans un premier temps, la réception des données depuis le moniteur série de l'IDE Arduino.

Visualisation sur le moniteur série d'Arduino, des donnés de la MPU

2.6 - Communication Arduino+Xbee / Xbee+PC

A présent, nous testons l'ensemble de notre module embarqué sur le drone. Donc à l'Arduino Nano, seront reliés le capteur d'altitude, la centrale inertielle, et le Xbee.

A l'aide de 2 Xbee Pro, dans un premier temps, nous ferons communiquer notre module drone avec l'HyperTerminal du PC.

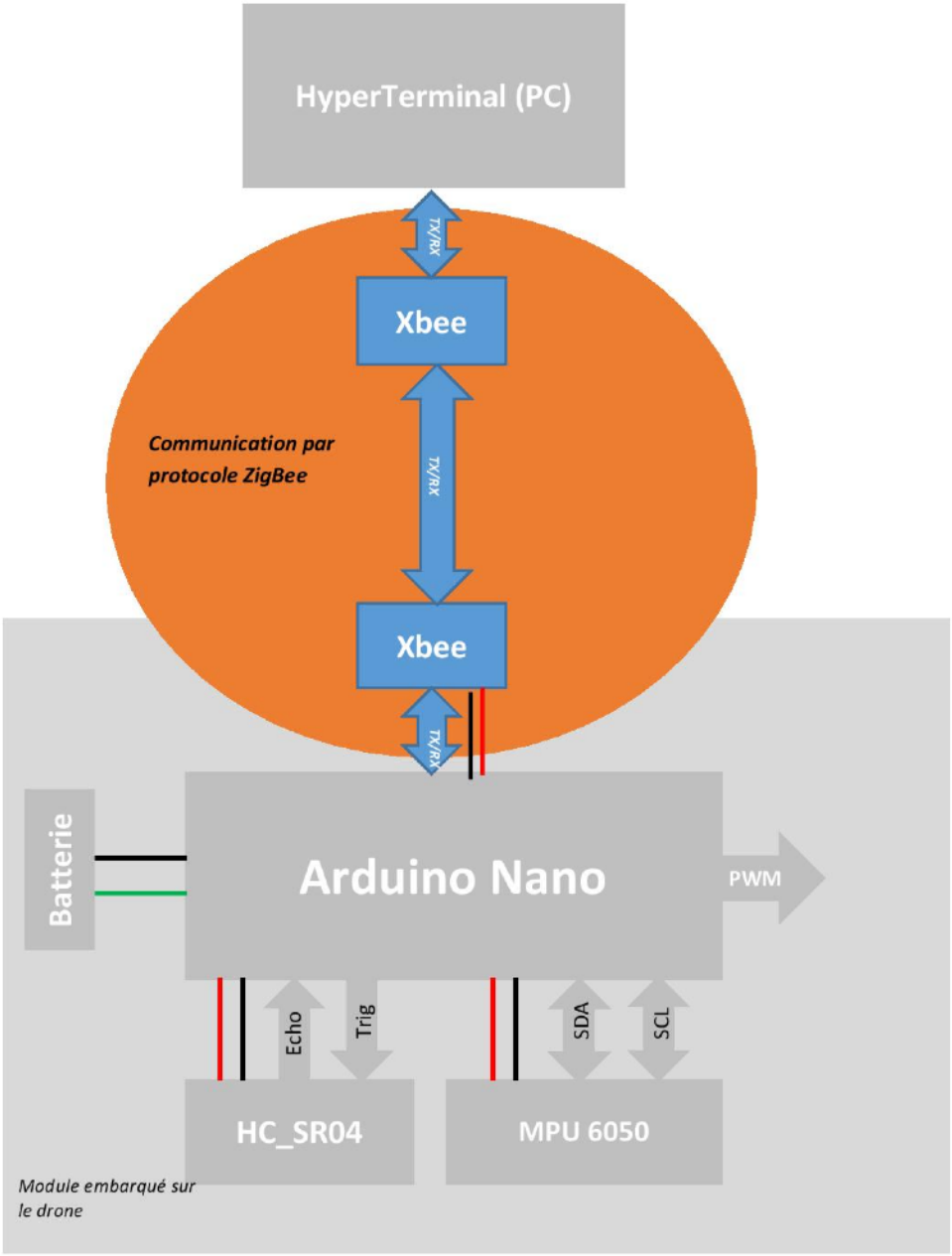


Schéma fonctionnel de la communication Xbee entre le module embarqué sur le drone et le PC

Préparation du Xbee DRONE

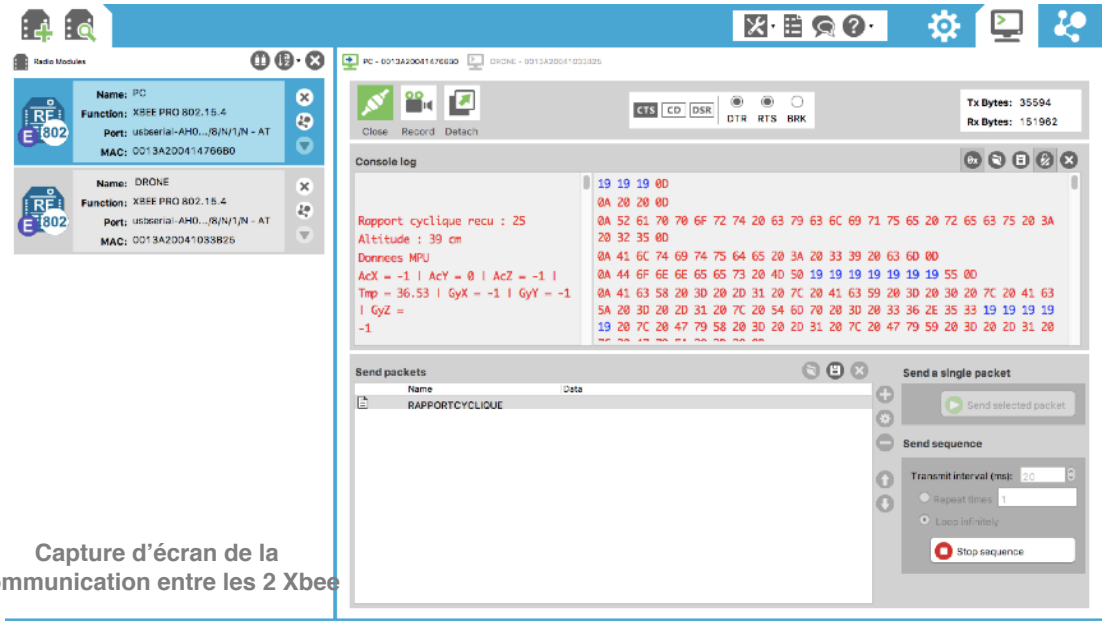
Le module embarqué sur le drone, comme nous l'avons expliqué dans la partie précédente, permettra de :

- Envoyer : l'altitude & données brutes de la centrale inertielle
- Recevoir : rapport cyclique du PWM

Nous câblons donc notre Xbee conformément à l'explication donnée en partie 2.
Notre module est à présent équipé de son Xbee, et donc prêt à transmettre/recevoir des données.

Visualisation sur l'HyperTerminal de XCTU

Nous rentrons enfin dans le vif du sujet. A présent, nous établissons la connexion entre les 2 Xbee précédemment configurés. Voici une capture d'écran de la communication :



Capture d'écran de la communication entre les 2 Xbee

Nous nous intéressons à la fenêtre *Console log*. Prenons du recul, sortons de la partie « module embarqué sur le drone » et mettons nous à la place du PC, donc de l'autre côté de la communication. Nous visualisons ici, notre module Xbee connecté au PC. C'est lui qui recevra/ transmettra des données depuis/vers le second module Xbee embarqué sur le drone.

QU'A-T-ON REÇU ?

Sur notre fenêtre s'affiche la valeur de l'altitude ainsi que les données brutes provenant de la centrale inertielle : l'accélération et la position angulaire des axes X, Y, Z. Ces données peuvent être visualisés à droite de la fenêtre en notation hexadécimale, ce sont les données de couleur rouge.

QU'A-T-ON ENVOYÉ ?

Nous pouvons lire sur la fenêtre « Rapport cyclique = 25 ». En effet, c'est la valeur du rapport cyclique du PWM qui servira à commander les régulateurs de vitesse, et donc les moteurs (cf. partie 1.3). Pour le test, on a décider d'envoyer cette valeur chaque 20ms sur une boucle infinie.

QUE TIRONS NOUS DE CE TEST ?

Notre test a permis de valider l'ensemble du bloc embarqué sur le drone. Le capteur d'altitude renvoie des valeurs correctes, la centrale inertielle renvoie des valeurs brutes qui nous semblent correctes comparé à la datasheet. De même, nous avons pu valider la réception de la valeur du rapport cyclique du PWM.

Partie 3 : Module télécommande

3.1 - Schéma fonctionnel TECHNIQUE de la télécommande

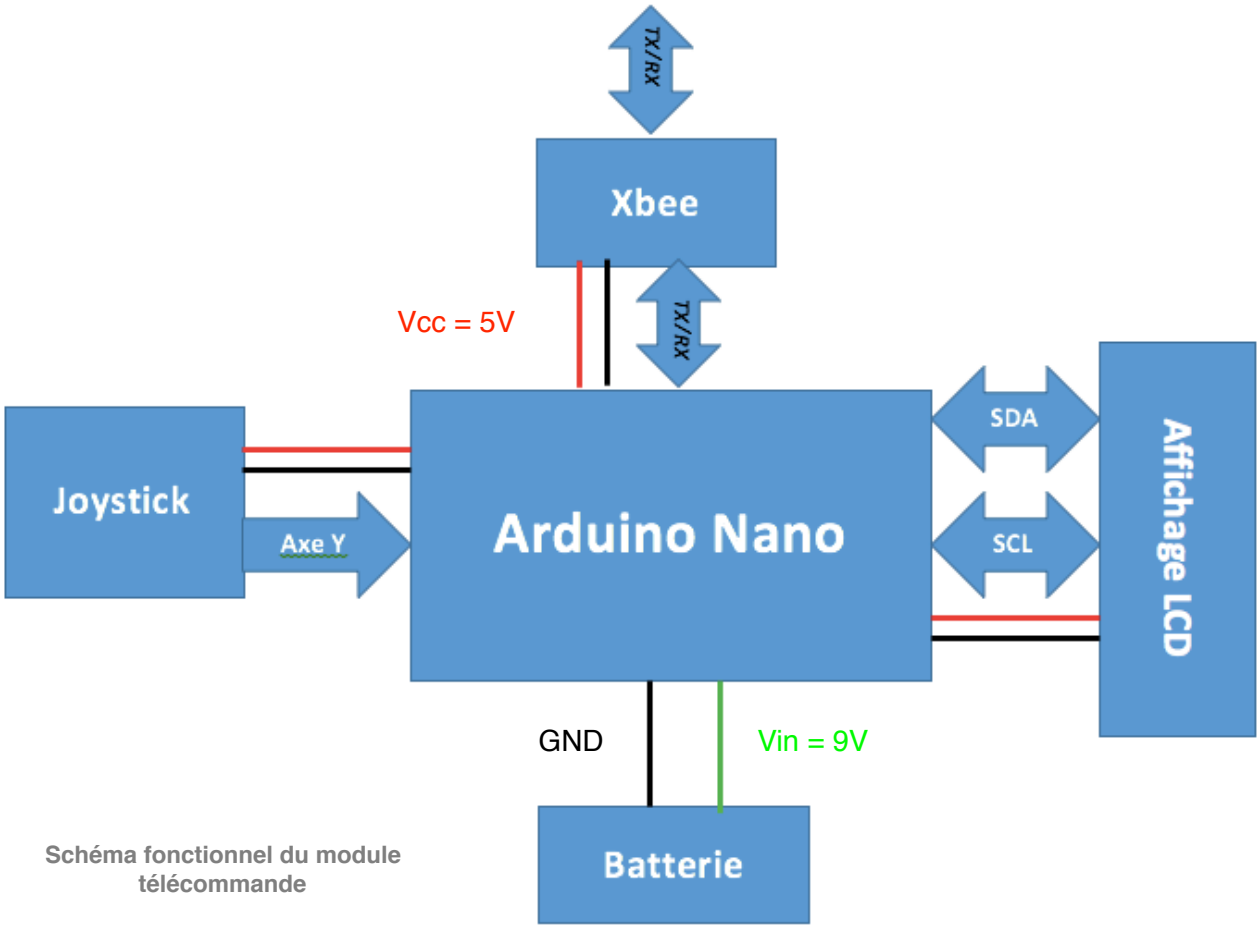


Schéma fonctionnel du module télécommande

Dans cette partie nous allons vous parler du bloc « télécommande » que nous avons conçue et qui va permettre le pilotage du drone à distance.

Elle est constituée d'un joystick qui permet le contrôle de la vitesse des moteurs, eux mêmes contrôlés par un ensemble "régulateur de vitesse + moteur" présenté précédemment. Dans cette étude, le Joystick ne permet que de contrôler la vitesse des quatre moteurs au même instant. De ce fait, nous ne pouvons contrôler que la montée et la descente de l'aéronef. Ce joystick est relié à notre Arduino NANO, il traitera les informations et les retransmettra au drone (grâce à la communication entre les deux modules Xbee connectés).

Ensuite, vient le bloc Arduino NANO qui s'occupe du traitement des données et du dialogue entre les différents blocs que l'on remarque ci-dessus. Cette transmission se fait via les ports TX-RX de la carte qui sont connectés aux mêmes ports du module Xbee dont on parlera ultérieurement. Choisie par nos soins, elle contient le programme qui permet la réception des données sur la position du drone afin de les afficher sur l'écran. Grâce à cela, les informations sont renvoyées via Xbee et les corrections sur la position peuvent être recalculées.

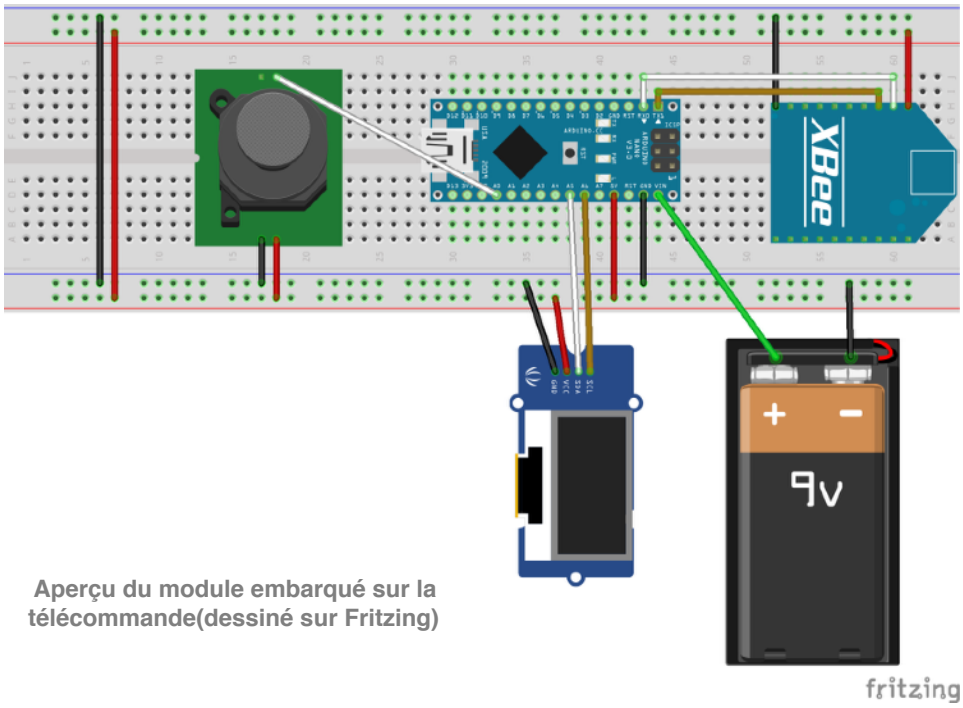
Comme exprimé précédemment, l'affichage LCD quant à lui permet de nous informer sur la position du drone (grâce à la centrale inertielle). Il est aussi utilisé pour nous fournir l'altitude exacte du drone (avec le capteur d'altitude) et la température.

Nous observons maintenant le bloc « Xbee », celui qui permet les échanges d'informations entre la partie « télécommande » et la partie « embarquée ». L'objectif est, après avoir jumelé les deux modules, de recevoir et d'envoyer les trames qui contiennent les informations essentielles au bon fonctionnement de l'aérodyne.

Le tout est alimenté via les sorties de l'Arduino NANO qui est elle-même alimentée grâce à une batterie de 9V.

Après cette brève présentation de la télécommande, nous allons pouvoir détailler expressément chacune des parties et vous montrer comment nous avons réussi, ou non, à réaliser toutes les instructions du cahier des charges.

3.2 - Arduino Nano



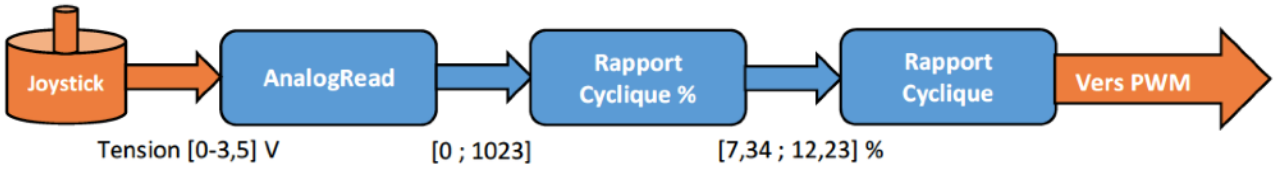
Aperçu du module embarqué sur la télécommande(dessiné sur Fritzing)

De la même façon que pour le drone, la carte est alimentée en 9V. Le programme qu'elle contient lui permet de générer une valeur du rapport cyclique à envoyer au module du drone. Le rapport cyclique sera calculé à partir de la valeur renvoyée par le Joystick relié à notre carte Arduino, et notre Xbee se chargera, en mode Transmission, d'envoyer cette valeur au module Drone.

De même, notre Xbee quand il sera en mode réception, récupèrera les données du drone concernant l'altitude et la centrale inertielle afin de les renvoyer à l'Arduino Nano. Cette dernière s'occupera d'envoyer ces informations pour un affichage sur LCD.

3.3 - Joystick

Parlons maintenant du joystick qui va permettre la gestion de la vitesse des moteurs.
Dans notre configuration, seule la montée/descente du drone nous intéresse. De ce fait, il nous suffit de contrôler la vitesse des moteurs en calculant le bon rapport cyclique à appliquer sur le régulateur de vitesse correspondant, dans le bloc drone.



Chaîne d'acquisition des données du Joystick,
transformées en rapport cyclique

Le Joystick délivre une tension comprise entre 0V et 3.5V sur une broche analogique de notre carte Arduino. La fonction *AnalogRead(),* disponible sur Arduino, nous permet de lire la tension et la convertira en un mot de 8 bits (compris entre 0 et 1023).

On a créé deux sous programmes qui s'occuperont de créer un rapport cyclique en fonction de la tension en entrée.

Rapport cyclique pourcentage

En entrée, on a le mot sur 8 bits correspondant à la position du Joystick. Nous convertissons cet intervalle en un intervalle de rapport cyclique, dont les valeurs nous sont imposées par le régulateur de vitesse. Comme nous l'avons vu dans la partie précédente, la valeur de l'état haut de notre signal PWM devra être comprise entre 1.2ms et 2.0ms. Cela correspond donc à un rapport cyclique compris entre 7.3% et 12.2%, avec la valeur de stabilité à 9.8%.

Rapport cyclique Arduino

Notre rapport cyclique étant prêt, il nous reste à l'appliquer sur une broche en sortie de notre Arduino Nano. Pour cela, nous utilisons une fonction *analogWrite(broche_sortie, valeur_rapport_cyclique)*. Seulement, cette fonction nous impose de convertir le rapport cyclique, jusqu'à lors en pourcentage, en une valeur comprise entre 0 et 255 (valeur sur 4 bits). Cette nouvelle valeur, que nous avons choisi d'appeler *Rapport_Cyclique_Arduino* sera donc envoyée par le biais du Xbee au module drone.

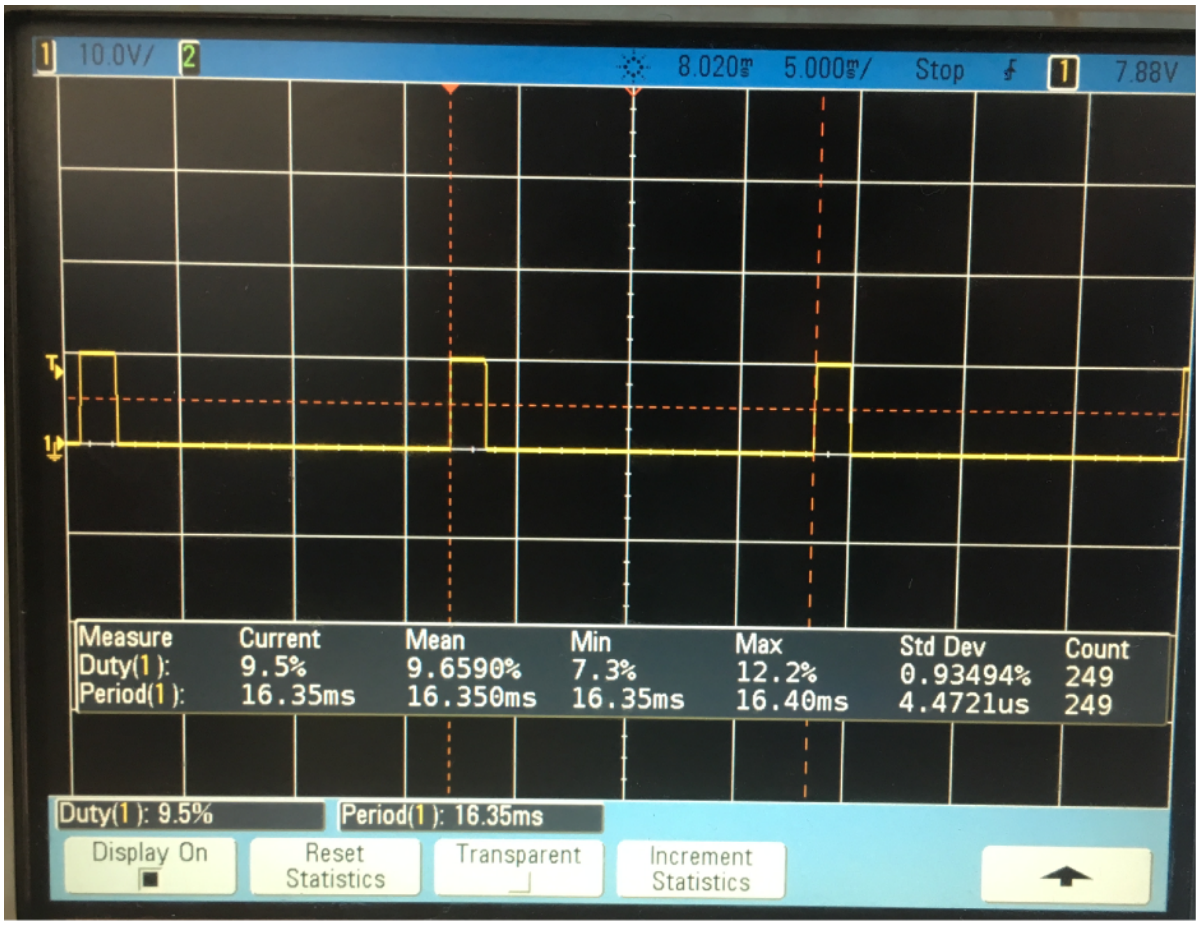
Le tableau ci dessous récapitule les valeurs de référence. Le schéma à la page 17 explique le choix de ces valeurs de référence.

Valeur de l'état haut (en ms)	Rapport cyclique en %	Rapport cyclique Arduino
1.2	7.3394%	19
1.6	9.7859%	25
2	12.2324%	32

Tableau de correspondance des valeurs
du rapport cyclique

Visualisation du signal PWM commandé par la télécommande

Nous avons décidé de tester le contrôle de la vitesse de rotation des moteurs. Comme prévu, le Joystick définira la valeur du rapport cyclique du signal PWM à appliquer sur le régulateur de vitesse. Nous avons donc visualisé ce signal PWM sur notre oscilloscope, depuis le bloc drone.



Visualisation du signal PWM appliqué en entrée du régulateur de vitesse

Sur cette figure, nous observons le signal PWM délivré à l'un des régulateurs de vitesse. Comme expliqué dans la partie 2.3 de la page 15, nous obtenons un signal PWM de période 16.35ms. Grâce à des fonctions de mesures intégrées à l'oscilloscope, nous avons pu récupérer les valeurs minimales et maximales du rapport cyclique de ce signal PWM.
Les valeurs observées correspondent approximativement aux valeurs attendues.

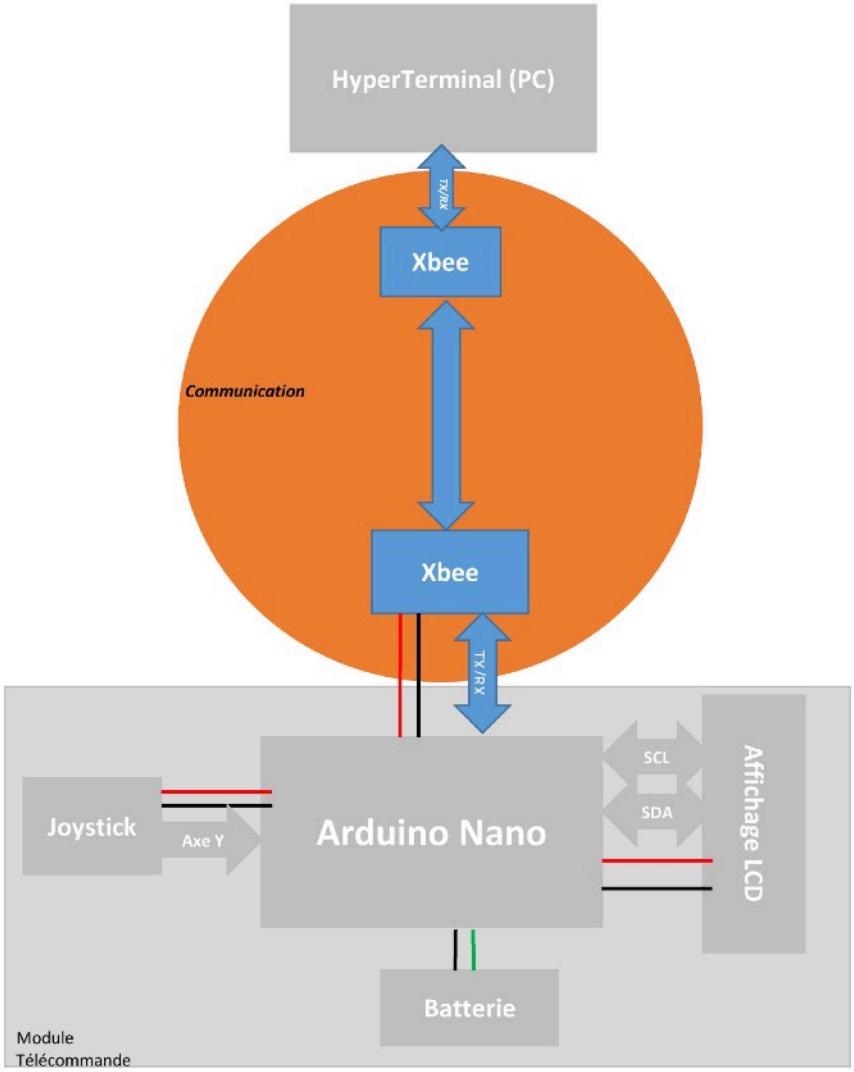
3.4 - Affichage des données sur écran LCD

Le Xbee du bloc télécommande reçoit l'altitude et les données brutes de la centrale inertielle. L'Arduino récupère donc ces informations et s'occupera de les afficher à l'utilisateur. Pour cela, nous avons décider de prendre un afficheur LCD par bus I2C.

3.5 - Communication Arduino+Xbee / Xbee+PC

Renvoie des informations sur la position du drone

Dès lors que nous étions capables de faire communiquer les deux modules, nous pouvions transmettre diverses informations qui seront importantes pour connaître la position du drone, l'altitude à laquelle il se situe, le rapport cyclique, et donc, intrinsèquement, la vitesse plus ou moins élevée des moteurs.



Préparation du Xbee TELECOMMANDE

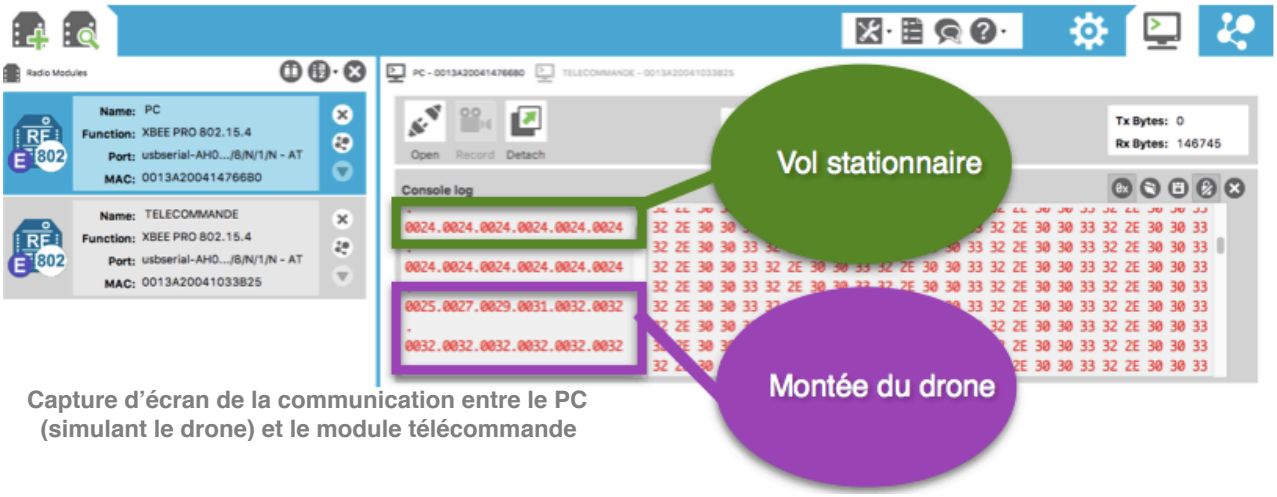
Le module télécommande, comme nous l'avons expliqué à la partie précédente, permettra de :

- Envoyer : rapport cyclique du PWM
- Recevoir : l'altitude et les données brutes de la centrale inertielle

Nous câblons notre Xbee conformément à l'explication donnée en partie 2.
Notre module est à présent équipé de son Xbee, et donc prêt à transmettre/recevoir des données.

Visualisation sur l'HyperTerminal de XCTU

A présent, on établit la connexion entre les 2 Xbee précédemment configurés. Voici une capture d'écran de la première communication entre le PC et la télécommande, vue depuis XCTU qui simule le drone.



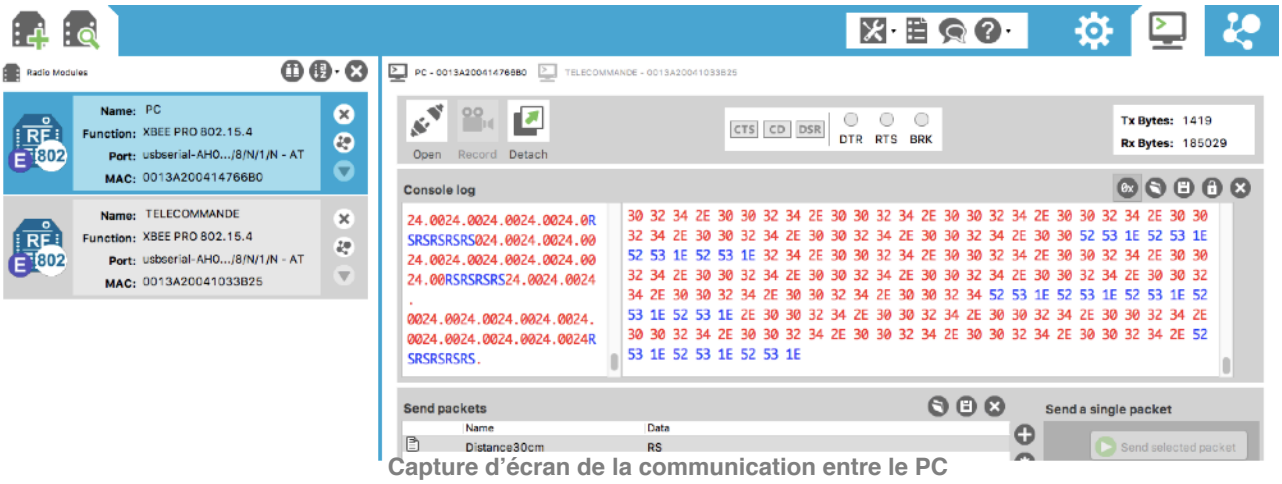
Capture d'écran de la communication entre le PC (simulant le drone) et le module télécommande

DEPUIS LA TÉLÉCOMMANDE, QU'A-T-ON ENVOYÉ ?

La télécommande envoie le rapport cyclique au module embarqué sur le drone. Sur la capture d'écran ci-dessus, la communication entre les deux Xbee nous permet d'observer deux des phases comportementales du drone.

La première, correspond à un vol stationnaire. Le rapport cyclique du signal PWM est de 24, correspondant à la valeur de stabilité du drone (cf partie 3.3).

La seconde, correspond à la phase de montée du drone. Le rapport cyclique passe de la valeur de stabilité à la valeur d'accélération maximale.



Capture d'écran de la communication entre le PC (simulant le drone) et le module télécommande

DEPUIS LA TÉLÉCOMMANDE, QU'A-T-ON ENVOYÉ ?

Sur la capture d'écran ci-dessus, les données en rouge correspondent au rapport cyclique envoyé par la télécommande.

DEPUIS LA TÉLÉCOMMANDE, QU'A-T-ON REÇU ?

Les données en bleu sont des données reçues par la télécommande. Elles correspondent à l'altitude du drone. Pour la simulation sur X-CTU, nous envoyons des paquets prédéfinis en hexadécimale que la télécommande reçoit.

QUE TIRONS NOUS DE CES DEUX TESTS ?

L'observation de cette communication nous a permis de valider le bon fonctionnement de la télécommande. En effet, nous envoyons les valeurs du rapport cyclique qui correspondent à nos attentes. De même, cela confirme la bonne réception des informations en provenance du drone.

Partie 4 : Critiques et suggestions

4.1 - Rassembler les 2 modules : problème de communication

A présent, nos deux blocs (drone et télécommande) fonctionnent quand ils sont testés séparément. En effet, le module embarqué sur le drone marchait parfaitement en communiquant avec l'HyperTerminal, de même que le module télécommande marchait en communiquant avec l'HyperTerminal.

A la fin des 50 heures de projet, le but était donc de relier ces deux modules pour qu'ils puissent communiquer ensemble via leurs Xbee respectifs. Nous nous sommes confrontés à un problème de communication à ce moment. En effet, jusqu'à présent, nous utilisions les Xbee pour envoyer quelques données qui ne seront « qu'affichés » sur l'HyperTerminal. Or, au moment de connecter les deux blocs, il nous fallait changer la programmation de nos 2 Xbee pour pouvoir faire un traitement de plusieurs données. En plus de cela, ce qui compliquait cette tâche était le fait que chacun des Xbee fonctionnait en émetteur ET récepteur.

La difficulté que l'on a eu, et par manque de temps car c'était la dernière étape, venait de la configuration des Xbee : nous aurions dû prendre un intervalle de temps donné et le diviser en deux. Durant la première partie, le Xbee embarqué sur le drone se mettrait en émetteur et le Xbee embarqué sur la télécommande en récepteur. Durant la seconde partie, on devrait intervertir les rôles. Le Xbee embarqué sur le drone deviendrait alors le récepteur et le Xbee embarqué sur la télécommande deviendrait alors l'émetteur. Nous nous heurtons donc à la limite de capacité des Xbee.

Nous pensons donc, pour une prochaine version de ce drone, à utiliser un autre mode de communication. Nous pourrons suggérer une communication par antenne, par bluetooth, par Wifi… Nous pourrions faire communiquer les 2 modules ensembles en même temps.

4.2 - Bloc télécommande : Affichage des informations sur écran LDC

Le bloc télécommande a été validé, sur le banc de test, comprenant l'Arduino Nano, le Xbee et le Joystick. Nous visualisons jusque là les informations transmises/reçues depuis la fenêtre de l'HyperTerminal. Dès le début, notre module télécommande était prévu pour embarquer un écran LCD à communication par bus I2C. Or, par manque de temps et s'étant pris la dernière séance pour raccorder l'écran, nous n'avons pas pu trouver d'écran disponible pour pouvoir l'embarquer sur le module télécommande.

Il aurait donc suffit de rajouter quelques lignes de code dans le programme de l'Arduino Nano pour afficher les informations qui nous auraient intéressé (principe de la communication série) et les rafraîchir toutes les secondes par exemple.

Nomenclature - Estimation du coût du projet

Composant	Nombre	Prix unitaire (en €)	total
Joystick	1	4.5	04.50
Arduino NANO	2	32.99	65.98
Modules Xbee (pro)	2	40.22	80.44
Socket Xbee	2	20.65	41.30
Shields Xbee	2	10.00	20.00
Structure du drone avec les 4 servomoteurs	1	149.90	149.90
Capteur d'altitude	1	10.99	10.99
Centrale inertielle	1	15.84	15.84
Platine embarquée	1	12.99	12.99
Batteries	2	19.90	39.80
Câbles (x100)	1	7.99	07.99
Afficheur LCD	1	41.04	41.04
Totaux :	17	Prix unitaire moyen : 28.87	Prix total : 490.77 €

Tableau regroupant l'ensemble des composants utilisés dans le projet ainsi que leurs prix

Les prix ont été trouvé sur le site internet de Conrad, sauf pour les composants Xbee qui ont été trouvés sur le site internet Lextronic.

Conclusion

Ces cinquante heures de projet nous ont permis de nous confronter à une tâche que nous pourrions retrouver en entreprise.

La majorité des manipulations que nous avons réalisées nous ont aider à engranger des connaissances dans un domaine que nous ne connaissions que peu. C'est principalement dans les domaines de l'électronique numérique et de la programmation que nos connaissances ont été approfondies.

Le fait d'avoir été impliqués dans toutes les tâches du projet nous a permis d'acquérir du savoir-faire en gestion et organisation de projet. En cours de projet, nous avons modifié quelques étapes qui nous ont permis de satisfaire le cahier des charges.

Même si nous avons rencontré des difficultés, le fait d'être en autonomie a accentué notre efficacité à réagir face aux problèmes.

Le fait d'avoir utilisé Arduino et Xbee, qui sont accessibles à tous, pourrait, dans un futur proche, nous inciter à les manipuler d'avantage.

Bibliographie

[1] C. Dupaty, « Mini drones Quadra et Hexa comment fonctionnent-ils, » Internet : http://genelaix.free.fr/IMG/pdf/drone_comment_fonctionne_un_drone.pdf, [consulté le 20/02/2017].

[2] H. Julien, « Un réseau sans-fil avec des Xbee, » Internet : http://www.pobot.org/Un-reseau-sans-fil-avec-des-XBee.html?lang=fr, [consulté le 25/02/2017].

[3] C. Tavernier, « Bus I2C, » Internet : http://www.tavernier-c.com/busi2c.htm [consulté le 12/04/2017].

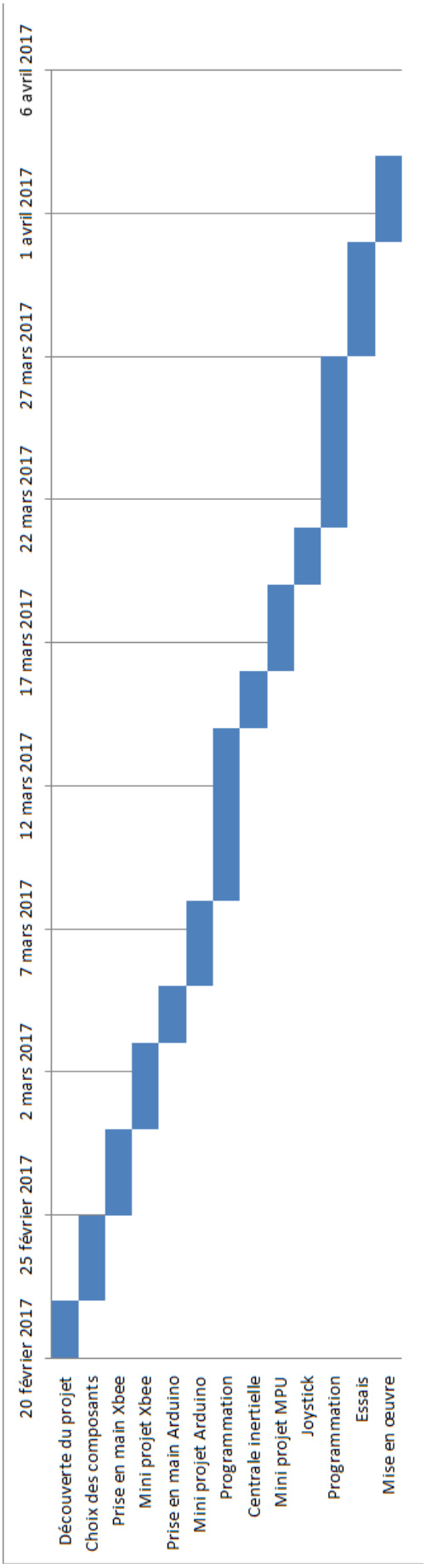
[4] Gotronic, « HC-SR04 - Module de détection aux ultrasons Utilisation avec Picaxe, » Internet : http://www.gotronic.fr/pj2-hc-sr04-utilisation-avec-picaxe-1343.pdf, [consulté le 17/03/2017].

[5] G. Aubret, « (Re)Decouvrez l'Arduino, » Internet : http://simple-duino.com/decouvrez-larduino/, [consulté le 20/02/2017].

T. Perisse, « Réalisation et contrôle d'un drone, » Internet : http://thierryperisse.free.fr/?page_id=228, [consulté le 20/04/2017].

ANNEXES

1 - Diagramme de Gantt



2 - Comparatif des Arduino

Carte Arduino	Microcontrôleur	Entrées/Sorties numériques	Entrées digitales	Tension de fonctionnement	Spécificité	Dimension	Prix (€)
Arduino UNO	AtMega328p	14/6	6	5 v		68x53 mm	20
Arduino MINI	AtMega328p	14/6	8	5 v		30x18 mm	15
Arduino NANO	AtMega328p	14/6	8	5 v		45x18 mm	22
Arduino MEGA 2560	AtMega2560	54/15	16	5 v		101x53 mm	40
Arduino 101	Intel Curie	14/4	6	7-12v	Centrale inertielle & bluetooth	68,6x53,4 mm	30
Arduino Lilypad	AtMega328p	14/6	6	5v			15
Arduino Leonardo	AtMega32u4	14/6	12	5 v		68x53 mm	20
Arduino DUE	AT91SAM3X8E	54/12	12	3,3 v		101x53 mm	40
Arduino MICRO	AtMega32u4	20/7	12	5 v		48x18 mm	22
Arduino YUN	AtMega32u4	20/7	12	5 v	WiFI + Linux		78
Arduino MKR1000	SAMD21 Cortex	8/12	7	3,3 v			50
Arduino Mega ADK	AtMega2560	54/15	16	5v		101x53 mm	50
Arduino Esplora	AtMega32u4						53

Comparatif des différentes cartes Arduino [5]

3 - Datasheet

Datasheet du MPU6050

InvenSense, « MPU-6000 and MPU-6050 Product Specification Revision 3.4, » Internet : https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf, [consulté le 27/03/2017].

Datasheet du HC_SR04

Elec Freaks, « Ultrasonic Ranging Module HC_SR04, » Internet : https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf, [consulté le 11/04/2017].

Datasheet du Xbee Pro

Digi International, « Xbee / Xbee-PRO, RF Modules, » Internet : https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf, [consulté le 11/04/2017].

Documentation de l'Arduino Nano

Arduino, « Arduino Nano, » Internet : https://www.arduino.cc/en/Main/ArduinoBoardNano, [consulté le 10/04/2017].

4 - Programme embarqué dans la télécommande

```
/* Ce programme sera embarqué dans la télécommande. Il lira la valeur du joystick
 * pour en déduire un rapport cyclique à envoyer aux sorties PWM de l'arduino du drone
 * Il recevra les données du capteur d'altitude et se chargera de les afficher
 */
#include <math.h>
#include <SoftwareSerial.h>

  const int PWM1 = 6;    //Utilisation de la sortie 6 car elle peut générer des signaux PWM
/**** Déclaration des variables et constantes *******/
const float Duree_Max_Etat_Haut = 2.0;                                  //en millisecondes e-3
const float Duree_Min_Etat_Haut = 1.2;                                  //en millisecondes e-3
const float Frequence_PWM = 0.061;                                      //en KiloHertz e+3
const float Periode_ms_PWM = 1.0/Frequence_PWM;                         //en millisecondes e-3
const float Rapport_Cyclique_Max = Duree_Max_Etat_Haut/Periode_ms_PWM;
const float Rapport_Cyclique_Min = Duree_Min_Etat_Haut/Periode_ms_PWM;
float valeur_num_joystick, alpha_pourcent, alpha_arduino;

/**** Prototypage des fonctions ************/
float Rapport_Cyclique_Pourcent(int valeur_num_potentio);
int Rapport_Cyclique_Arduino(float alpha_pourcent);
void etat_drone (int alpha_pourcent);


/********** PROGRAMME PRINCIPAL *************/

void setup() {
  Serial.begin(9600);
  TCCR0B = TCCR0B & B11111000 | B00000101; //Division de la fréquence du PWM par 1024, on a une fréquence de 61Hz
}



void loop() {

  //Chaine de traitement de l'information reçue du joystick
  valeur_num_joystick = analogRead(A0);     //Lecture de l'entrée analogique A0 : Potentiomètre [0-5]V --> [0;1023]
  alpha_pourcent = Rapport_Cyclique_Pourcent(valeur_num_joystick);
  alpha_arduino = Rapport_Cyclique_Arduino(alpha_pourcent);

    analogWrite(PWM1, alpha_arduino);  //Ecriture de la valeur du rapport cyclique sur la sortie PWM1

  //Serial.print("TELECOMMANDE - Rapport cyclique en pourcentage : ");
  Serial.print(alpha_arduino);
  //Serial.println(" %");


   //etat_drone ( alpha_pourcent);

}
```

```
/***************** SOUS PROGRAMMES ******************/

/* Calcul du rapport cyclique en fonction de la tension délivrée par le potentiomètre */
float Rapport_Cyclique_Pourcent(int valeur_num_joystick)
  {
    float alpha_pourcent;
    float pente, ordonnee_origine;

    pente = (Rapport_Cyclique_Max - Rapport_Cyclique_Min)/409 ;
    ordonnee_origine = Rapport_Cyclique_Max - pente*717;
    alpha_pourcent = (pente * valeur_num_joystick + ordonnee_origine)*100;

    return alpha_pourcent;
  }


  void etat_drone (int alpha_pourcent) {


    if(alpha_pourcent<=6.0)
       Serial.print("En dehors de la plage de valeurs");

    if(alpha_pourcent>6.0 && alpha_pourcent<9.0)
       Serial.print("Sens : DESCENTE ");

    if(alpha_pourcent>=9.0 && alpha_pourcent<=10.0)
       Serial.print("Sens : STABILISATION ");

    if(alpha_pourcent>10.0 && alpha_pourcent<13.0)
        Serial.print("Sens : MONTEE ");

    if(alpha_pourcent>=13.0)
        Serial.print("En dehors de la plage de valeurs ");

  }
```

5 - Programme embarqué sur le drone

```
/* Ce programme sert à récupérer les données du capteur d'altitude et les envoyer
 * par communication série à l'afficheur
 *  Il reçoit aussi par communication série le rapport cyclique
 *  envoyé par la télécommande, pour commander les 4 moteurs
 */

  #include <SoftwareSerial.h>
  #include<Wire.h>
  #define TrigPin 5
  #define EchoPin 3


  SoftwareSerial xbee(2,3);
  const int PWM1 = 6;    //Utilisation de la sortie 6 car elle peut générer des signaux PWM
  const int MPU_addr=0x68;  // I2C address of the MPU-6050
  int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

/* Prototype des fonctions */
  void generate_trig();

/******** CONFIGURATION *********/
void setup() {
  TCCR0B = TCCR0B & B11111000 | B00000101; //Division de la fréquence du PWM par 1024, on a une fréquence de 61Hz

  xbee.begin(9600);
  Serial.begin(9600);
  pinMode (PWM1, OUTPUT);                  //Configuration de la sortie PWM
  pinMode (TrigPin, OUTPUT);                  //Configuration du pin TrigPin en sortie
  pinMode (EchoPin, INPUT);                  //Configuration du pin EchoPin en entrée

  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);

  Serial.print("Configuration terminée");
}


/******* PROGRAMME PRINCIPAL *********/
void loop() {

  long duree=0, distance=0;
  int data_in;

 // Commande des moteurs
  if (Serial.available()>0) {
    data_in = Serial.read();
    Serial.print("Rapport cyclique recu : ");
    Serial.println(data_in);
  }

    analogWrite(PWM1, data_in);  //Ecriture de la valeur du rapport cyclique sur la sortie PWM1

  //Capteur altitude
  //NB : Signal pulse qui est envoyé à l'entrée TRIG. Durée état haut >10µs
  // NB : Récupération des données depuis la sortie ECHO du capteur
  generate_trig();
  duree = pulseIn(EchoPin, HIGH);        //Récupération de la durée de l'état haut (HIGH) du signal reçu sur EchoPin. Durée de l'état haut stockée dans la variable duree en µs
  distance = duree/58;    //Formule fournie par la Datasheet. Duree en µs, distance en cm
  if(distance<=3 || distance>=300)
    Serial.println("Mesure impossible");
  else{
```

```
  // Serial.print("Altitude : ");
  // Serial.print(distance);
  // Serial.println(" cm");
  }

   //MPU
   Wire.beginTransmission(MPU_addr);
   Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
   Wire.endTransmission(false);
   Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
   AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)       AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
   AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
   Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
   GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
   GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
   GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
   Serial.println("Donnees MPU");
   Serial.print("AcX = "); Serial.print(AcX);
   Serial.print(" | AcY = "); Serial.print(AcY);
   Serial.print(" | AcZ = "); Serial.print(AcZ);
   Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);  //equation for temperature in degrees C from datasheet
   Serial.print(" | GyX = "); Serial.print(GyX);
   Serial.print(" | GyY = "); Serial.print(GyY);
   Serial.println(" | GyZ = "); Serial.println(GyZ);
   Serial.println("  ");
}
```

```
/****** GENERATION DE SIGNAUX PULSE *********/
void generate_trig() {
 digitalWrite(TrigPin, HIGH);      //Génération d'un état haut sur la sortie TrigPin
  delayMicroseconds(30);            //Durée état haut : 20µs
  digitalWrite(TrigPin, LOW);       //Génération d'un état bas sur la sortie TrigPin
  delayMicroseconds(50);            //Durée état bas : 30µs
}
```