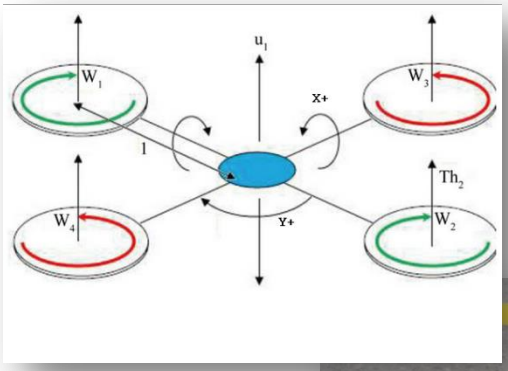
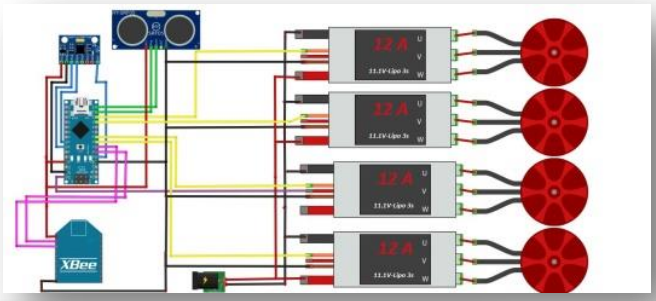
ESETDrone

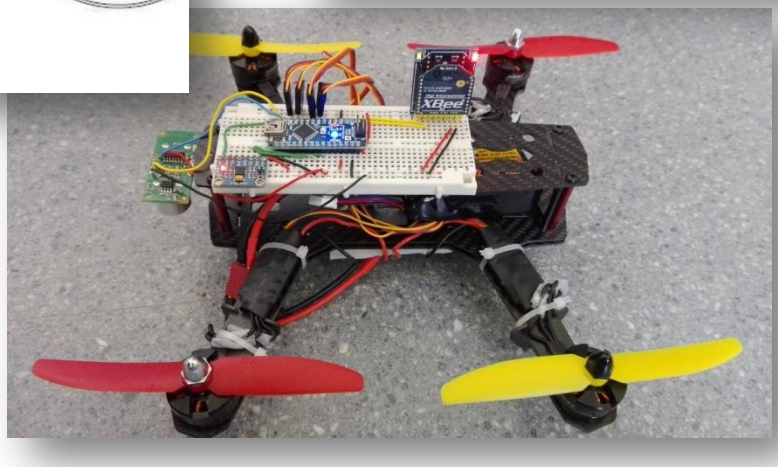
Rapport de Projet

EL KHAOUDI Samir , OUARGA Mohammed , LI Jiajun

5/23/2016







L'objectif de notre modeste travail et de concevoir un Drone tout entier en se basant sur un microcontrôleur de la famille ATmel ATmega assemblé dans la carte Arduino , le Drone sera commandé à distance via le protocole Zigbee

SOMMAIRE

REMERCIEMENT **3**

INTRODUCTION : **5**

MATERIEL ET TESTS **6**

I. ARDUINO 6
- a. *Introduction* *6*
- b. *Spécifications techniques* *6*
- c. *Fonctionnalités* *6*
- d. *Alimentation* *7*
- e. *Entrés – Sorties* *7*

II. LE CAPTEUR ULTRASONIQUE (HC-SR04) 8
- a. *Le télémètre ultrason HC-SR04* *8*
- b. *L'effet piézoélectrique dans le capteur ultrasonique :* *9*
- c. *La carte et le circuit électronique du HC-SR04 :* *9*
- d. *Paramètres techniques et principe de fonctionnement:* *9*
- e. *Résultats et testes* *10*

III. COMMANDE SANS FIL 10
- a. *Protocole Zigbee* *10*
- b. *Xbee* *11*
- c. *Caractéristiques* *12*
- d. *Communication avec l'ordinateur* *12*
- e. *Communication série* *12*
- f. *Configuration graphique* *13*
- g. *Configuration par uC (ici Arduino)* *14*

IV. GYROSCOPE, ACCELEROMETRE 15
- a. *Généralités* *16*
- b. *Principe de fonctionnement* *16*
- c. *Protocole I2C* *18*
- d. *Correction des angles par un PID* *19*
- e. *Testes et résultats* *20*

V. LE VARIATEUR DE VITESSE(ESC) 21
- **a.** *Définition :* *21*
- b. *Caractéristiques du variateur :* *21*

V. MOTEUR BRUSHLESS « SANS BALAIS » : 22
- a. *Petit rappel : Comment fonctionne un moteur électrique ?* *22*
- b. *Comment fonctionne un moteur brushless ?* *22*
- c. *Caractéristiques importantes des moteurs brushless sont en général :* *23*
- d. *Quelle puissance consomme et fournit le moteur ?* *24*
- e. *Les hélices :* *25*

ASSEMBLAGE FINAL **27**

A. UN DRONE : COMMENT ÇA VOLE ? 27
B. BILAN ENERGETIQUE : 28
C. SCHEMA DU DRONE ET CONTROLE 28
D. LES PROBLEMES RENCONTRES 31

Remerciement

Nous tenons à remercier dans un premier temps, toute l'équipe pédagogique de l'université Paul Sabatier et les intervenants professionnels responsables de la formation ESET (Electroniques des Systèmes Embarqués & Télécommunication).

Avant d'entamer ce rapport, nous profitons de l'occasion pour remercier et exprimer notre profonde reconnaissance à monsieur T.PERISSE notre encadrant de projet de fin semestre qui n'a pas cessé de nous encourager pendant la durée du projet, ainsi pour sa générosité en matière de formation et d'encadrement et sa disponibilité.

Nous le remercions également pour l'aide et les conseils concernant les missions évoquées dans ce rapport, qu'il nous a apporté lors des différents suivis, et la confiance qu'il nous a témoigné.

Nos sincères remerciements et gratitude s'adressent également aux membres du département EEA de nous avoir permis de travailler dans des bonnes conditions.

Sans oublier monsieur Frank pour toute l'aide technique qu'il nous a apporté.

Nous tenons également à remercier les membres de jury qui nous ont fait l'honneur de bien vouloir juger notre travail.

Finalement, merci à toute personne qui nous a aidés pour la réalisation de ce travail.

Merci…

Introduction :

Un drone : est un aéronef télécommandé ou autonome, c'est-à-dire qu'il ne nécessite pas de pilote. Il peut transporter diverses matériels (caméras, armes, sonde...) afin de réaliser des missions telles que la surveillance, l'exploration, le combat ou la cartographie.

Histoire de drone :

Le premier projet d'avion cible sans pilote radiocommandé date de 1916. Cependant, à cette date, il n'était réservé qu'à un usage militaire.

Le lancement du projet de drones en France remonte vers la fin de la 1ère Guerre Mondiale. En effet, George Clémenceau, alors président de la commission sénatoriale de l'armée, charge une équipe de la conception d'un aéronef pouvant être dirigé sans pilote dans le véhicule. Le projet aboutit en 1923, grâce au général Max Boucher et au scientifique Maurice Percheron. La nomination de drone n'est attribuée à ces avions qu'en 1930 au Royaume-Uni, qui signifiait à ces début "avion-cible"; mais un drone est plus scientifiquement appelé UAV ("Unmanned Aerial Vehicule" qui veut dire "Véhicule Aérien sans Pilote"). Leur première utilisation sur le terrain date de la Guerre du Vietnam vers 1960. Son usage se développa surtout durant la Guerre Froide, au cours de laquelle les Américains utilisèrent les drones afin d'espionner les soviétiques.

Détail du projet :

Avec ce projet, il est proposé au candidat de concevoir une liaison sans fil entre un organe de commande et un drone placé à distance sur lequel seront disposés plusieurs types de capteurs (capteur, caméra embarquée, etc...). Cette liaison devra être bidirectionnelle et à faible consommation, d'où le choix de la technologie Zigbee fonctionnant dans la bande des 2,4 GHz.

Dans un premier temps, le candidat aura en charge la mise en œuvre de modules Zigbee. Il sera amené à développer des routines de commande et de test en langage C. Il aura également en charge la conception de cartes d'interfaçage des modules avec le reste du système.

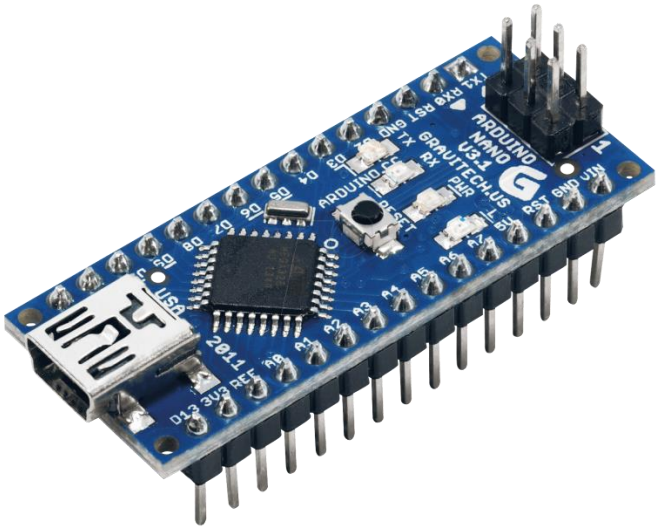
Dans un second temps, des tests grandeur nature sont prévus pour, d'une part, caractériser les modules (vérifier les puissances d'émission, sensibilité, consommation, etc...) et vérifier l'impact de différents paramétrages sur la portée, le débit et la robustesse de la transmission. La liaison bidirectionnelle sera également évaluée. Enfin, une dernière partie concernera la mise en place d'une gestion multi-nœud, caractéristique propre à la technologie Zigbee et qui offre de nombreuses perspectives. Par exemple, il serait possible d'interroger le même drone depuis différents endroits.

Matériel et tests

I. Arduino

a. Introduction

L'Arduino Nano est essentiellement un Arduino Uno conçu pour une utilisation directe sur breadboard. Il embarque son propre contrôleur USB permettant une communication série très facile.
La disposition de ses pins correspond à celui d'un Arduino Pro Micro (avec l'USB à l'opposé et quelques pins en plus, intelligemment placées de ce côté) ou d'un Pro Mini



Le Pro Mini, le Nano v3.1 et le Pro Micro : les pinouts sont compatibles.

b. Spécifications techniques

Micro-contrôleur	**Atmel ATmega328**
Tension de fonctionnement (niveau logique)	**5 V**
Tension d'alimentation (recommandée)	**7-12 V**
Tension d'alimentation (maximum)	**6-20 V**
E/S digitales	**14 (dont 6 peuvent fournir une sortie PWM, notées par un trait blanc)**
E/S analogiques	**8**
Courant disponible par pin E/S	**40 mA**
Mémoire flash	**32 KB (dont 2KB utilisés par le bootloader)**
SRAM	**2 KB**
EEPROM	**1 KB**
Vitesse d'horloge	**16 MHz**
Dimensions	**44mm x 18mm**

c. Fonctionnalités

- Reset automatique pendant l'upload
- Led bleue Power OK
- Leds verte (Tx), rouge (Rx) et orange (L)
- Alimentation à sélection automatique
- Connecteur USB mini-B de programmation et communication série
- Connecteur ICSP pour programmation directe
- Espacement des pins au format DIP (compatible breadboard)
- Bouton de reset manuel

d. Alimentation

Le Nano peut être alimenté par le PC via le connecteur USB mini-B, de 7 à 12V via la pin 30 (Vin) ou via une alimentation 5V régulée sur la pin 27 (5v). Le Nano sélectionne automatiquement son alimentation sur la source au voltage le plus élevé.

La puce FTDI FT232RL n'est alimentée que si le Nano est alimenté par le connecteur USB. La sortie 3.3v ne sera donc disponible que dans ce mode d'alimentation.

e. Entrés – Sorties

Les E/S digitales du Nano peuvent être utilisées en entrée ou en sortie avec un niveau logique de 5v. Chaque pin peut fournir 40mA au maximum et comporte une résistance de pull-up interne (désactivée par défaut) de 20-50 KOhms.
Certaines pins peuvent avoir des fonctions spéciales :

- Série : 0 (Rx) et 1 (Tx)
- Interruptions externes : 2 et 3. Ces pins peuvent être configurées pour déclencher une interruption sur n'importe quel changement d'état
- PWM : 3, 5, 6, 9, 10 et 11. Ces pins produisent une sortie PWM sur 8-bit
- SPI : 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Ces pins peuvent servir pour la communication avec d'autre périphériques SPI.
- LED : 13. C'est une LED sans fonction spéciale, connectée à la sortie digitale 13.

Le Nano a 8 entrées analogiques, chacune ayant une résolution de 10 bits (1024 valeurs). Les pins analogiques 6 et 7 sont les seules qui ne peuvent pas être reconfigurées en pins digitales.
Certaines pins ont également des fonctions spéciales :

- I^2C : 4 (SDA) et 5 (SCL). Supporte la communication I^2C (TWI) avec des composants externes, en utilisant la librairie Wire.

Il existe quelques autres pins spéciales sur le Nano :

- AREF : Voltage de référence pour les entrées analogiques.
- RESET : passez cette ligne à LOW pour redémarrer le micro-contrôleur. Habituellement utilisé pour rajouter une fonction ou un bouton de reset directement sur les shields/modules.

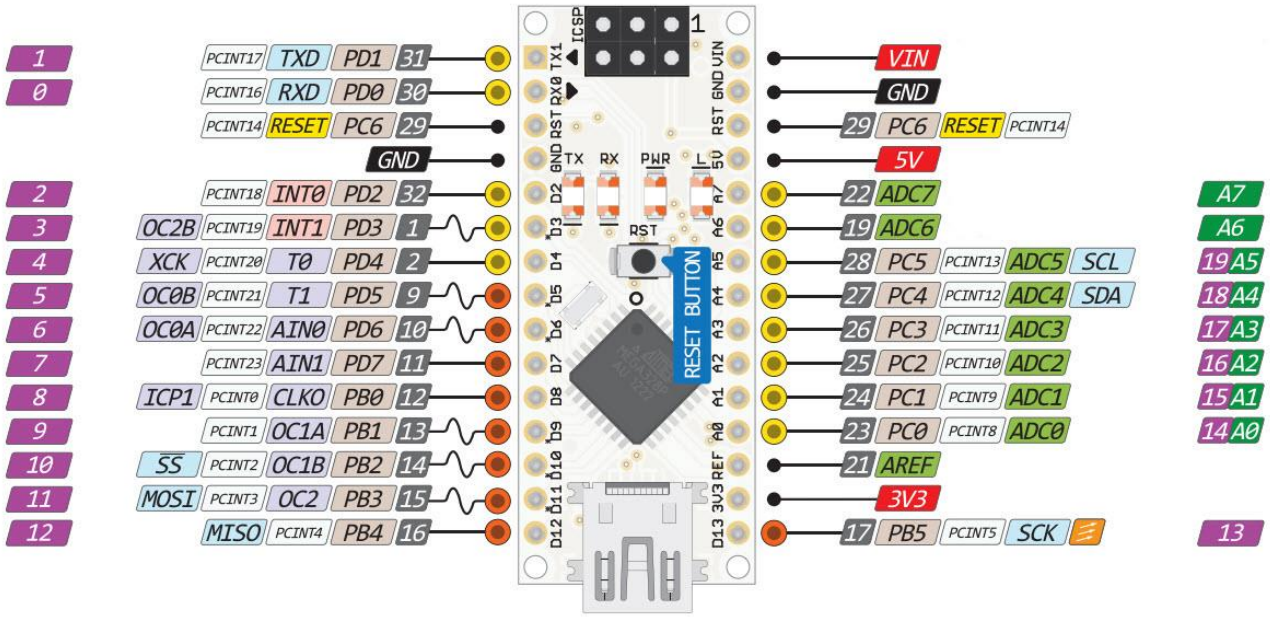


Figure 1 les pins de la carte Nano

II. Le capteur ultrasonique (HC-SR04)

Un capteur est un dispositif capable de transformer une grandeur physique (telle que la température, la pression, la lumière, la distance… etc.) en une autre grandeur physique manipulable (charge, tension, Courant ou impédance).

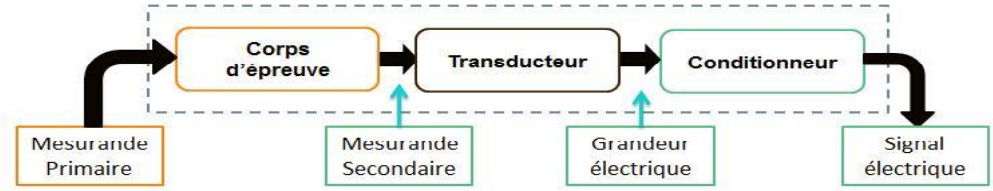


Figure 2 : Architecture d'un capteur

On peut d'ailleurs citer des exemples : un microphone est un capteur qui permet de transformer une onde sonore en un signal électrique, Une photorésistance permet de transformer un signal lumineux en une résistance variable selon son intensité… etc.

Les capteurs ultrasoniques sont les plus utilisés en robotique. La raison c'est qu'ils peuvent fournir une information très importante, à savoir la distance les séparant des obstacles autour d'eux, ici on l'utilise pour notre Drone pour mesurer l'altitude (d'ordre de quelques mètres pour le test), on peut le remplacer par d'autres capteurs l'altitude à savoir les capteurs de pression qui fournissent des résultats de l'ordre des kilomètres.

Son principe de fonctionnement repose comme son nom l'indique sur l'utilisation des ultrasons :Ce sont des ondes acoustiques dont la fréquence est trop élevée pour être audible par l'être humain.

Donc le fonctionnement est basé sur la génération des ondes ultrasoniques et leurs détections quand elles sont réfléchies par l'objet, le transport des ondes est assuré par la matière (figure 2).

Dans notre projet on a utilisé le capteur ultrason HC-SR04 dont le transport des ondes est assuré par l'air.

a. Le télémètre ultrason HC-SR04

est l'exemple classique du capteur renvoyant un créneau codant l'information. En effet, ce dernier mesure ce que l'on appelle un temps de vol.

Le HC-RS04 est un télémètre ultrasonique (figure 5) qui mesure la distance, il compte le temps que met une onde pour faire un aller - retour (Figure3). Un chronomètre est déclenché lors du départ de l'onde et est arrêté lorsque l'on détecte le retour de l'onde (une fois que celle-ci a "rebondi" sur un obstacle). Puisque l'on connait la vitesse de propagation (V) de l'onde dans l'air,on peut déterminer la distance (d) Nous séparant de l'objet

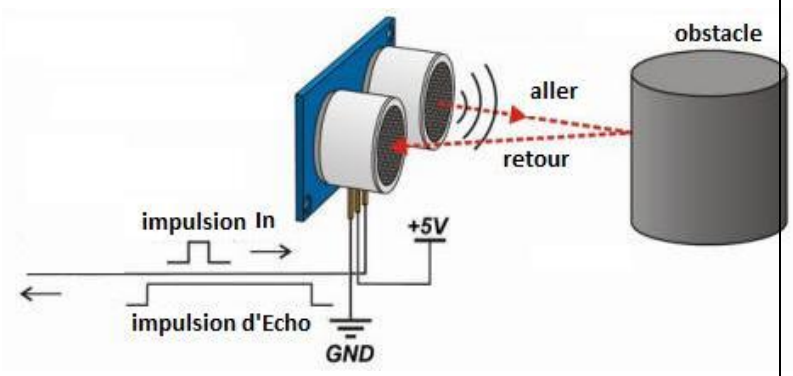


Figure 3

Le temps mesuré correspond à l'aller et au retour de l'onde, on a donc deux fois la distance. D'où la division par 2

HC-SR04 se compose de deux cylindres l'un des deux sur La platine envoie des ultrasons c'est l'émetteur (T), l'autre récupère ceux qui reviennent suite à la collision avec un objet c'est le récepteur (R), d'un oscillateur a base de quartz de vitesse 4Mhz et quatre broches de connexions.

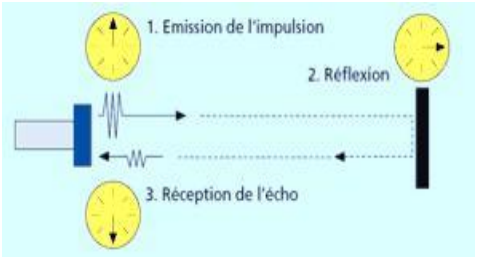


Figure 4

Les broches de connexions sont :

1. VCC ;
2. Trig: input ;
3. Echo : output ;
4. GND.

b. L'effet piézoélectrique dans le capteur ultrasonique :

La piézoélectricité est la propriété possédée par certains corps qui se polarisent électriquement sous l'action d'une contrainte mécanique. Inversement, ces corps peuvent se déformer sous l'action d'un champ électrique. Ce phénomène a été découvert au XIXème siècle par les frères Pierre et Jacques Curie

En appliquant un courant alternatif sur un cristal piézoélectrique, le cristal se comprime et se décomprime alternativement et émet un son dont la fréquence dépend des caractéristiques du cristal. Le même élément est utilisé pour transformer en courant électrique les ultrasons qui reviennent vers la sonde après avoir été réfléchis.

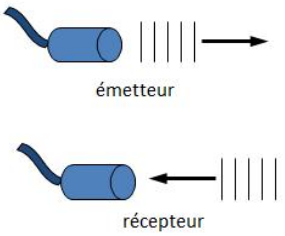


Figure 5 : Effet piézoélectrique

La sonde n'émet donc pas des ultrasons en continu, mais en salve. Pendant le reste de temps, la sonde est "à l'écoute" pour capter les ultrasons. La durée des salves est très courte, de l'ordre de quelques microsecondes, et correspond à l'émission de 3 cycles environ en moyenne. La durée de la période d'attente est plus longue, de l'ordre de la milliseconde. La fréquence de répétition du cycle est donc de l'ordre du kHz, ce qui donne l'impression d'une imagerie en temps réel.

c. La carte et le circuit électronique du HC-SR04 :

Le circuit comporte deux transducteurs un pour l'émission et l'autre pour la réception. Pour transmettre des impulsions ultrasonores d'une tension relativement élevée est nécessaire, c'est le rôle du MAX232 utilisé pour produire +/- 10V à partir de 5V. Le transducteur est connecté entre deux sorties de sorte qu'il est en fait alimenté à 20V. Le récepteur et l'émetteur sont contrôlés par un microcontrôleur cadencé à 4MHz. Le côté récepteur utilise le circuit intégré LM324 qui contient quatre amplificateurs opérationnels.

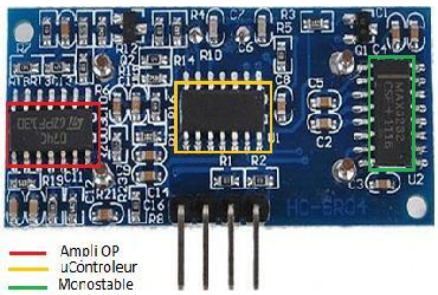


Figure 6 : Carte électronique du HC-SR04

Les principales caractéristiques techniques du HC-SR04 sont :

- Tension de fonctionnement : 5V (DC) ;
- Consommation de courant : moins de 2 mA ;
- Signal de sortie : signal électrique 5V niveau haut, 0V niveau bas ;
- Angle de visée du détecteur : moins de 15 degrés ;
- Fréquence : 40khz ;
- Portée : de 2 cm à 4m ;
- Haute précision: jusqu'à 0.3 cm ;
- Dimensions: 45 x 21 x 18 mm ;
- Calcul: distance (cm) = Vitesse / tepmps.

d. Paramètres techniques et principe de fonctionnement:

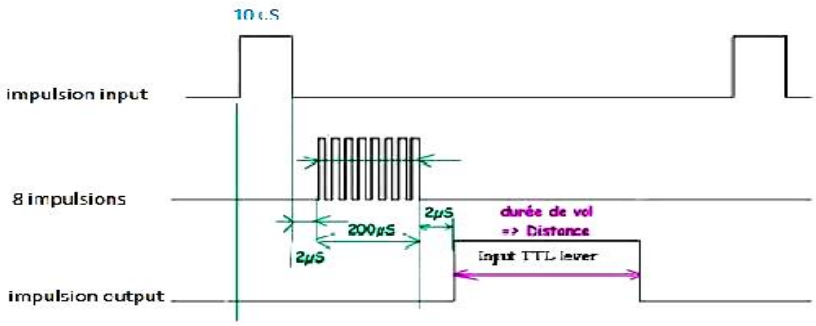


Figure 7 : principe de fonctionnement du HC-SR04

:

1. Le module émet une onde sonore composée d'une série de 8 impulsions à 40 kHz ;
2. En utilisant la broche trig, on envoie un signal pour activer le déclenchement de l'impulsion sonore ;
3. La sortie passe au niveau haut durant toute la période où l'onde voyage vers l'objet et revient après avoir été réfléchit par ce dernier ;
4. La distance est évaluée à partir du temps que la sortie est reste au niveau haut.

e. Résultats et testes

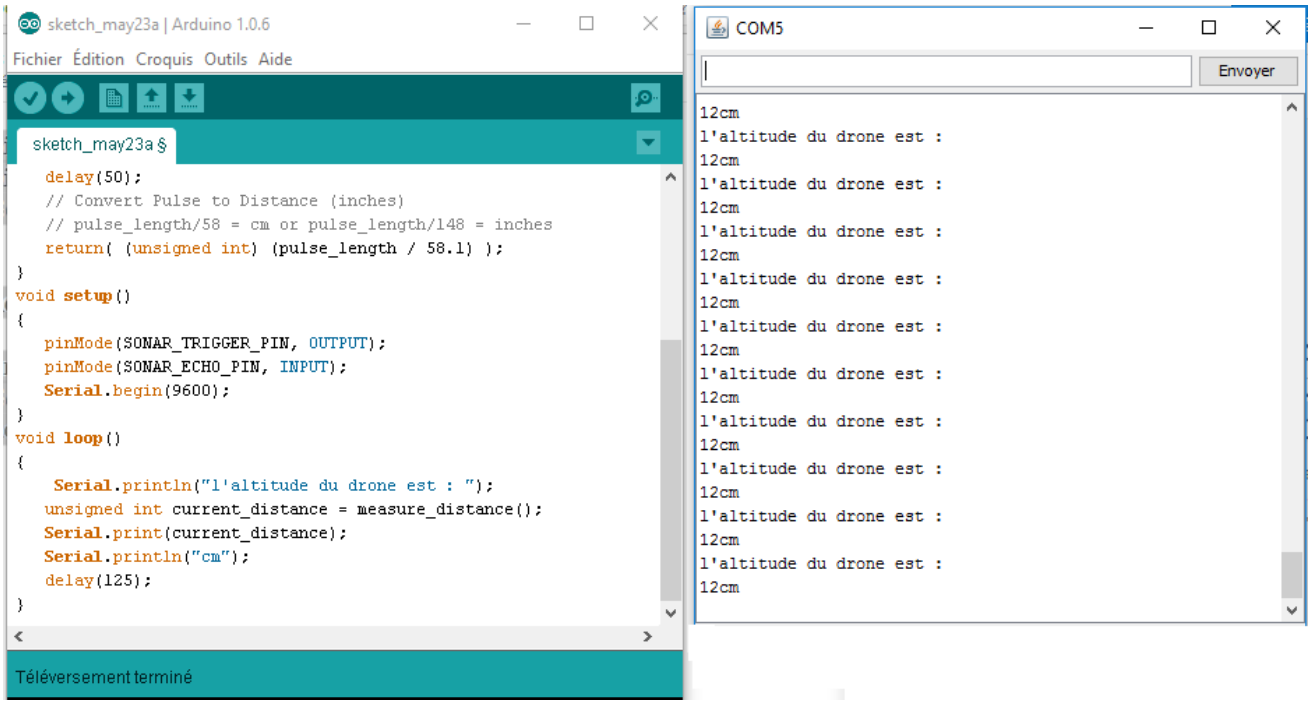


Figure 8 mesure d'altitude

III. Commande sans fil

a. Protocole Zigbee

ZigBee est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur la norme IEEE 802.15.4 pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPAN).

Ratifiées le 14 décembre 2004, les spécifications de ZigBee 1.0 sont disponibles auprès des membres de la communauté industrielle ZigBee Alliance.

Cette technologie a pour but la communication de courte distance telle que le propose déjà la technologie Bluetooth, tout en étant moins chère et plus simple. À titre d'exemple, les nœuds ZigBee classiques nécessitent environ 10% [réf. nécessaire] du code nécessaire à la mise en œuvre de nœuds Bluetooth ou de réseaux sans fil, et les nœuds ZigBee les plus élémentaires peuvent ainsi descendre jusqu'à 2%. [réf. nécessaire]

En 2006, une estimation du coût unitaire pour un nœud ZigBee a révélé un prix de 1.10$ par unité dans le cadre d'une production en très grand nombre. Il faut ajouter le prix du microcontrôleur qui commande le circuit, ce qui augmente légèrement le prix. À titre de comparaison, la technologie Bluetooth, lancée en 1998, annonçait à cette époque un prix unitaire de production compris entre 4 et 6 $,

La trame Zigbee :

Les topologies de réseaux possibles ne sont pas les mêmes. Avec la série 1, l'architecture est simple : point à point (pair) ou multipoint (star). La série 2 permet en plus de créer des réseaux plus complexes : maillés (mesh) ou en "arbre" (cluster tree).

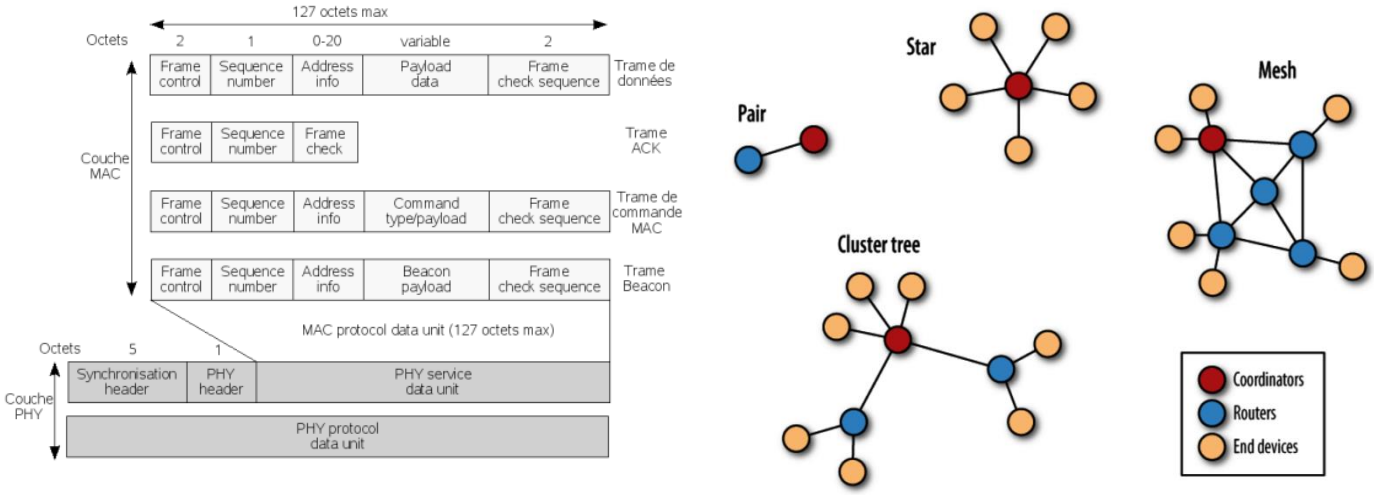


Figure 9 Trame Zigbee et les topologies possibles

b. Xbee

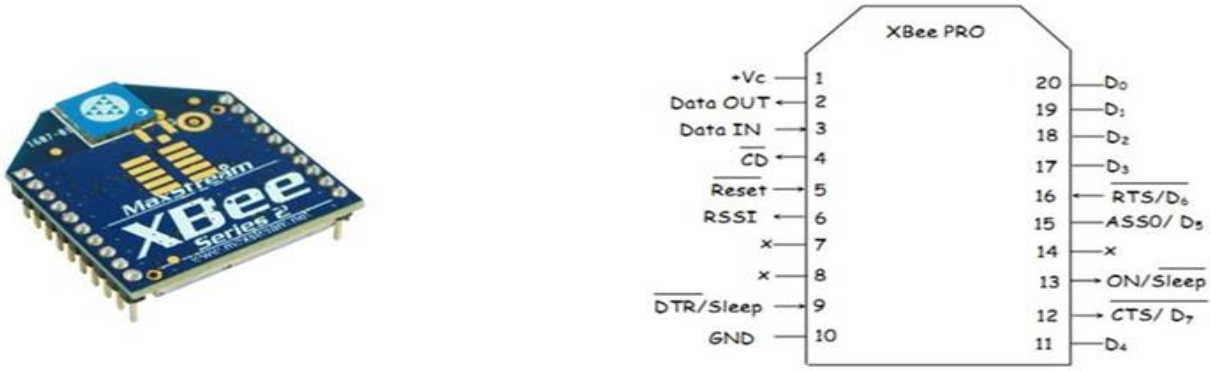


Figure 10 le XBee

Le module XBee utilise une liaison asynchrone pour communiquer avec un microcontrôleur. La configuration des paramètres de la liaison série (Vitesse, parité, start et stop, …) s'effectue à l'aide des commandes AT. Celles-ci peuvent être envoyées soit par un HyperTerminal (voie graphique grâce au logiciel XCTU fournit par le constructeur DIGI) soit via un microcontrôleur ou une carte intelligente comme exemple l'Arduino . C'est cette deuxième méthode qui est présenté. Cela nécessite d'utiliser au minimum les broches Vcc, GND, OUT et IN du module Xbee (voir le brochage paragraphe 3.1). Les broches OUT et IN doivent être connectées respectivement aux broches Rx et Tx de notre carte. Le module Xbee peut être alimenté entre 2,7 et 3,3 volts.

c. Caractéristiques

Fonctions	Module XBee série 1 Module XBee série 2	Module XBee-Pro série 1
Transmissions en intérieur	Jusqu'à 30 m Jusqu'à 40 m (série 2)	Jusqu'à 100 m
Transmission en extérieur	Jusqu'à 100 m Jusqu'à 120 m (série 2)	Jusqu'à 1500 m
Puissance de sortie	1 mW (0 dBm) 2 mW (+3dBm) (série 2)	60 mW (18 dBm) Ajustable par soft
Débit des données RF	250.000 bps	250.000 bps
Débit des données UART	1200 bps à 115.200 bps (autres débits non standards possibles)	1200 bps à 115.200 bps (autres débits non standards possibles)
Sensibilité	-92 dBm -98 dBm (série 2)	-100 dBm
Caractéristiques électriques		
Tension d'alimentation	2,8 V à 3,4 V	2,8 V à 3,4 V
Courant en émission	45 mA à 3,3 V 40 mA à 3,3 V (série 2)	10 dBm : 137 mA à 3,3 V 12 dBm : 155 mA à 3,3 V 14 dBm : 170 mA à 3,3 V 16 dBm : 188 mA à 3,3 V 18 dBm : 215 mA à 3,3 V
Courant en attente ou en réception	50 mA à 3,3 V 40 mA à 3,3 V (série 2)	55 mA à 3,3 V
Courant en mode «SLEEP»	< 10 µA 1 µA (série 2)	< 10 µA

d. Communication avec l'ordinateur

Figure 11

La communication en direct sans passer par une Arduino vous permet de configurer rapidement votre XBee. On verra plus loin dans les cas pratiques qu'on peut aussi configurer le module en le branchant à l'Arduino. Donc se procurer un explorateur n'est pas indispensable, mais c'est à conseiller pour débuter car c'est tout de même plus simple.

e. Communication série

La communication avec le module XBee s'établit par une communication série asynchrone. C'est très pratique, il suffit de quatre fils : deux pour l'alimentation (la masse et le +), un pour la réception et un pour l'émission. En effet le XBee permet de recevoir et d'émettre des données en même temps, on dit qu'il est full duplex, contrairement à la radio FM qui envoient les informations dans un seul sens (simplex) et au talkie-walkie qui ne permet pas à deux émetteurs de parler en même temps (half-simplex). On dit aussi que le XBee est un transceiver qui est la contraction de TRANSmitter (émetteur) et de reCEIVER (récepteur).

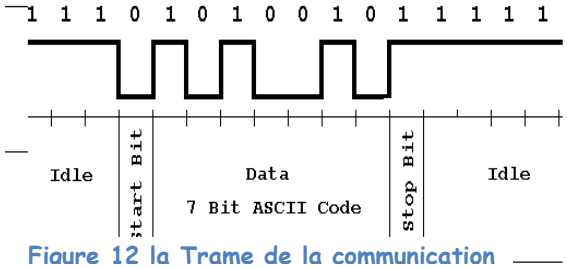


Figure 12 la Trame de la communication

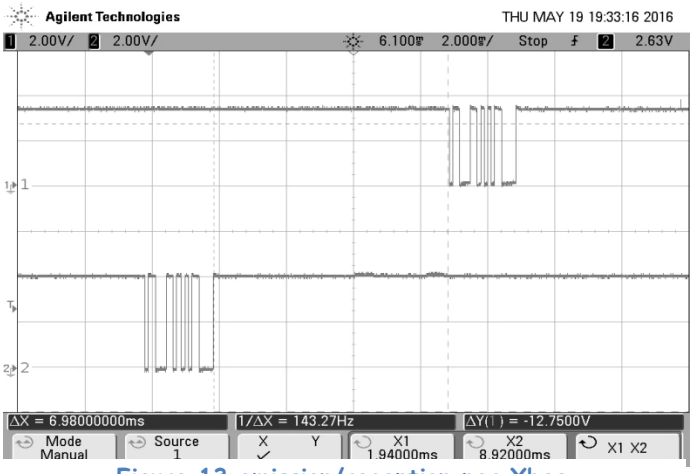


Figure 13 emission/reception par Xbee

En dessus, un exemple d'envoi d'un caractère par le premier Arduino (courbe inférieure), et le même caractère reçue par le deuxième (courbe supérieure), après un temps de transmission (7ms pour 1m de distance)

f. Configuration graphique

Pour la configuration du module Xbee on utilise le programme CTU (Configuratuin & test utility Software) fourni par le constructeur « Digi International » comme il est indiqué sur les figures ci-contre

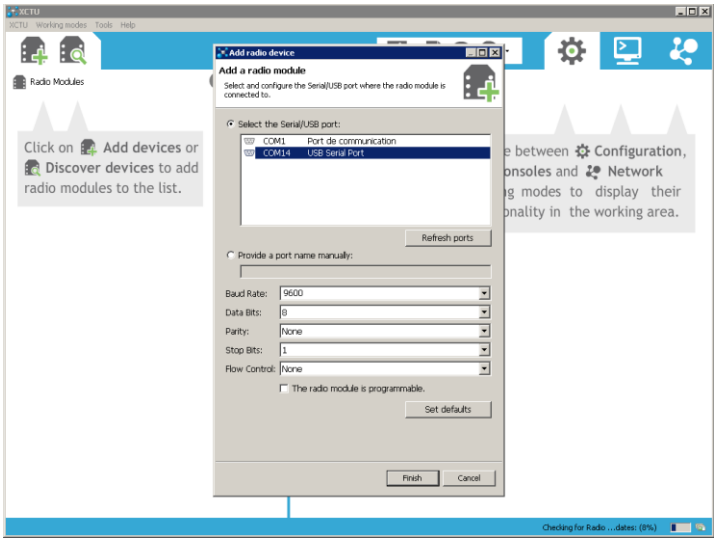


Figure 14 ajout d'un module connecté sur le port usb

On ajoute le module en appuyant sur add device et on remplie les configurations de la trame

- Baud rate : la vitesse de communication
- Data bits : le nombre de bits de données
- Parity : nombre de bits de parité
- Stop bits : les bits de stop (ici 1)
- Flow contrôle : contrôle de la trame(ici sans contrôle

On obtient l'écran suivant pour la configuration des paramètres de connexion entre les xbees

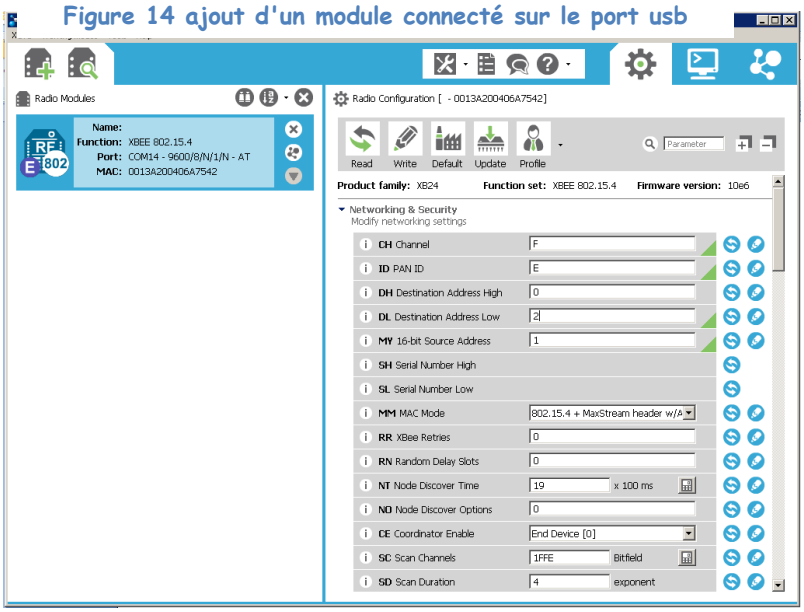


Figure 15 configuration de l Xbee

- **CH** : Le module peut émettre en RF sur 10 canaux différents (Cf doc). Le canal doit être le même pour les 2 modules
- **MY** : L'adresse du module en cours de configuration est donc $0001.On est en adresse courte (16 bits) car MY ≤ $FFFE
- **DH** : La partie haute de l'adresse du module de destination doit être '0' pour que le module en cours de configuration s'apparie avec le module de destination
- **ID** : adresse de réseau commun
- **DL** : L'adresse du module de destination est donc $0002 La puissance d'émission du module peut être réglée (Elle influe sur la portée du module)
- **SH**,**SL** : numéro de série haut et bas
- **MM** : Mode Mac
- **CE** : Mettre le dispositif comme coordinateur

g. Configuration par uC (ici Arduino)

Exemple série 2

Il faut juste envoyer les commandes comme étant de chaines de caractères pour configurer le XBee.

Commande AT envoyé par le microcontrôleur via l'UART	Réponse du module
+++ (mode AT)	OK <CR> (Entre dans le mode commande AT)
ATID 56 <entrer>	OK <CR> (positionne le PAN à 56)
ATMY <entrer>	{valeur actuel} <CR> (lit l'adresse réseau du module Xbee)
ATCH 2 <entrer>	OK <CR> (choix du canal 3)
ATBD 7 <entrer>	OK <CR> (vitesse de transmission 115200bps)
ATDH 0x2614AF <entrer>	OK <CR> (modifie l'adresse de destination haute)
ATDL 0x65AFC5D <entrer>	OK <CR> (modifie l'adresse de destination basse)
ATWR <entrer>	OK <CR> (sauve les paramètres dans mémoire non volatile)
ATCN <entrer>	OK <CR> (quitte le mode commande AT)

h. Communication entre deux Arduinos via xbee

Les cartes Arduino disposent d'une communication série matérielle sur les broches 0 et 1 (lesquelles sont également connectées à l'ordinateur via la connexion USB). Cette communication série matérielle est réalisée par un module matériel (intégré dans le circuit intégré du microcontrôleur de la carte) appelé UART (pour "Universal Asynchronous Receiver Transmitter" en anglais - émetteur-récepteur asynchrone universel). Ce module matériel permet au microcontrôleur Atmega de la carte Arduino de recevoir des communications série même lorsqu'il travaille à d'autres tâches, tant qu'il y a de la place dans les 64 octets de buffer de l'UART (le buffer est reçoit les données entrantes et les stocke en attendant qu'elles soient lues).

La librairie de communication série présentée ici a été développée pour permettre des communications série sur d'autres broches numériques de la carte Arduino, en utilisant un programme (le code de la librairie) pour répliquer la même fonction de communication série (d'où le nom de Software Serial pour communication série logicielle)

Un nouvel dispositif communicant via un port série est déclaré comme étant une variable de type SoftwareSerial :

- `SoftwareSerial mySerial= SoftwareSerial(rxPin, txPin);//rxPin et txPin a préciser (pins numériques )`
- La lecture d'une donnée sur XBee se fera alors grâce à la fonction XBee.read ()
- L'écriture d'une donnée sur XBee se fera alors grâce à la fonction XBee.print()

Le montage réalisé pour ce test est représenté par la figure

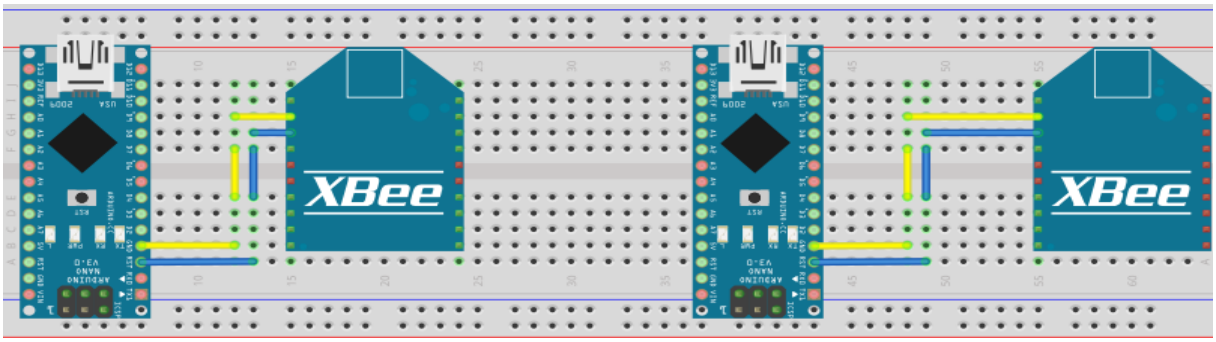


Figure 16 Montage d'essai arduino_xbee // arduino_xbee

Mise en œuvre de la communication

le sketch permettant cette communication tourne en boucle infinie, et détecte la présence d'une donnée dans le buffer de réception de XBee par la fonction Xbee.available() qui retourne 1 si la donnée et prête, si la condition est vraie il vas afficher le caractère correspond au Octet reçu jusqu'à ce que le buffer devient vide

Pour le coté émission la même procédure sauf que cette fois ci il test la présence d'une donnée dans le pour série (moniteur série COM5 ou COM4) et il envoie les octets par la fontion XBee.print()

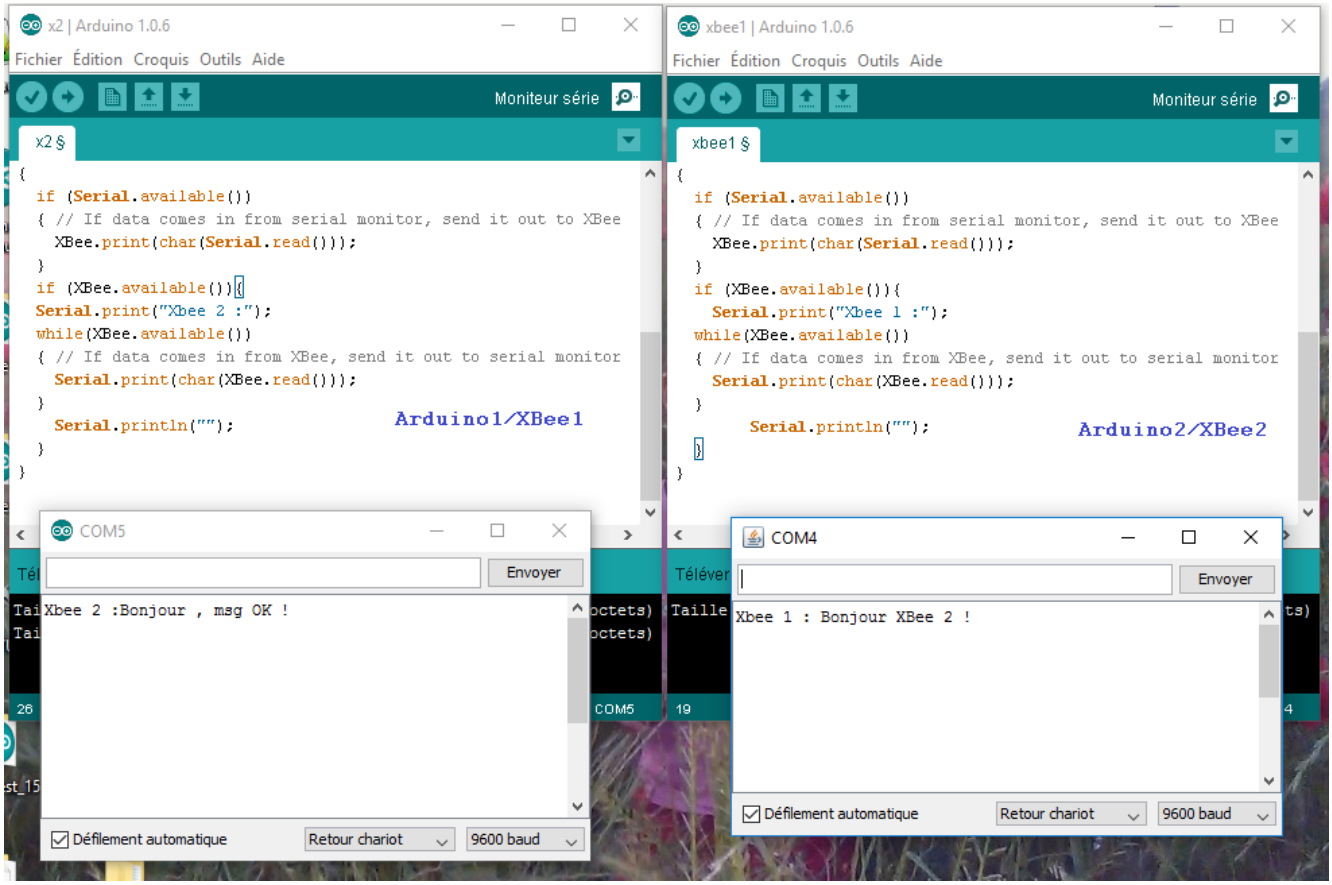


Figure 17 communicatio sanfil entre deux arduinos

IV. Gyroscope, accéléromètre

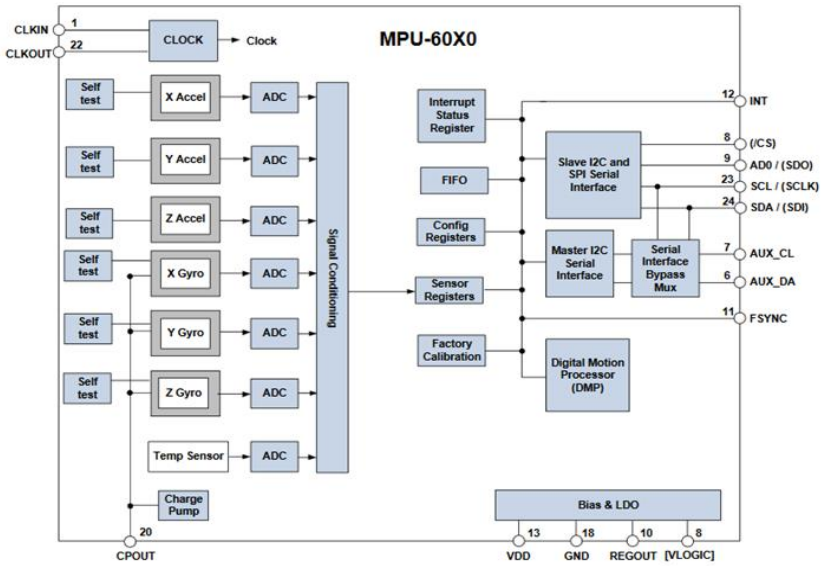


Figure 18 Block diagram de la MPU6050

a. Généralités

Une centrale à inertie est un équipement de navigation comportant six capteurs d'une précision métrologique :

- trois gyromètres mesurant les trois composantes du vecteur vitesse angulaire (vitesses de variation des angles de roulis, de tangage et de lacet); et
- trois accéléromètres mesurant les trois composantes du vecteur force spécifique (en). La force spécifique est la somme des forces extérieures autres que gravitationnelles divisée par la masse. Cette quantité a donc la dimension d'une accélération mais, contrairement à ce que suggère le nom de l'instrument de mesure, il ne s'agit pas exactement d'une accélération.

Le calculateur de la centrale à inertie réalise l'intégration en temps réel, uniquement à partir des mesures de ces six capteurs :

- des angles d'attitude (roulis, tangage et cap) ;
- du vecteur vitesse ; et
- de la position.

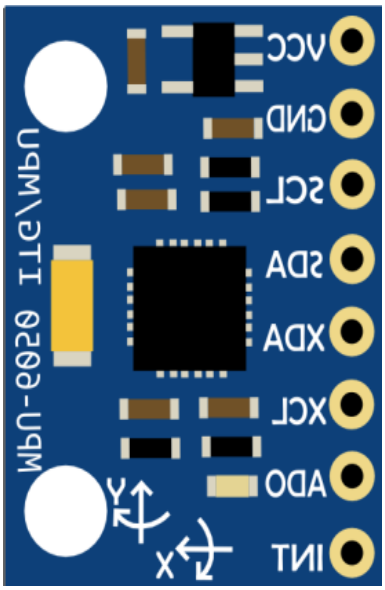


Figure 19 mpu6050

Des centrales à inertie sont installées à bord de véhicules terrestres, de navires, de sous-marins, d'avions, d'hélicoptères, de missiles et de véhicules spatiaux. Elles sont même parfois transportées par des piétons pour des applications de localisation de précision en génie civil quand le GPS n'est pas utilisable (forage de tunnels).
Les désignations habituelles des centrales à inertie sont des acronymes issus de l'anglais : Inertial Reference System (IRS), Inertial Navigation System (INS), ou Inertial Measurement Unit (IMU). Cette dernière appellation désigne le sous-système limité aux seuls capteurs inertiels (gyromètres et accéléromètres), sans calculateur.
On a choisi le MPU6050

b. Principe de fonctionnement

- Le gyroscope - accéléromètre MPU-6050 comporte 6 axes.
- Sa puce MEMS est très précise avec une conversion analogique-digitale sur 16 bits simultanée sur chaque canal, et une interface I2C (400 kHz).
- La lecture des mesures brutes de ce capteur est facile.
- Le capteur contient un registre FIFO de 1024 octets que le micro-contrôleur Arduino peut lire, étant prévenu par un signal d'interruption.
- Le module fonctionne en esclave sur le bus I2C vis à vis de l'Arduino (pins SDA, SLC) mais il peut aussi contrôler un autre dispositif en aval avec AUX-DA et AUX-CL, par exemple un magnétomètre 3 axes (mesure du champ magnétique terrestre) pour une orientation absolue dans l'espace (boussole).
- Sa consommation est faible, 3.9 mA avec les 6 capteurs activés.
- Le capteur possède un DMP (Digital Motion Processor) capable de faire des calculs rapides directement sur la puce à partir des mesures brutes du capteur (mais c'est malheureusement mal documenté).

Il est donc plus simple de traiter les mesures brutes sur sa carte Arduino.

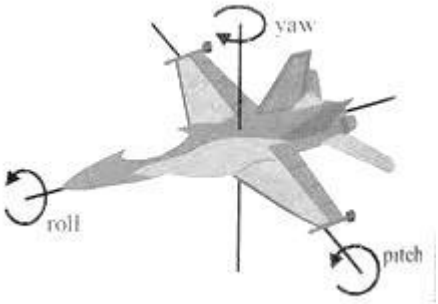
Mesures réalisées par le capteur

- Le gyroscope retourne une vitesse angulaire de rotation selon 3 axes, 0 si pas de rotation (degrés/seconde).
  Il ne donne pas directement un angle d'orientation (degrés).
  Un angle s'obtient par intégration dans le temps, en faisant attention au cumul des erreurs de dérive.
- L'accéléromètre retourne une force ou une accélération (m^2/s), la pesanteur terrestre seule si le module est fixe.

On peut la supprimer (par soustraction) dans le code si on ne veut que les accélérations.
La vitesse (m/s) peut s'en déduire par une première intégration dans le temps (à une vitesse initiale V0 près), en faisant attention au cumul des erreurs de dérive.
La position de déplacement (m) peut s'en déduire par une seconde intégration dans le temps.
Un filtrage des mesures brutes s'impose si on ne veut pas cumuler des petites erreurs et faire dériver la position exacte.

Filtrage des mesures brutes

L'accéléromètre doit être filtré si on veut piloter un servo de manière fluide sans vibrations.



- On peut utiliser une simple moyenne glissante de 40 mesures de 12 ms chacune par exemple, pour lisser les micros fluctuations
- ou utiliser un filtre de Kalman (méthode plus complexe).

Applications

- Pilotage automatique et stabilisation en radio modélisme
- Jouets, sports (Segway...)
- Reconnaissance de gestes
- Télécommandes 3D
- Capteur de mouvement à porter sur soi
- Stabilisation de caméra vidéo, nacelle photo (guimbal)
- Commandes par mouvements
- Détecteur de chocs

Lecture des données brutes du capteur

Les échelles de mesures suivantes sont programmables :

- 3 axes de gyroscope, échelles de ±250, ±500, ±1000, et ±2000 degrés/sec
- 3 axes d'accéléromètre, pleine échelle de ±2g, ±4g, ±8g et ±16g.
- les quaternions (w x y z).
  3 quaternions permettent de définir un axe, et le 4ème décrit la rotation autour de cet axe, ceci permet d'éviter les limitations au-delà de 180°.
  Voir l'effet de guimbal lock ici http://en.wikipedia.org/wiki/Gimbal_lock
- les angles d'Euler (psi, téta, phi).
  Les angles d'Euler sont les rotations autour des axes X Y et Z.
- les angles yaw , pitch, roll.
  YAW = direction(> 0 virage à droite)
  PITCH = tangage (> 0 bascule en avant)
  ROLL = roulis (> 0 à gauche)
- C'est la sortie la plus utile (guimbal)
- l'accélération, selon les axes du capteur
- l'accélération sans la pesanteur (yaw selon l'orientation initiale, sans magnétomètre).
- le format brut

Câblage

4 fils suffisent au montage (grâce au Bus i2c)
Le capteur est monté sur un PCB avec 2 x 5 pins --> Carte Arduino nano

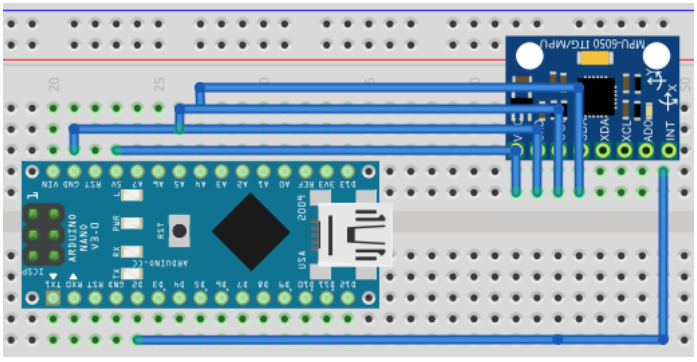
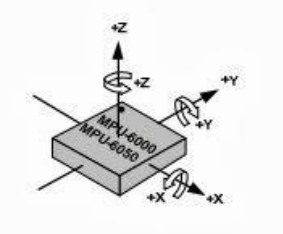


Figure 20 cablage mpu6050 avec arduino

- 5V
- 3V3 --> 3.3V (ne pas utiliser le 5v)
- SDA -->A4 (uno)
- SCL -->A5 (uno)
- GND --> GND
- INT --> pin2 si on utilise les interruptions
- ADO
- XDA
- XCL
- GND

« Brochage du bus I2C de la carte nano A4 =SDA, A5= SCL, et pour la carte »



Orientation spatiale des 3 axes

La puce comporte 3 axes (XY dans le sens anti-horaire vu de dessus)

x = selon la longueur, gauche > droite

y = selon la largeur , avant > arrière

z = selon la hauteur, bas > haut

La communication entre la carte Arduino et le MPU6050 se fait via le protocole I2C qu'on détaille ci-dessous

C. Protocole I2C

I2C (signifie : Inter-Integrated Circuit, en anglais) est un bus informatique qui a émergé de la « guerre des standards » lancée par les acteurs du monde électronique. Conçu par Philips pour les applications de domotique et d'électronique domestique, il permet de relier facilement un microprocesseur et différents circuits, notamment ceux d'une télévision moderne : récepteur de la télécommande, réglages des amplificateurs basses fréquences, tuner, horloge, gestion de la prise péritel, etc.

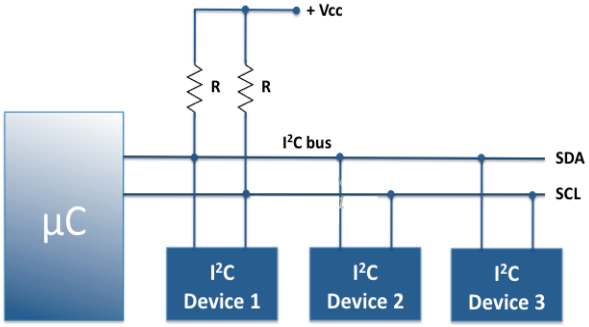


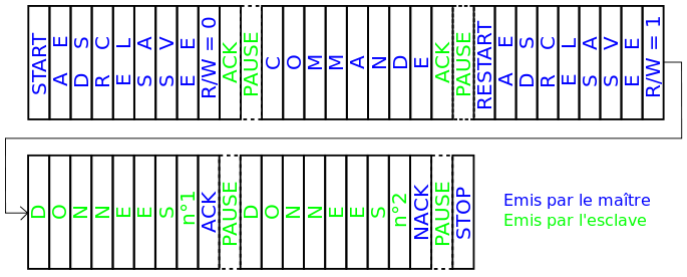
Figure 21 communication I2C avec plusieurs dispositif

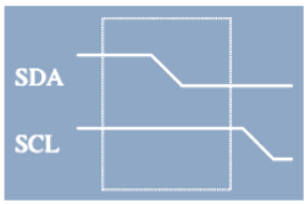
Il existe d'innombrables périphériques exploitant ce bus, il est même implémentable par logiciel dans n'importe quel microcontrôleur. Le poids de l'industrie de l'électronique grand public a permis des prix très bas grâce à ces nombreux composants.

Ce bus porte parfois le nom de TWI (Two Wire Interface) chez certains constructeurs.

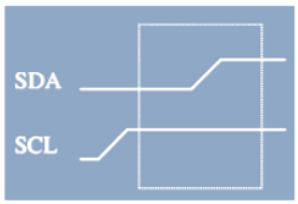
La trame I2c :

Communication Arduino-MPU6050





condition de demarrage



Condition d'arret

Figure 22 trame I2C

La communication entre l'Aduino se fait, comme nous l'avons signalé, via le protocole I2C, pour cela il faut d'abord télécharger deux bibliothèques d'arduino qui sont indispensables pour la communication :

Une bibliothèque I2Cdev qui contient les fonctions permettant la communication série I2C

Lien vers

Une bibliothèque MPU6050 qui contient toutes les adresses des registres de la centrale d'inertie et les fonctions permettant d'accomplir les différentes étapes pour d'initialisation, configuration, extraction des données etc…

Lien vers mpu

Le code permettant d'voir les données de la mpu6050 (voir annexe)

Le résultat de la communication :

d. Correction des angles par un PID

Comment régler les coefficients d'un PID ?

Le réglage des coefficients Kp, Ki et Kd d'un PID se fait de manière empirique par un cycle d'essais-erreurs. On ne règle jamais les 3 coefficients en même temps car il y a trop de combinaison possible.

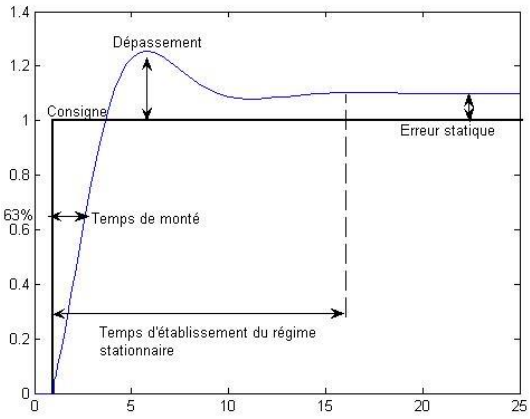


Figure 23 Régulation PID

- On commence par le régulateur proportionnel seul (Ki et Kd sont nuls). On améliore le temps de réponse du système. Le but étant de se rapprocher très vite de la consigne tout en gardant la stabilité du système, Si le système oscille trop, il est trop instable.
- Une fois ce coefficient réglé, on peut passer au coefficient Intégral Ki. Celui-là va permettre d'annuler l'erreur finale du système afin que celui-ci respecte exactement la consigne. On travaille sur la précision. Il faut donc régler Ki pour avoir une réponse exacte rapidement et en minimisant les oscillations apportées par l'intégrateur.
- Enfin, on peut passer au dernier coefficient Kd qui permet de rendre le système plus stable. Son réglage permet donc de diminuer les oscillations.
- C'est la consigne qui détermine nos calculs, ci-après un schéma reprenant les différentes notions mis en jeux:

Donc tout est une question de compromis. Le PID parfait n'existe pas:

- L'erreur statique, c'est l'erreur finale en régime stationnaire. Pour diminuer cette erreur, il faut augmenter Kp et Ki.
- Le dépassement, c'est le rapport entre le premier pic et la consigne. Ce dépassement diminue si Kp ou Ki diminuent ou si Kd augmente.
- Le temps de montée correspond au temps qu'il faut pour arriver à X% de la consigne (en général X = 67). Le temps de montée diminue si Kp ou Ki augmentent ou si Kd diminue.
- Le temps de stabilisation, c'est le temps au bout duquel l'erreur statique est inférieur à 5% de la consigne. Ce temps de stabilisation diminue quand Kp et Ki augmentent.

Pour résumer:

	Précision	Stabilité	Rapidité
P	↗	↘	↗
I	↗	↘	↘
D	↘	↗	↗

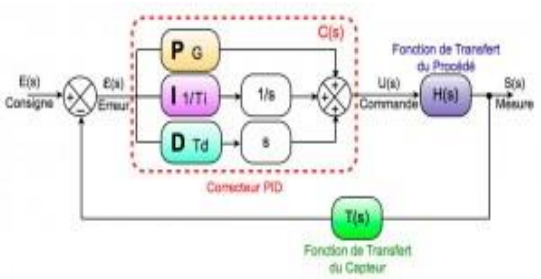


Figure 24 régulation PID

Important: les coefficients Ki et Kd dépendent de la fréquence d'échantillonnage du système ! En effet, l'intégrateur fera la somme des erreurs au cours du temps ! Si on échantillonne deux fois plus vite, on sommera deux fois plus d'échantillons. Du coup, le coefficient Ki devra être divisé par 2. A l'inverse, pour le dérivateur, si on double la fréquence d'échantillonnage, il faudra doubler le coefficient Kd afin de garder les mêmes performances du PID. Plus la fréquence d'échantillonnage est élevée et plus le PID sera performant. (En effet, plus on échantillonne souvent et plus l'intégration et la dérivée seront précises).

Dans notre cas l'asservissement des angles du drone se fait par le contrôle de la vitesse de chaque moteurs suivant sa position et les paramètres du gyro qui le positionnent :

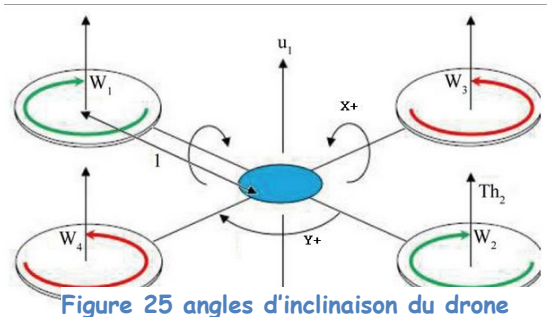


Figure 25 angles d'inclinaison du drone

Soit le drone a cette position, alors la variation de Y vers le côté positif signifie que les moteurs (W1,W4) doivent tourner plus vite pour rendre le drone plate, les moteurs (W2,W3) dans le cos ou Y et négatif. La même chose pour les moteur (W1,W3) pour les X positifs et (W2,W4) pour les X négatifs , comme ça on aura des corrections de la vitesse des moteurs comme ceci :

On suppose que tous les moteurs tournent à une vitesse initiale Val_speed

Alors après la correction on effectue les instructions suivantes :

```
mymotor1.write(val_speed+commande_y-commande_x);
mymotor4.write(val_speed-commande_x-commande_y);
mymotor3.write(val_speed+commande_x+commande_y);
mymotor2.write(val_speed+commande_x-commande_y);
```

avec mymotor1,2,3 et 4 sont les moteurs du drone et commande_x et commende_y sont les corrections calculés à partir de la position des trois axes tels que :

```
commande_y =Kp*erreur_y+Ki*somme_erreurs_y+Kd*variation_erreur_y;
commande_x =Kp*erreur_x+Ki*somme_erreurs_x+Kd*variation_erreur_x;
```

e. Testes et résultats

Après avoir embarqué le circuit de commande sur le drone on a réussi à récupérer sur la machine de contrôle l'ensemble des données à savoir l'altitude du drone et les données du gyroscope (tangage, roulis, a lacet) , l'écran qui s'affiche sur le moniteur série est le suivant :

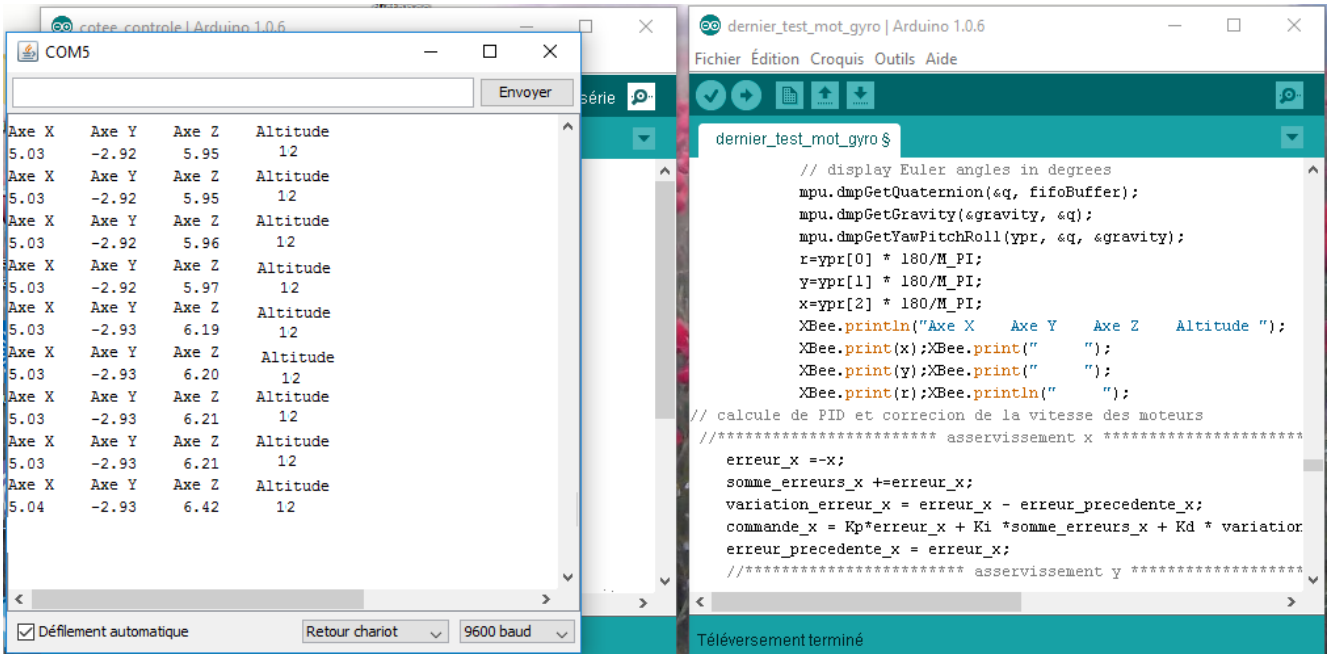


Figure 26 : les données récupérées

V. Le variateur de vitesse(ESC)

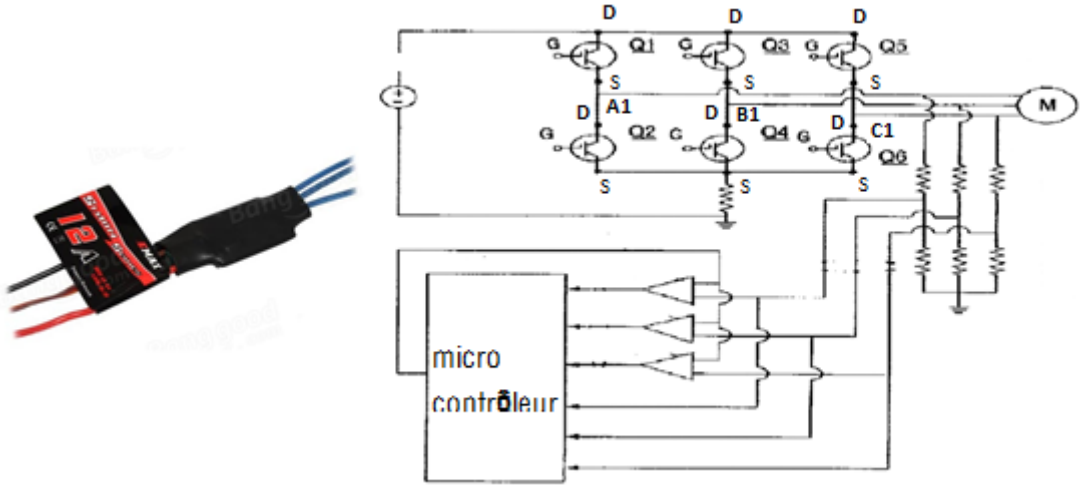


Figure 27 ESC simonk

a. Définition :

Un variateur de vitesse est un équipement permettant de fait varier la vitessse d'un moteur.

b. Caractéristiques du variateur :

Modèle	Continuous current	Burst current	Battery (cell)	Demension (L*W*H)	Weight (g)	programmable
Simon-12A	12A	15A	2-3	25*20*7	9	YES

Dans le drone, on va utiliser les moteurs sans balais qui fonctionnent avec un courant triphasé. De l'autre côté on a une batterie fournissant un courant continu. Il nous faut donc un variateur de vitesse pour convertir un courant continu à un courant triphasé afin de pouvoir faire tourner le moteur plus ou moins vite. Le diagramme du variateur est comme le circuit indiquer ci-dessus, le variateur est constitué par un microcontrôleur et les MOS. Le microcontrôleur dans le variateur toujours gère 3 fonctions :

- La commande des « gaz »
  Il produit un découpage de la tension de batterie pour modifier la tension moyenne appliquée au moteur, quand il reçoit la « gaz ».
- La commutation des tensions sur les bornes A1, B1, C1
- Le microcontrôleur pilote les interrupteurs des transistors MOS pour leurs état connectent en borne ou en circuit ouvert à la batterie.
- La mesure la tension induite dans la bobine laissée en circuit ouvert

Pour varier la vitesse du moteur, on utilise un signal de PWM(ou MLI) comme le signale l'entrée du variateur. Et le variateur va fournir les différentes fréquences du courant triphasé par rapport à le rapport cyclique du signal l'entré. Les signaux comme le figure suivant :

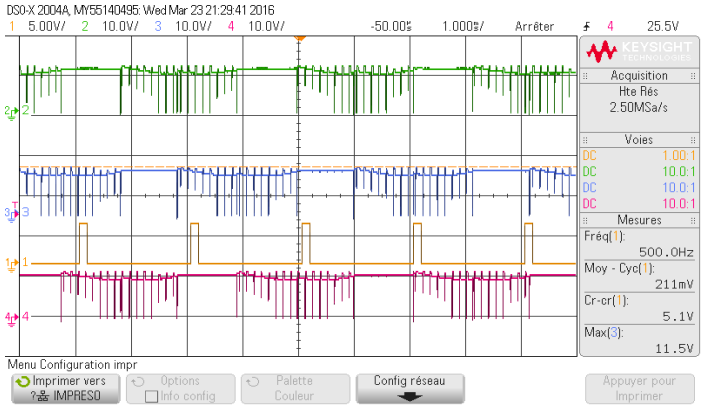


Figure 28 Rapport cyclique de 7%

La courbe jaune est le signal PWM d'entré, la courbe vert, bleu et rouge sont les courants triphasés générés par le variateur. Ces signaux triphasés sont générés par un PWM du rapport cyclique de 7%.

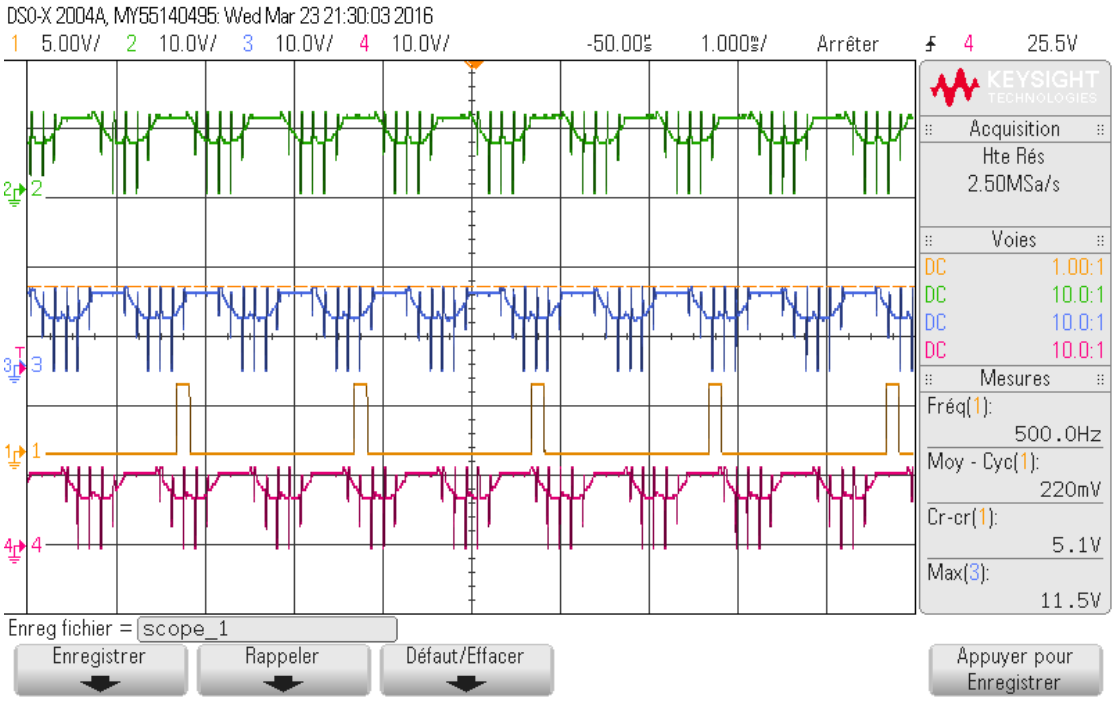


Figure 29. Rapport cyclique de 10%

On compare avec les deux figures qui sont générée par les différents rapports cycliques de signal PWM, et on trouve que le variateur de la vitesse varie le fréquence des signaux sorties, mais il ne varie pas le tension. La tension sortie du variateur est la même valeur que la tension de la batterie. Ensuit dans les deux figures, les signaux de sortie ne sont pas très lisses. C'est parce que le changement des bobines et des aimants va créer des pics.

V. Moteur brushless « sans balais » :

a. Petit rappel : Comment fonctionne un moteur électrique ?

Avant de parler des moteurs brushless en particulier, un petit rappel du fonctionnement des moteurs électriques "traditionnels", c'est à dire à charbons, ne paraît pas superflu. II faut savoir que, lorsqu'un courant passe dans un fil et que ce fil est soumis à un champ magnétique (aimant par exemple), ledit courant fait naître une force magnétique. Dans un moteur électrique, cette force, engendrée par le passage du courant dans les bobinages qui sont soumis au champ magnétique des aimants, est utilisée pour produire une rotation. Le moteur électrique est donc un "convertisseur" transformant l'énergie électrique en énergie mécanique... plus de la chaleur car le rendement n'est jamais de 100%. Pour créer le "champ tournant" des bobinages, il faut utiliser un système de commutation de manière à faire passer le courant dans ces bobinages dans un ordre précis et au bon moment. Dans un moteur à charbons ou balais (appelé brushed en anglais), cette commutation des bobinages est réalisée mécaniquement par l'intermédiaire des lamelles du collecteur situées sur l'axe moteur. Lors de la rotation, les charbons sont successivement en contact avec ces lamelles qui vont transmettre le courant continu délivré par le variateur aux bobines du moteur. Le passage du courant dans le fil des bobines fait naître une force magnétique qui repousse les aimants de même pôle et attire ceux de pôles opposés. Ce système de commutation mécanique détermine l'architecture type d'un moteur à charbons qui comprend : un rotor interne portant les bobinages, et un stator externe où sont fixés les aimants. Cette commutation mécanique est facile à réaliser mais le frottement des charbons (ou balais) sur le collecteur entraîne une perte, un échauffement du collecteur et une usure inévitable de celui-ci. D'où un rendement relatif et la nécessité d'entretenir son moteur en vérifiant régulièrement l'état des charbons et du collecteur.

b. Comment fonctionne un moteur brushless ?

Dans un moteur brushless (terme qui se traduit par "sans balais"), la commutation des enroulements est faite non pas mécaniquement comme précédemment mais de manière électronique par un système complexe appelé "contrôleur". Celui-ci transforme le courant continu en courant triphasé à fréquence variable et va alimenter successivement les bobines du moteur pour créer le champ tournant et donc la rotation qui nous intéresse. On comprend aisément qu'avec ce principe d'alimentation, il est impératif que les bobines soient fixes dans le moteur, et non pas en rotation comme dans un moteur à charbons. Tous les brushless ont donc la même architecture de construction : un stator fixe qui porte les bobines, et un rotor mobile sur lequel les aimants permanents sont collés. Les bobinages peuvent être réalisés de manières différentes : en étoile ou en triangle (appelé également delta), mais vous trouverez toujours trois fils à la sortie du moteur, qui réunissent les bobinages. La majorité des brushless possèdent un rotor interne qui tourne très vite jusqu'à 100.000 tr/mn. Si la vitesse est là, le couple est médiocre et il est alors obligatoire de les réducteur fortement de manière à pouvoir utiliser des hélices de taille adaptée.

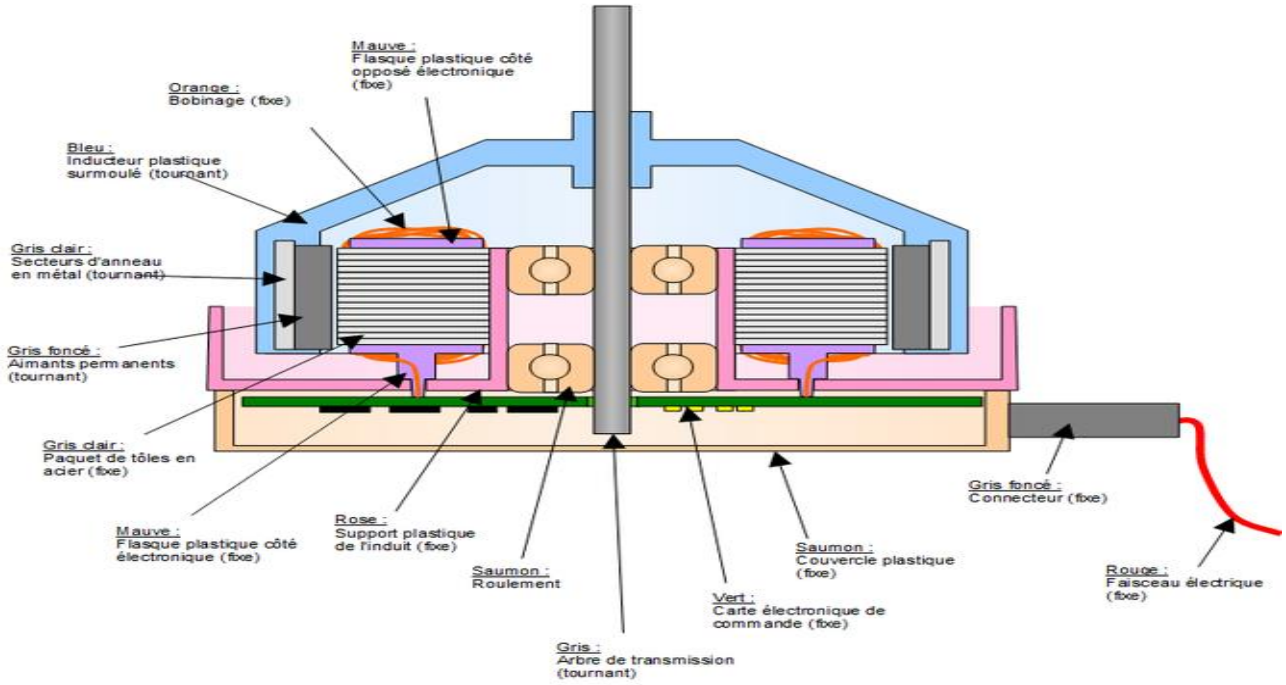


Figure 30: schéma d'un moteur brushless "sans balais"

C. Caractéristiques importantes des moteurs brushless sont en général :

- Le poids, c'est le premier paramètre qu'on regarde quand on fait son choix.
- Le rendement en %, c'est le paramètre sur lequel il faut faire le moins de compromis possible.
- Le Kv : coefficient de vitesse, nb de tours/min/volts, exemple : 2000kV sous 10V = 20 000 trs/min à vide, il est directement lié au nb de tours de bobinage.
- Les pôles, nombre de pôles magnétiques du rotor de 2 à 6 pour des moteurs classiques.
- Le régime maxi, limitée mécaniquement par les roulements et le rotor, ou la tension maxi d'utilisation.
- La température max de fonctionnement.

Concrètement :

LE POIDS: à rendement égal, plus le moteur sera lourd et plus le moteur pourra encaisser de puissance. Il faut bien comprendre que c'est le châssis qui va imposer au moteur la puissance à délivrer et non l'inverse : Le courant consommé par un moteur électrique est directement proportionnel au couple résistant qu'on lui applique. Ça veut dire que plus il "force" (rapport de transmission élevé, poids du châssis élevé) et plus il consomme d'ampères pour faire avancer le châssis. Et s'il n'est pas prévu pour consommer autant il va surchauffer et finira par cramer son bobinage ou démagnétiser les aimants du rotor. Inversement si le moteur est suffisamment gros, on peut lui en demander beaucoup et tirer des rapports de malade sans qu'il chauffe.

LE RENDEMENT : à poids égal, plus le moteur aura du rendement et moins il chauffera et consommera de courant pour la même application. Le rendement n'impactera pas les performances, mais plutôt le temps de roulage et la température du moteur en fin de pack. Il dépend surtout des matériaux utilisés pour la fabrication du moteur, et un peu du design électromagnétique ses entrefers et du bobinage. En fonctionnement il va dépendre évidemment du courant que le moteur consomme. Un moteur trop chargé aura un mauvais rendement, idem pour un moteur pas assez chargé. Pour l'application 1/8ème TT il est difficile de ne pas assez charger le moteur, c'est quasiment tout le temps le contraire. Expérience d'illustration : faire tourner à fond un moteur brushless à vide, vous constaterez qu'il chauffe assez vite car il n'a aucune charge. Le rendement dépend enfin de la température d'utilisation, d'où

l'intérêt d'avoir un moteur le plus froid possible. Mais il ne faut pas rêver, on ne gagne pas 5 min de roulage entre un moteur à 40°C et un moteur à 80°C.

LE KV : là on rentre dans le vif du sujet. A rendement et poids égaux, plus le KV est grand et moins le moteur a de couple par ampère, ou plus il consomme de courant pour le même couple mais plus il tourne vite. Donc inversement plus le KV est petit et plus le moteur à de couple par ampère, ou moins il consomme d'ampère pour un même couple mais moins le moteur tourne vite. Pour une même application, un moteur à petit Kv aura besoin d'une tension plus grand pour avoir les même performances qu'un moteur à KV plus grand, il fonctionnera sous un voltage plus élevé et à un courant plus faible pour une même puissance: P=UxI.

- Intérêt des grands KV : pas besoin d'alimenter avec des grosses tensions, mais consommation de courant élevée ce qui nuit au rendement du contrôleur et ce qui "force" sur le contrôleur et les accus. De plus la faible inductance des bobines donne une impression de violence à l'accélération, moteur plus vif.
- Intérêt des petits KV : rendement moteur en légère hausse, rendement contrôleur en hausse, impression d'onctuosité, moteur moins violent à l'accélération (retard de l'établissement du courant dans les bobines à cause de la plus grande inductance), mais besoin de plus de tension.

Le Kv dépend du bobinage du moteur, plus il y a de tours et moins le moteur a de Kv mais à poids égal, les fils du bobinage sont nécessairement plus fins et par conséquent le moteur doit travailler à un petit courant pour ne pas chauffer anormalement.
Voilà pourquoi on ne doit pas utiliser un moteur à petit KV à la même tension qu'un moteur à grand KV et avec un gros rapport pour compenser le manque de vitesse, sous peine de cramer le bobinage.
Vous remarquerez qu'à aucun moment je n'ai parlé de couple. Quand on dit qu'"un moteur a du couple", c'est assez subjectif en brushless. Dans des limites raisonnables d'utilisation, les deux moteurs précédents (petit Kv et gros Kv) auront rigoureusement le même couple, le gros KV consommera simplement plus de courant pour la même accélération mais à une tension plus petite. Le couple ressenti lorsqu'on enfonce sa gâchette à fond dépend en fait surtout du réglage du variateur qu'on verra dans le paragraphe suivant.

LES POLES: à part les moteurs Lehners, les moteurs brushless de qualité sont multiples (4), les 2 pôles sont souvent des moteurs d'entrée de gamme. En général plus de pôles signifie plus de douceur ("continuité" magnétique du rotor) et plus de rendement mais ce n'est pas tjrs le cas pour le rendement. On utilise presque toujours des 4 pôles dans notre application.

LE REGIME MAXI : les lois de la physique sont les même pour tous ! Un rotor qui tourne trop vite subit une très grosse force centrifuge, idem pour les roulements.
Exemple : les moteurs 15xx sont limités à 60 000trs/min, en pratique on utilise nos moteurs à des régimes entre 30 000 et 50 000trs/min.

LA TEMPÉRATURE DE FONCTIONNEMENT : si vous laissez chauffer votre moteur à une température trop élevée, vous allez le dégrader ses éléments : cramer le bobinage et démagnétiser les aimants. Dites-vous qu'il n'y a que 2 manières de casser un moteur brushless :
- un coup de marteau
- une grosse surchauffe, ou une chauffe répétée qui démagnétise petit à petit les aimants.

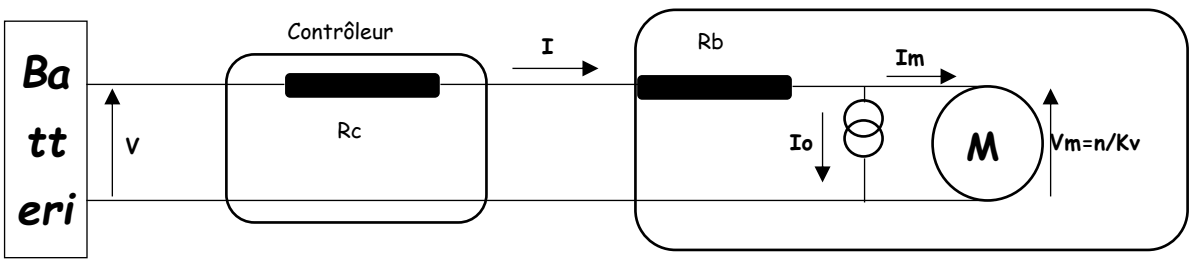
En général on ne doit pas dépasser les 80°C pour avoir une marge sûre de non détérioration du moteur. Certains moteurs bas de gamme n'acceptent que 70°C.

d. Quelle puissance consomme et fournit le moteur ?

On a regardé par le détail comment un moteur brushless fonctionne, pour déterminer ses propriétés de puissance et d'entraînement d'hélices, c'est un schéma électrique global qui sera utilisé.
Ce schéma débute à la sortie de la batterie d'alimentation qui donne V volts.

Ensuite le contrôleur dont les interrupteurs MOS aiguillent le courant, tout en présentant une résistance résiduelle Rc. On peut inclure Rc les résistances de connecteurs et câbles de liaisons dont la valeur peut aller de quelques mΩ à plus de la dizaine de mΩ.

Le moteur dont la résistance Rb des bobines s'oppose aussi au passage du courant, ainsi que la tension d'opposition engendrée par la vitesse de rotation. Dans la suite des communications elle se présente comme une tension quasi continue de valeur n/Kv, ou n est la vitesse de rotation en tours/min.



Enfin le courant Io. C'est le courant consommé par le moteur lorsqu'il tourne sans hélice. Il ne fournit aucune puissance mais le courant consommé n'est pas nul, parce qu'il faut un minimum de puissance pour vaincre les frottements, les pertes du circuit magnétique.

On pose ce courant du moteur à vide égal à Io, avec cette représentation, les pertes de puissance magnétiques et le frottement étant pratiquement constante, sont figurées par le courant Io, les pertes joule par Rb, il s'en suit que la puissance utile est donnée par :
$Ph=V_m.I_m=n/K_v$. Im en watts. C'est la puissance transmise à l'hélice.
Beaucoup de nos moteurs ont une valeur de « Io » comprise entre 1 et 2A.

Bilan des puissances :

- Puissance fournie par la batterie : ***Pbatt= V.I*** .
- Puissance perdue par effet Joule : ***Pj= (Rc+Rb) I²***.
- Puissance perdue par frottement et le circuit magnétique : ***Pf=Vm.Io***.

  Vm= V-(Rc+Rb) I, soit ***Pf= [V-(Rc+Rb) I] Io***.

Un critère usuel de bon fonctionnement est un bon rendement, rapport de la puissance utile sur la puissance fournie par la batterie **η** = ***Ph/Pbatt***

Avantages :

- ***Durée de vie et fiabilité :*** La longévité est équivalente aux moteurs AC et jusqu'à 4 fois supérieur à un moteur DC avec balais. Les soucis liés au frottement des charbons sur le collecteur disparaissent.
- ***Encombrement et poids :*** Plus compact que les moteurs AC et DC, un moteur sans balai est également 2 à 3 fois plus léger qu'un moteur DC traditionnel. Pour un poids et un encombrement identique.
- ***Rendement :*** bien supérieur à celui d'un moteur DC traditionnel à balais.
- ***Niveau sonore :*** Le bruit et les vibrations générés par un moteur sans balai sont inférieurs aux autres types de moteur.
- ***Variation de débit et asservissement :*** La technologie du moteur « brushless » avec son module électronique offre de nouvelles possibilités de variation de la vitesse. Plus de souplesse en effet, avec une plage de variation plus étendue et surtout le maintien du couple.
- Le moteur « brushless » se permet donc le cumul des avantages du moteur DC et AC. Voici une liste d'autres avantages intéressants : fort couple au démarrage, parasitage électrique limité, échauffement du moteur plus limité et refroidissement (perte joules) du moteur facilité par son architecture, indice de protection IP en général plus élevé du fait de l'absence de balais, moins d'inertie, moins de débris et résidus (pas d'usure des collecteurs).

Inconvénients :

- ***Cout*** : prix du moteur +contrôleur excessif.
- Consommation électrique.
- une certaine maitrise de l'engin est requise.
- transmission a renforcé.
- pièces de rechanges assez chers.

e. Les hélices :

Les caractéristiques techniques des hélices présentent deux valeurs : La longueur des pales et le pas géométrique, exprimé tous les deux en pouces.
Rappel sur ce qu'est le pas d'une hélice. Est appelé le pas, la distance que l'hélice parcourt en faisant un tour complet sans « glisser ». Autrement dit, en gardant une trajectoire linéaire. Le pas d'une hélice varie en fonction du nombre de pals de celle-ci, du poids de la machine et de la pression atmosphérique. C'est pourquoi il est important de faire la distinction entre le pas géométrique, qui la valeur théorique et inflexible, et le pas effectif, qui est le pas réel.

Pour résumer :

- Grande Hélice = beaucoup de portance → vol stable mais a besoin de puissance pour faire 1 tour entier → moteur important.
- Petite Hélice = peu de portance → vol moins stable et à besoin de moins de puissance pour faire 1 tour entier → moteur faible.

La taille de l'hélice ce fait en fonction de la portance (soit la masse à faire voler) dont on a besoin. Pour un Quadricopter par exemple, la taille varie entre 8" et 13" à cela il faut prendre en compte la taille du châssis : Pour un châssis de 30cm on peut monter à 9" max, sur 450cm on peut monter du 12" max.

- Petit pas = plus grande traction à faible vitesse, mais vitesse maxi limitée.
- Grand pas = plus petite traction à faible vitesse, mais vitesse maxi élevée.

La taille du pas géométrique ce fait en fonction de la vitesse que l'on veut atteindre.
Pour un Quadricopter, la moyenne et de 4,5". Il est toutefois possible de descendre à 3,5 (pour du vol lent) ou de monter à 6 (pour du vol rapide).

f. Testes et résultats :

Montage :

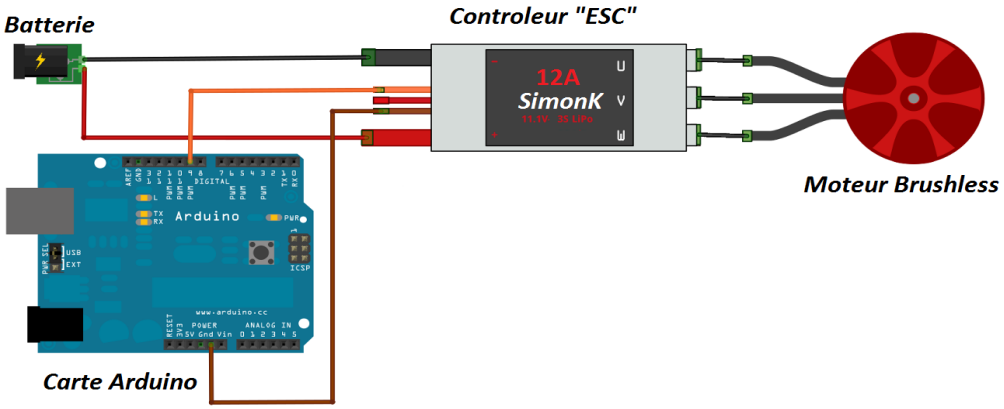


Figure 31 : schéma de câblage entre un moteur brushless et la carte arduino par le contrôleur "ESC"

Teste :

Dans l'arduino existe une bibliothèque qui s'appelle « Servo.h », cette dernière contient des commandes pour piloter les moteurs avec des signaux MLI ou « PWM ».
On a testé avec un programme qui commande les 4 moteurs avec un signal PWM de même rapport cyclique, en utilisant la commande « Servo mymoteur1...4 ; » pour définir le moteur à commander.
Donc pour faire tourner les moteurs faut envoyer un signal PWM au ESC avec un rapport cyclique entre 1ms et 2ms, puis le ESC il transfert ce signal à un signal triphasé compatible pour les moteur brushless.

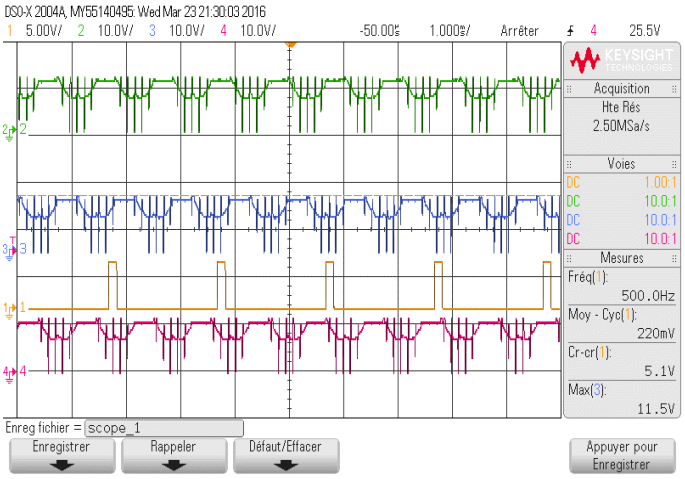


Figure 32 signal avant le ESC "jaune" et après le ESC "vert, bleu et rouge"

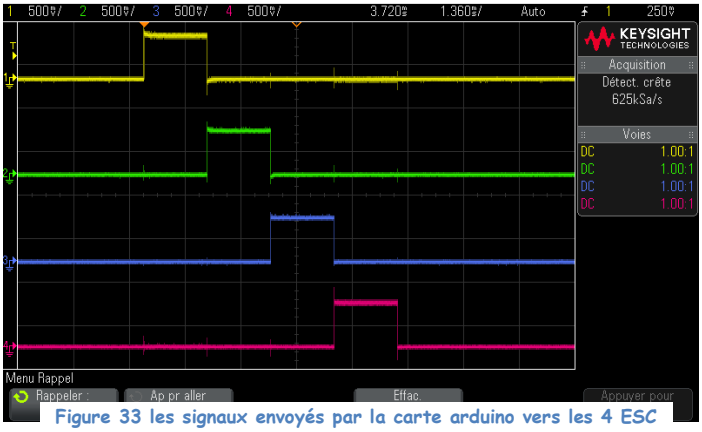
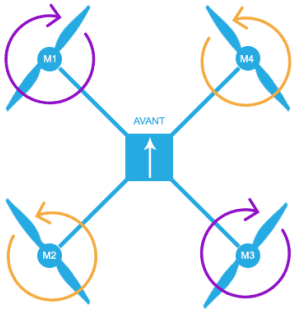


Figure 33 les signaux envoyés par la carte arduino vers les 4 ESC

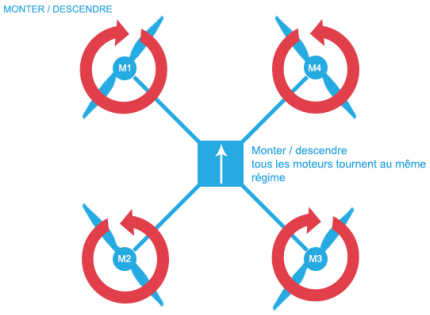
Assemblage final

a. Un Drone : comment ça vole ?

Ce chapitre est consacré au mode de déplacement d'un multicoptère dans les airs. En tournant, les hélices vont créer une force de sustentation (voir définition ci-dessous) qui vont compenser le poids de l'engin. Lorsque cette force est supérieure au poids du multicoptère, il s'élève dans les airs.
Définition sustentation : « la sustentation est l'effet d'une force qui maintient un corps à faible distance au-dessus d'une surface et sans contact avec elle » (source Wikipédia).



Le sens de rotation des hélices est très important, sur le schéma ci-dessous, on constate que les hélices situées sur le même axe tournent dans le même sens. En d'autres termes, M1 et M3 tournent dans le sens horaire et M2 et M4 dans le sens antihoraire. Pourquoi ? Cela annule le couple induit par l'effort sur chaque moteur pour faire tourner les hélices.

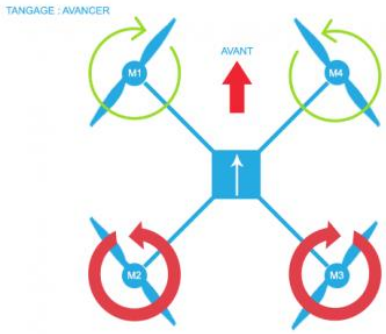


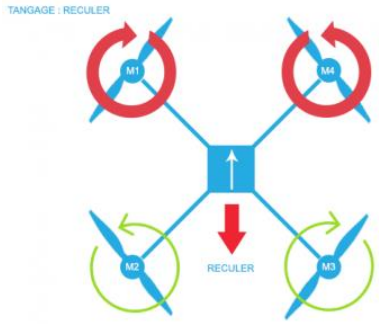
- **Monter / Descendre :**

Pour monter, on augmente la vitesse des moteurs simultanément, tous les moteurs tournent au même régime et inversement pour descendre, c'est la commande des gaz.

- **Le Tangage : Avancer / Reculer :**

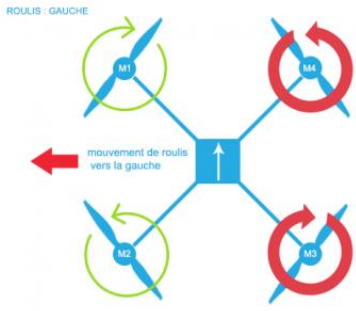
Pour avancer, on va diminuer la vitesse des moteurs avant et augmenter la vitesse des moteurs arrière et inversement pour reculer. On appelle cette action le « Tangage » ou « Pitch »

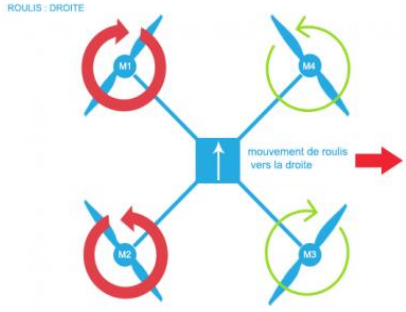




- **Le Roulis : Gauche / Droite**

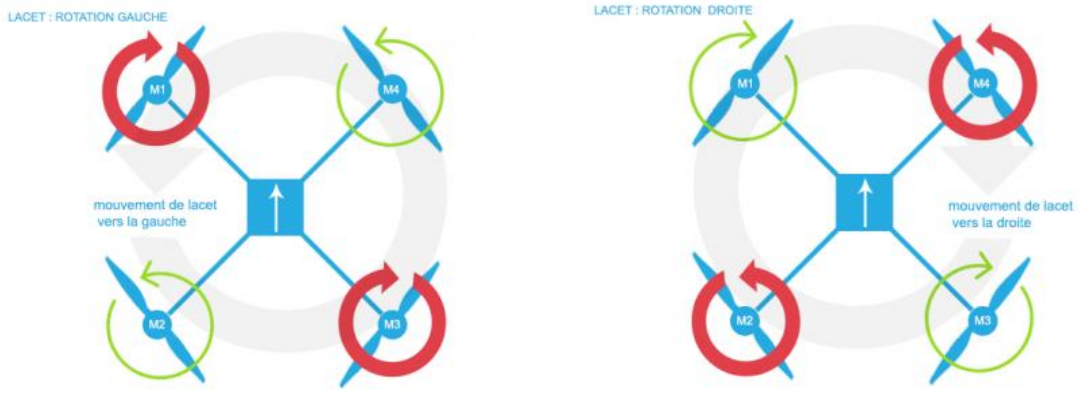
Pour incliner vers la gauche, on va diminuer les moteurs de gauche M1 et M2 et augmenter ceux de droite M3 et M4. Inversement pour incliner vers la droite. Cette action s'appelle le « Roulis » ou « Roll »





➢ **Le Lacet : Rotation :**

Pour un mouvement de rotation vers la droite ou la gauche, on va augmenter la vitesse d'une paire de moteurs sur le même axe tout en diminuant les moteurs du 2eme axe. Ceci est un mouvement de « Lacet » ou « Yaw »



b. Bilan énergétique :

	Moteur	ESC	Hélice	Batterie	Support	Contrôle
Quantité	4	4	4	1	1	1
Poids(g)	14	8	4	120	200	100
Poids Total	524g					

On choisit la batterie Lipo 3S avec la tension 11.1V et la capacité de 1300mah. On trouve dans le datasheet, la puissance de moteur est 3.6g/W. On calcule le temps de voler avec la formule suivant :

- Temps = 60/ (poids total / (tension * capacité * puissance))
- 60/(524/(11.1*1.3*3.6))=6mins

Par le calcul, on trouve que si nous achetons une batterie de 1300mah, le drone peut voler à vitesse maximal pour 6 mins. C'est suffit pour notre projet.

c. Schéma du Drone et contrôle

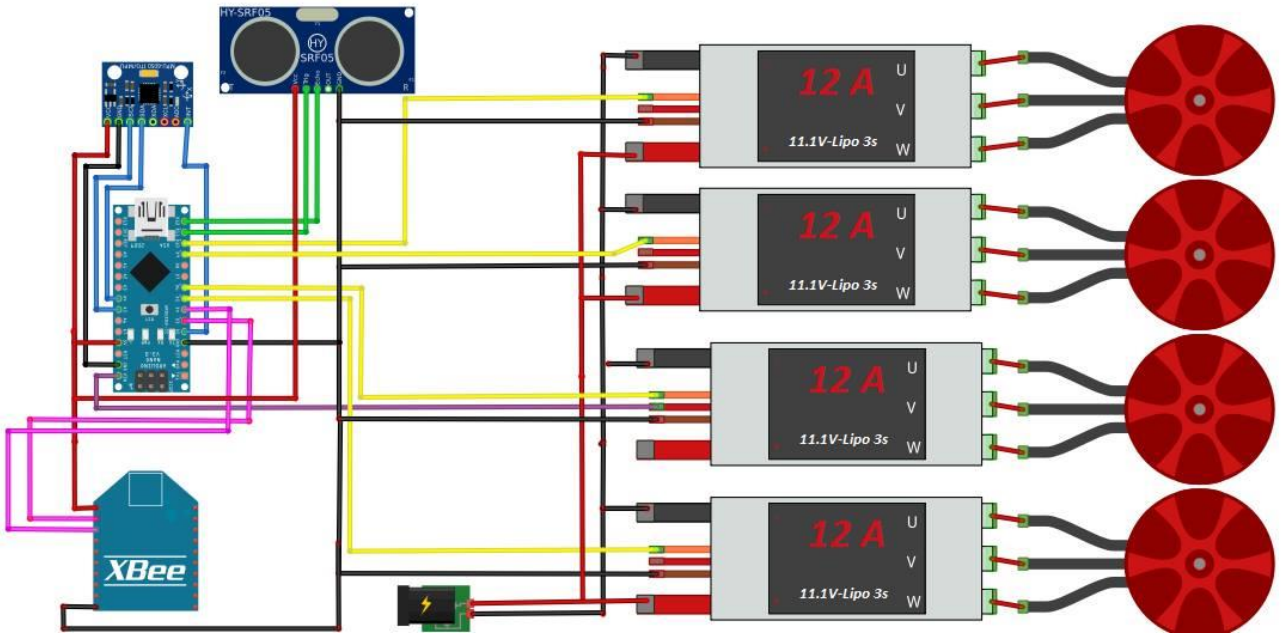


Figure 34 câblage du Drone

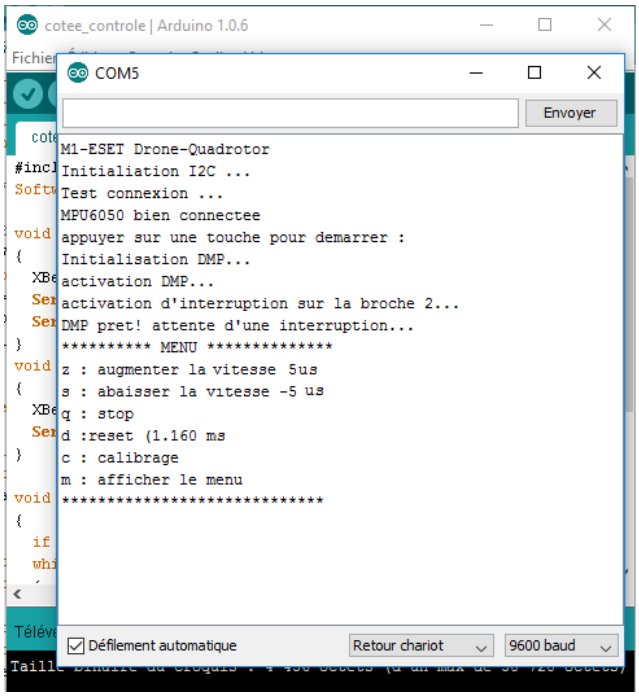


Figure 36 : démarrage de navigation et affichage de menu

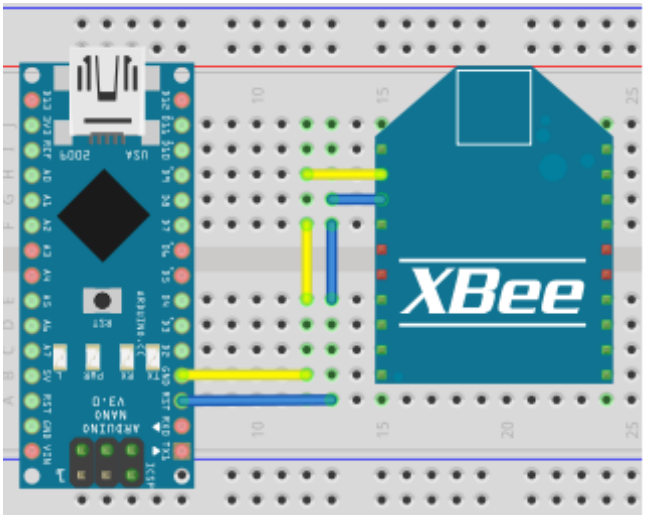
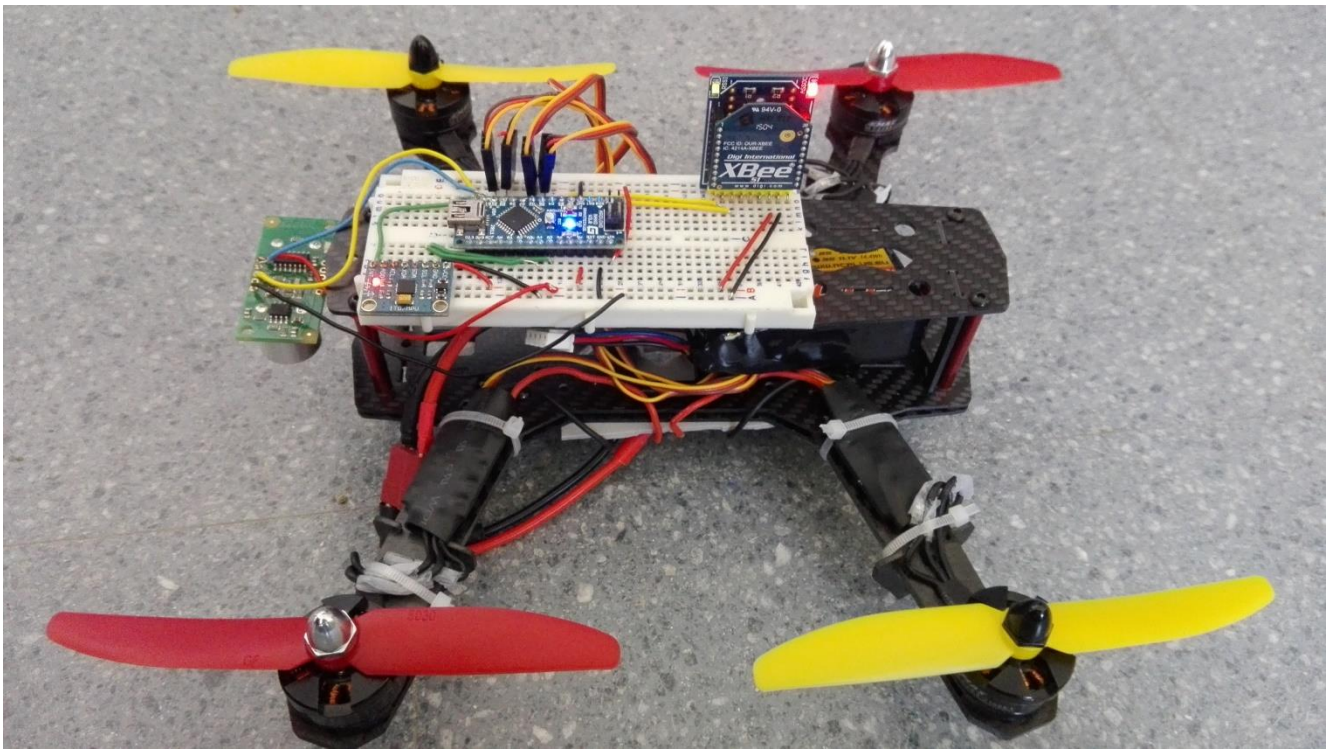
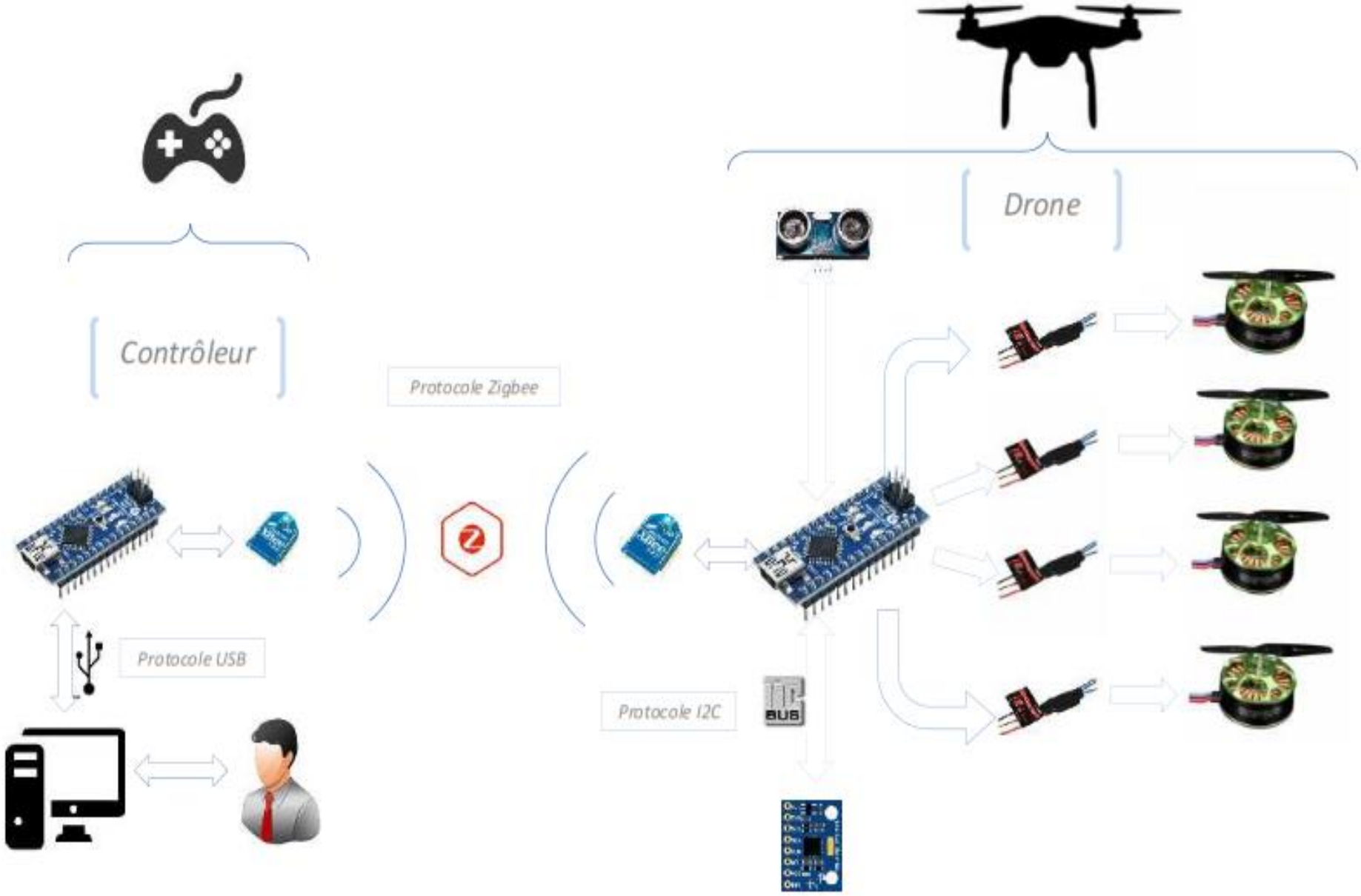


Figure 35 contrôleur de vol



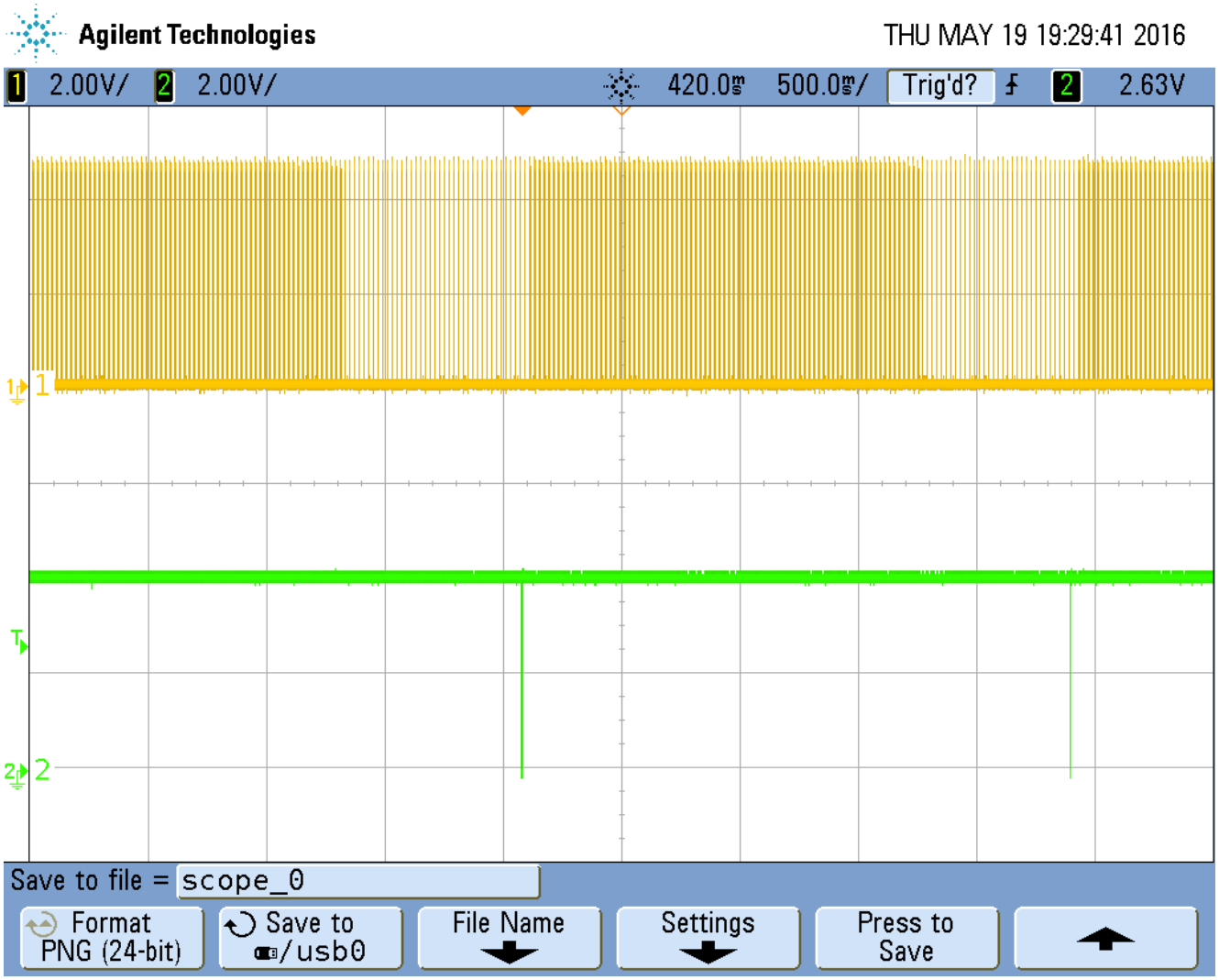


Shéma globale du système

d. Les problèmes rencontrés

Le long de la période de la réalisation de notre modeste travail, on a rencontré plusieurs problèmes matériels et logiciels, grâce au travail collectif qu'on a abordé on a réussi à les dépassé tous, mais malheureusement, à cause des contraintes du temps et l'obligation de la formation on n'a pas pu fixer le problème majeurs qui a bloqué le premier vol de notre drone, le problème se résume comme ceci :

La réception/émission des données via Xbee génère des perturbations inattendues au niveau des pins qui contrôlent les 4 moteurs comme le montre la figure suivante :



A chaque échange d'information/commande entre les deux systèmes (drone, contrôle) sur la voie verte génère un signal qui s'ajoute sur les moteurs de manière aléatoire, ce qui donne l'impression de la présence d'un régime transitoire qui dure trop long avant d'effectuer la commande et suivre la consigne.

On a procéder comme ceci pour le résoudre.

1. Vérifier les erreurs de communication : **pas d'erreurs de communication.**
2. Vérifier les rappels de courant causés par le xbee en lui séparant du système : **pas de rappels de courant**
3. Utiliser un autre uC que pour la prise de mesures à partir de la mpu6050 et qui communique par la voie série avec une deuxième carte qui fait que le pilotage des moteurs : **il y a toujours le problème**

En fin on a décidé de laissé le relai aux autres promos qui veulent travailler ou compléter notre travail.

Conclusion

Le but de ce travail consiste principalement à concevoir un drone piloté à distance a l'intermédiaire du protocole zigbee implémenté dans le XBee.

En fait, nous avons passé plus de temps dans l'intégration des composants utilisés, nous avons essayé de comprendre le fonctionnement de chaque composant en se basant sur les fiches technique.

Nous avons abordé d'une part les composants de la carte arduino et principalement son microcontrôleur, et d'autre part l'émission et la réception ultrasonique pour comprendre le fonctionnement de l'XBee et le gyroscope, et enfin la conception d'in système de contrôle basé sur un ordinateur, une carte arduino et un XBee.

L'étude menée dans le cadre de ce travail, nous a permis de :

- Comprendre le fonctionnement de la carte Arduino ;
- Comprendre le fonctionnement des capteurs ultrasoniques et le gyroscope;
- Elaborer un programme Arduino;
- Réaliser le drone sur un châssis commercialisé.
- Comprendre le protole Zigbee et I2C

Le présent rapport de fin d'étude traite dans son premier chapitre une présentation générale du matériel utilisé et les tests et montages réalisés, le capteur ultrason et les autres composants. Dans le deuxième chapitre nous avons parlé du montage final de notre drone ainsi que le contrôleur, les tests et la réalisation d'une maquette et après l'embarquement sur le châssis.

Enfin nous espérons, à travers ce travail, que notre projet soit à la hauteur des ambitions de notre encadrant, nos professeurs, nos familles et nos amis.

Bibliographie

http://embedded-lab.com/blog/lab-14-inter-integrated-circuit-i2c-communication/

http://jeromeabel.net/ressources/xbee-arduino

https://www.arduino.cc/

http://playground.arduino.cc/Main/MPU-6050

http://www.rcuniverse.com/magazine/article_display.cfm?article_id=1344

https://en.wikipedia.org/wiki/Electronic_speed_control

http://www.robotshop.com/blog/fr/comment-fabriquer-un-droneuav-leon-3-propulsion-4480

http://www.tomshardware.fr/articles/multirotor-quadcopter-fpv,2-914-6.html

http://electronique.marcel.free.fr/dsPIC/Programmes/BLDC/SVM_V2.pdf

http://www.jivaro-models.org/controleurs_brushless/page_controleurs_brushless.htm

http://cours.villemejane.net/geii/?wpfb_dl=133http://attie.co.uk/quad_copter/progress

http://www.droneaddict.net/comment-ca-vole/

http://www.wikipedia.net/

/************ parti Drone ****************/**

```
// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include <math.h>
#include <SoftwareSerial.h>
SoftwareSerial XBee(3,4);
#define SONAR_TRIGGER_PIN    11
#define SONAR_ECHO_PIN       12


#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h" //bibliotheque I2C
#endif
#include <Servo.h>
Servo mymotor1;//déclaration des moteurs
Servo mymotor2;
Servo mymotor3;
Servo mymotor4;
MPU6050 mpu;
float x,y,r;
#define OUTPUT_READABLE_YAWPITCHROLL
float erreur_precedente_x=0,erreur_x=0,variation_erreur_x=0,commande_x=0,somme_erreurs_x=0;
float erreur_precedente_y=0,erreur_y=0,variation_erreur_y=0,commande_y=0,somme_erreurs_y=0;
float val_speed=0,Kp=0.15,Ki=0.007,Kd=0.08;
float refx=0,refy=0;
char mode=' ';
int alt;
#define LED_PIN 13 // Led pour signaler le fonctionnement du gyro
bool blinkState = false;
bool dmpReady = false;  // met true si initialisation du DMP réussi
uint8_t mpuIntStatus;   // statut d'interruption
uint8_t devStatus;      // statut de dispositif
uint16_t packetSize;    // taille du packet DMP
uint16_t fifoCount;     // les octets dans le FIFO
uint8_t fifoBuffer[64]; // le buffer FIFO
Quaternion q;           // [w, x, y, z]        quaternion container
VectorInt16 aa;         // [x, y, z]           accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]           gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]           world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]           gravity vector
float euler[3];         // [psi, theta, phi]   Euler angle container

float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };

// ================================================================
// ===               ROUTINE  DE DETECTION D4INTERRUPTION       ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
   mpuInterrupt = true;
}
/************************** MENU DE COTROLEUR *********************************/
```

```
void menu(void)
{
   pinMode(LED_PIN, OUTPUT);
   XBee.println("********** MENU **************");
   XBee.println("z : augmenter la 5ms ");
   XBee.println("s : abaisser la vitesse -5 ms");
   XBee.println("q : stop ");
   XBee.println("d :reset (1.160 md) ");
   XBee.println("c : calibrage ");
   XBee.println("m : afficher le menu ");
   XBee.println("******************************");
//  char com=char(XBee.read());
  while ( !XBee.available () ){}
}
// ================================================================
// ===                    setup SETUP                       ===
// ================================================================
void setup() {
  pinMode(SONAR_TRIGGER_PIN, OUTPUT);
  pinMode(SONAR_ECHO_PIN, INPUT);

 // config moteurs
 XBee.begin(9600);
 Serial.begin(9600);
 mymotor1.attach(5);
 mymotor2.attach(6);
 mymotor3.attach(9);
 mymotor4.attach(10);

 mymotor1.write(0);
 mymotor2.write(0);
 mymotor3.write(0);
 mymotor4.write(0);
 XBee.println("M1-ESET Drone-Quadrotor");

 //**************************************************************

  // join I2C bus (I2Cdev library doesn't do this automatically)
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
     Wire.begin();
     TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
     Fastwire::setup(400, true);
  #endif

  while (!Serial);

  // initialisr device
  XBee.println("Initialiation I2C ...");
  mpu.initialize();

  // verify connection
  XBee.println("Test connexion ...");
  XBee.print(mpu.testConnection() ? "MPU6050 bien connectee" : "MPU6050 pas connectee");

  // attendre qu'il soit pret
  XBee.println("\nappuyer sur une touche pour demarrer : ");
  while (XBee.available () && XBee.read()); // buffer vide
  while (!XBee.available ());                 // attente des données
  while (XBee.available () && (XBee.read())); // buffer vide

  // load and configure the DMP
  XBee.println("Initialisation DMP...");
  devStatus = mpu.dmpInitialize();

  // fixation des OFFsets
  mpu.setXGyroOffset(220);
```

```
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 par defaut

    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        XBee.println("activation DMP...");
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        XBee.println("activation d'interruption sur la broche 2...");
        attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        XBee.println("DMP pret! attente d'une interruption...");
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        XBee.print("DMP Initialisation echouée ");
        XBee.print(devStatus);
        XBee.println(")");
    }
menu();

}

//================================================
//    fin partieMUP6050                    //
//================================================

// ================================================================
// ===                MAIN PROGRAM LOOP                  ===
// ================================================================


void get_ref()
{ //prendre la valeur de reference une seule fois
          refx=ypr[2] * 180/M_PI;
          refy=ypr[1] * 180/M_PI;
          XBee.println ("les valeurs de references sont refx et refy:");
          XBee.println("refx        refy");
          XBee.print(refx);
          XBee.print("    ");
          XBee.println(refy);
          XBee.println ("les commande:");
          XBee.println("Cx        Cy");
          XBee.print(commande_x);
          XBee.print("    ");
          XBee.println(commande_y);

          XBee.println(" Appuyer sur une touche pour demarrer");
          do{}while(!XBee.available ()); // wait fo a caracter for start
        }
//********************************** ultrason
int current_alt()
{
  // Trigger the SRF05:
  digitalWrite(SONAR_TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(SONAR_TRIGGER_PIN, LOW);
```

```
  // Wait for Echo Pulse
  unsigned long pulse_length = pulseIn(SONAR_ECHO_PIN, HIGH);

  // Ensure the ultrasonic "beep" has faded away
  delay(50);

  // Convert Pulse to Distance (inches)
  // pulse_length/58 = cm or pulse_length/148 = inches
  return( (unsigned int) (pulse_length / 58.1) );
}


void loop() {

if (!dmpReady) return;
   // wait for MPU interrupt or extra packet(s) available
   while (!mpuInterrupt && fifoCount < packetSize) {
      //    ne rie faire si le buffer et plein ou il y a pas d'interruption
      // .
      // .
      //
      //
      //
      // .
      // .
      // .
   }

   mpuInterrupt = false;
   mpuIntStatus = mpu.getIntStatus();
   fifoCount = mpu.getFIFOCount();
   if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
      mpu.resetFIFO();
   } else if (mpuIntStatus & 0x02) {
      while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
      mpu.getFIFOBytes(fifoBuffer, packetSize);
      fifoCount -= packetSize;
      #ifdef OUTPUT_READABLE_YAWPITCHROLL
         //display Euler angles in degrees
         mpu.dmpGetQuaternion(&q, fifoBuffer);
         mpu.dmpGetGravity(&gravity, &q);
         mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
           if(XBee.available ()){
             mode=char(XBee.read());
         }
         alt = current_alt();
         if(mode=='c') get_ref(); //prendre la reference
         r=ypr[0] * 180/M_PI;
         y=ypr[1] * 180/M_PI;
         x=ypr[2] * 180/M_PI;

// calcule de PID et correcion de la vitesse des moteurs
 //************************ asservissement x *****************************/
   erreur_x =refx-x;
   somme_erreurs_x +=erreur_x;
   variation_erreur_x = erreur_x - erreur_precedente_x;
   commande_x = Kp*erreur_x + Ki *somme_erreurs_x + Kd * variation_erreur_x;
   erreur_precedente_x = erreur_x;

   //************************ asservissement y *****************************/
   erreur_y =refy-y;
   somme_erreurs_y +=erreur_y;
   variation_erreur_y = erreur_y - erreur_precedente_y;
   commande_y =Kp*erreur_y+Ki*somme_erreurs_y+Kd*variation_erreur_y;
   erreur_precedente_y = erreur_y;
   //*********************************************************************/
```

```
      #endif
      blinkState = !blinkState;
      digitalWrite(LED_PIN, blinkState);
   }
    /****************** MENU ***************************/

 if (mode=='m') menu();

 if (mode=='z') {
            val_speed=val_speed+5;
            XBee.println("je suis dans z");
             }
 if (mode=='s') {

             val_speed=val_speed-5;
            XBee.println("je suis dans s");
            }
  if (mode=='q') {
             val_speed=0;
             mymotor1.write(val_speed);
             mymotor2.write(val_speed);
             mymotor3.write(val_speed);
             mymotor4.write(val_speed);
             XBee.println("je suis dans q");

             }
    if (mode=='d') {
             val_speed=1160;
             mymotor1.write(val_speed);
             mymotor4.write(val_speed);
             mymotor3.write(val_speed);
             mymotor2.write(val_speed);
             XBee.println("je suis dans d");
                }
     if (mode=='a') {
                XBee.print("alt : ");
                XBee.println(alt);
                while(!XBee.available());
               }
      mymotor1.write(val_speed+commande_y-commande_x);
      mymotor4.write(val_speed-commande_x-commande_y);
      mymotor3.write(val_speed+commande_x+commande_y);
      mymotor2.write(val_speed+commande_x-commande_y);
//*****************************************************************
         XBee.println("Axe X    Axe Y    Axe Z    Alt ");
         XBee.print(x);XBee.print("     ");
         XBee.print(y);XBee.print("     ");
         XBee.print(r);XBee.print("     ");
         XBee.println(current_alt());
mode=' ';
}
```

/********partie contrôle **********/**

```
#include <SoftwareSerial.h>
SoftwareSerial XBee(3, 4); // RX, TX

void setup()
{
  XBee.begin(9600);
  Serial.begin(9600);
  Serial.begin(9600);
}
void send_char(char c)
{
  XBee.print(c);
  Serial.println(c);
```

```
}

void loop()
{
  if (XBee.available()){
  while(XBee.available())
  { // If data comes in from XBee, send it out to serial monitor
    Serial.print(char(XBee.read()));
  }
  }
  if(Serial.available()){
  while(Serial.available())
  { // If data comes in from XBee, send it out to serial monitor
   XBee.print(char(Serial.read()));
  }
  }
}
```