Rapport de projet

Mai 2013

Réalisation de commande sans fil et télémétrie sur un drone

Tom OBRY, Maxime ROBLIN
L3 EEA REL 2012/2013
Faculté Paul Sabatier, 31000 TOULOUSE

Sommaire

Partie 1
Préparation

I. Présentation du projet

Le sujet de ce projet est la réalisation d'une télémétrie sur un quadrirotor piloté par une télécommande : le UFO 5 du fabricant Walkera. Ce projet a été en partie réalisé par un élève (master 1ere année) l'année précédente.

Le but du projet est donc de terminer, optimiser et améliorer son travail à l'aide de son rapport et de nos connaissances.

Voici le schéma fonctionnel de l'ensemble du projet (Drone + organes de commande) :

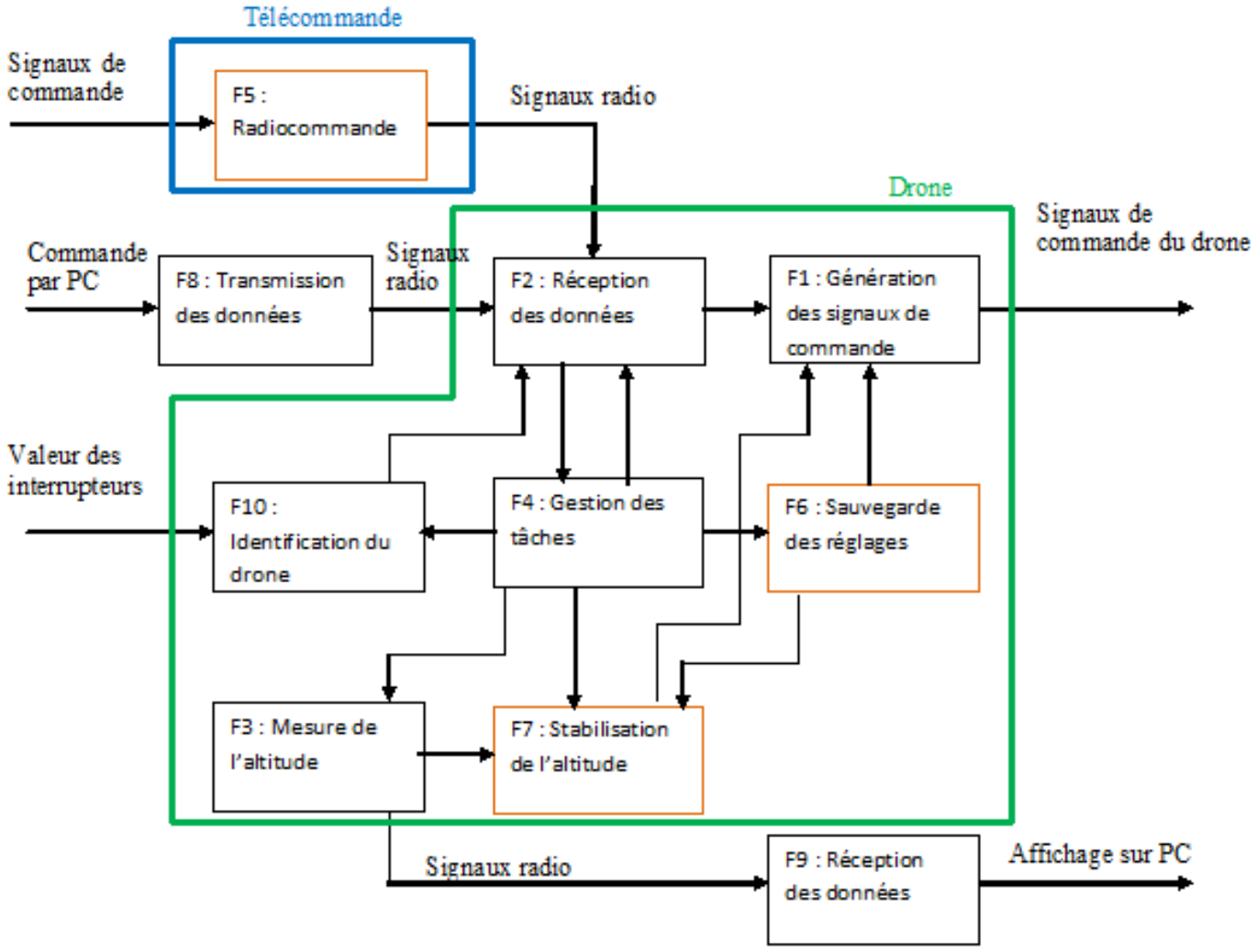


Schéma fonctionnel
Les fonctions en orange sont des fonctions ajoutées afin d'améliorer le projet.

Entrées :

- Signaux de commande : Signaux 0-5V (PWM avec mixage temporel) image des commandes (sticks)
- Commande par PC : Nombre(s) codés en hexadécimal
- Valeur des interrupteurs : états logiques 0 ou 1

Sorties :

- Signaux de commande du drone: 4 signaux 0-5V (PWM avec mixage temporel) permettant de contrôler le drone dans toutes les directions
- Affichage sur PC : Nombres décimaux

II. Choix du matériel

Pour créer les prototypes, nous avons gardé le materiel de notre prédécesseur :

- Programmation sous plateforme ARDUINO : http://www.arduino.cc/
  C'est une plateforme open-source comprenant un logiciel de programmation gratuit ainsi que des carte de prototypage préconçues que l'on peut acheter ou fabriquer soi-même à partir des schémas open-source. La carte de base « Uno » en notre possession comporte un nombre suffisant d'entrées/sorties et est peu onéreuse.

Carte Arduino Uno

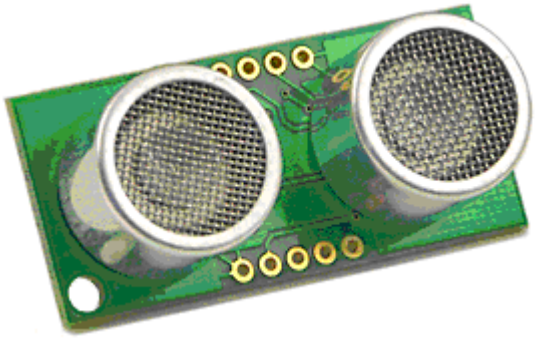
- Communication sans fil par XBee : http://www.digi.com/fr/products/wireless/point-multipoint/xbee-series1-module
  Utilise le protocole standardisé ZigBee. Il est très répandu donc nous pouvons trouver très facilement des tutoriels sur internet, des librairies, il est pas trop onéreux et surtout, le XBee est facile d'utilisation.

Nous avons donc utilisé 2 XBee pro serie : 1 pour réaliser une liaison série sans fil et 1 avec le Arduino Shield pour le connecter à la carte Arduino Uno.

Arduino Shield

XBee Pro Serie 1

- Capteur d'altitude SRF05 : (voir annexes)
  Capteur à ultrasons. Peu onéreux, précis (de l'ordre du cm), simple de mise en oeuvre.



Capteur d'altitude SRF05

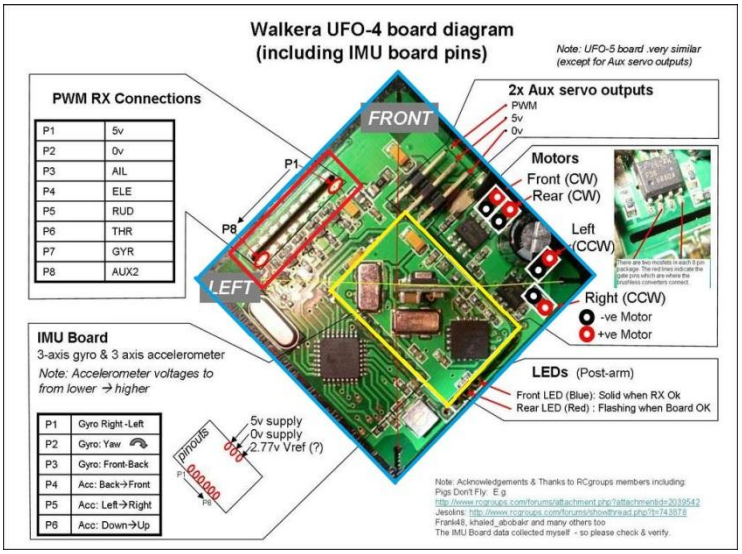
Partie 2
Travaux réalisés

I. Préparation du drone

Le drone est constitué de plusieurs cartes :

- Une carte principale
- Une carte gyroscope / accéléromètres
- Une carte récepteur

La carte principale a pour rôle de gérer l'alimentation ainsi que la gestion des moteurs en fonction des différents signaux reçus.
La carte gyroscope / accéléromètres permet de gérer l'équilibre et la stabilité du drone automatiquement.
La carte du récepteur permet de recevoir les données émises par la télécommande.

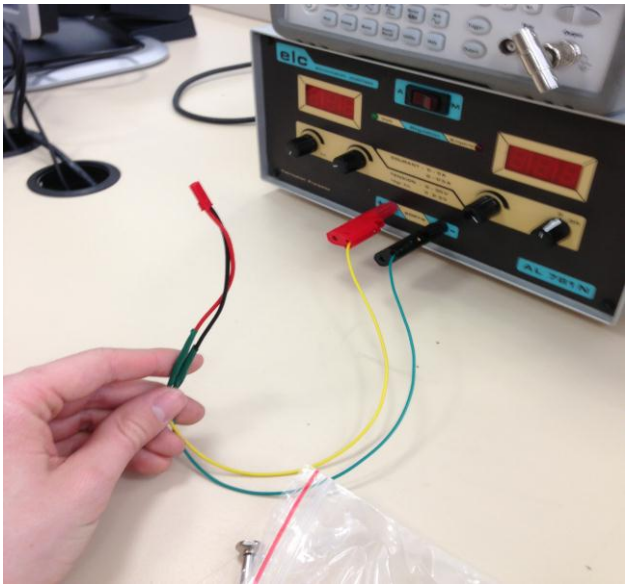


: Carte principale
: Carte récepteur
: Carte gyro / acc

Schéma de l'ensemble de contrôle embarqué sur le drone

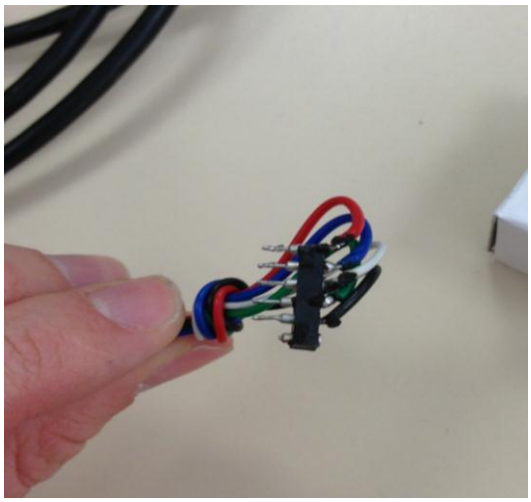
La première chose que nous avons remarquée lors du premier contact avec le drone est que l'élève précédent avait laissé les 2 fils permettant de brancher la batterie, or n'ayant pas à ce moment de batterie adaptée aux besoins du quadrirotor, nous avons pris la décision d'enlever ces fils pour souder des fils avec des fiches bananes. Cette manipulation nous a permis d'alimenter le drone pour les tests avec une alimentation stabilisée de laboratoire en attendant les fils et batteries adaptées.

Par la suite, nous avons soudé des connecteurs BECs (connecteurs d'origine) à des fiches banane (cf photo ci-après) ainsi que sur la carte récepteur pour pouvoir switcher facilement entre alimentation par batterie et par alimentation stabilisée.



Connecteur BEC avec fiches bananes

De plus, nous utilisons les fils de commande (réalisés par notre prédécesseur) pour les tests de chaque fonction mais avec un connecteur pour pouvoir facilement les remplacer par une carte récepteur :



Fils de commande avec connecteur

Il y a 4 fils de commande (1 par voie) et 1 fil de masse.

II. Programmation

Pour créer les différents programmes, nous avons téléchargé et installé le logiciel Arduino à l'adresse suivante : http://arduino.cc/en/Main/Software
Une fois le logiciel installé, nous nous sommes aidés principalement du tutoriel disponible sur le site Arduino : http://arduino.cc/fr/Main/Debuter

La fonction F10 étant reprise de l'élève précédent, nous ne la détaillerons pas : il s'agit simplement de récupérer une valeur décimale à partir d'interrupteurs avec un codage binaire. De plus, nous ne l'avons pas incluse dans le programme final car trop contraignante pour la mise au point. Nous avons donné directement une valeur ID_reseau = 15 dans l'initialisation des variables.
Nous n'avons également pas décrit les programmes principaux des phases de test car il s'agit d'un simple appel de fonction.

Les appels de fonction dans les schémas fonctionnels proviennent du programme principal (gestion des tâches).

Important : il est conseillé de lire le programme final en annexe car certains morceaux du programme utilisé par plusieurs fonctions (tel que la déclaration de variable) ne sont pas réécrits.

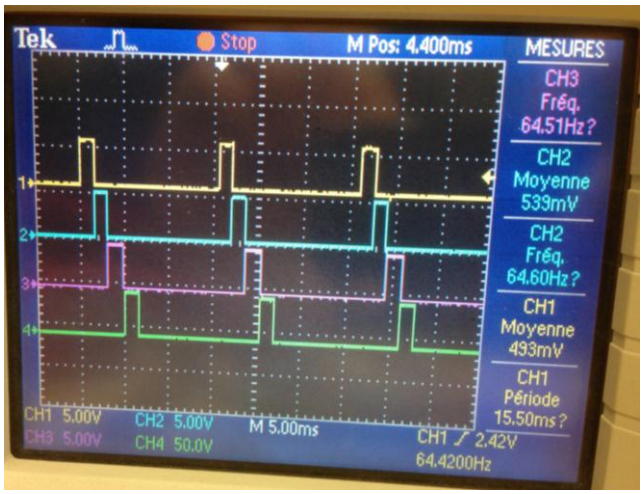
1) F1 : Génération des signaux de commande

A la sortie du récepteur, le drone (carte principale) reçoit 4 signaux de type PWM (en 0-5V max) de période 15,5ms multiplexés dans le temps (CH1 puis CH2 puis CH3 puis CH4). (plus d'info à cette adresse : http://home.nordnet.fr/~fthobois/MRA-1.htm)
Ils sont de rapport cyclique variables correspondant à la position de chaque manette et donc au pourcentage de chaque commande :

- *Voie/CH 1 : Avancer/Reculer*
- *Voie/CH 2 : Gauche/Droite*
- *Voie/CH 3 : Puissance des moteurs*
- *Voie/CH 4 : Rotation*

Le rapport cyclique varie entre 1ms et 2ms, 1ms correspondant à la position basse, 2ms à la position haute et 1.5ms à la position neutre.

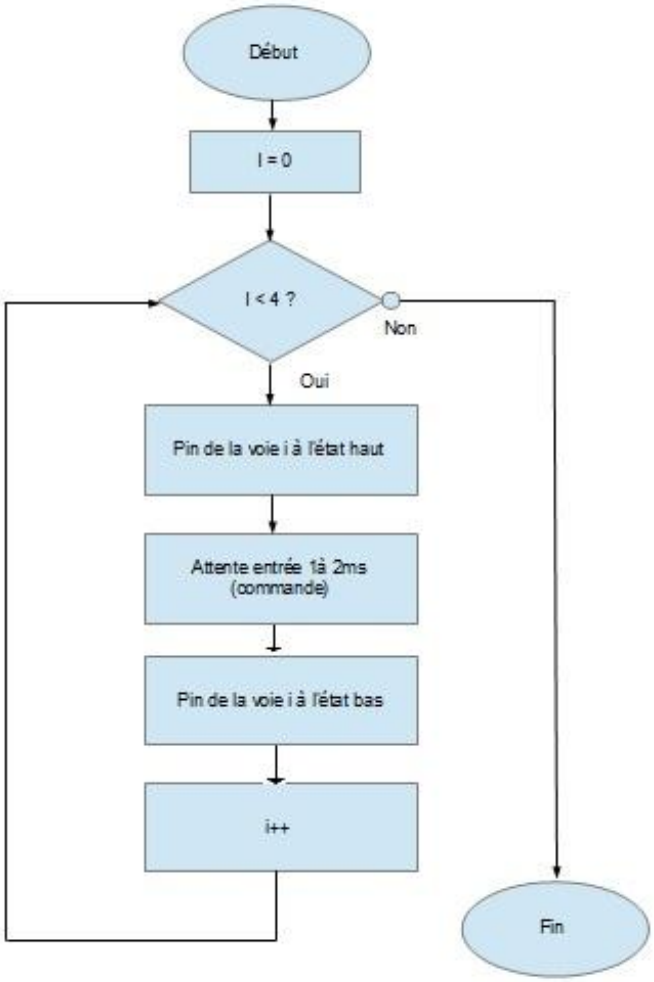
Une trame est ainsi composée de 4 signaux carrés de durée variable dans cet ordre : CH1 puis CH2 puis CH3 puis CH4.
Les trames se répètent toutes les 15,5ms, mais il semble que le drone réagit correctement si l'on dépasse cette valeur, même de manière aléatoire.



Composition d'une trame

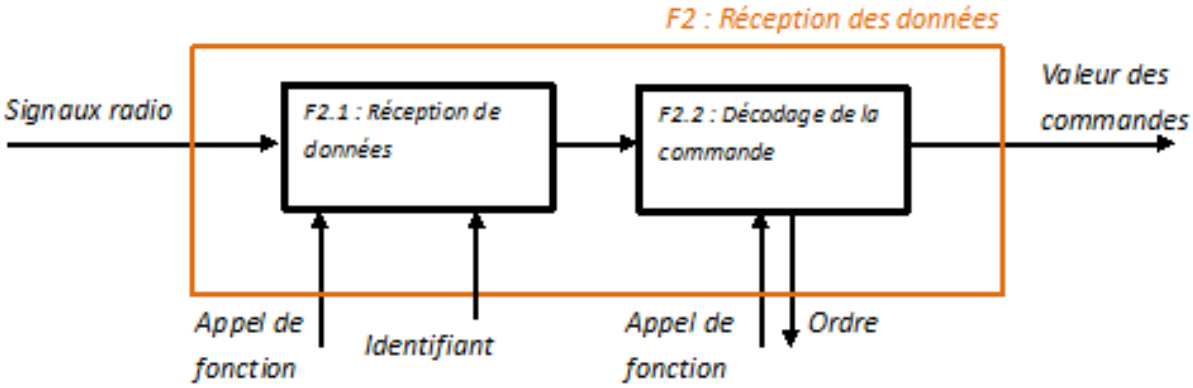
Nous avons tenté différentes sortes de programme (par interruption, en fonction du temps…) afin de gagner du temps d'exécution pour les autres fonctions, mais cela n'était finalement pas nécessaire (nous avons testé le comportement du drone pour des valeurs aléatoires de temps entre chaque trame très tard).

Voici l'algorithme du programme de génération de signaux que nous avons mis au point :



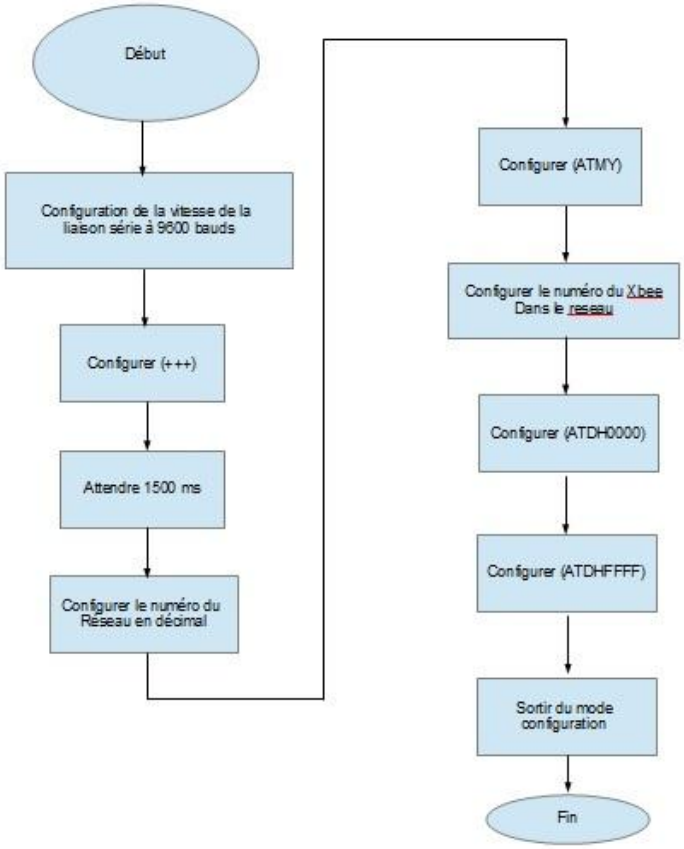
Algorigramme pour générer les signaux de commande

2)F2 : Réception des données



La fonction F2.1 est réalisée par l'ensemble XBee et USART du microcontrôleur(préprogrammé), le XBee servant de transmission série entre l'organe de commande et le drone.
La fonction F2.2 est réalisée uniquement par le microcontrôleur : elle permet de lire les données reçues et au programme principal de déterminer que doit faire le drone (augmenter/diminuer la puissance moteur, renvoyer l'altitude, commande manuelle…)

Le rôle du module XBee est donc de créer une liaison sans fil entre 2 de ces modules ou plus au sein d'un même réseau pour faire passer des informations tel que des signaux de commandes de pilotage de drone ou encore des information sur l'altitude à laquelle se situe le drone.
Il faut dans un premier temps configurer le XBee :

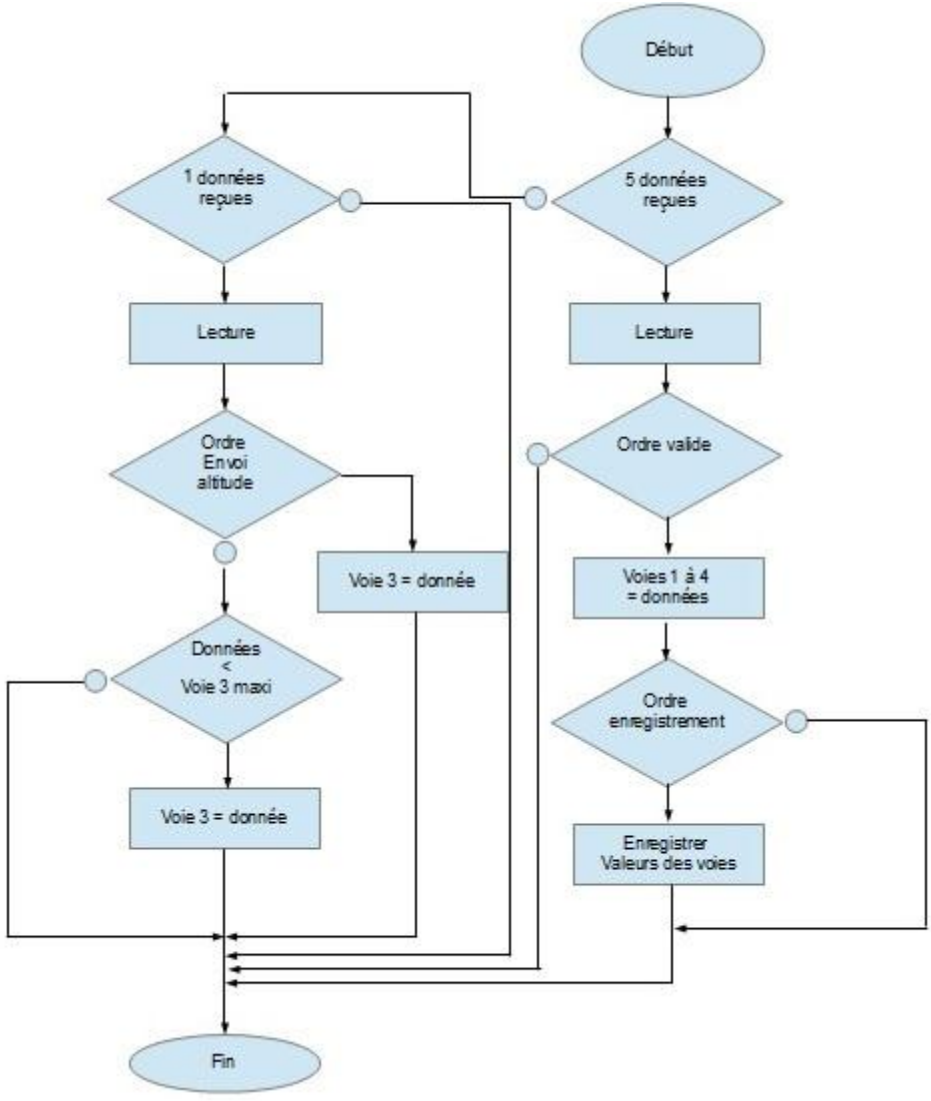


Algorigramme d'initialisation du XBee

Voici l'explication des différentes commandes de ce programme :

- (+++) : permet de mettre le XBee en mode configuration.
- (ATMY) : permet de configurer l'identifiant du XBee.
- (ATID) : permet de configurer l'identifiant (numéro) du réseau.
- (ATDH) : permet de configurer la limite haute des adresses de provenance acceptées par le XBee.
- (ATDL) : permet de configurer la limite basse des adresses de provenance acceptées par le XBee. (ici toutes)
- (ATCN) : permet de stopper la réception des données.

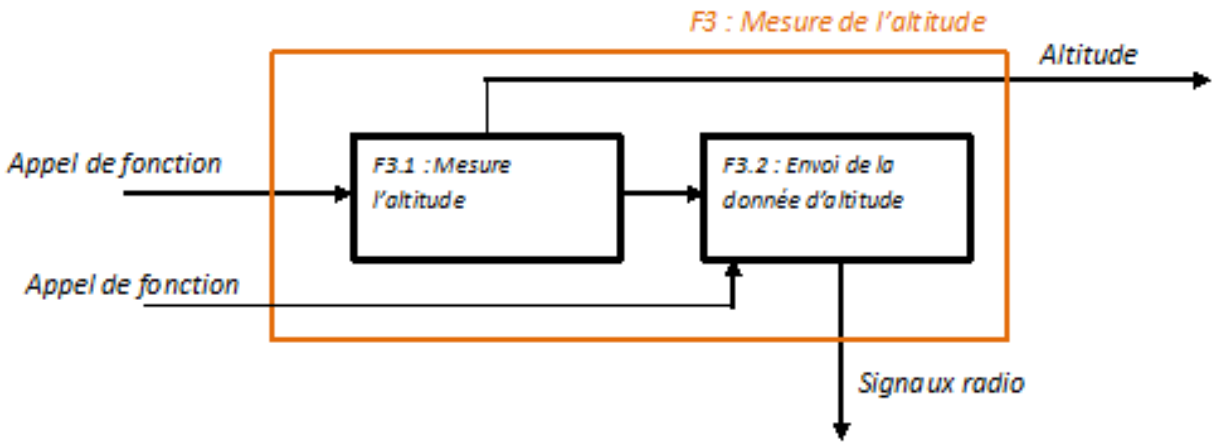
Nous programmons ensuite la lecture des données (F2.2) :



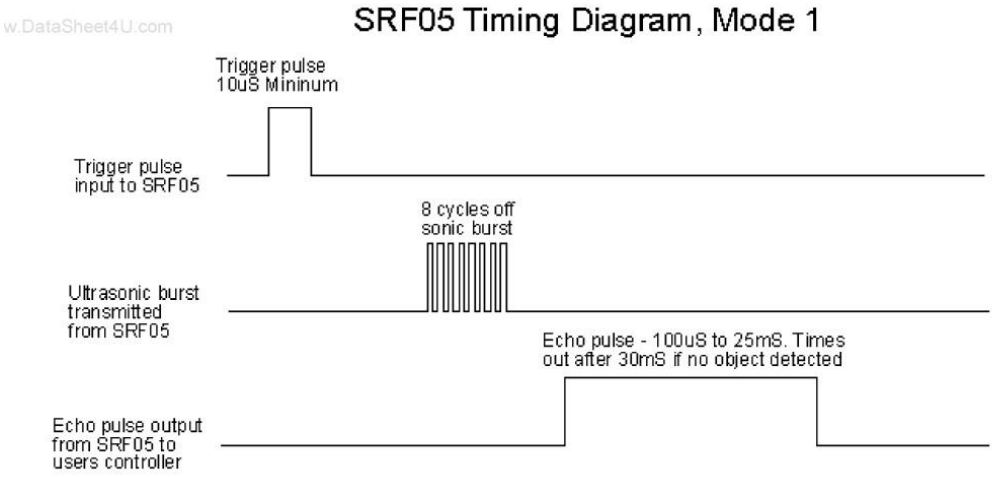
Algorigramme de lecture des trames

Cet algorithme permet de lire 1 donnée (octet) ou un paquet de 5 données, en fonction du nombre d'octets disponible sur la liaison série et les enregistre dans un tableau. Une partie de F2.2 est en réalité programmée sur F4 : il s'agit de la gestion des signaux du drone en fonction de l'ordre reçu.

3)F3 : Mesure de l'altitude

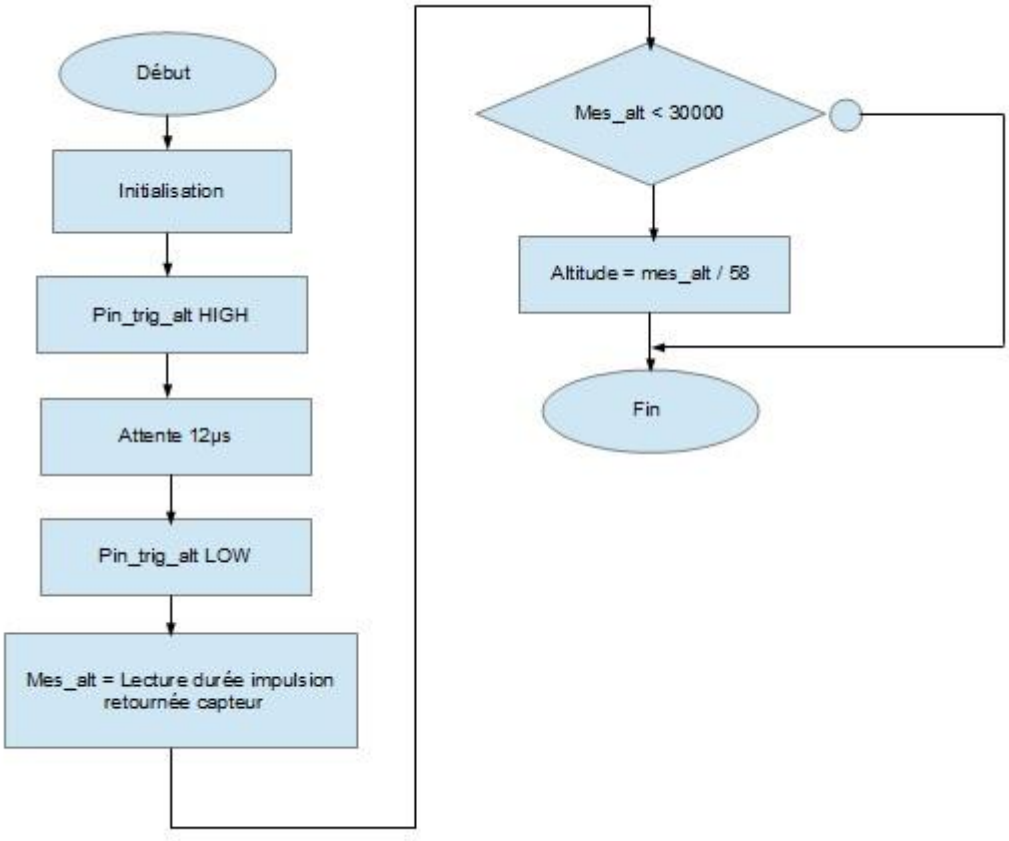


Le capteur d'altitude (contenu dans F3.1) permet de retourner une distance lorsque nous envoyons une certaine commande. Le principe est le suivant : Nous envoyons une impulsion de minimum 10µs. Cette impulsion représente l'ordre qui permet de lancer la mesure d'altitude. Une fois cet ordre lancé, le capteur envoie 8 impulsions d'ultrasons. Une fois cette trame envoyée, le capteur attend le retour de cette dernière. Le capteur détermine enfin la distance entre lui-même et le sol grâce au temps d'écho et renvoi cette valeur sous forme d'un signal carrée dont la durée est proportionnelle à la distance.
Voici un chronogramme expliquant le mode de fonctionnement du capteur :



Chronogramme de temps du fonctionnement du capteur

Voici l'algorithme permettant d'obtenir la distance entre le capteur et le sol :



Algorithme permettant d'obtenir la distance entre le capteur et le sol

La fonction 3.2 permet d'envoyer à l'ordinateur l'altitude sous forme ASCII (via liaison série/XBee)

4) F8 et F9 : Transmission et réception de données avec un PC

Ces fonctions sont réalisées grâce à la carte « Xbee explorer USB » (marque Sparkfun, référence WRL-08687) et le logiciel X-CTU.

Xbee explorer USB avec XBee

Il faut d'abord commencer par télécharger les pilotes à l'adresse suivante :
http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=0&s=316&tp=1
Puis lancer l'application CDM20600.exe.
Puis télécharger le logiciel XCTU à l'adresse suivante :
http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=57&s=316&tp=5&tp2=0
Puis lancer l'application XSCTU 40002637.exe.
Un guide d'utilisation de ce logiciel (assez sommaire) est disponible à l'adresse suivante :
http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=57&s=316&tp=3&tp2=0

Il faut au préalable configurer le XBee de la manière suivante (dans l'onglet « modem configuration », read) :

- ATMY : différent de 5.
- ATID : identifiant (numéro) du réseau (pour nous 15)
- ATDH : 0000 ATDL : FFFF

Puis cliquer sur « write ». Nous pouvons maintenant utiliser X-CTU et envoyer des commandes en hexadécimal (onglet « Terminal », « Assemble Packet » et sélectionner « Hex »).

III. Tests

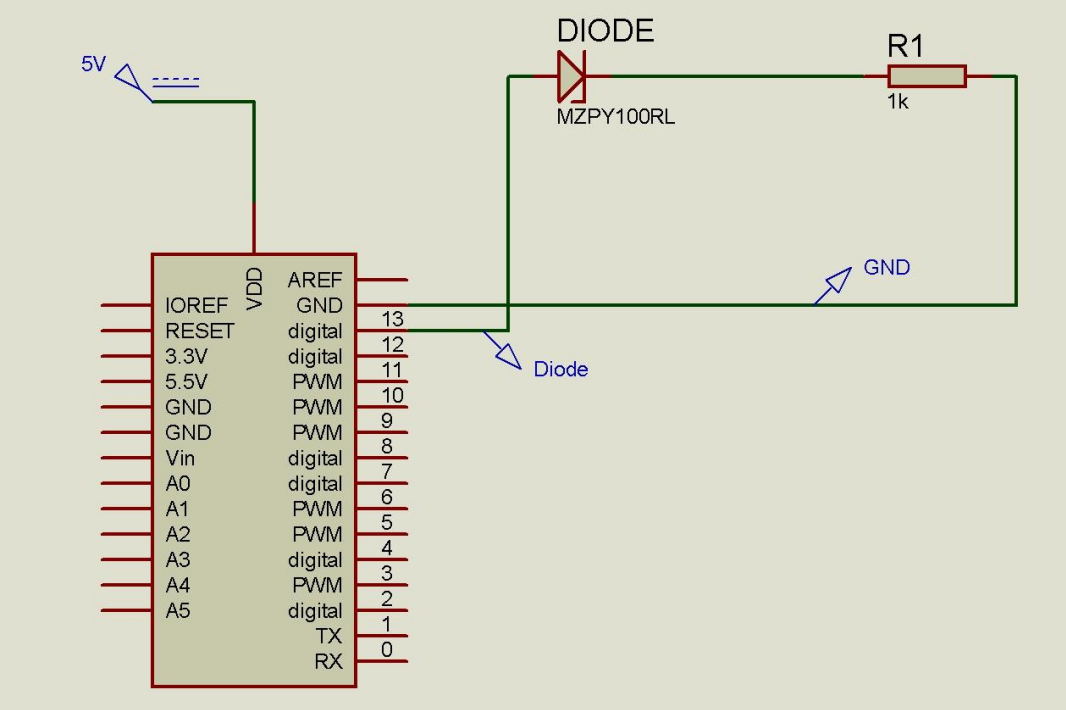
Avant de mettre au point le programme complet, nous avons dû faire une série de tests afin de valider chacune de nos fonctions. Malheureusement, l'environnement Arduino ne possède pas de débugger : il faut écrire dans le programme l'envoi des valeurs des variables qui nous intéressent.

1) Génération des signaux de commandes

Les éléments de cette maquette sont :

- 4 potentiomètres qui permettent de simuler la télécommande.
- Le Arduino Shield qui est la carte de prototypage.
- Le drone connecté avec les fils et non la carte récepteur.

Tout d'abord, voici le câblage de test mis au point pour tester 1 voie :

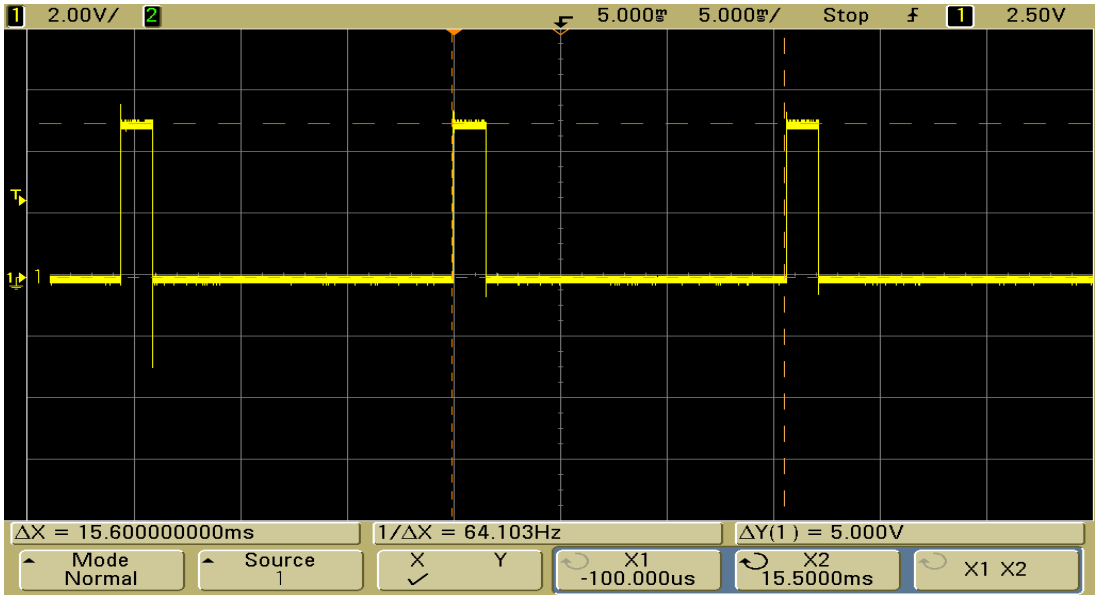


Câblage de test du programme de génération d'un signal

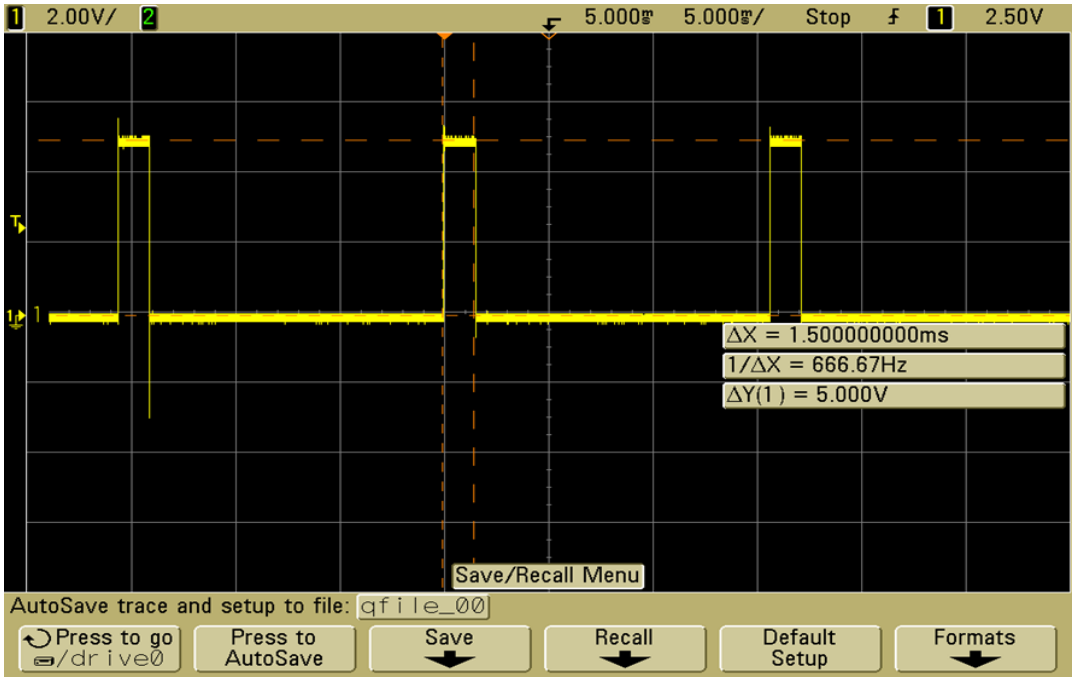
Voici le programme permettant de générer ce signal :

```
int ledPin = 13; // La led est connectée a la broche 13
void setup ()
{
 pinMode (ledPin, OUTPUT); // Je mets la broche en sortie
}
void loop ()
{
 digitalWrite (ledPin, HIGH); // j'allume la led
 delayMicroseconds (1500); // je mets le delay à 1.5 ms en position haute
 digitalWrite (ledPin, LOW); // j'éteins la led
 delayMicroseconds (14000); // je laisse la led éteinte durant 14 ms
```

Voici les résultats observés à l'oscilloscope pour 1 voie (Diode) :



Signal PWM de période 15.5 ms et une tension de +5V



Temps « haut » de 1.5 ms

Une fois ce test concluant, nous avons programmé et testé un programme permettant de gérer les différentes commandes de pilotage du drone avec 4 potentiomètres, dont voici le montage :

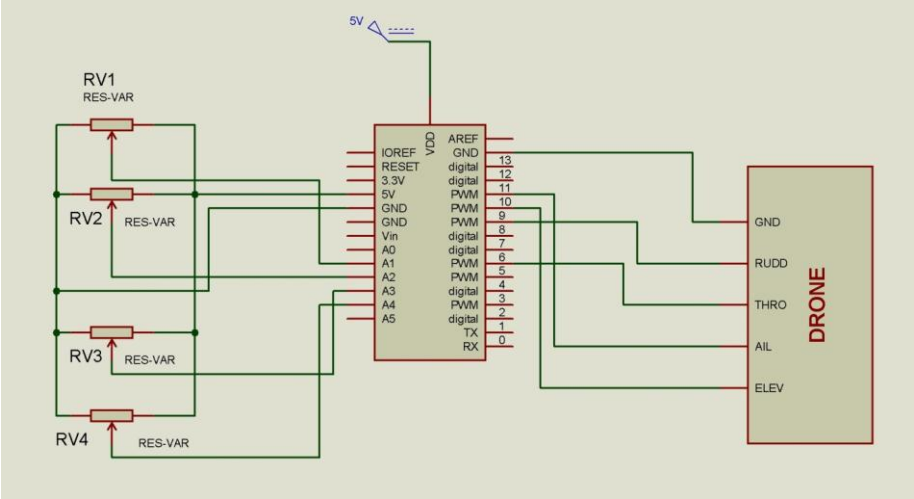


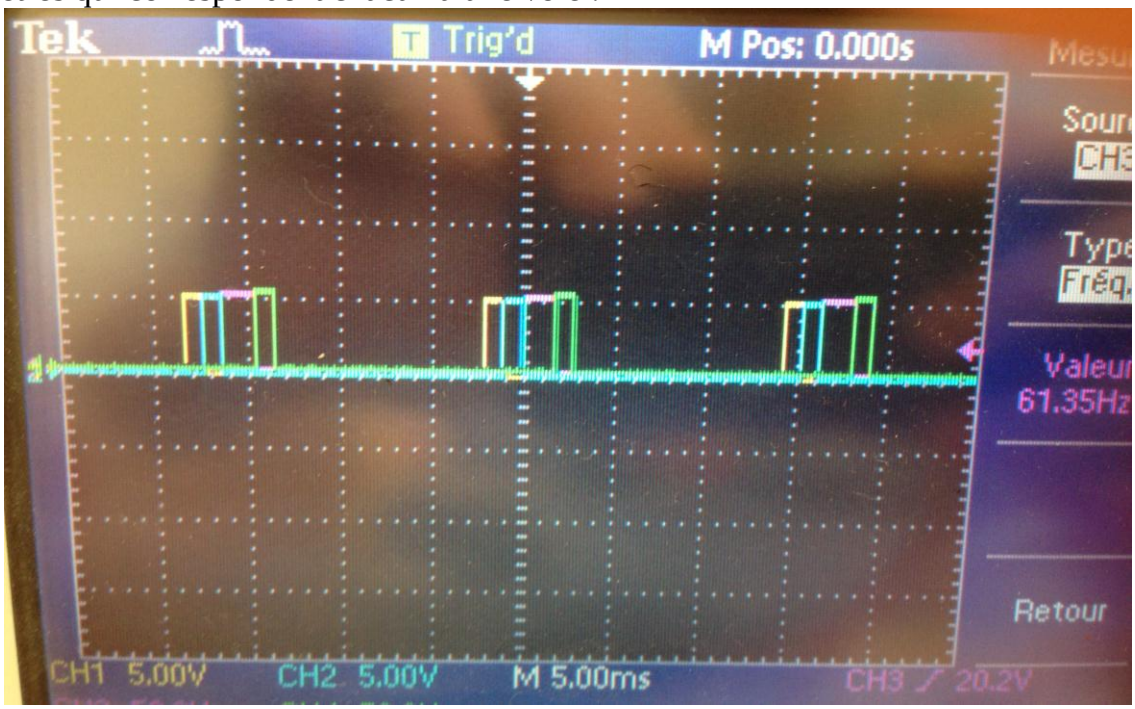
Schéma de câblage de test avec les 4 potentiomètres

Voici le sous-programme permettant de générer les signaux (la lecture des potentiomètres n'y est pas décrite mais se situe dans le programme principal, où la lecture des potentiomètres puis l'appel de la fonction de génération des commandes est effectué en boucle)

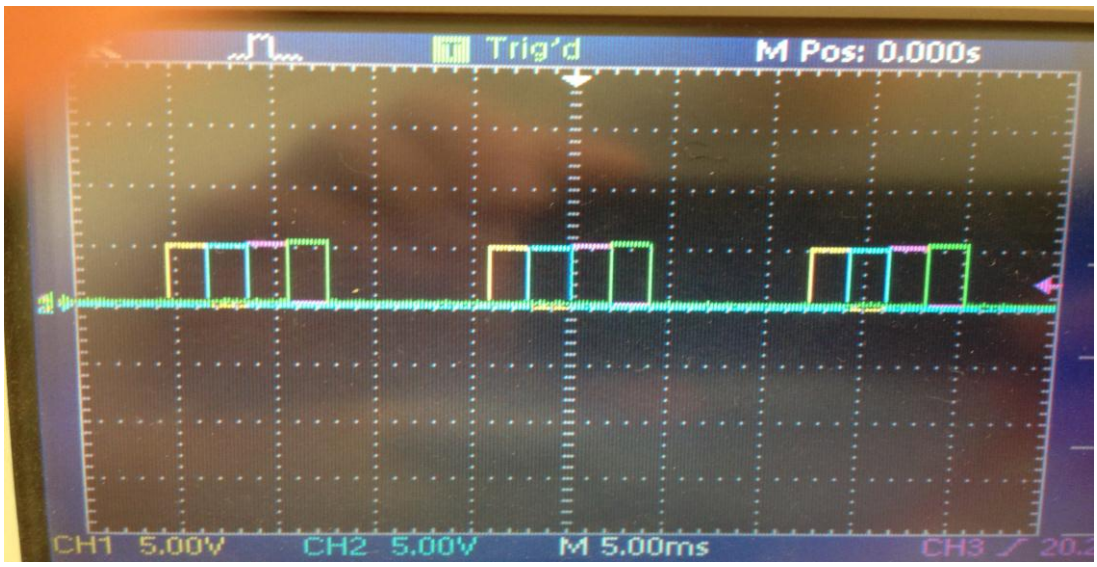
```
//Déclaration des variables
const int pin_CH[4]={9,10,11,12}; //Numero de chaque pin selon la voie
const int pot_CH[4]={1,2,3,4};  //Numero de chaque pin selon le potentiomètre
unsigned int CH[4]={0,0,0,0};    //Valeur de commande de chaque voie

//Generation des commandes
void gen_commande()
{
  int i;
  for (i=0 ; i<4 ; i++)           //Pour chacune des voies
  {
    digitalWrite(pin_CH[i], HIGH);     //Mise à l'état haut
    delayMicroseconds(950+CH[i]);  //Attente correspondant au rapport cyclique (valeur du potentiomètre)
    digitalWrite(pin_CH[i], LOW);      //Puis remise à l'état bas
  }
}
```

Programme génération de commandes

Nous avons effectué différentes captures d'écrans à l'oscilloscope en manipulant les différents potentiomètres qui correspondent chacun à une voie :



Relevé de chaque voie en position « basse »



Relevé de chaque voie en position « haute »

Nous avons donc les 4 signaux PWM (1 par voie) de période 15,5ms et de rapport cyclique variant (1 à 2ms) selon la valeur de son potentiomètre associé, comme voulu.

2) Le capteur d'altitude :

Le capteur d'altitude permet de retourner une valeur d'altitude à n'importe quel moment, à partir du moment où l'on envoie la commande « FF ».
Nous avons repris le programme de l'étudiant précédent pour nous inspirer de la structure du programme.

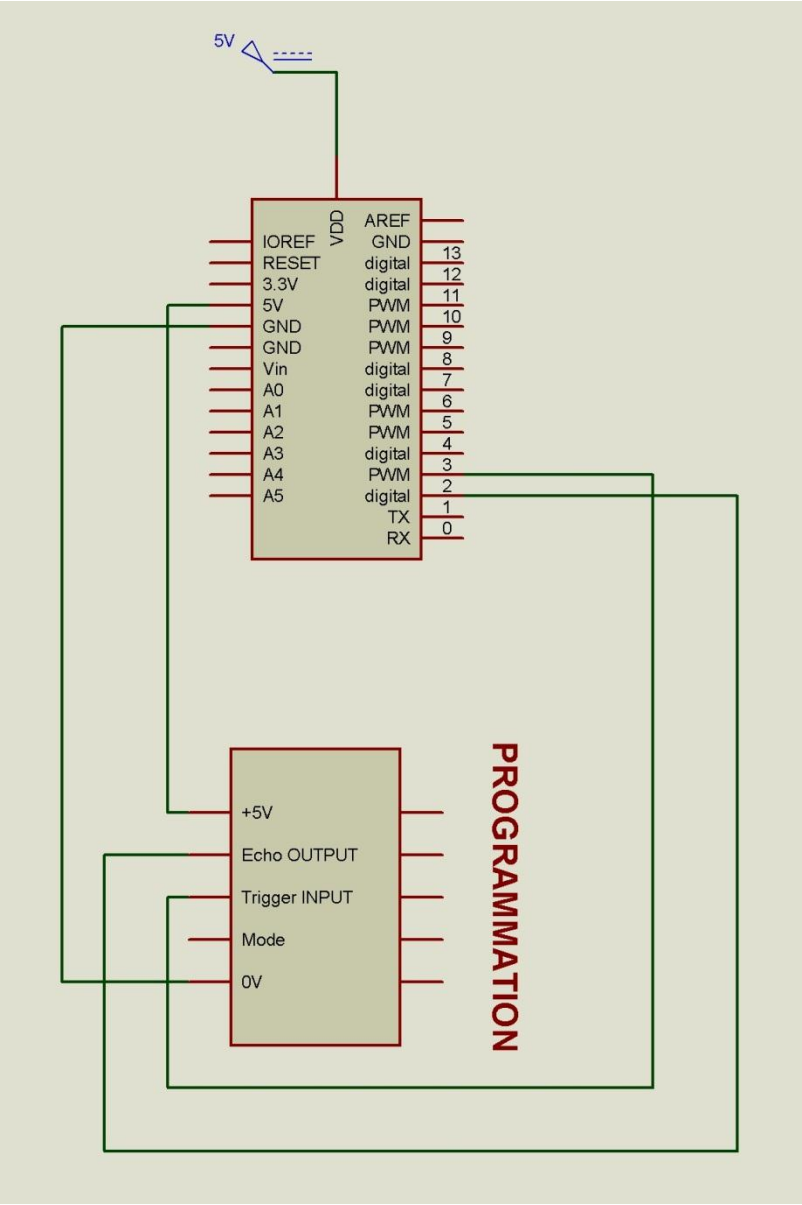
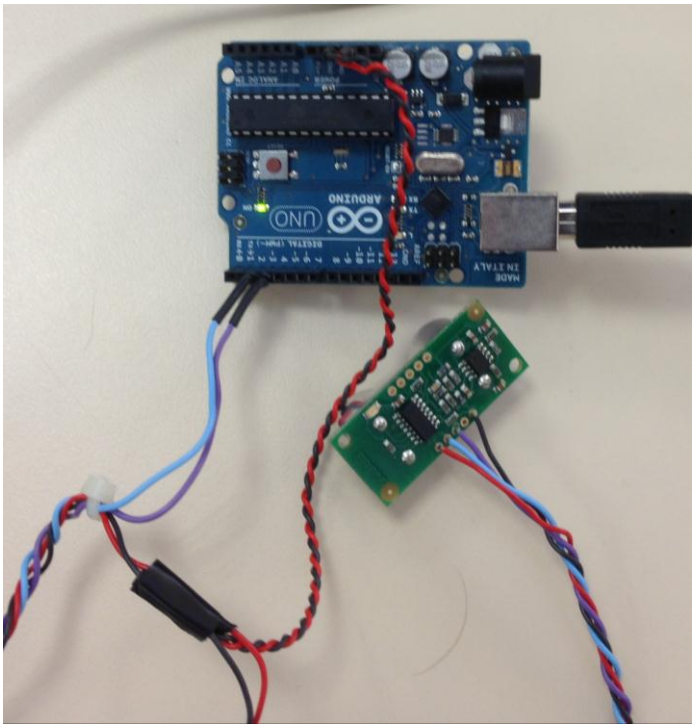


Schéma de câblage du capteur d'altitude



Carte Arduino reliée au capteur d'altitude

```
//Mesure d'altitude
unsigned int altitude=0;
const byte pin_trigg_alt = 3; // Broche d'envoi de la demande d'altitude au module ultrason (Trigger) sur pin 3
const byte pin_echo_alt = 2; // Broche de réception de l'altitude du module ultrason (Echo) sur pin 2
unsigned long timeout_alt=50000; //Timeout de la fonction PulseIn (mesure du temps de echo) en microsecondes

// Mesure de l'altitude
void mes_altitude()
{
  unsigned long mes_alt = 0;   // variable pour socker la largeur de la pulsation en micro secondes
  digitalWrite(pin_trigg_alt, HIGH);  // envoie d'une pulsation (mise à l'état haut de la sortie,
  delayMicroseconds(12);   // attente de 10 micro seconde,
  digitalWrite(pin_trigg_alt, LOW);  // puis passage à l'état bas.
  mes_alt = pulseIn(pin_echo_alt, HIGH, timeout_alt); // lit la durée de l'impulsion retournée par le capteur
  if (mes_alt<30000)   //Si la valeur du temps de « echo » ne correspond pas à « aucun obstacle »
  {
    altitude = mes_alt/58.0;  // conversion d'un temps en microseconde en distance en cm. La valeur du dénominateur est issue des mesures et de la datasheet
  }
}
```

Programme capteur d'altitude

Pour les tests, le programme principal appel la fonction et envoie sur la liaison série (non XBee) la valeur de l'altitude en cm (en ASCII), qui est directement lisible par le terminal inclus dans Arduino.

3) La réception des trames du XBee :

La maquette est constituée ainsi :
XBee branché sur PC (avec X-CTU), Adruino XBee shield connecté à l'Arduino Uno
Il faut tout d'abord initialiser le XBee :

```
//XBee
const byte ID_reseau=15;
const byte ID_XBee=5;

//Initialisation XBee
void init_XBee()
{
  Serial.begin(9600);  // initialisation de la liaison série (à 9600 bauds)

  Serial.print ("+++"); //passage en mode configuration
  delay(1500);
  Serial.print ("ATID"); //configuration de l'ID réseau
  Serial.print (ID_reseau, DEC); //numero du reseau
  Serial.print ("\r");
  Serial.print ("ATMY");      //configuration de l'ID XBee
  Serial.print (ID_XBee, DEC); //numero du XBee dans le reseau
  Serial.print ("\r");
  Serial.print ("ATDH0000\r");  //Communique avec tout le monde sur le réseau
  Serial.print ("ATDLFFFF\r");  //Communique avec tout le monde sur le réseau
  Serial.print ("ATCN\r"); //Sortie du mode configuration
}
```

Nous pouvons ensuite lire des trames de 1 ou 5 octets :

```
//F2.2 : Lecture d'une trame(paramètre : tableau de 5 octets)
void lecture_XBee(byte data[5])
{
  int i;
  for (i=0 ; i<5 ; i++)
  {
    data[i]=0xAA; //Initialisation à la valeur par défaut AA en hexadécimal
  }
 if(Serial.available() == 5) //lecture de la commande, s'il y a 5 donnée disponible sur la liaison série
 {
    data[0] = Serial.read(); //la donnée disponible est stockée dans la variable data
    data[1] = Serial.read();
    data[2] = Serial.read();
    data[3] = Serial.read();
    data[4] = Serial.read();
 }
 else if(Serial.available() >0) data[0] = Serial.read(); //Sinon lire uniquement la première donnée reçue

 Serial.flush(); //Effacement des données en attente sur la liaison série
}
```

Pour les tests, le programme principal initialise le XBee puis, dans une boucle, appelle la fonction de lecture et en fonction de la/des donnée(s) reçue(s), renvoi une certaine valeur lisible sur X-CTU.
Après une longue mise au point, le programme fonctionne correctement.

IV. F4 : Gestion des tâches (Programme principal)

C'est le programme final. Il contiens également d'autre fonctions : F6, prise en charge de F5(contrôle manuel) et F3.2 . Le mode_auto fait partie de F1(mise en position neutre des voies 1, 2 et 4). Le programme final complet est donné en annexe.

Initialisation :

```
void setup()
{

//Configuration des E/S
  pinMode(pin_echo_alt, INPUT);
  pinMode(pin_trigg_alt, OUTPUT);
  int i;
  for (i=0 ; i<=3 ; i++)
  {
    pinMode(pin_CH[i],OUTPUT);
  }

init_XBee();

//Chargement des données du mode auto (stable), non utilisé pour l'instant (aucun réglage sauvegardé, on utilise les valeurs « par défaut » écrites lors de l'initialisation des variables)
for (i=0 ; i<4 ; i++)
{
//  stable[i]=EEPROM.read(10*(i+1));
}

delay(temps_demarrage);
}
```

Fonctionnement en boucle :

```
void loop()
{

   gen_commande(); // Génération des signaux de commande du drone

  mes_altitude();  //Mesure d'altitude

  lecture_XBee(cmd);         //Lecture des données XBee

  //Changement d'altitude/Gestion des commandes
  int i;
  if (((cmd[0]== 0xCA) || (cmd[0]== 0xCE)) && (cmd[1] <= CHX_max) && (cmd[2] <= CHX_max)&& (cmd[3] <= CHX_max)&& (cmd[4] <= CHX_max)) //Si réception d'une trame valide contenant l'ordre « contrôle manuel »
  {
    for (i=0 ; i<4 ; i++)
    {
    CH[i]=cmd[i+1]*10;       //Affectation des valeurs des commandes du drone à partir des données recues par le XBee
    }
    if (cmd[0]== 0xCE)
    {
      for (i=0 ; i<4 ; i++)
      {
        stable[i]=cmd[i+1]*10;
        EEPROM.write(stable[i],(10*(i+1))); //Enregistrement de la position stable à la réception de la commande "CE"
```

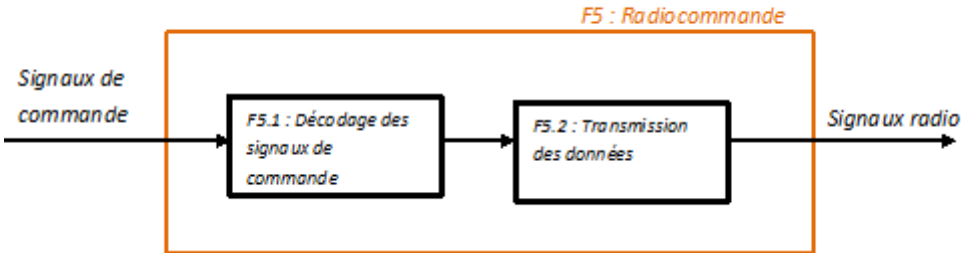
```
    }
   }
  }
  else if(cmd[0]==0xFF)
  {
   Serial.println(altitude,DEC);        //Envoi de l'altitude si on reçoit "FF"
  }
  else if (cmd[0]<=gaz_max)             //Sinon si la commande est valide (valeur recue inferieur à la valeur max de CH3)
  {
   CH[2]=cmd[0]*10;                     //Affectation de la valeur de commande puissance moteurs (CH3)
   Serial.println(cmd[0],DEC);          //Renvoi de la valeur affectée à la puissance moteurs
  }
  else mode_auto();          //Sinon garder la commande moteur actuelle et une position stable (parallèle au sol)
}
```

Fonctionnement final :

La carte Arduino initialise les E/S, le XBee puis passe en fonctionnement en boucle : il génère les signaux de commande du drone puis mesure l'altitude et lit les données reçues par le XBee.
Si cette donnée contient 5 octets dont l'ordre « contrôle manuel» (premier octet), les valeurs des 4 derniers octets sont affectés aux 4 signaux de commande du drone. De même si l'ordre reçu est l'enregistrement des commandes, mais le programme sauvegarde également la valeur des 4 commandes en tant que « position stable ».
Si la donnée reçue contient 1 seul octet et qu'il est valide (cohérent, <0x64), le drone monte/descend plus ou moins vite selon la valeur reçue.
Si la donnée reçue est 1 octet et que c'est l'ordre d'envoi d'altitude (FF), le drone renvoi la valeur de son altitude en ASCII (pour une lecture simple sur écran).
Si le drone reçoit n'importe quelle autre type de donnée (ou pas de type du tout), il reste dans sa position actuelle.

Partie 3
Travaux inachevés

I. F5 : Copie des signaux de la télécommande



La fonction F5.1 lit les signaux provenant du cordon de simulation de la télécommande (2 fils) préalablement adapté en 0-5V par un montage inverseur simple (pull-up) avec 1 transistor 2n2222a et 1 résistance 10kOhms. On obtient les signaux suivant :

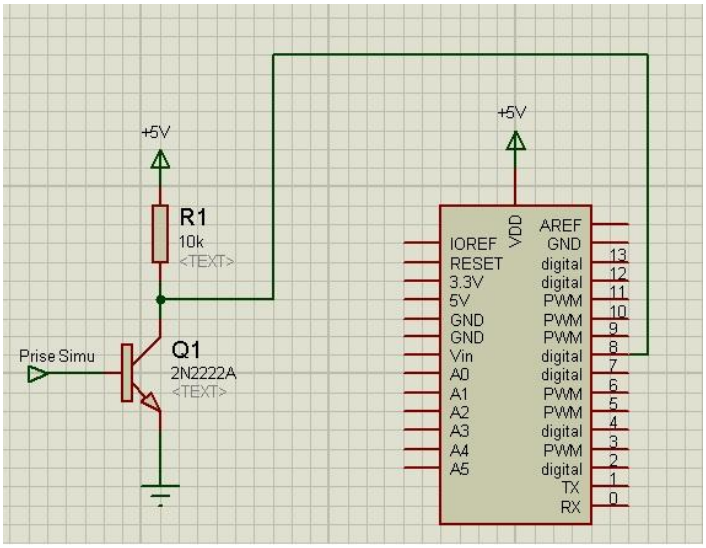
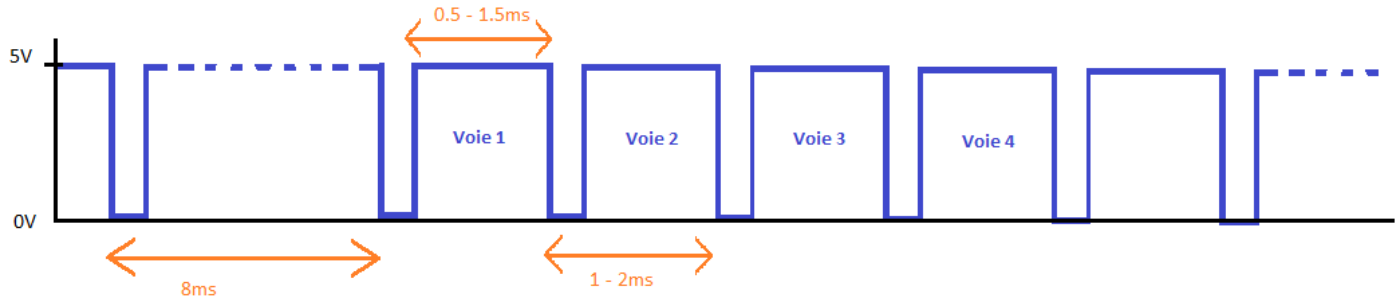


Schéma de câblage



Forme des signaux obtenus

Il faut donc mesurer le temps durant lequel le signal est à l'état haut pour chaque voie, après le début de trame (temps d'état haut >5s). Voir programme en annexe.
Ce programme n'est pas au point.

La fonction 5.2 est un envoi par XBee (comme pour la fonction 3.2 mais l'information est codée en hexadécimal)

II. F6 : Enregistrement des réglages

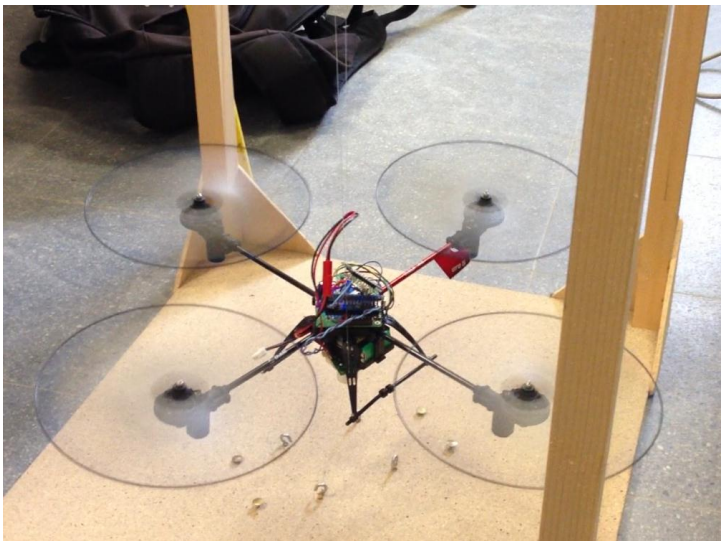
Cette fonction enregistre dans la mémoire interne EEPROM du microcontrôleur la valeur des commandes du drone lorsque l'ordre lui est envoyé (en contrôle manuel seulement, à distance par appuis sur un bouton poussoir placé sur la télécommande). Elle n'a pas encore été testée

III. F7 : Auto-stabilisation du drone en altitude

Non-testée. Les modifications apportées au programme final sont écrites en annexe.
Cette fonction a pour but de garder le drone à une altitude constante. En effet nous commandions jusqu'ici la puissance moteur, ce qui correspond à monter/descendre plus ou moins vite. Avec ce programme, il aurait été possible d'envoyer au drone l'altitude souhaitée et le drone s'y serait stabilisé automatiquement. Nous n'avons cependant pas pris en compte dans ce programme la variation des commandes en fonction du niveau de charge de la batterie.

Bilan

Nous avons remplacé la carte Arduino Uno et ses fils de commande par une carte plus petite embarquée sur le drone possédant les mêmes caractéristiques : la Arduino Nano ; sur laquelle nous avons soudé un module XBee. Elle est directement alimentée en 5V par la carte principale du drone et aux broches de commande.
Cependant, fabriquer une carte intégrant le XBee et un meilleur connecteur serait une solution idéale, car notre montage est relativement fragile (à concevoir à partir des plans disponibles en annexe et à l'adresse http://arduino.cc/en/uploads/Main/ArduinoNano30Schematic.pdf et http://www.arduino.cc/en/uploads/Main/XbeeShieldSchematic.pdf).



Drone sur son support

Carte Nano

Le drone, placé sur un support ne le laissant que monter et descendre, fonctionne correctement malgré un petit déséquilibre vers l'avant au décollage à cause de l'effet de sol, ainsi qu'une tendance au lacet (rotation). Ce petit souci mineur mis de côté (qui nécessitera un simple réglage des valeurs « stables »), nous avons atteint les objectifs principaux de ce projet qui étaient les suivants :

- Commander le drone à distance par envoi d'un ordre (puissance moteur) en hexadécimal
- Permettre la télémétrie en rendant la liaison bidirectionnelle et l'ajout d'un capteur d'altitude sur le drone.

Par manque de temps, nous n'avons pas pu améliorer le drone (F5, F6 et F7) ni effectuer les réglages des commandes « stables ». Nous aurions également voulu lire directement l'altitude sur un écran LCD.

Le fonctionnement amélioré que nous souhaitions mettre au point aurait été le suivant :

- Mode auto : Envoi de commande d'altitude en hexadécimal par PC ou avec le manche des gaz (voie 3). Le drone se place et se stabilise à l'altitude désirée et y reste.

- Mode manuel : Par basculement d'un interrupteur placé sur la télécommande, le drone se pilote comme d'origine, c'est-à-dire que l'on commande les 4 voies.

En mode manuel, l'appui sur un bouton poussoir permettrait d'enregistrer la valeur des commandes du drone afin de pouvoir régler précisément sa position « stable » (valeur des 4 voies pour que le drone reste parfaitement immobile en vol).
A tout moment, l'envoi d'une commande « FF » ou l'appui sur un bouton poussoir (télécommande) ordonne au drone d'envoyer son altitude actuelle, qui s'affiche à l'écran du PC et sur l'écran LCD.
Ajouter la possibilité de configurer le réseau XBee sur lequel est connecté le drone par interrupteurs sur le drone et la télécommande(F10).

Ce projet a été pour nous l'occasion d'utiliser nos connaissances et d'appliquer les techniques acquises lors de nos formations. Il nous a également permis d'apprendre à utiliser la plateforme Arduino pour créer des prototypes et mieux comprendre l'utilisation du XBee.

Nous avons pris beaucoup de plaisir à travailler sur ce projet.

Remerciements

Nous tenons à remercier tout particulièrement Mr Thierry PERISSE pour son aide et ses conseils bienveillants ainsi que Frank (nous ne connaissons pas son nom) pour nous avoir fournis en composants et matériel de soudure.

Merci pareillement à l'ambiance de la classe dans laquelle nous avons pris beaucoup de plaisir à travailler.

Annexes

SRF05 - Ultra-Sonic Ranger

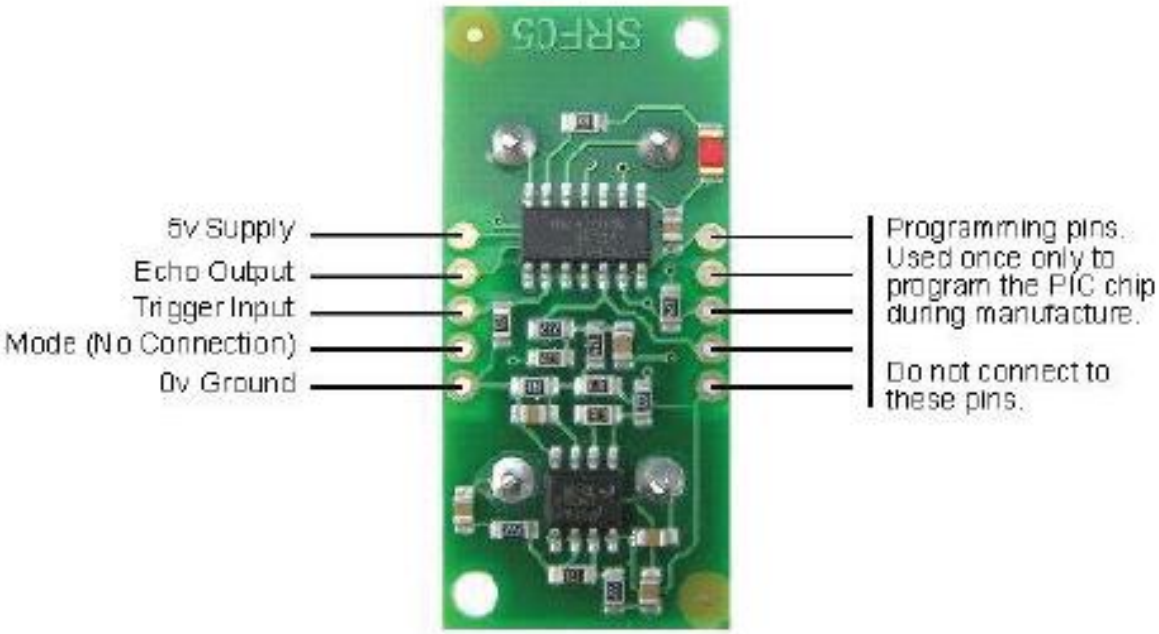
Technical Specification

Introduction

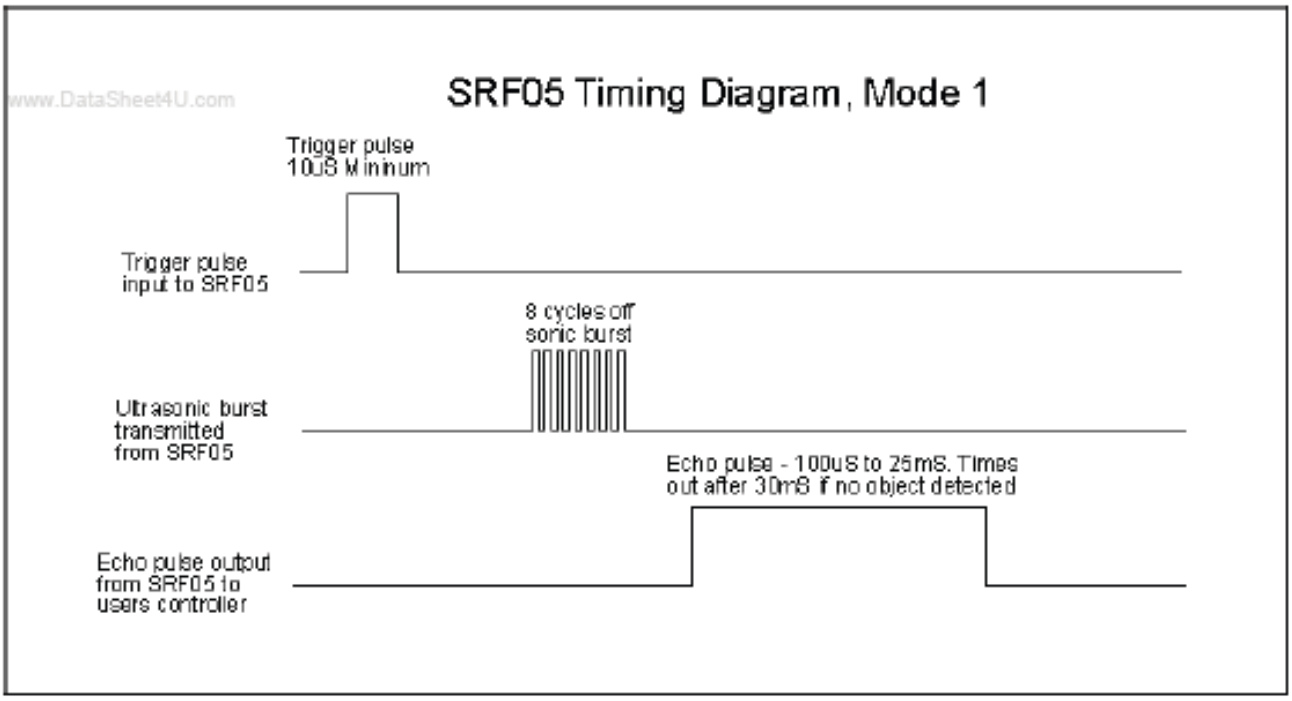
The SRF05 is an evolutionary step from the SRF04, and has been designed to increase flexibility, increase range, and to reduce costs still further. As such, the SRF05 is fully compatible with the SRF04. Range is increased from 3 meters to 4 meters. A new operating mode (tying the mode pin to ground) allows the SRF05 to use a single pin for both trigger and echo, thereby saving valuable pins on your controller. When the mode pin is left unconnected, the SRF05 operates with separate trigger and echo pins, like the SRF04. The SRF05 includes a small delay before the echo pulse to give slower controllers such as the Basic Stamp and Picaxe time to execute their pulse in commands.

Mode 1 - SRF04 compatible - Separate Trigger and Echo

This mode uses separate trigger and echo pins, and is the simplest mode to use. All code examples for the SRF04 will work for the SRF05 in this mode. To use this mode, just leave the mode pin unconnected - the SRF05 has an internal pull up resistor on this pin.

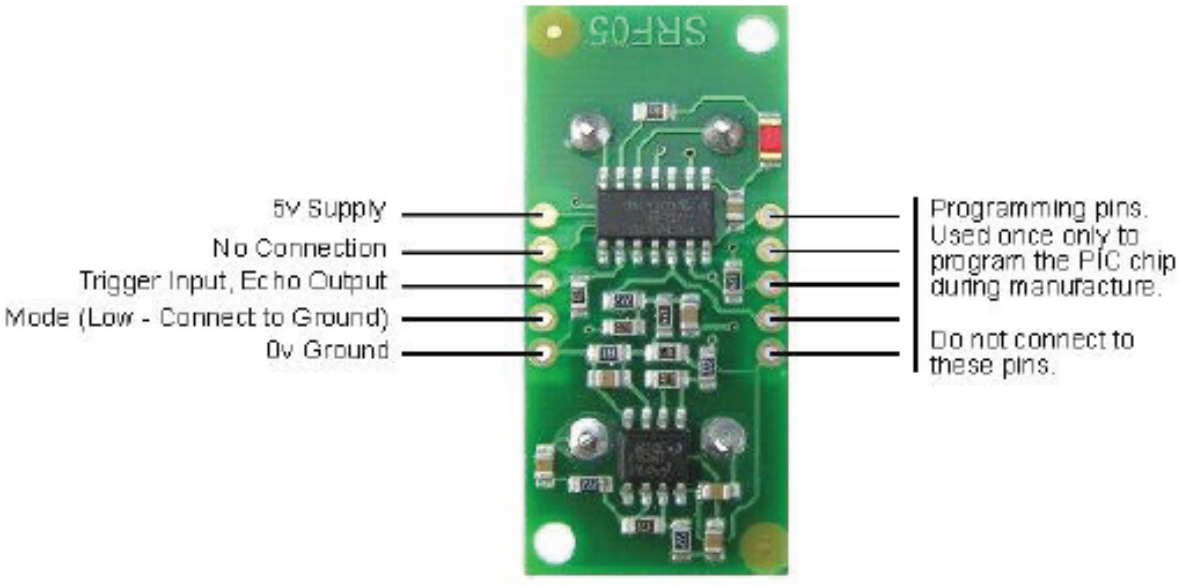


Connections for 2-pin Trigger/Echo Mode (SRF04 compatible)

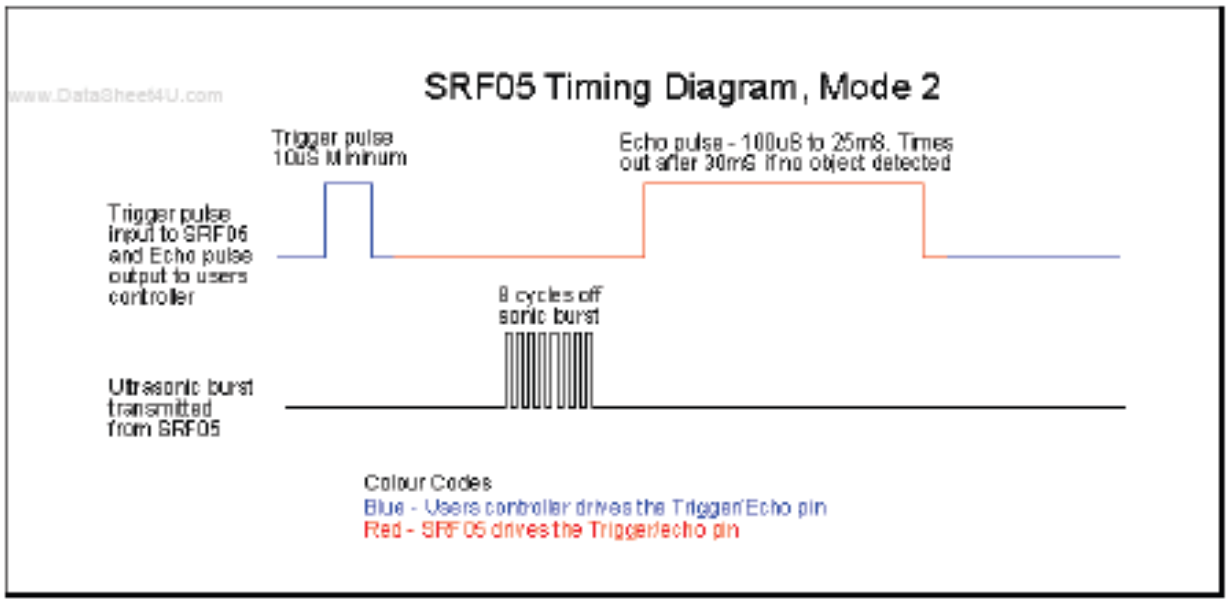


Mode 2 - Single pin for both Trigger and Echo

This mode uses a single pin for both Trigger and Echo signals, and is designed to save valuable pins on embedded controllers. To use this mode, connect the mode pin to the 0v Ground pin. The echo signal will appear on the same pin as the trigger signal. The SRF05 will not raise the echo line until 700uS after the end of the trigger signal. You have that long to turn the trigger pin around and make it an input and to have your pulse measuring code ready. The PULSIN command found on many popular controllers does this automatically.



Connections for single pin Trigger/Echo Mode



To use mode 2 with the Basic Stamp BS2, you simply use PULSOUT and PULSIN on the same pin, like this:

```
SRF05 PIN 15              ' use any pin for both trigger and echo
Range VAR Word            ' define the 16 bit range variable

SRF05 = 0                 ' start with pin low
PULSOUT SRF05, 5          ' issue 10uS trigger pulse (5 x 2uS)
PULSIN SRF05, 1, Range    ' measure echo time
Range = Range/29          ' convert to cm (divide by 74 for inches)
```

Calculating the Distance

The SRF05 Timing diagrams are shown above for each mode. You only need to supply a short 10uS pulse to the trigger input to start the ranging. The SRF05 will send out an 8 cycle burst of ultrasound at 40khz and raise its echo line high (or trigger line in mode 2). It then listens for an echo, and as soon as it detects one it lowers the echo line again. The echo line is therefore a pulse whose width is proportional to the distance to the object. By timing the pulse it is possible to calculate the range in inches/centimeters or anything else. If nothing is detected then the SRF05 will lower its echo line anyway after about 30mS.

The SRF04 provides an echo pulse proportional to distance. If the width of the pulse is measured in uS, then dividing by 58 will give you the distance in cm, or dividing by 148 will give the distance in inches. uS/58=cm or uS/148=inches.

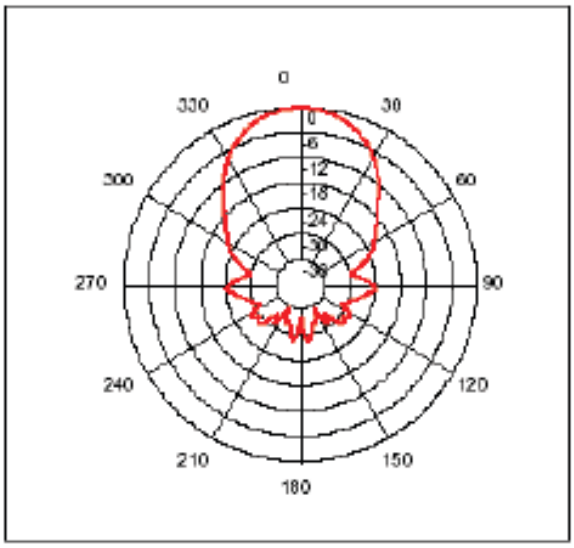
The SRF05 can be triggered as fast as every 50mS, or 20 times each second. You should wait 50ms before the next trigger, even if the SRF05 detects a close object and the echo pulse is shorter. This is to ensure the ultrasonic "beep" has faded away and will not cause a false echo on the next ranging.

The other set of 5 pins

The 5 pins marked "programming pins" are used once only during manufacture to program the Flash memory on the PIC16F630 chip. The PIC16F630's programming pins are also used for other functions on the SRF05, so make sure you don't connect anything to these pins, or you will disrupt the modules operation.

Changing beam pattern and beam width

You can't! This is a question which crops up regularly, however there is no easy way to reduce or change the beam width that I'm aware of. The beam pattern of the SRF05 is conical with the width of the beam being a function of the surface area of the transducers and is fixed. The beam pattern of the transducers used on the SRF05, taken from the manufacturers data sheet, is shown below.



There is more information in the sonar faq.

Programme final

```
/*
Programme final :
Il contiens les fonction F1, F2, F3, F4 et prend en compte les fonctions F8 et F9
Fonctionnement :
Initialise les E/S,  le XBee
Génère les signaux de commande du drone
Mesure l'altitude
Lit les données reçues par le XBee.
Gère les routines à éxecuter selon la donnée reçue :
- contrôle manuel ou fonctionnement semi-automatique (contrôle de montée/descente)
- renvoi la valeur de son altitude en ASCII
*/

#include <EEPROM.h>     //Utile pour la sauvegarde de la position stable (non-utilisé ici)

// Déclaration des variables par fonction:
//F4
const unsigned int temps_demarrage=1000; //Temps d'attente avant activation du drone après branchement de la batterie (en millisecondes)


//F3 : Mesure d'altitude
float altitude=0;       // Stock la valeur de l'altitude mesuré
const byte pin_trigg_alt = 11; // Broche d'envoi de la demande d'altitude au module ultrason (Trigger) sur pin D11
const byte pin_echo_alt = 12; // Broche de réception de l'altitude du module ultrason (Echo) sur pin D12
unsigned long timeout_alt = 50000; //Timeout de la fonction PulseIn (mesure du temps de echo) en microsecondes

//F1 : Generation des signaux de commande
const byte gaz_max=100; //valeur max des gaz (en %)
const byte pin_CH[4]={17,18,15,16}; //Numero de chaque pin selon la voie
unsigned int CH[4]={0,0,0,0};    //Valeur de commande de chaque voie
const byte CHX_max=100; //Valeur maximale d'une voie (en %)
unsigned int stable[4]={600,600,500,600}; //Valeur de commande (temps) de chaque voie en mode stable


//F2 : Réception des données XBee
const byte ID_reseau=15;
const byte ID_XBee=5;
byte cmd[5];

// ############## FONCTIONS/ROUTINES ###################

//F1 : Generation des commandes
void gen_commande()
{
 int i;
 for (i=0 ; i<4 ; i++)
 {
  digitalWrite(pin_CH[i], HIGH);
  delayMicroseconds(950+CH[i]);
  digitalWrite(pin_CH[i], LOW);
 }
}

//F3 : Mesure de l'altitude
void mes_altitude()
{
 unsigned long mes_alt = 0;   // variable pour socker la largeur de la pulsation en micro secondes
 digitalWrite(pin_trigg_alt, HIGH);  // envoie d'une pulsation (mise à l'état haut de la sortie,
 delayMicroseconds(12);   // attente de 10 micro seconde,
 digitalWrite(pin_trigg_alt, LOW);  // puis passage à l'état bas.
 mes_alt = pulseIn(pin_echo_alt, HIGH, timeout_alt); // lit la durée de l'impulsion retournée par le capteur
 if (mes_alt<30000)
 {
```

```
   altitude = mes_alt/58.0;  // conversion d'un temps en microseconde en distance en cm. La valeur du dénominateur est issue des mesures et de la datasheet
  }
}


//F2.1 : Initialisation XBee
void init_XBee()
{

 Serial.begin(9600);  // initialisation de la liaison série (à 9600 bauds)

 Serial.print ("+++"); //passage en mode configuration
 delay(1500);
 Serial.print ("ATID"); //configuration de l'ID réseau
 Serial.print (ID_reseau, DEC); //numero du reseau
 Serial.print ("\r");
 Serial.print ("ATMY");  //configuration de l'ID XBee
 Serial.print (ID_XBee, DEC); //numero du XBee dans le reseau
 Serial.print ("\r");
 Serial.print ("ATDH0000\r");  //Envoi à tout le monde sur le réseau
 Serial.print ("ATDLFFFF\r");  //Envoi à tout le monde sur le réseau
 Serial.print ("ATCN\r");   //Sortie du mode configuration
}


//F2.2 : Lecture d'une trame (paramètre : tableau de 5 octets)
void lecture_XBee(byte data[5])
{
 int i;
 for (i=0 ; i<5 ; i++)
 {
   data[i]=0xAA; //Initialisation à la valeur par défaut AA en hexadécimal
 }
 if(Serial.available() == 5) //lecture de la commande, s'il y a 5 donnée disponible sur la liaison série
 {
   data[0] = Serial.read(); //la donnée disponible est stockée dans la variable data
   data[1] = Serial.read();
   data[2] = Serial.read();
   data[3] = Serial.read();
   data[4] = Serial.read();
 }
 else if(Serial.available() >0) data[0] = Serial.read(); //Sinon lire uniquement la première donnée reçue

 Serial.flush(); //Effacement des données en attente sur la liaison série/
}


// F2.2 : Mode semi-automatique : commande de la puissance moteur uniquement
void mode_auto()
{
 CH[0]=stable[0];  //On commande les 3 autres voies avec leur valeur neutre par défaut
 CH[1]=stable[1];
 CH[3]=stable[3];
}

//####### F4 : PROGRAMME PRINCIPAL #########

void setup()
{

//Configuration des E/S
 pinMode(pin_echo_alt, INPUT);  //Pin reliée à la broche "Echo" du capteur d'altitude configurée en entrée
 pinMode(pin_trigg_alt, OUTPUT);  //Pin reliée à la broche "Trigger" du capteur d'altitude configurée en sortie
 int i;
 for (i=0 ; i<=3 ; i++)
 {
   pinMode(pin_CH[i],OUTPUT); //Pins de commande du drone reliées au drone (à la place de la carte récepteur d'origine) configurée en sortie
 }
```

```
init_XBee();  //F4/F2.1 : Initialisation du XBee

//Chargement des données du mode auto (stable) (non-testé)
for (i=0 ; i<4 ; i++)
{
//  stable[i]=EEPROM.read(10*(i+1));
}

delay(temps_demarrage);  //Attente avant démarage : permet à l'utilisateur de s'écarter après branchement de la batterie
}


void loop()  //Executer en continu
{

 gen_commande(); // F1 : Génération des signaux de commande du drone

 mes_altitude();  // F3 : Mesure de l'altitude

 lecture_XBee(cmd);  //F2.2 : Lecture des données XBee

 // F4/F2.2 : Changement d'altitude/Gestion des commandes
 int i;
 if (((cmd[0]== 0xCA) || (cmd[0]== 0xCE)) && (cmd[1] <= CHX_max) && (cmd[2] <= CHX_max)&& (cmd[3] <= CHX_max)&& (cmd[4] <=
CHX_max)) //Si réception d'une trame valide contenant l'ordre « contrôle manuel » ou « enregistrement »
 {
  for (i=0 ; i<4 ; i++)
  {
  CH[i]=cmd[i+1]*10;          //Affectation des valeurs des 4 commandes du drone à partir des données recues par le XBee
  }
  if (cmd[0]== 0xCE)
  {
   for (i=0 ; i<4 ; i++)
   {
    stable[i]=cmd[i+1]*10;
    EEPROM.write(stable[i],(10*(i+1))); //Enregistrement de la position stable à la réception de la commande "CE"
   }
  }
 }
 else if(cmd[0]==0xFF)
 {
  Serial.println(altitude,DEC);             //Envoi de l'altitude si on reçoit "FF"
 }
 else if (cmd[0]<=gaz_max)   //Sinon si la commande est valide (valeur recue inferieur à la valeur max de CH3)
 {
  CH[2]=cmd[0]*10;                          //Affectation de la valeur de commande puissance moteurs (CH3)
  Serial.println(cmd[0],DEC);//Renvoi de la valeur affectée à la puissance moteurs
 }
 else mode_auto();              //Sinon garder la commande moteur actuelle et une position stable (parallèle au sol)
}
```

Programme F5 :
Copie des signaux de la télécommande

```
/* Programme réalisant la fonction F5 :
- F5.1 : Lecture des signaux sortant du cordon simulateur de la télécommande
- F5.2 : Envoi des commandes (valeur numérique) décodées des signaux de la télécommande
- Prise en compte de la fonction F6
*/

//Déclaration des variables
const byte pin_simu=8;  //Broche où est connecté le câble du simulateur
const byte pin_save=7;  //Broche où est connecté le BP pour sauvegarder la position stable
unsigned long CH[4]={0,0,0,0};    //Valeur de commande de chaque voie

//XBee
const byte ID_reseau=15;
const byte ID_XBee=3;

//####### FONCTIONS/ROUTINES #########

//Initialisation XBee
void init_XBee()
{
// initialisation de la liaison série (à 9600 bauds)
  Serial.begin(9600);
//initialisation de l'ID réseau
  Serial.print ("+++");
  delay(1500);
  Serial.print ("ATID");
  Serial.print (ID_reseau, DEC); //numero du reseau
  Serial.print ("\r");
  Serial.print ("ATMY");
  Serial.print (ID_XBee, DEC); //numero du XBee dans le reseau
  Serial.print ("\r");
  Serial.print ("ATDH0000\r");  //Envoi à tout le monde sur le réseau
  Serial.print ("ATDLFFFF\r");  //Envoi à tout le monde sur le réseau
  Serial.print ("ATCN\r");
}

//####### PROGRAMME PRINCIPAL #########

void setup()
{

//Configuration des E/S
  pinMode(pin_simu, INPUT);
  pinMode(pin_save, INPUT);

init_XBee();

}

void loop()
{


// F5.1 : Lecture des voies de la télécommande
  if ( pulseIn(pin_simu,HIGH)>3000)  //Si on est au début de la trame de commande
  {
    int i;
    for (i=0 ; i<=3 ; i++)
    {
```

```
    CH[i] = (pulseIn(pin_simu,HIGH)); //Mesure du temps pendant lequel la pin connectée au cordon simulateur est à l'état haut pour les 4 premiers signaux
   }
  }
  else
  {
   //F5.2 : Envoi des données
   if(digitalRead(pin_save)==HIGH) Serial.write(0xCE);  //Si le bouton poussoir de sauvegarde est appuyé (état haut), envoyer en premier la commande d'enregistrement
   else   Serial.write(0xCA);    //Sinon envoyer la commande "mode manuel" en premier
   Serial.write(map(CH[0],515,1465,0,255));  // Envoyer ensuite la valeur des voies (convertion du temps relevé précédement entre 0 et 255)
   Serial.write(map(CH[1],500,1490,0,255));
   Serial.write(map(CH[2],460,1380,0,255));
   Serial.write(map(CH[3],470,1520,0,255));

/*  Utilisé pour tester la copie des commandes en envoyant directement les valeurs en ASCII (lecture sur terminal Arduino)
   Serial.print(map(CH[0],515,1465,0,255),DEC);
   Serial.print(' ');
   Serial.print(map(CH[1],500,1490,0,255),DEC);
   Serial.print(' ');
   Serial.print(map(CH[2],450,1390,0,255),DEC);
   Serial.print(' ');
   Serial.print(map(CH[3],470,1520,0,255),DEC);
   Serial.println();
*/
   delay(100);
  }

}
```

Routine F7 : Stabilisation en altitude

```
//Déclaration des variables

[…]
//F7 : Changement d'altitude
const byte acc_moteur=1; //Vitesse d'accélération verticale du drone en mode auto (en %)
byte alt_cmd=0; //Altitude demandée (0 - 255) en cm
const byte prec_alt=2;  //Precision de l'altitude (cm)
const int alt_max=100; //Altitude maximale (en cm)
const int alt_min=6;  //valeur de l'altitude 0 (en cm)
[…]


// ############## FONCTIONS/ROUTINES ###################
[…]
// F2.2 : Mode semi-automatique : commande de l'altitude
void mode_auto()
{
  CH[0]=stable[0];
  CH[1]=stable[1];
  CH[3]=stable[3];

  if ((alt_cmd)<=(altitude-prec_alt))  //Si on est trop bas
  {
    CH[2]=stable[2]+acc_moteur*10; //Augmenter la commande puissance moteur
  }
  else if ( alt_cmd>altitude && altitude>alt_min || altitude>alt_max)  //Sinon si on est trop haut
  {
    CH[2]=stable[2]-acc_moteur*10; //Diminuer la commande puissance moteur
  }
  else if(altitude<=alt_min)  //Sinon si on est au sol
  {
    CH[2]=0;              // Arrêt des moteurs
  }
  else CH[2]=stable[2];           //Sinon rester à l'altitude actuelle
}
[…]

// ############## PROGRAMME PRINCIPAL ################### (modifier la fin du programme comme suit)

[…]
     for (i=0 ; i<4 ; i++)
     {
       stable[i]=cmd[i+1]*10;
       EEPROM.write(stable[i],(10*(i+1)));
     }
    }
  }
  else if (cmd[0]<=alt_max)    //Si la commande reçue est valide (<0x64)
  {
    alt_cmd=cmd[0];               //Affecter cette valeur à la variable "alt_cmd"
    Serial.println(cmd[0],DEC);               //Renvoyer la valeur affectée en ASCII
  }
  else if(cmd[0]==0xFF)
  {
    Serial.println(altitude,DEC);
  }
  else mode_auto();
}
```

Schéma de la carte Arduino Nano

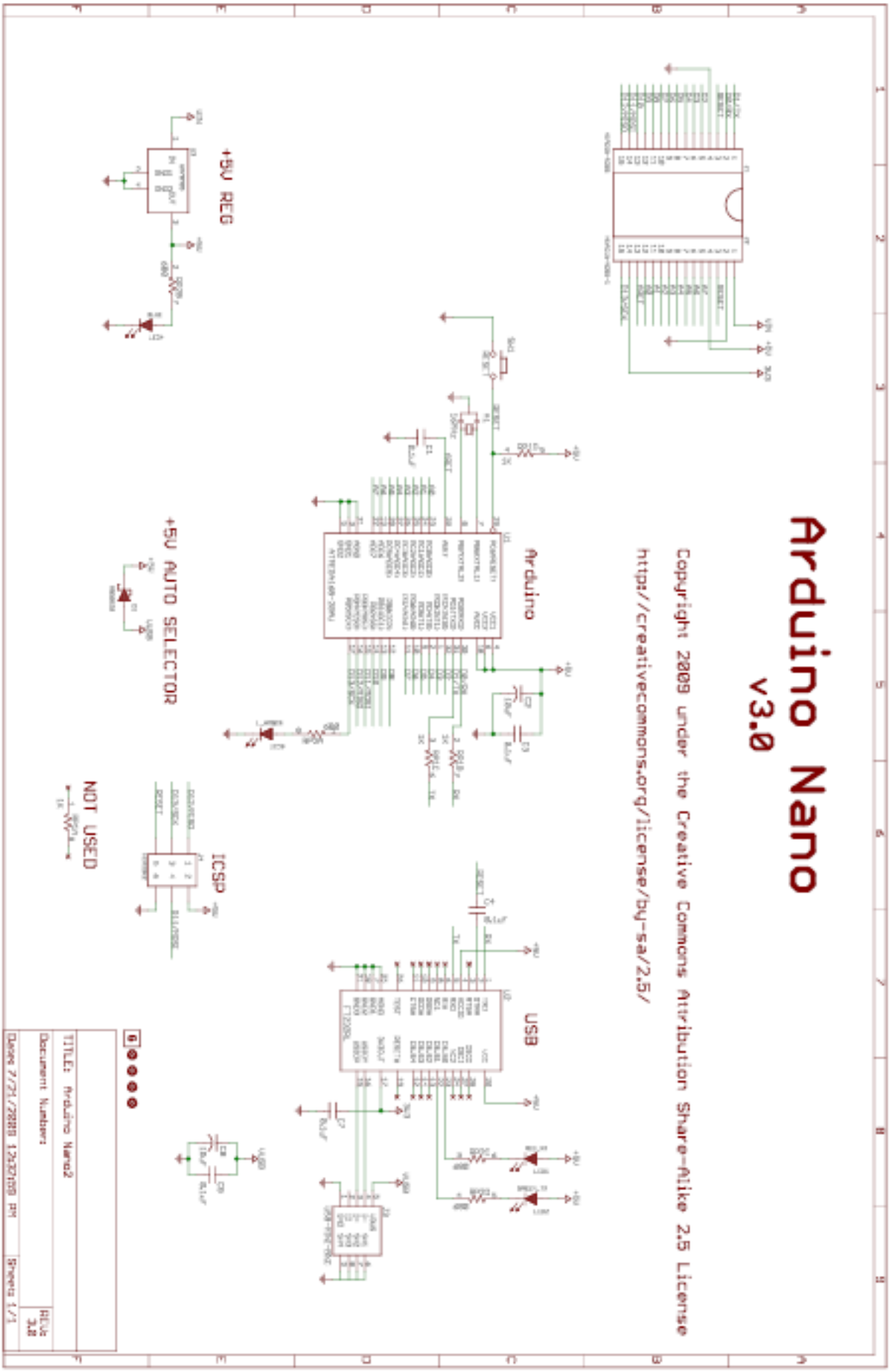
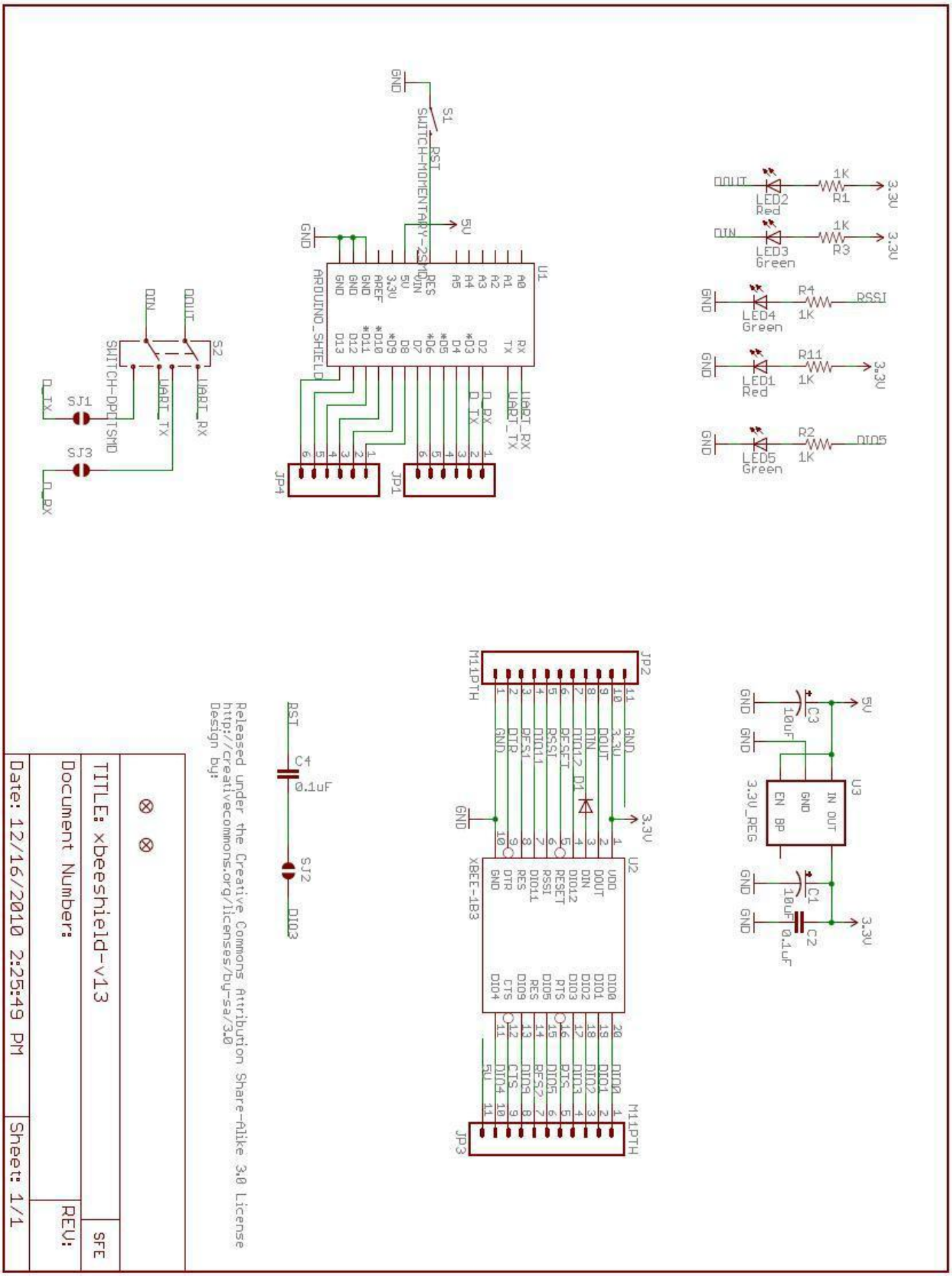


Schéma de l'adaptateur « XBee Shield »



Bibliographie – Liens Internet

http://home.nordnet.fr/~fthobois/theorieRC.htm

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

http://domotique.benchi.fr/robotique/arduino/tutorial-arduino-xbee-shield-partie-1/

http://www.siteduzero.com/sciences/tutoriels/arduino-pour-bien-commencer-en-electronique-et-en-programmation

http://193.49.146.171/~cottence/ArduinoCottenceau1112.pdf

http://arduino.cc/fr/Main/Debuter

http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=57&s=316&tp=3&tp2=0

Summary

The subject of this project is the realization of telemetry on a radio-controlled quadrotor : the UFO 5 by Walkera.

The drone, placed on a stand not letting that go up and down, must have the following characteristics:
- Allow to control the drone remotely by sending a command (engine power) in hexadecimal base
- Allow telemetry making the bidirectional wireless link and adding an altitude sensor on the drone.

To do this, we use the Arduino platform including programming software (close to C language) and different prototyping cards whose plans are available on the Arduino.cc website. To make the wireless communication, we used XBee RF modules.

This project was an opportunity for us to use our knowledge and apply the skills learned during our learning. It also allowed us to learn how to use the Arduino platform to create prototypes and understand the use of the XBee.

We really enjoyed working on this project.