Projet WepDrone

Par KHODR Remy et LESSIEUX Maxime.

Introduction

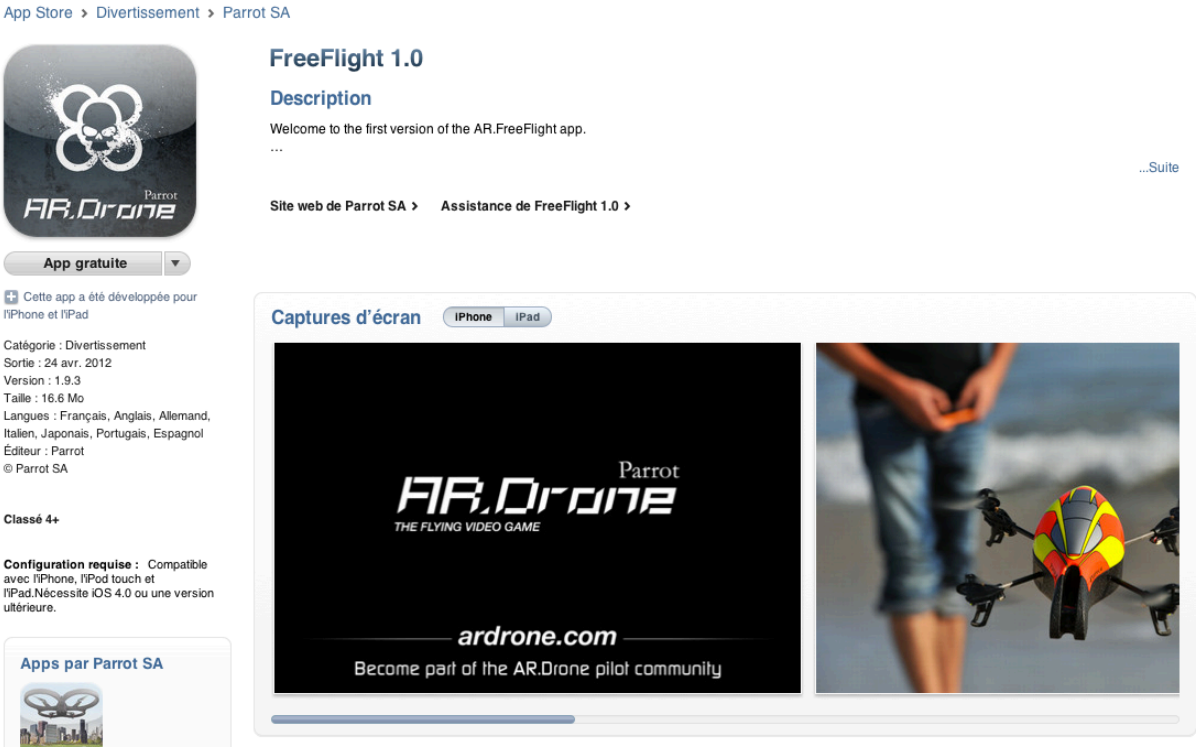
Ce projet a pour but de « customiser » un ArDrone de chez Parrot afin qu'il soit contrôlable à distance et donc affranchi de la portée du Wifi.
De plus en lui greffant une FoxBoard nous voulons le rendre capable de « hacker » des clés Wep via le programme AirCrack.
La FoxBoard servirai d'interface de commande entre le Drone et l'utilisateur, les 2 puces (GSM et GPS) seraient donc connectées à celle-ci, les coordonnées nous seraient transmises via la puce GSM, celle ci recevrait également les commandes.

L'objectif final était d'avoir un drone que l'on piloterait depuis notre ordinateur, sans le voir, en se fiant uniquement aux images qu'il nous enverrait via sa camera intégrée. Nous le localiserions sur une carte via les données de la puce GPS. Le programme que nous exécuterions sur la Foxboard nous demanderait vers quel point d'accès Wifi s'orienter et si oui ou non nous désirions se procurer la clé WEP. Si oui, le drone se poserait le temps de l'opération (économie d'énergie).

Sommaire

1. Prise en main

Image 1 : Présentation application FreeFlight



Nous commençons d'abord par télécharger l'application iPhone pour découvrir les fonctionnalités du drone. Après quelques instants nous nous rendons compte que sa prise en main est très intuitive :

- On incline l'iPhone dans la direction vers laquelle on veut aller, et le drone s'incline de la même manière que le téléphone (l'angle d'inclinaison est synonyme de vitesse ici).
- Pour tous les mouvements ne nécessitant pas d'inclinaison du drone, il y a des sticks virtuels sur l'écran qui nous permettent de faire monter ou descendre le drone (en altitude) ou de le faire pivoter sur lui même.
- Il y a un bouton pour le faire décoller/atterrir, ainsi que le plus important de tous : EMERGENCY qui envoie un signal prioritaire qui fait atterrir le drone immédiatement.

Image 2 : Screenshot de l'application FreeFlight

2. Connexion Wifi

Le drone communique par Wifi, nos premiers tests étant réalisés sur l'iPhone, nous avons du réaliser une connexion wifi manuelle entre le PC et le drone.

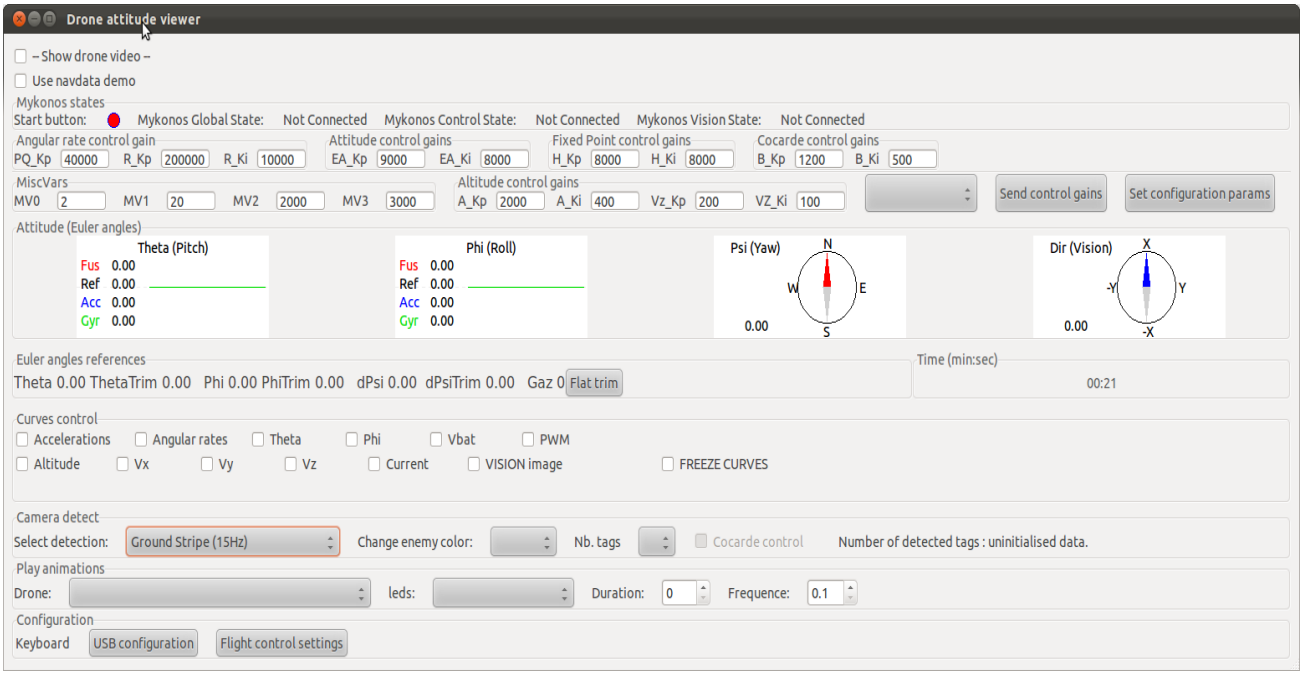
Pour cela nous avons installé les paquets nécessaires au bon fonctionnement de la clé wifi (cf. support de TP IMA4 sur rex.plil.fr, section « Ajout de paquetages »).

Ensuite afin de renseigner correctement le fichier interfaces (dans le répertoire /etc/network), nous avons cherché le SSID du drone via la commande : iwlist, « iwlist wlan0 scan | grep SSID » (qui nous donne la liste de tout les SSID détectés, ensuite nous

avons trouvé celui du drone, qui est sous la forme essid_ardroneXXXXXX, les XXXXXX étant réglables, et changent à chaque reset du drone).

3. Premiers pas avec le logiciel

Image 3 : Logiciel de contrôle du drone (Linux)



Afin d'assurer une connexion entre le PC et le drone, il faut veiller à ce que celle-ci réponde au protocole DHCP (affectation automatique des paramètres IP…) et non en « static ». (Ndlr : nous avons perdu beaucoup de temps avant de nous en rendre compte).

Nous avons essayé de contrôler le drone via le logiciel, mais nous n'avons jamais réussi, nous arrivions à voir la réponse des capteurs et à recevoir les données des 2 webcams intégrées au drone.

Devant l'inefficacité du logiciel nous n'avons eu d'autre choix que d'élaborer notre propre programme de pilotage.

4. Elaboration de notre programme C

Après étude de la documentation du drone, nous devions établir la liste des commandes AT avec lesquelles communiquer avec le drone :

```
Décollage = "AT*REF=%d, 290718208\r" ;
Atterrissage = "AT*REF=%d, 290717696\r";
Planer = "AT*PCMD=%d, 1, 0,0, 0,0\r";
Go_up = "AT*PCMD=%d, 1, 0, 0, 1036831949,0\r";
Go_down = "AT*PCMD=%d, 1, 0, 0,-1110651699,0\r";
Avancer = "AT*PCMD=%d, 1, 1036831949, 0,0,0\r";
Reculer = "AT*PCMD=%d, 1,-1110651699,0, 0,0\r";
Rotate_right ="AT*PCMD=%d, 1, 0, 0, 0,1036831949\r";
Rotate_left = "AT*PCMD=%d, 1, 0,0,0,1110651699\r";
Gauche = "AT*PCMD=%d, 1, 0, 1036831949, 0,0\r";
Droite = "AT*PCMD=%d, 1, 0,-1110651699, 0,0\r";
reculer30 = "AT*ANIM=%d, 0,1000\r";
avancer30 = "AT*ANIM=%d, 1,1000\r";
```

Le « %d » correspond au numéro de la commande en question, qui commence à 0 et qui est incrémenté à chaque commande envoyée.

Maintenant que nous savons quelles commandes envoyer, il faut trouver le moyen de les envoyer.
La fonction « nomVersAdresse » (source : rex.plil.fr) permet de résoudre l'adresse d'une machine en fonction du nom, ensuite nous créons une socket dans laquelle seront envoyées les commandes.

Image 4 : Programme C, Elaboration de la socket

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_TAMPON 30
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>


int nomVersAdresse(char *hote,struct in_addr *adresse)
{
    struct hostent *h;
    h=gethostbyname("192.168.1.1");
    if(h!=NULL) memcpy(adresse,h->h_addr_list[0],h->h_length);
    else return -1;

    return 0;
}

int main(int argc,char *argv[])
{

    int s;    /* Descripteur de SOCKET */
    struct sockaddr_in adresse; /* Adresse de la SOCKET du serveur */
    int statut;   /* Stocke le statut des commandes */

/* Creation de la SOCKET du client */
    s=socket(PF_INET,SOCK_DGRAM,0);
    if(s<0){perror("socket"); exit(-1);}

/* Preparation de la structure adresse du serveur */
    nomVersAdresse("192.168.1.1",&adresse.sin_addr);
    adresse.sin_family=AF_INET;
    adresse.sin_port=htons(5556);
```

On note que l'on communique sur le port 5556 du drone.

Une fois cette connexion établie, le programme nous dit comment envoyer chaque commande en entrant une lettre (pour plus de simplicité), ensuite on prépare la commande convenablement (en incrémentant le numéro de séquence) puis celle-ci est envoyée au drone dans la socket.

Image 5: Programme C, envoi de commande

```
int i;
char cmd='x';
char ordre[100];

printf("Decollage: a\nAtterissage: z\nrotateleft: l\nrotateright: r\ngoleft: L\ngoright: R\ngoup:
       u\ngodown: d\navancer30: o\nreculer30: m\navancer: O\nreculer: M\nChaine: C\n");

scanf("%c",&cmd);

switch (cmd) {
    case 'a':   //Take off
            sprintf(ordre,decollage,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
            break;
        case 'z':   //Landing
        sprintf(ordre,atterrissage,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
            break;
        case 'u':   //Go Up
        sprintf(ordre,goup,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
    break;
        case 'd':   //Go Down
        sprintf(ordre,godown,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
    break;
        case 'l':   //Rotate Left
        //for(i=0;i<4;i++)
            sprintf(ordre,rotateleft,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
    break;
        case 'r':   //Rotate Right
        //for(i=0;i<4;i++)
        sprintf(ordre,rotateright,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
    break;
    case 'L':   //Go left
        sprintf(ordre,gauche,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
            break;
    case 'R':   //Go Right
            sprintf(ordre,droite,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
            break;
        case 'p':   //SpaceBar
            sprintf(ordre,planer,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
            break;
    case 'o':   //avancer30
        //for(i=0;i<30;i++)
            sprintf(ordre,avancer30,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
        break;
    case 'm':   //reculer30
            //for(i=0;i<30;i++)
            sprintf(ordre,reculer30,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
    break;
    case 'O':   //avancer
        sprintf(ordre,avancer,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
            break;
    case 'M':   //reculer
        sprintf(ordre,reculer,(*sequence)++);
                sendto(s,ordre,strlen(ordre),0,(struct sockaddr *)&adresse,sizeof adresse);
            break;
    default:
            break;
    }
```

Tout cela sans oublier de fermer la socket via la commande close(int socket).

Après beaucoup de travail, nous n'avons pas réussi à faire avancer ou reculer notre drone via le programme C, nous arrivons à le faire décoller/atterrir, pivoter sur lui même mais nous devons lui envoyer une commande pour qu'il s'arrête.

Afin d'avancer dans le projet, nous allons maintenant essayer de contrôler le drone via la FoxBoard pour ensuite la scotcher sur le drone afin de pouvoir le piloter à distance.

5. Essai avec la FoxBoard

Nous avons scotché la FoxBoard avec son Alimentation (4 piles LR6) sur le « dos » du drone, et à notre grand regret celui-ci n'arrive pas à décoller en raison du poids. Ou il décolle et se crashe aussitôt.

La solution la plus raisonnable était de fabriquer un régulateur de tension qui alimenterait la FoxBoard (5V) via la batterie du drone (11.1V).

Cependant nous avons eu l'idée d'alimenter la FoxBoard via la fiche de recharge de la batterie qui compte 4 prises. Une pour la masse, une pour 4V, une pour 7V, une pour 11V. En combinant la masse et la prise 4V nous arrivons à alimenter notre FoxBoard en s'affranchissant du poids des batteries. Cependant cette solution est « dangereuse » étant donné que ces prises sont régulées par un microcontrôleur à l'intérieur de la batterie et que lors de nos premiers tests nous avons grillé une FoxBoard.

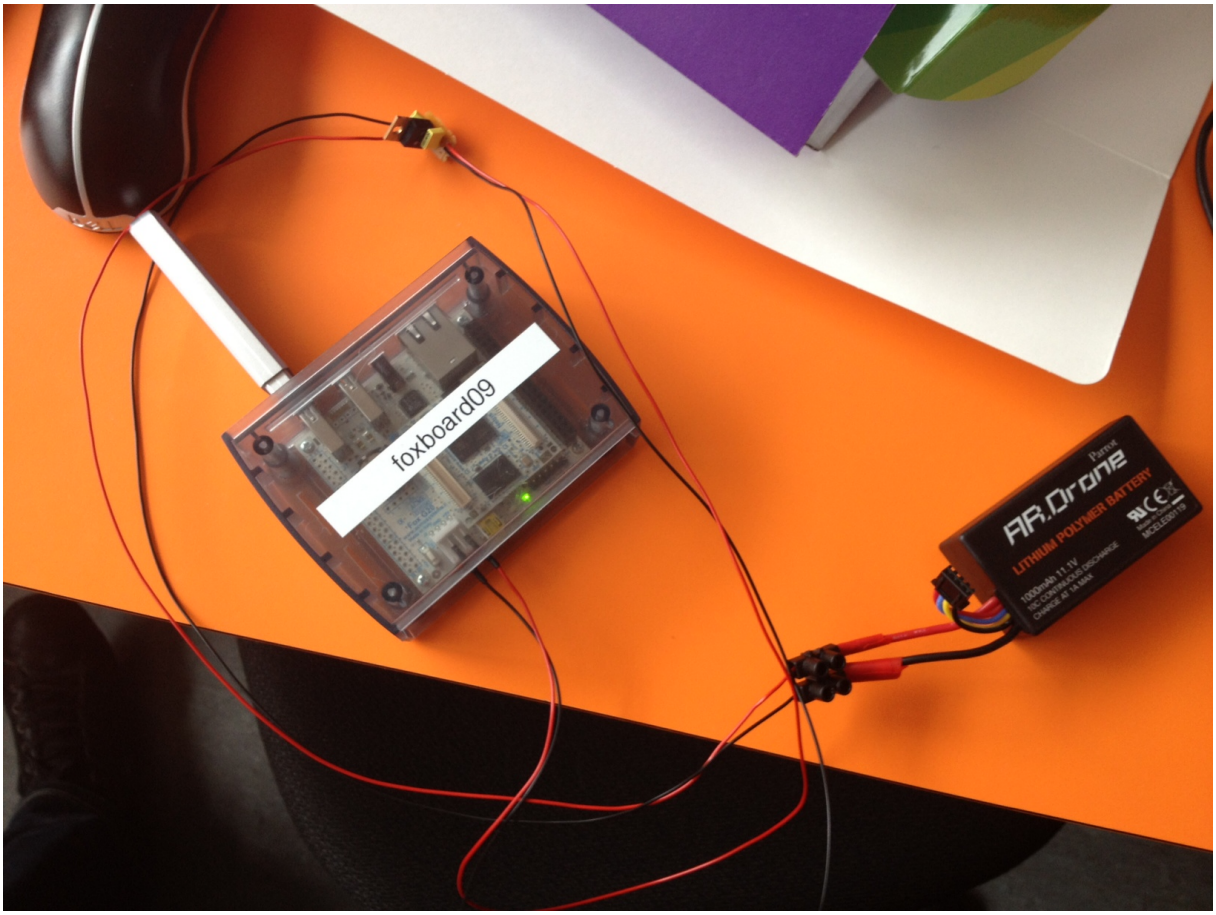
Image 6 : FoxBoard sur le Drone

En fixant la FoxBoard sur le drone ainsi, celui-ci, décolle sans trop de mal et se stabilise sans soucis. Nous contrôlons le drone depuis la FoxBoard (via notre programme C) mais

bien entendu un membre du binôme est toujours connecté avec son iPhone pour un éventuel arrêt d'urgence quand le programme ne répond plus.

Après plusieurs tests, et devant la dangerosité de notre montage, nous avons demandé de l'aide pour la réalisation d'un régulateur de tension, nous avons par la même occasion « customiser » la batterie afin d'alimenter à la fois le drone et la FoxBoard.

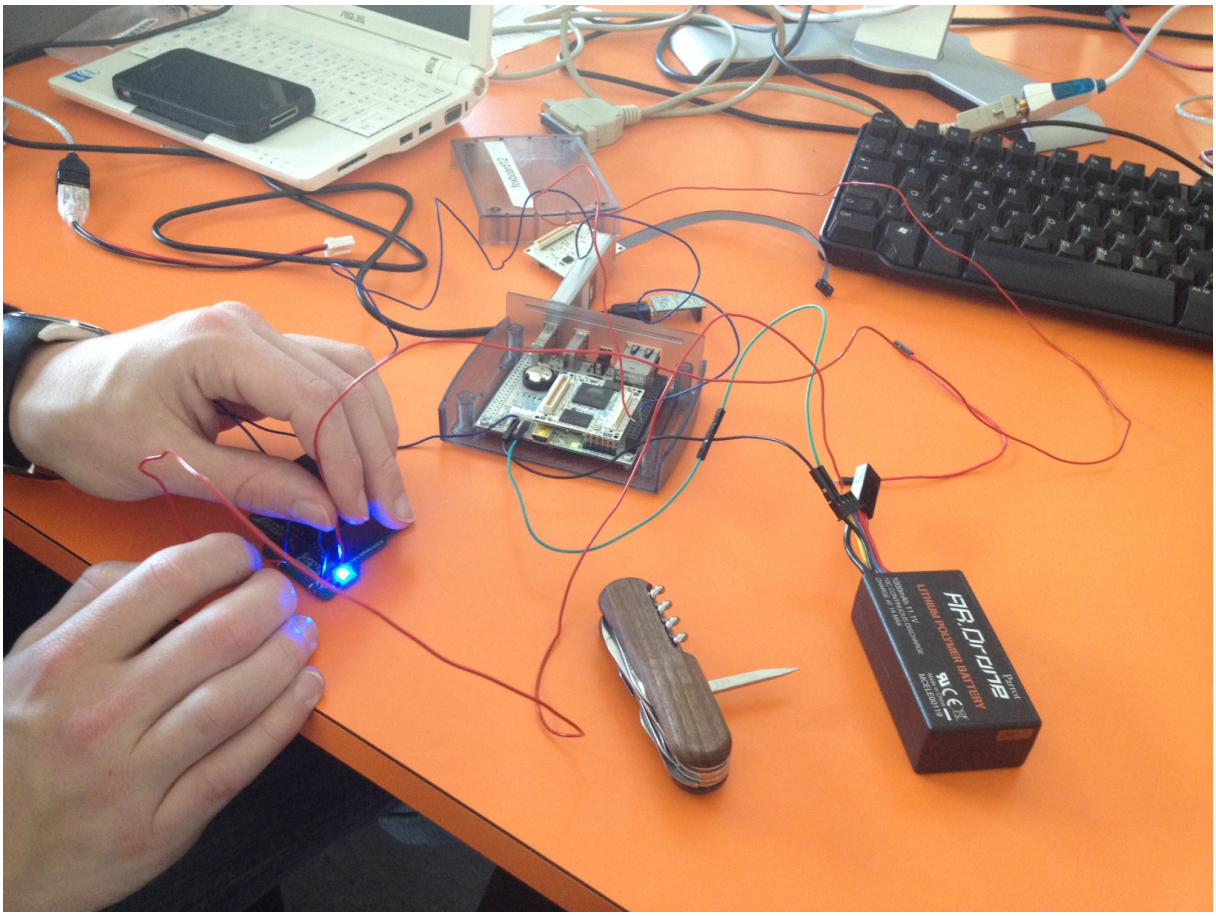
Image 7 : Régulateur de tension et batterie



6. Puce GPS

Nous tentons maintenant de localiser notre FoxBoard et donc notre drone via une puce GPS.
Nous voulons récupérer les données de celle-ci via une communication I2C avec la FoxBoard. Mais tout comme la FoxBoard, nous devons résoudre le problème de l'alimentation de la puce GPS. Nous pouvons soit l'alimenter via le régulateur de tension (5V) soit via les pins de la FoxBoard (3,3V).

Image 8 : Puce GPS alimentée par la FoxBoard



Une fois alimentée, il faut récupérer les données via les broches I2C de la FoxBoard que l'on a trouvé grâce à ce schéma :

FOX J6 pinout

Fox pins	Signal	Kernel ID	
J6.1	3.3V		3.3 volt power line
J6.2	3.3V		3.3 volt power line
J6.3	PB28	92	General purpose I/O
J6.4	PB7	71	General purpose I/O
J6.5	PB6	70	General purpose I/O
J6.6	PB29	93	General purpose I/O
J6.7	PB26	90	General purpose I/O
J6.8	PB5	69	General purpose I/O
J6.9	PB4	68	General purpose I/O
J6.10	PB27	91	General purpose I/O
J6.11	N.C.		Not connected
J6.12	5V		5 volt power line
J6.13	PB11	75	General purpose I/O
J6.14	PB10	74	General purpose I/O
J6.15	PB13	77	General purpose I/O
J6.16	PB12	76	General purpose I/O
J6.17	PB21	85	General purpose I/O
J6.18	PB20	84	General purpose I/O
J6.19	PB31	95	General purpose I/O
J6.20	PB30	94	General purpose I/O
J6.21	PA31	63	General purpose I/O
J6.22	PA30	62	General purpose I/O
J6.23	N.C.		Not connected
J6.24	PA6	38	General purpose I/O
J6.25	PA7	39	General purpose I/O
J6.26	PA9	41	General purpose I/O
J6.27	PC3	99	General purpose I/O
J6.28	PC2	98	General purpose I/O
J6.29	PC1	97	General purpose I/O
J6.30	PC0	96	General purpose I/O
J6.31	SCL		I2C Clock
J6.32	SCD		I2C Data
J6.33	AVDD		Clean 3.3V out for A/D circuitry
J6.34	VREF		A/D voltage reference input
J6.35	AGND		Analog ground
J6.36	PA10	42	General purpose I/O
J6.37	PA22	54	General purpose I/O
J6.38	PA11	43	General purpose I/O
J6.39	GND		Signal ground
J6.40	GND		Signal ground

Nous avons trouvé un programme C pour la communication I2C de la FoxBoard :

```
..
// Example to read A/D values from a
// 4 channel / 8 bit AD converter PCF8591
// through I2C using the I2C driver improved by
// Geert Vancompernolle
// http://www.acmesystems.it/?id=10
//*****************************************************

#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "sys/ioctl.h"
#include "fcntl.h"
#include "time.h"
#include "string.h"

#include "i2c_errno.h"
#include "etraxi2c.h"


int main( int argc, char **argv ) {
  int rtc;
  int fd_i2c;
  I2C_DATA i2c_d;
  int ch;

  printf("Reading from a PCF8591 (4 chanel A/D at 8 bits with I2C bus)\n");

  fd_i2c = open( "/dev/i2c-0", O_RDWR );
  if (fd_i2c<=0)     {
    printf( "Open error on /dev/i2c\n" );
    exit( 1 );
  }

  // PCF8591 address scheme
  // |  1 |  0 |  0 |  1 | A2 | A1 | A0 | R/W |
  i2c_d.slave =(0x09<<4)|(0x01<<1);

  for (ch=0;ch<=3;ch++) {
    // Select the A/D channel
    i2c_d.wbuf[0] = ch;
    i2c_d.wlen  = 1;
    if ((rtc=ioctl(fd_i2c,_IO( ETRAXI2C_IOCTYPE, I2C_WRITE), &i2c_d))!=EI2CNOERRORS)  {
      close(fd_i2c);
      printf( "Error %d on line %d\n",rtc,__LINE__);
      return ( -1 );
    }

    i2c_d.rlen  = 3;
    if ((rtc=ioctl(fd_i2c,_IO( ETRAXI2C_IOCTYPE, I2C_READ), &i2c_d))!=EI2CNOERRORS)  {
      close(fd_i2c);
      printf( "Error %d on line %d\n",rtc,__LINE__);
      return ( -1 );
    }

    // Show the voltage level
    printf("Chanel %d = %.2fv (%02X hex)\n",ch,i2c_d.rbuf[2]*0.012941,i2c_d.rbuf[2]);
  }

  close(fd_i2c);
  return(0);
}
```

Cependant nous celui-ci nous renvoie toujours une erreur « Open error on /dev/i2c ». Nous avons installé les paquets nécessaires à l'I2C sur la FoxBoard, mais sans succès.

Nous avons donc décidé de réinstaller le kernel de la FoxBoard. Après l'avoir fait, le problème n'était toujours pas résolu et nous avions malheureusement atteint l'échéance du projet.

Conclusion

Lorsque nous avons choisi ce projet, nous savions que celui-ci était long, et qu'il englobait beaucoup de domaines différents.
Mais cela ne nous a pas rebuté. Au fur et à mesure que les séances passaient, nous passions d'une problématique purement informatique (programmation), à une problématique plus « pratique » (comment faire décoller le drone avec la FoxBoard) ou encore électronique (alimentation, régulateur de tension). Il est sur que cela ne nous a pas aidé à avancer rapidement, mais nous avions vraiment le sentiment de faire avancer un vrai projet, qui englobait beaucoup de notions que ne nous maîtrisions pas forcément, et cette difficulté était notre motivation.

Ce projet nous a permis de gagner en autonomie, même si l'aide des encadrants nous était souvent de grande utilité.