Projet Avancé en Système Embarqué

ARDrone Parrot

IPB
ENSEIRB
MATMECA
BORDEAUX

DEVELLE Kevin (E3A SE)
ESCANDE Léo
FLAMENBAUM Raphaël
WOUTERS Thomas

ARDrone Parrot

T. AHMED 2012/2013

Table des matières

1. ARDRONE PARROT.

L'ARDrone PARROT est une plateforme quadri-rotors équipée de moteurs brushless, lui assurant sa sustentation. Chaque moteur se situe à l'extrémité d'un des quatre bras.

Afin de piloter les moteurs, une carte spécialement dédiée à leur manipulation se situe près d'eux. Elle permet de retransmettre les ordres envoyés par le noyau et la carte mère du drone. En effet, celui-ci est équipé de son propre noyau. Un microprocesseur ARM 9 est ainsi embarqué et gère le drone. Par exemple, lorsque le drone doit se déplacer longitudinalement, l'ARM va communiquer avec chaque carte de gestion des moteurs afin de pouvoir leur transmettre les ordres. C'est par les différences de vitesses de rotation de chaque moteur que le mouvement est insufflé au drone. Cela est également possible grâce au fait que chaque paire d'hélice est contrarotative par rapport à l'autre.

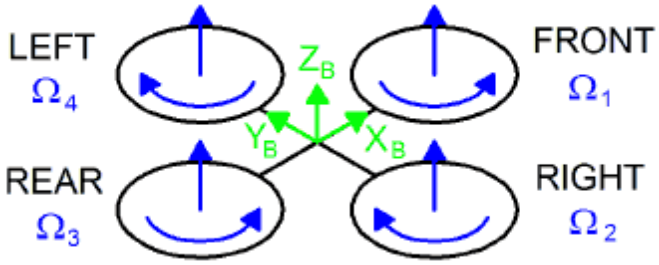

Figure 1 : Illustration des hélices contrarotatives.

Même si le drone est soumis à la dynamique des vols pour assurer sa stabilité, différents gyroscopes, une caméra orientée vers le sol et des senseurs à ondes ultrasonores sont présents pour que l'ARM puisse stabiliser le drone et posséder des informations quant à la vitesse et la distance au sol de la plateforme. Toute cette batterie de senseurs est présente pour rendre automatique le décollage, l'atterrissage ainsi que certains types de hover de la part du drone. Les trims sont également gérés automatiquement, grâce à la présence d'un fichier dans le noyau du drone, définissant les valeurs par défaut.

Il sera crucial de bien comprendre les différents aspects du vol pour pouvoir manipuler convenablement les évolutions du drone.

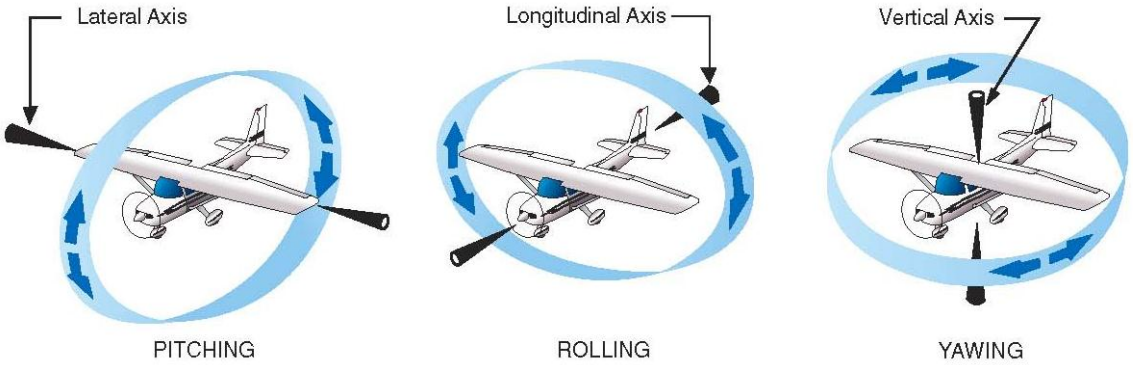

Figure 2 : Illustration des trois axes autour desquels le drone va évoluer dans l'espace (dans l'ordre : tangage, roulis, lacet).

Le Parrot possède également une caméra embarquée permettant de filmer l'avant du véhicule. Celle-ci peut être utilisée pour retransmettre en temps réel les images. En effet, le drone est une plateforme wifi et peut donc communiquer avec diverses plateformes interactives (smartphone, tablette, ordinateur, …).

Figure 3 : Caméra embarquée sur l'ARDrone Parrot.

2. CAHIER DES CHARGES.

Bien qu'il s'agisse d'un drone du commerce, celui-ci peut se transformer en une plateforme pour la programmation, et ce grâce à son propre noyau que l'on peut modifier. On peut également trouver le kit de développement internet, mais celui-ci possède la limite d'avoir une modularité limité, voir même parfois qui présente quelques bugs (pour la version du Parrot que nous utilisons).

Nous avons trois objectifs :

- Réaliser un code embarqué qui s'exécute sur le drone lui-même. Il s'agit en réalité d'une trajectoire prédéfinie implantée en dur. Ainsi, la plateforme doit, de son propre chef, effectuer des évolutions précises, définies par le programmeur.
- Réaliser un asservissement depuis un ordinateur qui échange en permanence avec le drone. Le poste fixe envoie des commandes au Parrot qui les exécute et renvoie les informations importantes sur sa situation.
- Réaliser une stratégie "Sense & Acquire". Grâce à la caméra embarquée à l'avant, le Parrot doit se déplacer vers des cibles (marqueurs fluorescents) et se mettre dessus. Il doit ainsi détecter/reconnaître les tags et ensuite mettre en place une stratégie pour acquérir la cible.

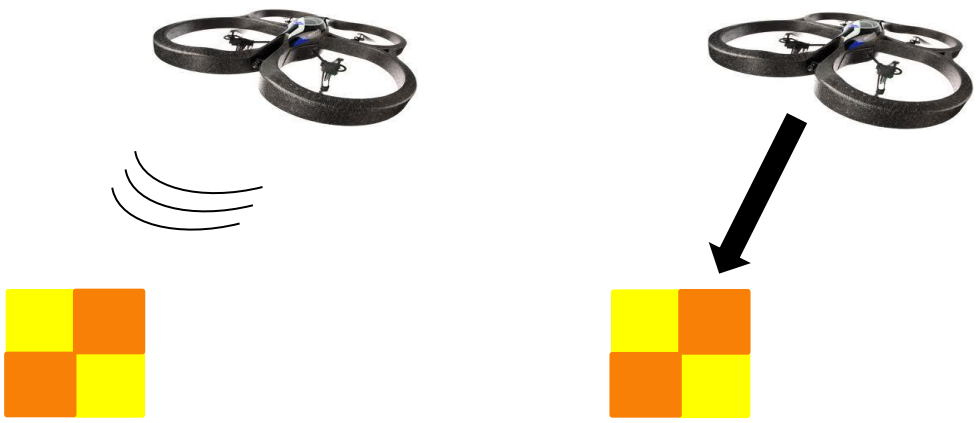

Figure 4 : Illustration de la stratégie "Sense & Acquire".

3. PREMIERS PAS AVEC LE DRONE.

Plutôt que de se précipiter en commençant à coder, nous avons décidé de nous familiariser à la manipulation du drone par le biais de l'application pour smartphone ainsi que l'utilisation du logiciel de pilotage développé par Parrot, disponible sur internet.

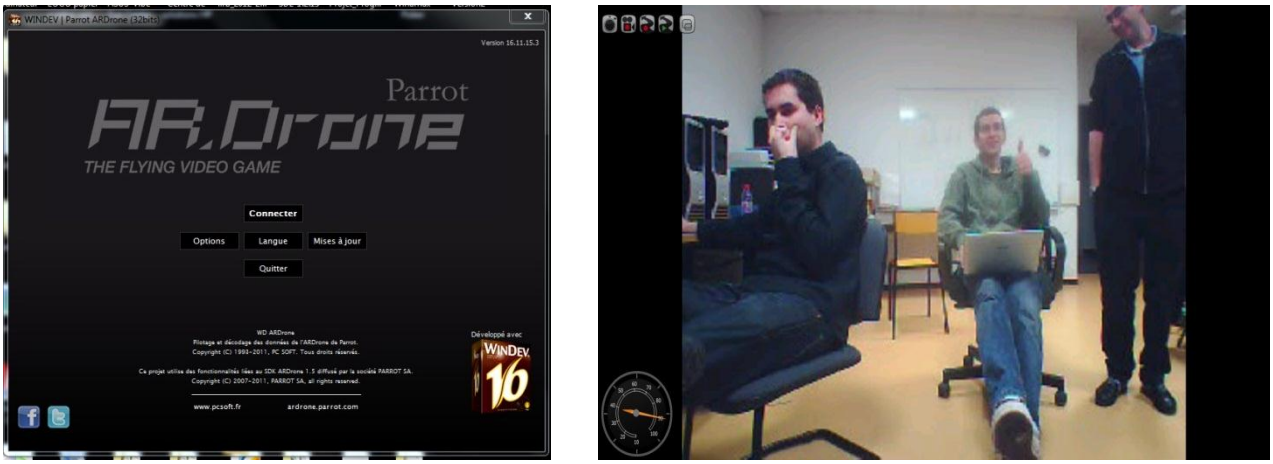

Figure 5 : Utilisation du logiciel Parrot pour se familiariser avec le maniement du drone.

4. CODE EMBARQUE.

L'objectif de cette partie est de réaliser un code qui est embarqué en dur sur le drone. Deux options s'offrent à nous : lancer l'exécution du code grâce à un ordinateur et le drone devient ainsi autonome, ou alors faire en sorte que le programme s'exécute une fois la phase d'initialisation passée.

4.1. Cross-compilation et test basique

L'ARDrone Parrot posséde un ARM embarqué, afin de pouvoir coder la trajectoire fixe, nous avons procédé par étapes.

Tout d'abord, il nous faut rechercher un cross-compilateur pour que le code compilé puisse être interprété par le processeur. Ainsi, nous avons été à même de trouver un cross-compilateur pour le Parrot : "`arm-none-linux-gnueabi-gcc`" Pour ce faire, nous avons utilisé un script permettant de passer en environnement de cross-compilation pour l'ARM. Cela nous a permis par la suite de compiler un "Hello World".

```
kevin@ubuntu:~$ codesourcery-arm-2009q3.sh
Type 'exit' to return to non-crosscompile environment
NOW in crosscompile environment for arm (arm-none-linux-gnueabi-)
kevin@ubuntu:~$ arm-none-linux-gnueabi-gcc hello.c -o hello_arm
```

Figure 6 : Environnement de cross-compilation pour ARM.

Afin de vérifier le bon fonctionnement, nous devons uploader le code sur le drone. Pour ce faire, nous nous connectons sur le port 5551, de l'adresse 192.168.1.1, pour pouvoir le déposer. Celui-ci est ainsi disponible dans le dossier update de la plateforme.

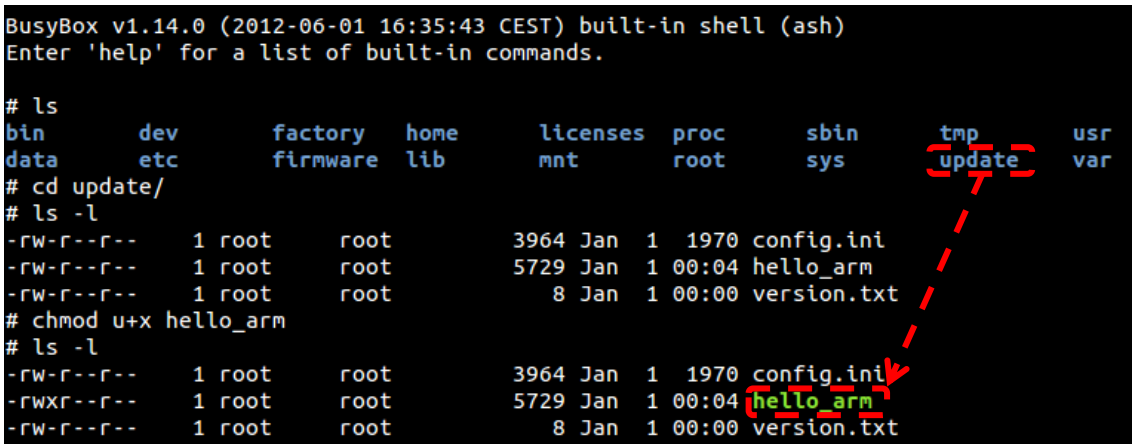

Figure 7 : Contenu du dossier Update du drone.

Afin de pouvoir exécuter le programme, nous nous connectons en Telnet pour pouvoir lancer l'exécutable.

```
# ./hello_arm
Hello world
```

Figure 8 : Exécution du "Hello World!" cross-compilé sur la plateforme.

Cette première étape était très importante. En effet, même si le code de notre programme se résumait à un simple `printf("Hello World\n") ;` ne présentant aucune difficulté, cela nous a permis de confirmer la bonne installation des différents outils nécessaires à la réalisation de programmes embarqués pour processeur ARM.

4.2. Code embarqué

4.2.1. Utilisation de la SDK

L'objectif de cette partie est d'embarquer un programme de vol autonome sur notre drone. A l'aide des fonctions fournies dans le "Source Development Kit" (SDK) du drone, nous avons souhaité réaliser un programme permettant à notre drone de décoller, d'exécuter successivement plusieurs consignes de vol et enfin atterrir, le tout sans intervention extérieur une fois le programme lancé. Dans les bibliothèques de la SDK, de nombreuses fonctions liées à l'utilisation de notre drone sont fournies. En voici quelques exemples :

`ardrone_tool_set_ui_pad_start(int value) ;`

➔ Il s'agit de la fonction permettant d'assurer le décollage (value=1) et l'atterrissage (value=0) du drone

```
ardrone_at_set_progress_cmd(int flags, float phi, float teta, float gaz, float yaw) ;
```

➔ Il s'agit de la fonction qui permet de contrôler les déplacements du drone. L'argument flag est un entier qui permet d'activer les commandes progressives du drone lorsqu'il vaut 1. Les autres arguments sont des réels compris entre -1 et 1 : les valeurs positives permettant de se déplacer dans un sens, les négatives dans l'autre. L'argument phi permet de gérer les déplacements latéraux, teta permet d'avancer ou reculer, gaz permet de monter ou descendre, yaw sert à assurer les rotations du drone sur lui-même (lacet).

Ces fonctions étant clairement définies, la réalisation de programme de vol autonome semblait à première vue assez simple. Cependant, nous nous sommes très rapidement heurtés à de nombreuses difficultés liées au kit de développement fourni. Notre première approche a été d'essayer de réaliser un programme simple permettant de faire clignoter les LEDs du drone et un second s'assurant de le faire décoller. Après avoir recherché dans le guide de la SDK les fonctions adéquates pour nos programmes ainsi que les fichiers "header" à inclure, nous avons réalisé le programme de l'animation des LEDs. Cependant, lors de la compilation de ce dernier, le problème suivant est apparu, des fichiers headers, a priori non-nécessaires, manquaient.

Après avoir essayé de résoudre ce problème en ajoutant le répertoire de ce premier fichier "header" à l'aide de l'option –I de `gcc`, nous nous sommes aperçus qu'un second manquait à l'appel. Nous avons donc regardé les différents fichiers ".h" nécessaires à notre projet et nous nous sommes rendus compte du gros problème de cette SDK : sa non-modularité. En effet, elle présente deux gros problèmes lors de son utilisation. Le premier est l'imbrication des fichiers "headers" entre eux. Chaque fichier ".h" en appelle plusieurs autres qui à leurs tours vont en appeler plusieurs, et devant le nombre très conséquent de fichiers ".h" que contient la bibliothèque du drone, ce problème est apparu très rapidement insurmontable. Voici un schéma résumant le problème dans le cas de notre programme de clignotement de LEDs.

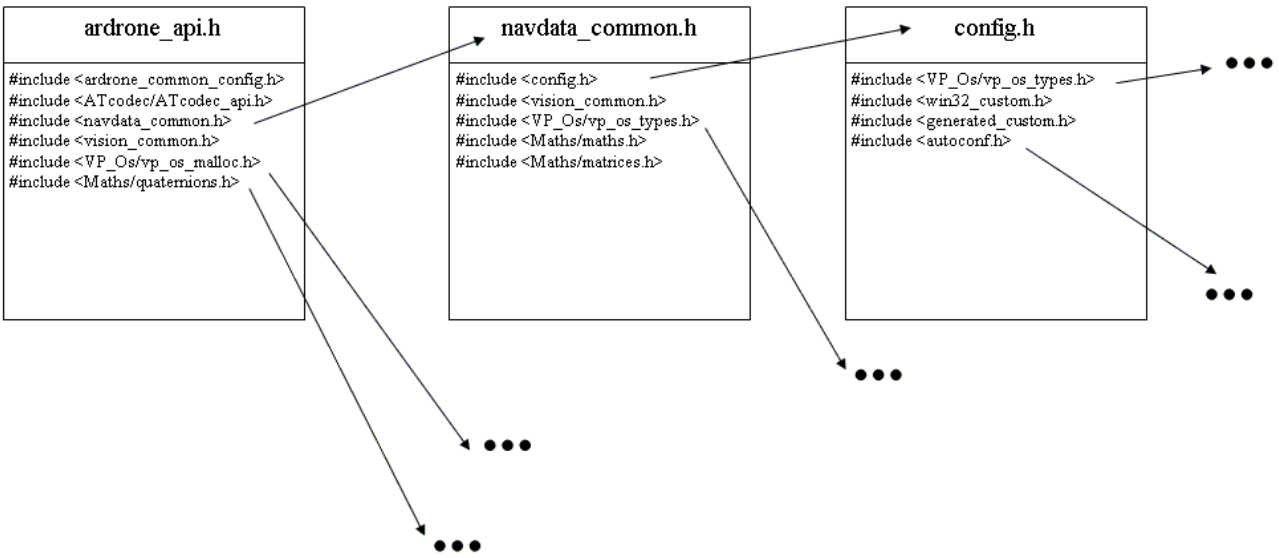

Figure 9 : Illustration du problème d'inclusion des headers imbriqués

Un deuxième problème d'inclusion des "headers" est également apparu. Au sein d'un même fichier, l'inclusion d'un fichier ".h" n'est pas toujours réalisée de la même façon. Les inclusions s'effectuant en chemin relatif n'aurait pas dû poser de problème au premier abord, mais nous avons découvert quelques problèmes d'utilisation des chemins d'inclusions. En voici un exemple :

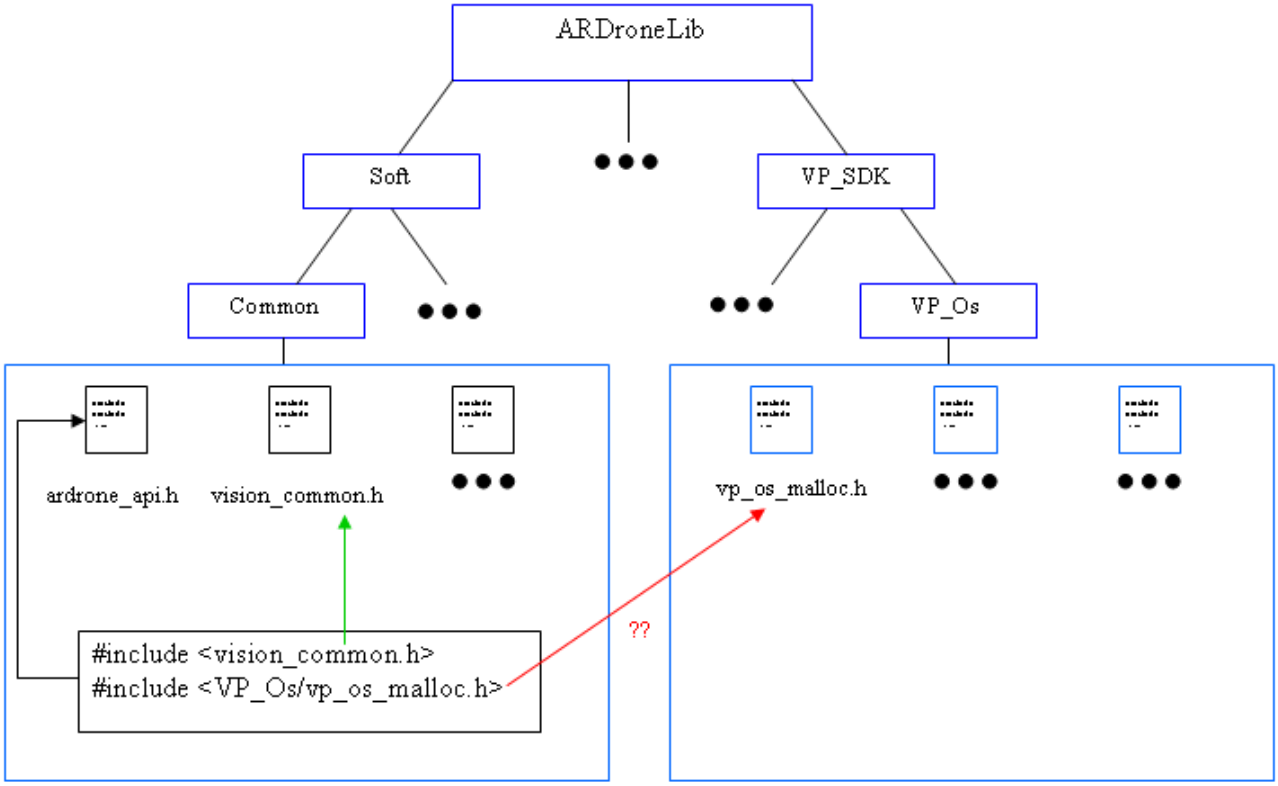

Figure 10 : Illustration du problème des chemins d'inclusion

Sur cette figure, nous pouvons constater que le chemin vers le fichier vp_os_malloc.h n'est pas correctement défini.

Ces deux problèmes nous ont poussés à utiliser une autre approche, la réutilisation des makefile fournis dans la SDK. Cependant, ces derniers étant très complexe, très nombreux, avec de nombreux fichiers permettant de les configurer, nous n'avons pas réussi à les adapter à notre outil de cross-complication ni au type de programme que nous souhaitions réaliser. Les informations relatives à l'installation et l'utilisation d'un cross-compilateur se trouvent en annexe 1. Pour la réalisation d'un programme embarqué, nous avons donc essayé une autre approche, sans utilisation de la SDK.

4.2.2. Programmation directe

Devant les problèmes rencontrés avec la SDK, nous avons décidé de réaliser une deuxième approche pour réaliser un programme embarqué de vol autonome, commander directement les moteurs du drone. En agissant différemment sur chacun des rotors du drone, nous pouvons lui faire exécuter des déplacements. Cette approche nous permet de nous affranchir des inconvénients liés aux bibliothèques de la SDK, mais en comporte d'autres. Le premier est la gestion des commandes de vol, celles fournies dans la SDK étaient très simples, tandis qu'avec notre programme, nous devons agir sur chaque hélice et bien étudier les conséquences de chaque fonction. Le second concerne l'aspect stabilisation du drone. En effet, celui-ci est prévu pour se stabiliser automatiquement à l'aide de ses senseurs et du traitement des informations en interne. Enfin le dernier est un problème de sécurité, contrairement au programme utilisant la SDK Parrot, notre programme n'assure plus la coupure des moteurs en cas de retournement ou bien l'atterrissage en douceur lorsque la batterie atteint un niveau critique. Pour cela, nous avons réutilisé un programme d'un autre développeur en le modifiant pour pouvoir réaliser une application plus proche de nos attentes. Cependant, par manque de temps, nous n'avons pas pu réaliser une vrai stratégie de vol embarqué, ce programme ne reposant pas sur la SDK, nous n'avions eu à notre disposition les fonctions toutes prêtes tels que celle réalisant un décollage. Les explications concernant ce programme se trouve en annexe 2.

5. MANIPULATION VIA PC.

5.1. Sous Linux (Ubuntu)

Dans cette partie, nous utilisons la SDK et les exemples fournis pour réaliser un programme de vol autonome, commandé par le PC. En étudiant les sources du programme « linux_sdk_demo », nous avons repéré le thread qui gère l'utilisation d'une manette ou d'un joystick pour piloter le drone. En utilisant ce thread, nous avons pu insérer directement nos commandes et ainsi gérer le pilotage du drone. Le thread gérant la mise à jour des commandes de la manette étant mis à jour à intervalle régulier. Pour le mesurer, nous avons utilisé un compteur que nous incrémentons à chaque passage dans le thread. Nous avons chronométré 4000 cycles pour obtenir une mesure précise du temps de cycle afin de gérer les temporisations. Le drone dont nous avons cherché la cadence a un temps de cycle de 20ms, cependant, nous avons constaté que ce temps de cycle pouvait varier selon les drones. Toutes les informations relatives à l'utilisation de notre programme, les outils nécessaires à installer ainsi que le code du programme se trouve en annexe 3. Contrairement à la partie ou nous embarquions le code sans utiliser la SDK, pour cette partie nous l'avons conservée et donc avons pu réaliser un programme de vol avec décollage, rotation du drone sur lui-même, déplacement, ascension et enfin atterrissage.

La réalisation de notre programme réutilisant directement un exemple de la SDK, nous avons pu conserver intact certaines parties intéressantes du programme. Par exemple, il y a un thread qui permet d'afficher en permanence les données de vol du drone, pendant l'exécution du programme que nous avons réalisé.

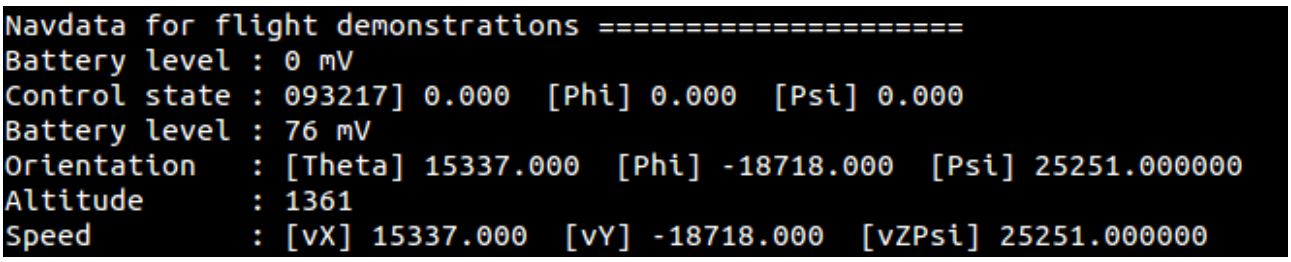

Figure 11 : Affichage des données de vol

5.2. Sous Windows

Cette partie est très similaire à la précédente, nous avons exploité l'exemple Windows fourni par la SDK pour réaliser le même programme que précédemment. Nous avons décidé de travaillé en parallèle sous Linux et Windows pour plusieurs raisons. La première était de permettre d'avancer dans deux voies séparées en cas de problème avec une approche. Ce choix fut judicieux, tous les PC Windows ne permettant pas cette approche, ayant eu des problèmes avec l'installation de certains outils. De plus, cette partie était le premier pas de l'approche pour la partie "Sense & Acquire", cela nous permettait de disposer de deux bases de travail distinctes pour la suite. Les informations relatives à l'installation des outils nécessaires et leurs utilisations se trouvent en annexe 4.

6. STRATEGIE “SENSE & ACQUIRE”.

Afin de réaliser la stratégie “Sense & Acquire”, nous comptons utiliser un ordinateur qui sera en permanence relié au drone et analysera les flux vidéos qui lui sont transmis par ce dernier en temps réel. La caméra avant va donc filmer et envoyer les images. L’ordinateur devra de son côté recevoir les images et les analyser pour reconnaître les tags. La reconnaissance vidéo est ainsi la première des étapes à réaliser. Pour cela, nous comptons employer la bibliothèque OpenCV.

OpenCV est une bibliothèque graphique permettant de réaliser des opérations de traitement des images. Par son utilisation, il est possible de pouvoir discerner les tags disposés sur le sol.

Une fois que la partie reconnaissance est effectuée, le drone doit ainsi se déplacer vers la cible. L’ordinateur devra faire en sorte de rapprocher le drone de la cible.

Ainsi, lorsque l’ordinateur aura analysé une image en provenance du drone, il va mettre la cible au centre de la vision du drone par diverses manœuvres automatisées. Puis, le drone ira tout droit et lorsque la cible disparaîtra de son champ de vision, la cible sera alors considérée comme acquise. Le drone ne fera qu’exécuter les commandes que le PC lui enverra.

Remarque : Suites aux difficultés rencontrées et au manque de temps, nous n’avons pas été à même de réaliser cette stratégie, en effet avant de pouvoir réaliser cette approche, nous devions être certains de disposer d’une ordinateur ou tous les outils nécessaires à l’utilisation de la SDK devait être proprement configuré, cette partie dépendant donc de la validation de l’étape précédente.

7. CONCLUSION.

L'objectif de ce projet était d'implémenter trois stratégies différentes pour la plateforme AR Drone Parrot.

Dans un premier temps implémenter une trajectoire fixe au sein du noyau du drone basé sur le SDK officielle, nous avons rencontré quelques difficultés pour compiler l'ensemble des bibliothèques fournies. Cependant, nous avons été à même de contourner cette difficulté repartant d'un logiciel développé par un particulier. Nous avons ainsi pu réaliser un programme embarqué faisant tourner un à un les moteurs du drone. Cependant, en réalisant une telle manipulation, nous avons dû faire un "kill" du processus principal présent sur le drone qui est /usr/bin/program.elf. Ce processus permettant au drone d'utiliser sa batterie de senseurs et qui par traitement lui permet d'être stable. Si l'on avait eu le temps de continuer sur notre lancée, nous aurions dû incorporer notre code dans ce processus. Le drone parrot se veut donc ouvert, mais en réalité, la non-modularité de sa SDK complique très vite la tâche de quiconque souhaitant développer des programmes pour celui-ci.

Dans un second temps, nous avons travaillé sur le développement d'une stratégie où le drone est « l'esclave » d'une station sol qui lui envoie en permanence des ordres. Ainsi, grâce à la connexion wifi directe sur le drone, nous avons été en mesure de faire évoluer le drone dans un environnement. Plutôt que de nous baser sur des relevés métriques pour définir la trajectoire, nous avons utilisé la temporisation du drone.

Dans un troisième temps, il aurait fallu réutiliser la stratégie précédente pour implémenter une stratégie "Sense & Acquire". Nous aurions ainsi utilisé le retour vidéo du drone sur une station sol où les images auraient été analysées grâce à l'emploi de la bibliothèque OpenCV. En effet, grâce aux différentes API, nous aurions été en mesure de pouvoir reconnaître les tags présents au sol. Par la suite, les ordres auraient été donnés au drone pour que ce dernier s'en approche.

Ce qui aurait pu clôturer le projet aurait été de réaliser un vol en formation de plusieurs drones. Ainsi, un premier drone volerait de lui-même par le biais d'une stratégie "Sense & Acquire", ou bien par une trajectoire fixe, puis les autres drones l'auraient suivi. Pour cela, le chef de formation aurait été équipé du fuselage "taggué" et les autres auraient utilisé la stratégie "Sense & Acquire" pour le suivre.

Ce projet n'a pas également été qu'un travail technique, mais surtout une expérience humaine. En effet, il a fallu s'organiser pour essayer de réaliser le maximum d'objectifs définis. Nous nous sommes donc tout d'abord attelé à la familiarisation de la plateforme, puis nous nous sommes répartis les tâches en équipes. Afin de pouvoir réaliser le maximum de choses en parallèle. Cependant, les différentes équipes étaient en totale interaction pour pouvoir communiquer sur l'état d'avancement du projet. Cette flexibilité nous a également permis de pouvoir à tout moment faire face à plusieurs problèmes, notamment en cas d'absence de certains, afin de pouvoir combler le manque de moyens humains.

Annexe 1 : la cross-compilation

1) Sous Windows

Pour cross-compiler des programmes sous Windows, un seul outil est nécessaire : Sourcery G++ Lite 2011.03-41 for ARM GNU/Linux. Il peut être trouvé à l'adresse suivante :

http://www.codesourcery.com/sgpp/lite/arm/portal/release1803

Après l'avoir correctement installé (nous l'avons utilisé sous Windows 7, 64bits), il doit être utilisé en console à l'aide de la commande :

arm-none-linux-gnueabi-g++.exe

Cette commande s'utilise de la même manière que gcc sous un environnement unix, il suffit juste d'indiquer le nom du fichier à compiler. Comme pour gcc, il est possible d'utiliser des makefile nommés « make.bat » sous windows et d'ajouter des options à la compilation.

2) Sous Ubuntu

L'environnement de cross-compilation sous linux est identique à celui utilisé sous windows. Pour pouvoir le télécharger et l'installer, il suffit d'exécuter le script qui peut être téléchargé à cette adresse :

http://taghof.github.com/Navigation-for-Robots-with-WIFI-and-CV/downloads/codesetup.sh

Deux problèmes peuvent être rencontrés lors de cette étape :

- Selon le pc, le script ne sera peut être pas exécutable par défaut, pour cela, la solution est un simple chmod u+x
- Les connections internet comme celle à l'ENSEIRB-MATMECA nécessite l'utilisation de proxy, cela causera des erreurs lors du téléchargement du fichier. Dans ce cas, il suffit de saisir ces deux lignes dans le terminal et de relancer l'exécution du script :

```
export https_proxy="http://proxy.enseirb-matmeca.fr:3128"
export http_proxy="http://proxy.enseirb-matmeca.fr:3128"
```

Une fois le programme cross-compilé, il suffit de se connecter sur le drone et envoyer le fichier de sortie du cross-compilateur à l'aide du logiciel filezilla. Par défaut, la connexion sur un drone se fait à l'adresse IP 192.168.1.1 avec le port 5554 pour envoyer le fichier. Celui-ci sera par défaut placé dans le dossier update du drone.

La dernière étape est de se connecter via telnet à l'aide de la commande suivante :

telnet 192.168.1.1

Par défaut, sous Windows, le client telnet n'est plus activé, pour l'activer, il faut se rendre dans le menu « désinstaller ou modifier un programme », puis sur la gauche aller dans le menu « activer ou désactiver des fonctionnalités Windows ».

Une fois connecté, il ne reste plus qu'à rendre le fichier exécutable à l'aide d'un chmod u+x et enfin l'exécuté comme l'on exécute tout programme sous un environnement Unix.

Annexe 2 : Réalisation d'un programme embarqué

Cette approche de programme embarqué est une approche visant à remplacer le programme embarqué sur le drone. En effet, celui-ci n'est pas ouvert et notre approche visant à utiliser les fonctions de la SDK n'ayant pas abouti. Nous avons trouvé les projets suivant qui nous a servi de base de travail :

http://blog.perquin.com/prj/ardrone/CustomFirmwareArDrone.zip

Pour cette partie, nous avons utilisé uniquement le contenu du dossier ardrone/motorboard. Celui-ci contenant quelques fichiers ainsi qu'un make.bat pour les compiler. Nous avons apporté nos modifications au main_motorboard.c pour réaliser la démonstration de notre soutenance, à savoir démarrer un à un les moteurs puis les éteindre.

Après nous être placé dans le dossier motorboard, nous avons cross-compilé notre programme à l'aide d'un simple « make ». Nous l'avons ensuite embarqué dans le drone comme expliqué en annexe 1. Avant de pouvoir lancer ce nouveau programme, nous avons dû nous placer sous telnet dans le dossier /usr/bin du drone et saisir l'instruction suivante :

```
killall program.elf
```

Nous avons ainsi stoppé le programme d'origine du drone, il ne nous reste plus qu'à exécuter notre propre programme. Ce programme embarqué n'est qu'un petit aperçu des possibilités de programme embarqué, cependant, sans la SDK, nous n'avons pas eu suffisamment de temps pour faire réellement voler le drone.

Il est nécessaire de modifier les droits du programme motorboard afin de le rendre exécutable.

```
Chmod 777 motorboard
```

Enfin, on peut lancer notre premier code embarqué !

```
./motorboard
```

Remarque : Pour relancer le programme d'origine, il suffit de se replacer dans le dossier /usr/bin et saisir la commande :

```
program.elf &
```

Notre programme main_motorboard.c est le suivant :

```
#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <stdlib.h>  //exit()
#include <pthread.h>
#include <ctype.h>    /* For tolower() function */
#include <math.h>

#include "../util/type.h"
#include "../util/util.h"
#include "mot.h"


int main()
{

  mot_Init();
  float step=0.01;

  //main loop
  while(1)
  {
          printf("DEBUT BOUCLE\n");
          if(n<2){
                                  mot_Run(.50,0,0,0);//rotation d'un moteur à 50%
                                  sleep(2);
                                  mot_Run(.50,.50,0,0);//rotation de 2 moteurs à 50%
                                  sleep(2);{
                                  mot_Run(.50,.50,.50,0);
                                  sleep(2);
                                  mot_Run(.50,.50,.50,.50);//rotation des 4 moteurs à 50%
                                  sleep(2);
                                  mot_Run(0,.50,.50,.50);
                                  sleep(2);
                                  mot_Run(0,0,.50,.50);
                                  sleep(2);{
                                  mot_Run(0,0,0,.50);
                                  sleep(2);
                                  mot_Run(0,0,0,0);//arret des moteurs
                                  sleep(2);
                          }
                  printf("ARRET");
                  printf("FIN BOUCLE\n");

        }

        //yield to other threads
        pthread_yield();

  mot_Close();
  return 0;
  }
  }
}
```

Annexe 3 : Manipulation à distance via le PC (Ubuntu)

Pour réaliser un programme de manipulation à distance, permettant de définir une stratégie de vol autonome au drone, la plupart des informations se trouvent dans l'ARDrone SDK 1.7 Developer Guide disponible à cette adresse :

https://projects.ardrone.org/login?back_url=http%253A%252F%252Fprojects.ardrone.org%252Fattachments%252Fdownload%252F365%252FARDrone_SDK_1_7_Developer_Guide.pdf

Pour réaliser notre programme de vol, nous avons réutilisé un exemple fourni dans la SDK disponible à cette adresse :

https://projects.ardrone.org/login?back_url=http%253A%252F%252Fprojects.ardrone.org%252Fattachments%252Fdownload%252F373%252FARDrone_SDK_Version_1_8_20110726.tar.gz

Une fois la SDK téléchargé, la première étape est d'installé les outils nécessaires à cette dernière.

Sous Ubuntu, la commande à exécuter est la suivante :

```
sudo apt-get install libsdl-dev libgtk2.0-dev libiw-dev
```

Il faut ensuite modifier un fichier de la SDK pour pouvoir compiler les exemples sans erreur. Il s'agit du fichier custom.makefile, qui se trouve dans le répertoire :

```
(SDK)/ARDroneLib/Soft/Build
```

La première modification a effectué se trouve ligne 17 en changeant le « no » en « yes » de « USE_LINUX ». La seconde se trouve ligne 50 en changeant le « yes » en « no » de « FFMPEG_RECORDING_SUPPORT ». Ces modifications effectuées, il est maintenant possible de compiler l'exemple qui nous a servi de base de travail. Il faut commencer par compiler la bibliothèque du drone en se rendant dans le dossier :

```
(SDK)/ARDroneLib/Soft/Build
```

Et en tapant la commande `make`.

Il est maintenant possible de compiler le programme en se rendant dans le dossier :

```
(SDK)/Examples/Linux/sdk_demo/Build
```

Et de taper à nouveau la commande `make`.

Le programme est maintenant exécutable, pour cela, il suffit de se connecter au drone par l'interface wifi (il est important qu'un seul PC et aucun iPhone ne soit connecté dessus), l'exécutable `linux_sdk_demo` se trouve dans le dossier :

```
(SDK)/Examples/Linux/Build/Release
```

A moins de disposer d'une des mannettes déjà configurées avec le drone (la liste se trouve dans le fichier gamepad.c), ce programme ne pourra pas faire voler le drone, mais vous aurez la confirmation de son bon fonctionnement avec l'affichage des données de vol comme sur la figure 11.

Après avoir étudié l'ensemble des fichiers compris dans le dossier sdk_demo, nous avons constaté que nous pouvions réaliser un programme de vol autonome en modifiant le fichier ui.c. Il s'agit du fichier gérant la mise à jour des commandes à envoyer au drone lors de l'appui sur les touches de la mannettes. Nous l'avons donc modifié pour implanter directement nos commandes dedans. Le principe est assez simple, la fonction `C_RESULT custom_update_user_input` est appelée à intervalle régulier, nous utilisons donc une variable globale comme compteur, ainsi, nous pourrons envoyer chaque commande pendant une durée suffisante pour agir sur le drone. Le fichier ui.c que nous avons réalisé est le suivant :

```c
#include <config.h>
#include <ardrone_api.h>
#include <UI/ui.h>

int c=0;

// les valeurs de test sur c sont à modifier selon le drone car tous ne sont pas cadencés à la même fréquence

C_RESULT custom_reset_user_input(input_state_t* input_state, uint32_t user_input )
{

  return C_OK;
}

C_RESULT custom_update_user_input(input_state_t* input_state, uint32_t user_input )
{
        if(c<50)
        {
                ardrone_tool_set_ui_pad_start(1);
                if(c==1)
                        printf("Je decolle\n");
        }
        else if(c>=50 &&c <100)
        {
                ardrone_at_set_progress_cmd(0,0,0,0,1);
                if(c==50)
                        printf("Je tourne sur moi même dans un sens\n");
        }
        else if(c>=100 &&c <150)
        {
                ardrone_at_set_progress_cmd(0,0,0,0,-1);
                if(c==100)
                        printf("Je tourne sur moi même dans l'autre sens\n");

        }
        else if(c>=150 &&c <200)
        {
                ardrone_at_set_progress_cmd(0,0,0,1,0);
                if(c==150)
                        printf("Je monte\n");
        }
        else if(c>=200 &&c <250)
        {
                ardrone_at_set_progress_cmd(0,0,0,-1,0);
                if(c==200)
                        printf("Je descends\n");
        }
        else if(c>=250 &&c <300)
        {
                ardrone_at_set_progress_cmd(0,0,1,0,0);
                if(c==250)
                        printf("J'avance\n");
        }
        else if(c>=300 &&c <350)
        {
                ardrone_at_set_progress_cmd(0,1,0,0,0);
                if(c==300)
                        printf("Je recule\n");
        }
        else if(c>=350 &&c <400)
        {
                ardrone_at_set_progress_cmd(0,1,0,0,0);
                if(c==350)
                        printf("Deplacement lateral a droite\n");
        }
        else if(c>=400 &&c <450)
        {
                ardrone_at_set_progress_cmd(0,-1,0,0,0);
                if(c==400)
                        printf("Deplacement lateral a gauche\n");
        }
```

```
        else
        {
                ardrone_tool_set_ui_pad_start(0);
                if(c==450)
                        printf("Je me pose\n");
        }
  return C_OK;
}
```

Annexe 4 : Manipulation à distance via le PC (Windows)

Pour cette partie, le point de départ est le même que pour l'annexe 3 avec le travail sous ubuntu. Une fois la SDK téléchargés, les outils à installer sont les suivants :

- La SDK windows (plusieurs versions selon les OS et processeurs, nécessite Framework 4) http://www.microsoft.com/en-us/download/details.aspx?id=8279
- La SDK de direct X (plusieurs versions selon les OS et processeurs) http://www.microsoft.com/en-us/download/details.aspx?id=6812
- L'environnement de développement Visual C++ : http://msdn.microsoft.com/fr-fr/gg699327

Nous avons réussi à les installer sur Windows 7 32 bits et 64 bits, mais uniquement sur des PC disposant de processeurs AMD.
Une fois les outils installés, il a fallu modifier le fichier ArDrone_properties.vsprop contenu dans le dossier :

(SDK)/Examples/Win32/VCProjects/ARDrone

Dans ce fichier, il a fallu indiquer le chemin du dossier ARDroneLib et du dossier Win32ClientDir.

L'étape suivante de préparation de notre projet est d'ouvrir le fichier vp_os_signal_dep.h (dans l'explorateur visual C++, après avoir ouvert le projet ARDrone.sln). Il faut que la macro #define USE_WINDOWS_CONDITION_VARIABLES ne soit pas commenté et que #define USE_PTHREAD_FOR_WIN32 le soit. Le dernier point consiste à vérifier dans le fichier win32_custom.h l'adresse ip qui est défini correspond bien à celle du drone. (Il y a cependant des problèmes lorsque celle-ci n'est pas 192.168.1.1, donc il vaut mieux éviter d'avoir 2 drones dans la même pièce).

Une fois toutes ces étapes terminées, il ne reste plus qu'à compiler le projet à l'aide visual C++. Pour lancer le programme, il faut lancer le fichier Win32Client qui se trouve dans le dossier :

(SDK)/Examples/Win32/VCProjects/ARDrone/Debug

Une fois tous les outils correctement configurés, nous avons pu implanter le même programme de vol contrôlé à distance que sous ubuntu. Cette fois-ci, un code identique à celui de ui.c a été implanté dans la fonction C_RESULT_update_dx_keyboard(void). Cette deuxième version de programme à distance, cette fois-ci réalisé sous Windows nous a donné les mêmes résultats.