POLYTECH LILLE

Université Lille1 Sciences et Technologies

CONTROLE DE DRONE AVEC TELECOMMANDE A GESTE

Rapport de projet

Quatrième année département

Informatique, Microélectronique et Automatique

Auteurs :

Sébastien DELTOMBE

Mathieu GERIER

Nom de la matière :

Projet de 4ème année

Année :

2013/2014

A l'attention de :

Mr. Jérémie DEQUIDT

Remerciements

Nous tenons tout d'abord à remercier, toute l'équipe pédagogique de Polytech'Lille et les responsables de la formation Informatique, Microélectronique et Automatique de m'avoir enseigné les bases pour réaliser ce projet dans de bonnes conditions.

Nous tenons à remercier et à témoigner toute notre reconnaissance à Monsieur Jérémie DEQUIDT, notre responsable de projet, pour son soutien permanent et sa disponibilité tout au long du projet.

Nous souhaitons également remercier Monsieur Xavier REDON pour nous avoir fourni tout le matériel nécessaire et pour avoir été là pour répondre à toutes nos questions.

Nous souhaitons également remercier l'ensemble des employés de L'INRIA pour leur accueil et leur gentillesse.

Sommaire

Introduction

Les télécommandes à gestes sont en passe de devenir des outils incontournables dans les domaines de l'informatique, de la robotique et du multimédia. Déjà présentes dans nos consoles et nos télévisions, ces dispositifs proposent différents types de reconnaissance de nos mouvements. Il en existe de différentes formes comme la Kinect qui permet de reconnaître des gestes par caméra ou encore la Wiimote qui dispose d'un accéléromètre et d'un gyroscope (en option) afin de détecter les angles d'inclinaison. Le but du projet était d'utiliser une telle technologie afin de contrôler le déplacement d'un drone. Dans un premier temps, nous verrons les objectifs fixés par le cahier des charges, dans un second temps, le déroulement du projet et enfin les impressions sur ce projet.

Présentation du projet

1. Présentation générale

Figure 1 : *Drone*

Le cahier des charges du projet spécifie qu'un drone quadricoptère du commerce soit piloté par le biais d'une interface à gestes. Un drone est un aérodyne sans pilote embarqué et télécommandé. Le drone est en général utilisé au profit des forces armées ou de sécurité (police, douane, etc.) mais il peut avoir d'autres applications comme par exemple la commande d'un tel engin à l'aide d'une télécommande Myo comme une voiture téléguidée.

Figure 2 : *Brassard Myo*

L'entreprise Thalmic Labs a créé un brassard-télécommande. Lorsqu'il est porté sur l'avant-bras d'une personne, il permet de piloter des périphériques à distance comme des ordinateurs, des consoles de jeux ou des drones. Il interprète l'activité électrique des muscles lors de l'utilisation de la main, des doigts et du bras. Cela permet d'associer les mouvements de l'utilisateur à des tâches à envoyer aux appareils compatibles. Le brassard ne peut se connecter qu'à des appareils ayant le Bluetooth (moyen de communication).

Le drone ne communique que par le moyen d'une communication de type Wifi, ce qui pose problème pour la communication entre le drone et la télécommande.

Il est donc nécessaire de passer par un système, une passerelle, qui permet de faire communiquer la télécommande avec le drone.

2. Description du projet

Le projet s'articule autour de trois points :

- La télécommande qui communique les mouvements de l'utilisateur par liaison Bluetooth au système embarqué.
- Le système embarqué qui fera la passerelle entre l'interface à gestes et le drone. Il a néanmoins quelques contraintes. En effet, il doit disposer du Wifi et du Bluetooth afin de pouvoir communiquer avec les autres entités. De plus, le système permettra d'interpréter les mouvements qu'envoie la télécommande afin de les retransmettre au drone pour qu'il puisse effectuer correctement les ordres voulus.
- Le drone qui doit être pilotable par le Wifi.

Le schéma suivant illustre comment s'effectuent les échanges entre les différents systèmes :

Figure 3 : *Illustration des échanges entre les différents systèmes*

3. Matériel nécessaire

Pour réaliser à bien ce projet, Un ArDrone version 1 de la société Parrot SA nous a été fourni. Il fallait également un smartphone sous Android (version 4.0.3) où Android est un système d'exploitation pour smartphones, tablettes tactiles, PDA et les terminaux mobiles. C'est un

système Open source (possibilité de libre redistribution, d'accès au code source et de créer des travaux dérivés) utilisant le noyau Linux (système d'exploitation). Le choix d'un tel système était de pouvoir disposer d'un système embarqué léger qui puisse être porté par l'utilisateur pour faciliter le confort. En effet, si l'on utilisait un ordinateur, il serait lourd encombrant et pénible pour l'utiliser juste pour faire voler le drone.

Ensuite, un brassard Myo et/ou une télécommande Wiimote étaient nécessaires pour le projet.

Figure 4 : *Wiimote*

La Wiimote, appelée « télécommande Wii » est le nom du contrôleur de la console de jeu Wii de Nintendo. Il s'agit d'une manette de jeu de forme rectangulaire. Elle est tenue généralement à une main. Elle possède une multitude de capteurs (accéléromètre, gyroscope, etc.) qui lui permettent de se repérer dans l'espace et de retranscrire les mouvements à l'écran. Elle possède également des actionneurs comme des boutons qui permettent de donner des ordres pour que le drone les exécute.

Le brassard Myo étant en précommande, il était nécessaire d'avoir une interface à gestes afin de ne pas bloquer le projet suite à une indisponibilité de celui-ci. Pour cela nous allons travailler avec une Wiimote. De manière à avoir un programme qui puisse s'adapter au contrôleur (interface à gestes), il est nécessaire de bien penser l'application.

Figure 5 : *Logos du logiciel NetBeans et du Java*

Nous avons choisi pour notre application Android le langage Java. Il s'agit d'un langage de programmation informatique orienté objet. L'avantage de ce type de langage est que les logiciels écrits dans ce dernier sont facilement portables sur des systèmes d'exploitation tels que Linux, Windows ou Mac OS car le langage Java utilise une machine virtuelle disponible sur chacun d'eux. Les applications Android sont généralement codées en Java. C'est pourquoi, nous avons choisi ce langage pour ce projet. La programmation se fera via le logiciel NetBeans qui est un environnement de développement intégré. Il est très intuitif et intelligent. Il permet de programmer vite car il

dispose d'un système proposant de compléter une partie du code de façon automatique et permet ainsi d'éviter des erreurs. Il intègre également des outils de débogage afin de détecter les erreurs plus facilement.

4. Etapes du projet

Afin de mener à bien le projet, différents objectifs ont été posés :

- Prise en main du drone
- Prise en main de la Wiimote (en attendant le brassard)
- Prise en main du brassard Myo
- Réalisation d'un prototype sur PC
- Intégration dans Android

Les tâches ont été attribuées à chacun de manière à optimiser le temps et l'efficacité du déroulement de projet.

a. Prise en main du drone

L'objectif de cette partie est de développer une portion du programme permettant d'envoyer des ordres (avancer, reculer, rester statique, monter, descendre, aller à gauche, etc.) au drone en passant par le Wifi.

b. Prise en main de la Wiimote (en attendant le brassard)

Il est question de développer la portion de code permettant la communication avec une Wiimote à l'aide d'une librairie open source existante. Il s'agit d'une collection de fonctions (portion de code effectuant une tâche ou un calcul relativement indépendant du reste du programme) déjà écrites permettant la communication Bluetooth avec la Wiimote. Cette étape va permettre de

récupérer les informations venant de la télécommande comme par exemple, l'ordre de déplacement lorsque l'on incline la Wiimote, de décollage lorsque que l'on appuie sur tel bouton ou d'atterrissage.

c. Prise en main du brassart Myo

Une fois le brassard en notre possession, des batteries de tests seront réalisées afin d'étudier le brassard. En effet, les mouvements à effectuer doivent être déterminés pour le pilotage. Une fois cette phase d'étude terminée, la portion de code permettant de gérer les commandes à envoyer au drone, si l'on fait telle ou telle action, devra être conçue.

d. Réalisation d'un prototype sur PC

Lorsque les portions de code des étapes précédentes seront réalisées, l'objectif sera d'effectuer les tests avec, comme passerelle entre le drone et la télécommande, un ordinateur. Ce test permettra de mettre en commun les parties permettant de récupérer les informations arrivant de la télécommande et d'envoyer les informations au drone à partir de la passerelle.

e. Intégration sous Android

Cette ultime étape consistera à intégrer les librairies conçues dans une application Android.
Tout ce qui aura été fait dans la section « d. » devra être implanté sur un système embarqué Android permettant plus de confort.

II. Déroulement du projet

1. Préambule

A une semaine de la fin du projet, le brassard n'a malheureusement pas été livré. La solution de secours qui était l'utilisation d'une Wiimote fait donc finalement partie de la version finale livrée avec le projet.

Nous avons décidé tout au long du projet d'utiliser un gestionnaire de code source afin de pouvoir partager notre code et conserver un historique de nos modifications. Nous avons choisis pour cela Github qui est un service web d'hébergement et de gestion de développement de logiciels, utilisant le programme Git. Il est gratuit et accessible directement sur Internet où même à partir du logiciel NetBeans. Un tel dispositif permet de récupérer les données d'un autre programmeur sur Git afin d'avoir la dernière version du programme. Par exemple, un concepteur modifie tel ou tel fichier du projet, il le stocke sur Git et un autre développeur peut récupérer les fichiers modifiés dans le répertoire dans lequel il travaille.

Figure 6 : *Logo de Github*

2. Gestion du drone

a. Recherche de solution existante de pilotage du drone

Dans cette partie nous avons recherché les différentes techniques possibles pour communiquer avec le drone. Des renseignements ont été naturellement faits sur le SDK (Software Development

Kit) fournit par la société Parrot qui commercialise le drone. Le SDK est une librairie, mise à disposition des développeurs, permettant l'intégration du pilotage du drone dans un autre programme.

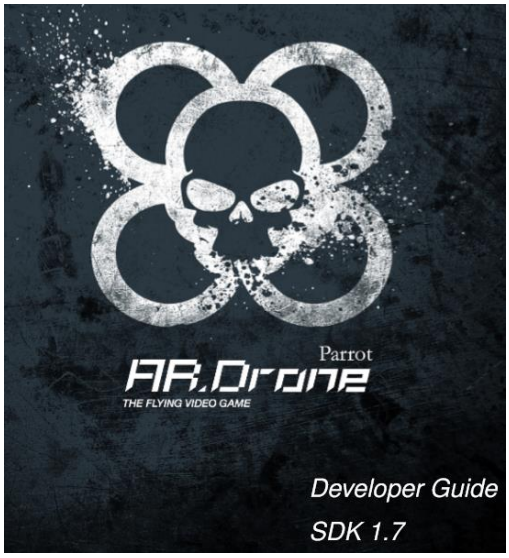


Figure 7 : *Software Développement Kit*

Nous avons donc téléchargé le SDK ainsi que sa documentation (cf. Annexe 2). Lors de la lecture de celle-ci nous nous sommes aperçu que le SDK était une librairie développée en langage C sous linux. Elle est utilisable par n'importe quel programme capable d'appeler des fonctions de celle-ci.

Après quelques recherches, nous avons également constaté que les sources de l'application Android officiel de pilotage du drone, de la société, étaient disponibles et réutilisables.

La récupération de l'application officielle ainsi que son adaptation pour notre projet nous a paru une bonne solution. En effet les sources étaient directement en Java ce qui correspond au langage cible choisit pour notre projet.

Nous avons donc téléchargé l'application pour pouvoir l'étudier. Lors de nos observations nous avons établi plusieurs constats :

- ⇨ L'application est très riche et contient de nombreuses parties inutiles dans le cadre de notre projet.
- ⇨ Le niveau de complexité du code est très élevé, il nous est donc difficile de détecter les parties du code permettant de réellement communiquer avec le drone.

- ⇨ Le projet, une fois compilé, ne fonctionne pas sur les différents smartphones testés.
- ⇨ Le mode de débogage de l'application est inutilisable en raison de la volumétrie de celle-ci. Ce mode permet de pouvoir analyser le code pendant son fonctionnement directement depuis un ordinateur.

A la suite de ces constats, le temps d'adaptation de l'application nous a paru trop conséquent comparer au temps de réécriture de notre propre application en utilisant le SDK. Cette piste a donc été abandonnée.

Une autre contrainte nous a restreint à ne pas pouvoir utiliser le SDK tel quel. En effet sur Android toutes les librairies de base ne sont pas forcément disponibles. Il est donc difficile d'être certain que le SDK lui-même n'utilise pas une librairie incompatible. D'ailleurs lors de l'analyse de l'application Android officiel, nous n'avons pas détecté de trace de l'utilisation du SDK.

En revanche la documentation du SDK nous a tout de même été très utile car elle explique le principe de base de la communication avec le quadricoptère en UDP (User Datagram Protocol). Il s'agit d'un protocole de communication permettant une transmission de données très simple entre deux entités. Les informations du SDK relatent également de façon précise les trames des différentes commandes à utiliser pour les échanges avec le drone.

Nous avons donc décidé pour être sûrs de n'avoir aucun problème de compatibilité lors de notre passage sur Android, de créer notre propre librairie en java permettant la communication avec le drone. L'avantage de cette solution est que nous nous reposons sur du code simple qui sera multiplateforme et dont nous avons la maitrise.

b. Généricité et portabilité du programme

Lorsque le choix de la méthode permettant de commander le drone a été fait, nous avons réfléchi sur la façon de concevoir notre programme. En effet deux points importants étaient à prévoir dès les premiers pas de la conception :

1) Notre programme devait être utilisable sur plusieurs plateformes (Windows, Linux, Android). Il était donc nécessaire de réécrire certaines parties du programme selon le système d'exploitation cible. Un bon exemple pour illustrer cela est la partie qui permet d'effectuer des traces (informations échangées entre les dispositifs affichées à l'écran afin de les contrôler) dans notre programme. Il est bien évident qu'écrire une trace sous Windows ou Linux ne se fait pas de la même façon que sous Android.

2) Afin de permettre au programme de pouvoir facilement s'adapter au brassard Myo lors de sa réception, il est important de disposer d'une solution permettant la gestion de plusieurs types de contrôleurs.

1. Portabilité

Pour répondre à la problématique de portabilité, nous avons décidé de rassembler toute la partie commande du drone et gestion des contrôleurs dans une seule et même librairie. Nous avons appelé cette librairie « DroneControllerLib ». Elle représente le moteur de chacun de nos programmes. Le but de cette librairie est de contenir le code pouvant être utilisé tel quel, sans modifications de notre part, sur chacun des systèmes d'exploitation cibles.

La partie du code spécifique à la plateforme (système d'exploitation) se trouvera quant à elle dans une librairie à part qui sera utilisée en fonction du système d'exploitation.

Voici un exemple d'utilisation de ces librairies :

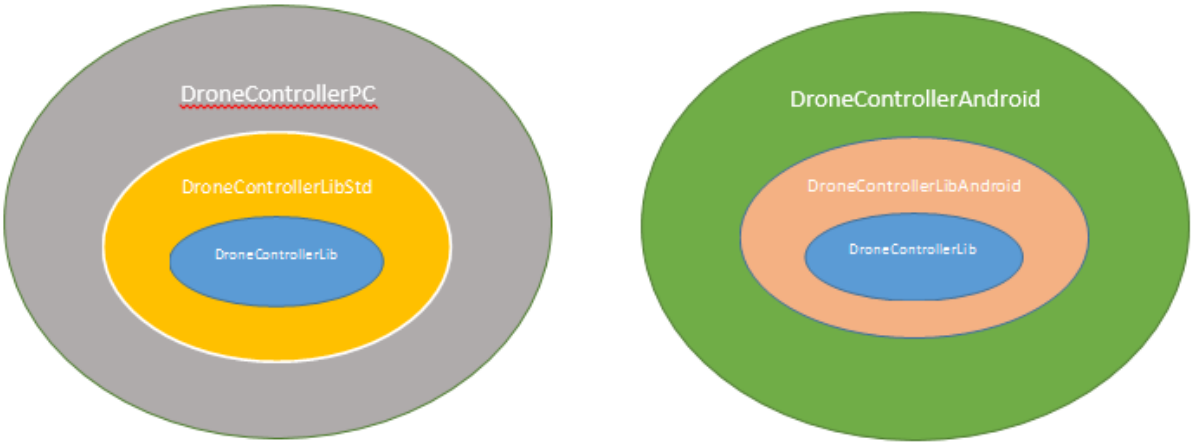


Figure 8 : *Exemples des librairies sur différentes plateformes*

La figure 8 montre un exemple d'utilisation pour deux applications qui fonctionnent sur des plateformes différentes (PC, Android). La partie en bleu représente notre moteur « DroneControllerlib » qui contient le code utilisable des deux côtés. Nous pouvons ensuite voir nettement la surcouche qui permet d'utiliser les fonctions spécifiques aux systèmes (DroneControllerLibStd pour PC et DroneControllerLibAndroid pour Android) et bien sûr les applications en elles-mêmes qui utilisent nos librairies. Par exemple, l'application DroneControllerAndroid utilise la librairie DroneControllerLibAndroid.

2. *Gestion de plusieurs contrôleurs*

La programmation orientée objet permet d'utiliser différentes techniques pour résoudre le problème de la gestion de plusieurs contrôleurs. En créant un système capable de s'adapter de façon dynamique à plusieurs types de contrôleur, le programme est capable d'appeler les fonctions nécessaires afin de gérer correctement le contrôleur connecté. (cf. Annexe 1).

c. Prototype PC

Une fois la partie « moteur » créée, nous voulions tester, dans un premier temps, notre solution directement sur PC. Ces tests ont été faits pour des questions pratiques. En effet, il était ainsi plus facile de débugger notre programme avant son adaptation sur Android.

Pour pouvoir compléter et tester facilement cette partie nous avons créé, dans la librairie spécifique PC, un contrôleur de type clavier. Ce dernier permet d'envoyer les ordres directement au drone sans passer par la Wiimote. Durant cette phase, nous avons eu quelques problèmes de stabilité et de communication avec le drone. En effet toutes les commandes n'étaient pas documentées, des recherches supplémentaires ont été nécessaires. Elles étaient basées sur des exemples de communication réalisés par d'autres développeurs dans d'autres langages afin de trouver l'ensemble des commandes manquantes. Nous n'avons pas rencontré de problèmes majeurs par la suite sur cette partie.

d. Adaptation sur Android

Le code étant destiné à être multi-plateformes, il n'a fallu pour cette partie que réécrire la partie du code spécifique à Android dans la librairie DroneControllerLibAndroid. Certains problèmes ont toutefois été rencontrés lors de la compilation. Cette dernière permet d'analyser le code de façon syntaxique et sémantique afin de créer un fichier exécutable. Le JDK Android (Java Développement Toolkit) ne permettait pas l'utilisation de certaines syntaxes disponibles dans les dernières versions de Java.
Le JDK représente les librairies Java permettant de concevoir un programme sous Android.
Cependant peu de modifications ont été nécessaires pour arriver à faire fonctionner cette partie directement sous Android.

e. Adaptation sur PcDuino

Figure 9 : *PcDuino*

Suite aux difficultés rencontrées pour faire fonctionner la Wiimote sur Android (voir partie gestion Wiimote) nous avons également testé la possibilité de faire fonctionner notre programme sur un système embarqué de type PcDuino fonctionnant sur Linux. Le PcDuino est un miniPC sous la forme d'une carte électronique embarquant des connectivités d'entrées/sorties analogiques et numériques. Le PcDuino fonctionnant sous Linux, aucune adaptation du programme n'a été nécessaire car toutes les fonctions relatives au fonctionnement sur PC étaient implémentées dans la librairie DroneControllerLibStd.

Remarque :

Nous avons utilisé un PcDuino version 2 qui dispose d'un module Wifi intégré facilitant la communication avec le Drone.

3. Gestion de la Wiimote

a. Recherche de solutions existantes pour la communication avec une Wiimote

Comme pour le drone nous avons recherché dans cette partie une façon de communiquer avec la Wiimote en Bluetooth. Le but est de récupérer les informations de commande comme l'appui ou le relâchement d'un bouton ou les données concernant l'accéléromètre pour envoyer les

informations à la passerelle afin qu'elle les traite. Les recherches ont été concentrées sur une librairie disponible sur Windows (équipé de base sur nos ordinateurs) en vue de la réalisation du premier Prototype.

Lors de nos recherches, nous avons trouvé une librairie Java dénommée WiiuseJ correspondant à nos attentes. Cette librairie devait s'utiliser conjointement avec un driver permettant la communication avec la Wiimote.

Ce driver est nécessaire car la télécommande utilise deux niveaux de protocoles bien particuliers pour communiquer :

- ⇨ Le protocole L2CAP (Logical Link Control & Adaptation Protocol) permet de définir des canaux de communication de type Bluetooth. Il s'agit de la couche logicielle la plus proche du matériel.
- ⇨ Le protocole HID qui est un standard permettant de définir la façon dont doivent être constitués les messages échangés.

Le système d'exploitation Windows n'étant pas la plateforme cible nous n'avons pas cherché longtemps pour trouver le driver nécessaire. Nous avons donc utilisé la version d'évaluation du driver BlueSoleil, dont nous avions connaissance de sa compatibilité avec WiiuseJ.

b. Méthodes de tests

Dans cette partie, nous allons expliquer qu'elles étaient les étapes de tests de la Wiimote. Nous pouvons en noter 4 :

- Simulation sur la passerelle de la gestion des boutons de la Wiimote sans l'accéléromètre.
- Test réel sur le drone de la gestion des boutons sans l'accéléromètre.
- Simulation sur la passerelle de la gestion des boutons et de l'accéléromètre.
- Test réel sur le drone de la gestion des boutons et de l'accéléromètre.

Pour chaque test, nous passions tout d'abord par une phase de simulation (cf. figure 10) où l'on regardait les traces envoyées par la Wiimote à la passerelle et celles censées être envoyées au

drone. En effet, nous désactivions la fonction de notification, permettant d'envoyer les informations au drone, afin d'éviter tout problème. Par exemple, si l'on appuyait sur un bouton de la Wiimote afin que le drone avance et que la commande envoyée (visible par les traces d'envoi des données de la passerelle au drone) restait figée dans cette position pour telle ou telle raison, il y avait risque de collision avec un objet ou une personne. Ceci étant trop dangereux, c'est pourquoi nous nous sommes donné une règle à suivre pour éviter tout dommage.
L'accéléromètre étant un peu plus difficile à prendre en main, nous faisions tout d'abord les tests des boutons pour valider que tout fonctionnait correctement puis nous passions aux tests avec les boutons et l'accéléromètre.

Voici les traces s'affichant sur l'ordinateur pour la phase de simulation :

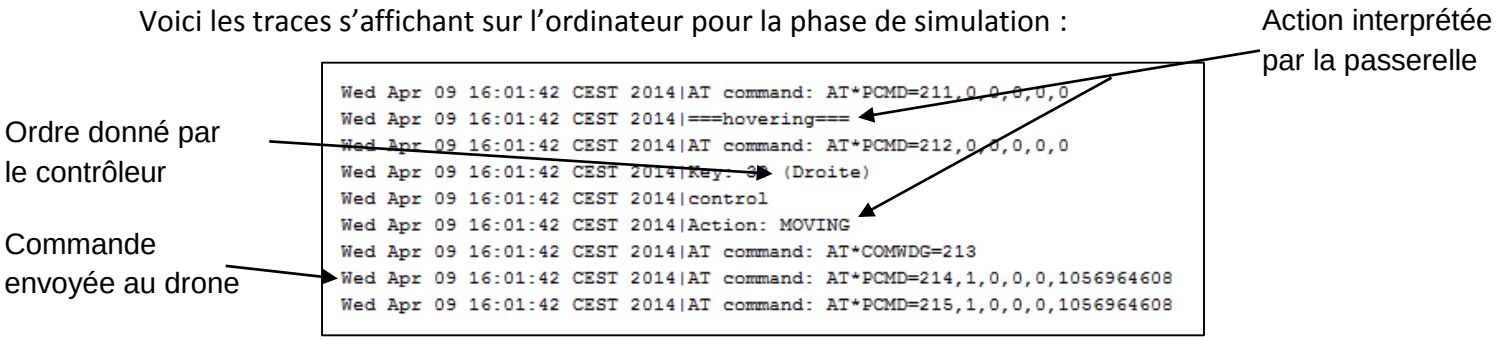


Figure 10 : *Exemple de trace s'effectuant sur la passerelle*

c. Prototype PC (avec BlueSoleil et WiiuseJ)

Le but du prototype PC était de valider le principe de fonctionnement avant de commencer à travailler sur Android. Nous avons pour cela créé du code qui utilise la librairie WiiuseJ tout en utilisant en parallèle le moteur de commande du drone. Nous avons rapidement obtenu un résultat satisfaisant qui nous a confortés dans nos choix techniques de conception. En effet nous arrivions à piloter le drone avec des commandes basiques mais également avec l'accéléromètre. Nous avons cependant remarqué une perte de communication assez fréquente de la wiimote avec BlueSoleil. Ce problème ne nous a cependant pas inquiété étant donné que ce driver n'avait d'importance que le temps du prototype une autre solution devra être trouvé pour Android.

d. Adaptation sur Android (avec BlueZ IME)

Dès lors que nous avions validé le prototype nous nous sommes renseignés sur la façon d'utiliser nos deux protocoles L2CAP et HID sur Android. Dans un premier temps, nous avons découvert que la gestion du protocole L2CAP était directement intégrée par le système d'exploitation Android. Pour le protocole HID nous avons trouvé une application largement utilisée appelée Bluez IME permettant la communication avec plusieurs types de manettes dont la Wiimote par le biais du Bluetooth.

BlueZ IME s'installe sous la forme d'un service qui fonctionne en tâche de fond, c'est-à-dire que qu'il se lance au démarrage du système et n'a pas besoin d'avoir une interface homme-machine active pour fonctionner. Nous pouvons ensuite interroger ce service par le biais d'un mécanisme de communication inhérent aux applications java sous Android.

Nous n'avons pas réussi lors de nombreux tests à faire fonctionner BlueZ IME sur notre smartphone disposant d'Android. A notre grande surprise et après de nombreuses recherches nous nous sommes aperçu que de nombreux utilisateurs rencontrés les mêmes anomalies de connexions que nous. Ce problème est lié à notre version d'Android, nous utilisions la version 4.3 qui a subit depuis la version 4.2, la réécriture d'une partie du code permettant la communication Bluetooth. Après recherche nous avons découvert de nombreux problèmes de compatibilités, notamment avec le protocole L2CAP, qui étaient présents depuis cette mise à jour. Une anomalie traitant de ce sujet a depuis un certain temps été répertoriée dans le système de bug tracking de Google concernant Android.

N'ayant plus la certitude de pouvoir résoudre ce problème dans les temps impartis pour rendre le projet, nous avons alors décidé de travailler en parallèle sur un système embarqué de type PcDuino (cf. section suivante). L'avantage de ce type de système est qu'il est léger et nécessite

une faible consommation, il répond donc également au cahier des charges que nous nous étions fixés au départ.

Nous avons cependant essayé de continuer de travailler sur Android mais à partir d'une tablette disposant d'une version antérieure à la mise à jour concernant le Bluetooth (version 4.0.3).
Après vérifications du bon fonctionnement de BlueZ IME dans cette version nous avons implémenté le code permettant la communication avec celui-ci. Finalement nous sommes arrivés à une solution fonctionnelle en très peu de temps. Seul problème après cela, l'activation de l'accéléromètre, qui a nécessité une modification de notre part dans le code de BlueZ IME lui-même à cause d'un petit bug mineur.

Même si notre solution ne fonctionne pas actuellement sur la dernière version d'Android, nous ne doutons pas que, lors de la correction du problème concernant le L2CAP par Google, notre programme puisse fonctionner un jour sur celui-ci (avec peu ou pas d'adaptation).

e. Adaptation sur PcDuino (avec Wiic)

Le PcDuino utilisant le système d'exploitation Linux, nous avons recherché une librairie compatible avec ce système permettant l'utilisation d'une Wiimote. Nous sommes très vite tombés sur une librairie dénommée Wiic très populaire pour l'utilisation d'une Wiimote sous Unix. Le driver utilisé par cette librairie, pour la gestion des protocoles, se nomment libbluetooth-dev qui est en fait ni plus ni moins que BlueZ IME sous Linux.
Le problème que nous avions rencontré avec WiiC, est que cette librairie est écrite en C. Cela complexifie son intégration dans notre programme. Nous n'avons malheureusement pas trouvé d'équivalence en Java. Ensuite d'autres difficultés ont été rencontrées comme par exemple, l'activation et la désactivation de l'accéléromètre ou alors les valeurs renvoyées par celui-ci pour l'inclinaison. Elles ont très vite été corrigées.

Nous avons alors profité de notre système de Contrôleur dynamique afin de concevoir du code capable de se connecter à un autre programme par le biais d'une connexion de type TCP (Transmission Control Protocol).

Le TCP est un protocole de transport fiable, en mode connecté, c'est-à-dire qu'il gère des messages d'acquits tout au long de la communication entre le client et son serveur.

L'avantage de cette solution est que nous disposons d'une architecture de type Client Serveur. Dont le schéma de principe est le suivant :

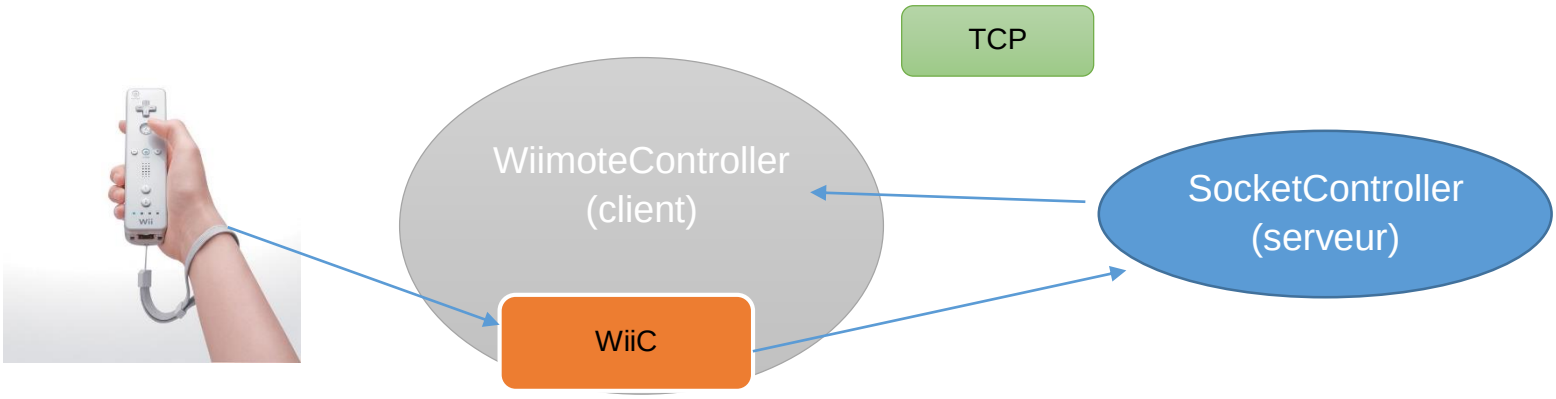


Figure 11 : *Exemple de la communication TCP avec la librairie WiiC*

Remarque : La partie cliente est donc un programme C qui se connecte au serveur Java par le biais du TCP. La partie SocketController n'utilisant que du code compatible PC et Android nous avons pu placer ce composant directement dans la librairie DroneControllerLib qui est le moteur de nos différents programmes.

De ce fait on pourrait très bien envisager facilement de créer à l'avenir un autre client qui utilise un autre contrôleur pour se connecter à notre partie serveur. Il suffira alors d'utiliser le même format de message qu'actuellement.

Ci-dessous, un état des lieux des actions effectuées et restant à faire :

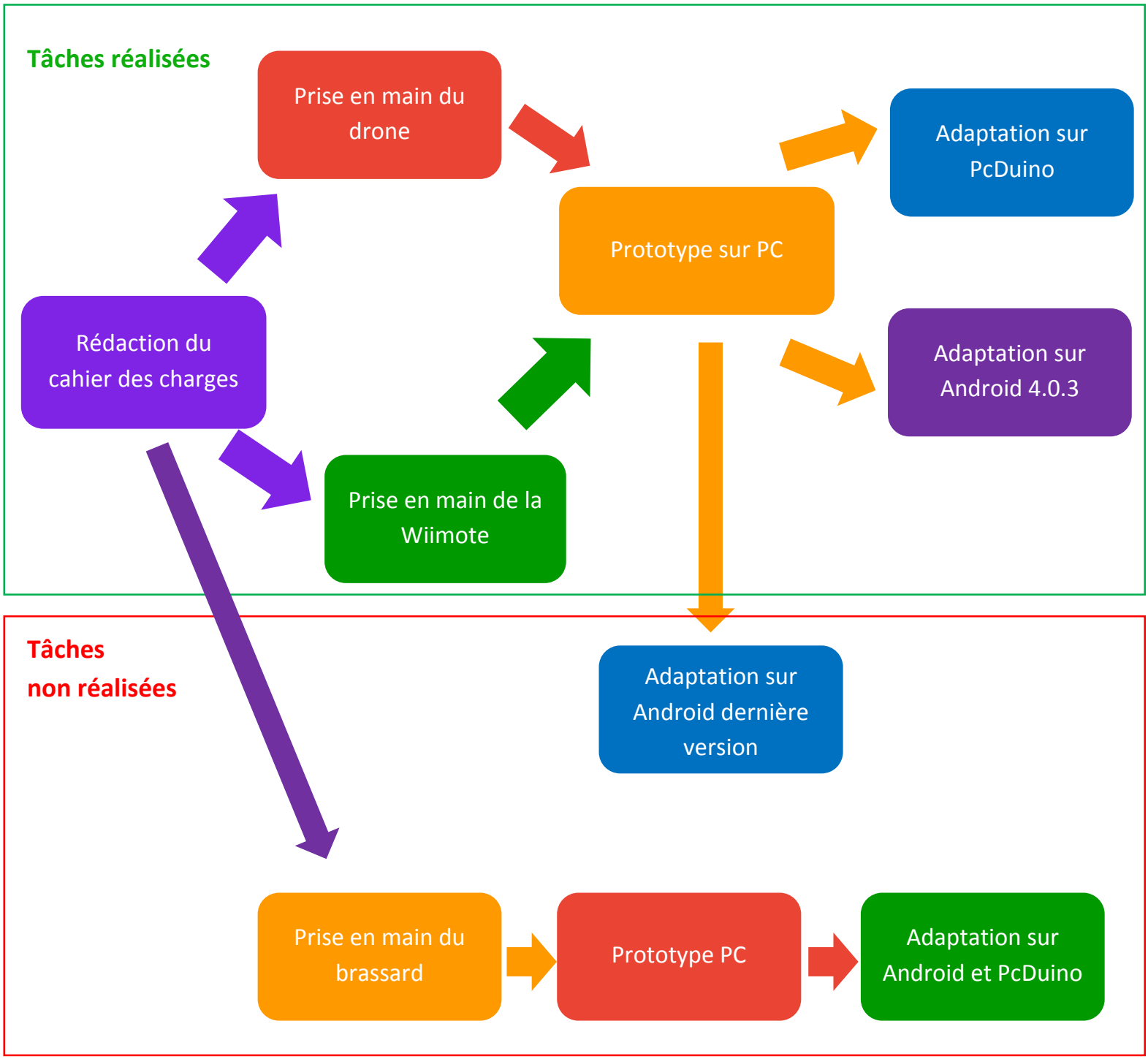


Figure 12 : *Présentation des différentes étapes de ce projet*

III. Ressentis du projet

1. Difficultés rencontrées

L'un des premiers problèmes que nous avons rencontrés a été le lieu pour faire voler le drone. En effet, les salles de projets n'ayant pas beaucoup de places, nous avons dû trouver un endroit où l'espace était assez grand afin d'effectuer des tests de vol du drone. Nous nous sommes renseignés sur les endroits possibles où nous pouvions faire nos tests. Monsieur DEQUIDT nous a alors proposé de faire les essais dans le laboratoire de L'INRIA où il travaille en tant que chercheur.

Nous avons également rencontré d'autres difficultés durant le projet. La gestion de la Wiimote a été l'une des plus importantes. En effet, la gestion du Bluetooth sur Android a changé pour les dernières versions. De plus, les problèmes étant encore d'actualité, il nous a fallu trouver des solutions très rapidement afin de finir, dans les temps impartis, le projet.
Et même si nous les avons finalement trouvées pour chacun de nos problèmes, cela nous a appris à ne pas sous-estimer une partie à l'avance.

Une troisième partie nous a posé des difficultés. Au départ, nous utilisions un drone qui n'avait pas une grande stabilité. Nous avons décidé de tester un autre drone afin de les comparer. Le deuxième était beaucoup plus stable. Nous avons donc, par la suite, fait les tests avec celui-ci. Cependant, un des moteurs a malencontreusement explosé. Nous avons dû faire de la maintenance en mettant un moteur du premier drone à la place de celui qui avait été détérioré. A cause de ces péripéties, nous avons perdu un peu de temps dans la continuité du projet.

2. Bilan personnel

Nous avons trouvé ce projet très enrichissant tout d'abord d'un point de vue technique, il nous a permis de mettre à profit nos compétences acquises au cours de l'année, comme la programmation orientée objet, la communication TCP et UDP.
Ensuite d'un point de vue gestion de projet nous avons dû apprendre à se répartir les tâches afin de rendre notre travail plus productif et efficace. Il est également nécessaire d'anticiper les risques possibles du projet, comme par exemple envisager le fait de ne pas recevoir le brassard Myo à temps comme ce fût le cas où encore la problématique liée au fonctionnement de la Wiimote sur la dernière version Android. Nous avions alors dû envisager de faire un portage de notre solution sur un autre système embarqué.
Il est très difficile de suivre les objectifs fixés sans avoir de problèmes et c'est pourquoi, c'est à nous d'anticiper pour qu'il se déroule sans anicroche.
Nous serons confrontés au même choix dans notre vie professionnelle, nous aurons des délais à respecter, nous devrons alors prendre les dispositions nécessaires pour cela.

Nous aurions voulu ajouter plus de fonctionnalités sur l'interface de notre application créée sur Android afin par exemple de visualiser l'état de la batterie, d'afficher des indicateurs de connexions du drone et de la Wiimote ainsi que de sélectionner le contrôleur permettant de commander le drone. Ces étapes n'ont pas été réalisées car le temps ne nous le permettait pas.

Nous pouvons retenir qu'il faut toujours faire un prototype avec les fonctionnalités minimales afin de respecter le cahier des charges puis de customiser le projet par la suite. En effet, la partie Interface Homme Machine par exemple prend un certain temps, si l'on privilégie ceci avant les fonctionnalités principales, on peut vite se retrouver en difficulté.

Le fait d'avoir utilisé un service web d'hébergement GitHub a été très profitable. Nous pouvions, en effet, directement prendre en compte les changements effectués par l'autre.

Imaginons que nous ne fonctionnons pas avec ce système.
Deux membres A et B travaillent sur un même projet. Le membre A doit utiliser des fonctions que le membre B crée, le membre A doit récupérer les informations par le biais d'une clé USB ou par mail, il doit prendre en compte toutes les modifications faites par le membre B pour son programme. Ceci prend beaucoup de temps pour regarder quelle partie a changé et c'est très fastidieux. Avec le service proposé par GitHub, nous n'avons plus ce genre de problèmes. En effet, les informations sont automatiquement corrigées et si les deux membres ont travaillé sur une même partie, le service permet d'avertir l'utilisateur que deux mêmes parties sont différentes et lui propose de régler le problème.

Le cahier des charges a été respecté correctement dans les temps voulus. Notre seul regret est de ne pas avoir pu travailler sur le brassard Myo qui représentait pour nous un des intérêts majeurs du projet. Mais nous pensons que nous pourrions facilement adapter notre programme à celui-ci si l'occasion se présente.

Conclusion

Nous avons effectué ce projet dans le cadre de notre quatrième année dans la spécialité Informatique Micro-électronique et Automatique à Polytech'Lille.

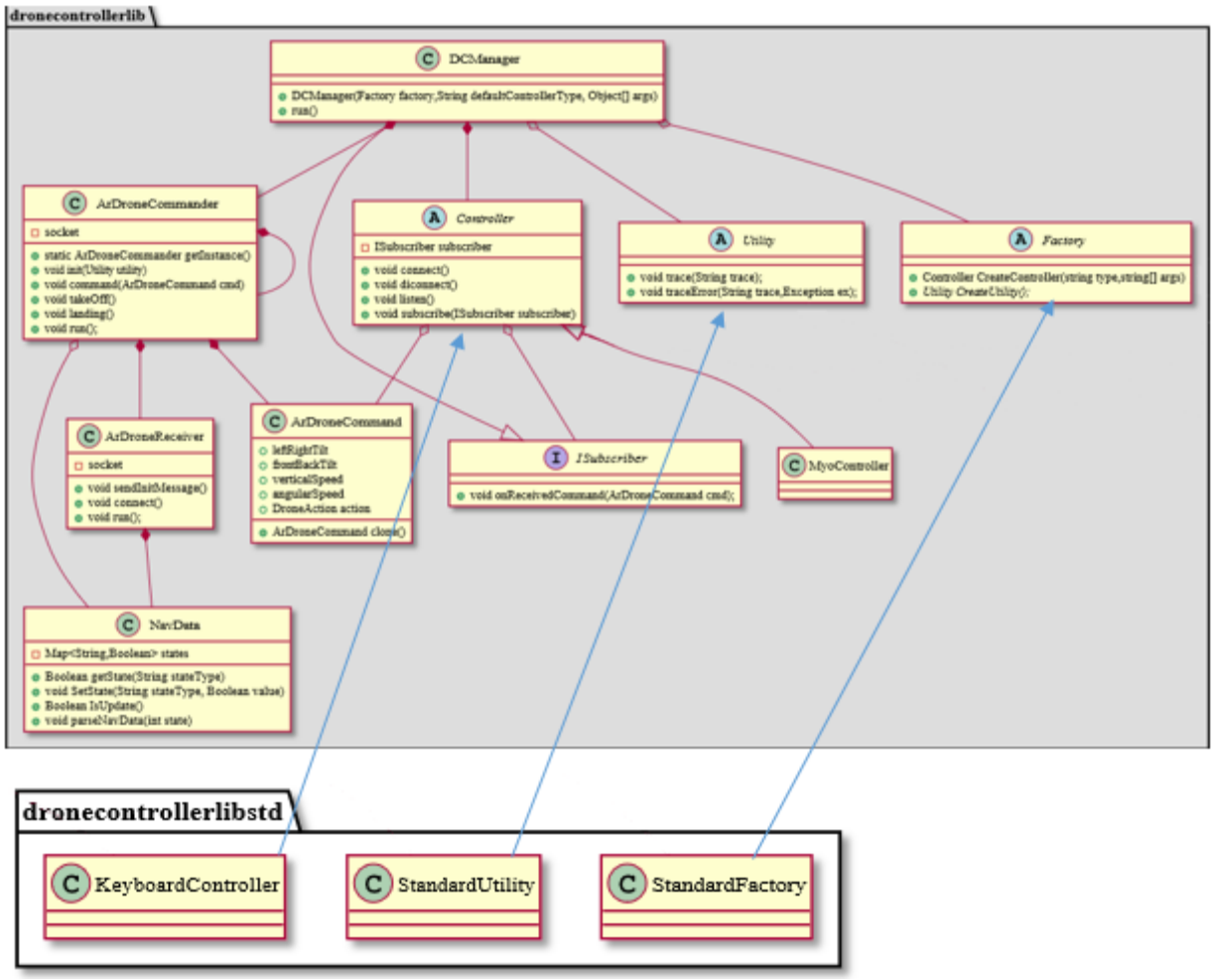
Le projet consistait à créer un système permettant le contrôle d'un drone grâce à une interface à gestes. La rédaction d'un cahier des charges, la conception d'un programme en Java, l'utilisation de librairies Java et C pour gérer les communications avec le drone ainsi que la Wiimote ont été les différentes étapes qui nous ont été nécessaires afin de créer un tel dispositif.

Nous avons pu, lors de ces neufs semaines de projet, mettre en pratique nos connaissances acquises durant la formation et d'en développer de nouvelles. Cette expérience nous a permis d'anticiper les différents problèmes qui se sont dressés devant nous, de travailler en équipe ainsi que de gagner en maturité et de prendre confiance.

Annexes

Sommaire

Annexe 1 : Diagramme de classes (avec exemple de surcouche pour PC)



Légende :

: Librairies

: Classe qui contient des attributs (booléen, float…) et des méthodes (fonctions)

: Agrégat (signifie que la classe où le symbole est collé possède une liste de classes vers laquelle il pointe)

: Dans ce sens, la classe accolée à ce dispositif qui est à gauche est une sous classe de la classe de droite

: Comme l'agrégat mais signifie qu'il n'est pas responsable de la durée de vie de l'objet

Remarque :

Le diagramme n'est pas exhaustif il nous a surtout permis dans un premier temps de nous donner une idée d'ensemble de notre architecture.

L'un des points essentiels à retenir du diagramme est l'utilisation du pattern « Abstract Factory » qui nous permet de construire dynamiquement nos contrôleurs ainsi que nos classes utilitaires en fonction du système cible. Cela nous permet de conserver une librairie en tant que moteur « DroneControllerLib » et de la réutiliser aussi bien sur Pc que sur Android.

Annexe 2 : Description des commandes du drone

AT*REF

Summary : Controls the basic behaviour of the drone (take-off/landing, emergency stop/reset)

Corresponding API function : *ardrone_tool_set_ui_pad_start*
Corresponding API function : *ardrone_tool_set_ui_pad_select*

Syntax : **AT***REF=%d,%d<LF>

Argument 1 : the sequence number
Argument 2 : an integer value in $[0..2^{32}-1]$, representing a 32 bit-wide bit-field controlling the drone.

Description :
Send this command to control the basic behaviour of the drone. With SDK version 1.5, only bits 8 and 9 are used in the control bit-field. Bits 18, 20, 22, 24 and 28 should be set to 1. Other bits should be set to 0.

Bits	31 .. 10	9	8	7 .. 0
Usage	Do not use	Takeoff/Land (aka. "start bit")	Emergency (aka. "select bit")	Do not use

La figure ci-dessus montre la commande à envoyer au drone afin de le faire décoller ou atterrir. Par exemple, pour le faire décoller, il faut envoyer la commande :
AT*REF=256, 290718208

Dans cet exemple, on renseigne AT*REF pour que le drone sache qu'il doit décoller ou atterrir. On envoie pour cela le compteur de commande qui permet la gestion du séquencement. Il s'agit de la valeur 256 dans ce cas. Pour éviter tout problème avec un autre utilisateur, si ce dernier possède le compteur le plus élevé, il aura la main sur le drone.
Ensuite la valeur 290718208 à le bit 9 à 1 et permet ainsi le décollage (0 pour l'atterrissage).

Il existe d'autres commandes pour effectuer les déplacements (avancer, tourner à gauche, monter en altitude, etc.) et obtenir des informations sur l'état du drone comme par exemple le niveau de la batterie.

Annexe 3 : Exemple d'une classe Java

```
package dronecontrollerlibstd;

import dronecontrollerlib.pkg.Utility;
import dronecontrollerlib.pkg.Factory;
import dronecontrollerlib.pkg.Controller;
import wiimote.*;
/**
 *
 * Classe StandardFactory qui dérive de Factory
 */
public class StandardFactory extends Factory {

   public static final String KEYBOARD_TYPE = "KEYBOARD";
   public static final String WIIMOTE_TYPE = "WIIMOTE";

   public StandardFactory(Utility utility) {
      super(utility);
   }

   @Override
   public Controller createController(String type,Object[] args)
   {
      if(type == KEYBOARD_TYPE)
      {
         return new KeyboardController(utility,args);
      }else if(type == WIIMOTE_TYPE)
      {
         return new WiimoteController(utility,args);
      }

      return super.createController(type,args);
   }

   @Override
```

```
    public Utility createUtility() {
        return new StandardUtility();
    }
}
```

Cette classe en langage Java montre bien la généricité du code. Elle se situe dans la librairie DroneControllerLibStd. Elle permet de fournir les contrôleurs compatibles Linux et Windows. En effet, nous utilisons le pattern (bon principe de modélisation objet) « Abstract Factory » qui nous permet d'instancier dynamiquement les contrôleurs voulus en fonction de leur type.
Pour que le contrôleur soit utilisable dans le système, il faut qu'il dérive de la classe Controller.