Université de Bordeaux

Projet d'étude et de développement

Numérisation de façades par un essaim de drones

Etudiants
Guillaume Dupont
Jérémy Poirier
Alexandre Troncy

Responsables
Vincent Autefage
Serge Chaumette
Pascal Desbarats

Mars 2015

Table des matières

1 Introduction au domaine **3**

1.1 Introduction au projet . . . 3

1.2 Etude de l'existant . . . 3

1.2.1 Projets existants utilisant les drones pour la modélisation . . . 4

1.2.2 Projets existants utilisant la photogrammétrie . . . 4

2 Besoins **6**

2.1 Besoins fonctionnels . . . 6

2.2 Besoins non-fonctionnels . . . 7

3 Spécificités du Parrot Bebop **8**

3.1 Particularités . . . 8

3.2 BebopXT . . . 8

3.3 Initialisation de la communication . . . 9

3.4 Protocole ArDrone 3 . . . 9

3.4.1 En-tête . . . 9

3.4.2 Message d'envoi de données avec ou sans acquittement . . . 10

3.4.3 Message d'acquittement . . . 10

3.4.4 Message à faible latence . . . 11

3.4.5 Concaténation de message . . . 12

3.4.6 Plugins Wireshark . . . 12

4 Scénario d'utilisation **13**

4.1 Initilisation . . . 13

4.2 Numérisation de la façade . . . 13

4.3 Récupération et traitement de la vidéo . . . 13

5 Architecture **15**

5.1 Modèle - Vue - Contrôleur . . . 15

6 Résultats **17**

6.1 Commandes 17

6.2 Parser 18

6.3 IHM 20

6.4 Système de numérisation de façade automatisé 20

6.5 Récupération de vidéo et conversion 20

6.5.1 Récupération de vidéo 21

6.5.2 Conversion de vidéo 21

7 Difficultés rencontrées **22**

7.1 Mise à jour du firmware 22

7.2 Défaillance du drone 22

8 Conclusions et perspectives **23**

Introduction au domaine

Ce chapitre a pour objectif de cerner avec le moins d'ambiguïté possible le projet dans sa globalité ainsi que les différents comportements et services attendus par celui-ci. Nous définirons dans un premier temps le contexte du projet, puis nous ferons un état de l'art des projets existants ayant des problématiques proches.

1.1 Introduction au projet

La modélisation 3D est un domaine novateur aujourd'hui et de nombreuses solutions sont à l'étude quant à l'automatisation de ce procédé. Le projet consiste ici à numériser en 3D la façade d'un bâtiment en utilisant les caméras 2D de plusieurs drones de manière simultanée, et ce en utilisant le principe de la photogrammétrie. L'objectif est de reconstituer le nuage de points correspondant à la surface de la façade. Ce projet sera mené en collaboration par des étudiants ayant des spécialités différentes, à savoir les domaines de l'image et celui des réseaux.

Drone Bebop

Le type de drone qui sera utilisé durant ce projet est le Parrot Bebop Drone. C'est un drone de loisir sorti en novembre 2014 dont les performances sont nettement supérieures à ses prédécesseurs (notamment en terme de stabilité de l'appareil, de la caméra ainsi que de la portée). On peut noter entre autres quelques caractéristiques :

- Processeur double coeur Parrot P7
- Processeur graphique quadri-coeur
- Mémoire vive de 8 Go
- Appareil photo "Fisheye" de 14 Mpx

1.2 Etude de l'existant

Nous présenterons dans cette partie les différents projets ayant des problématiques globales relativement proches du projet à effectuer. L'objectif est d'avoir une vue d'ensemble quant à ce qui existe déjà dans le domaine de la numérisation 3D effectuée à l'aide de drones.

1.2.1 Projets existants utilisant les drones pour la modélisation

Dans cette partie seront exposés les différents projets ou solutions d'entreprise relatant des problématiques proches du projet à réaliser.

senseFly

senseFly [1] est une entreprise suisse filiale de Parrot proposant une solution de cartographie 2D et 3D à l'aide de drones autonomes. Ces cartographies permettent de récolter des informations utilisées notamment dans les domaines agricoles, miniers, humanitaires et de surveillance.

L'utilisateur définit dans un premier temps le plan de vol du drone à l'aide d'une application eMotion prenant en compte la topographie de la zone de vol et la force des vents. Ensuite, l'utilisateur lance simplement le drone qui suit son plan de vol et récupère des photos lors de son parcours. Enfin, une fois les photos récupérées, celles-ci peuvent être traitées via un logiciel de rendu 3D, PostFlight-Terra3D, ou tout autre logiciel de processus métier. Quelques caractéristiques de la solution :

- Les drones utilisés sont de type avionique
- Les drones ont une autonomie de 50 min
- Les photos récoltées sont de type aériennes
- Pas de gestion de flotte de drones

Drone Aero Services

En réponse à une demande d'une institution régionale, l'entreprise a eu pour mission de reconstituer un terrain nécessitant une mise à jour cartographique [2]. Le drone utilisé a balayé le site à l'aide d'un GPS centimétrique sur plusieurs hauteurs et plusieurs angles. Une fois les clichés pris, des points servant de référence ont été définis et ont constitué un nuage de points, et c'est grâce à celui-ci que la reconsitution 3D fut réalisée.

Studio Fly

Dans la même optique et en optant pour la même démarche de reconnaissance avec les drones, l'entreprise Studio Fly [3] propose des solutions pour la modélisation dans divers domaines tels que le BTP, la géologie ou l'archéologie.

iDroneX

L'entreprise iDroneX [4] propose également ce type de services dans des applications majeures comme les chantiers, monuments ou projets immobiliers.

1.2.2 Projets existants utilisant la photogrammétrie

PhotoModeler

PhotoModeler permet de créer des modèles 3D à partir de photos. Il est utilisé entre autres pour l'archéologie, l'architecture ou encore la biologie.

1. https://www.sensefly.com/home.html
2. http://droneaeroservices.com/octobre-2014-photogrammetrie-et-reconstitution-3d-
3. http://www.dronethermographie.fr/modelisation3d-drone-studiofly-technologie/
4. http://www.idronex.fr/modelisation-3d/

Dans un premier temps, l'utilisateur doit charger les photos d'un objet, prises depuis différents points de vue. Ensuite, il y a plusieurs constructions du nuage possibles :

— L'utilisateur entre manuellement les points communs entre les différentes photos
— L'utilisateur a pris des photos avec peu de différences entre chaque angle de vue, ce qui permet au logiciel de reconnaitre la proximité de chaque photo
— L'utilisateur a placé des marqueurs sur l'objet au moment de prendre la photo, ce qui permet ensuite une reconnaissance automatique de la position de l'objet par le logiciel

Une fois le nuage de points calculé (quelque soit la méthode utilisée), un modèle 3D peut être obtenu.

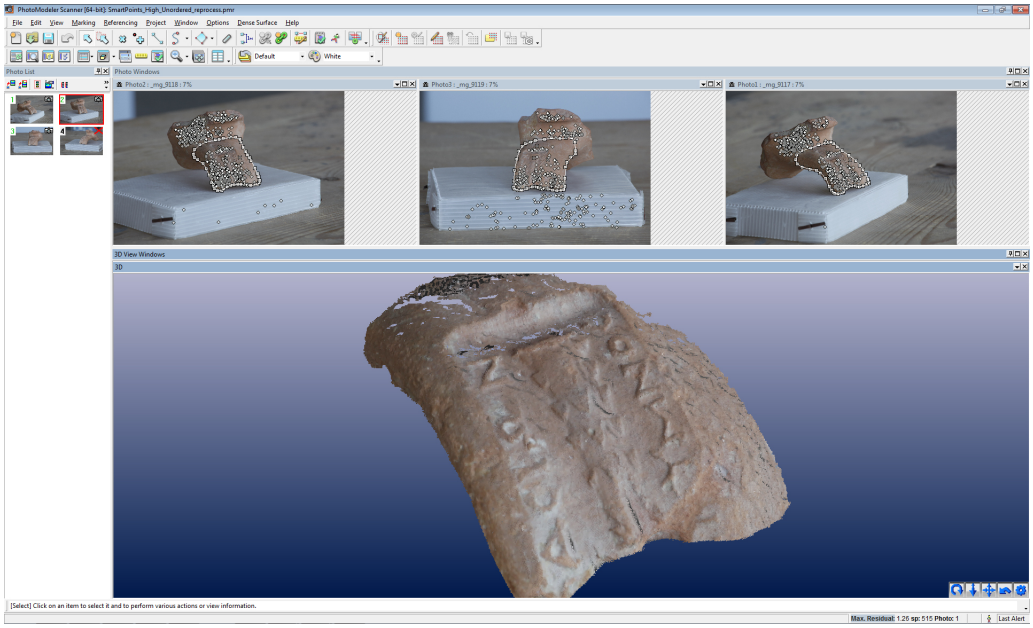

FIGURE 1.1 – Capture d'écran du logiciel PhotoModeler

Free-D

Free-D est un logiciel de reconstruction 3D. Il offre les fonctionnalités suivantes :

— Segmentation des images
— Recalage des images
— Reconstruction des surfaces
— Affichage graphique des modèles

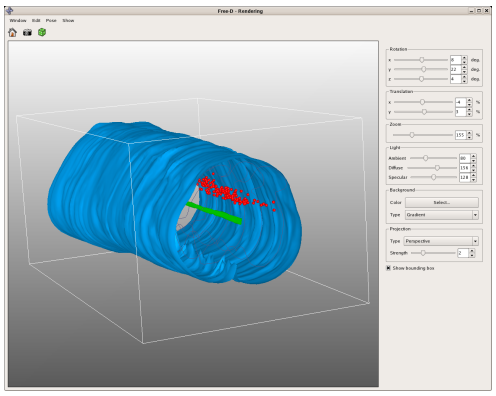

FIGURE 1.2 – Capture d'écran du logiciel Free-D

Besoins

Cette seconde partie est destinée à définir précisément les besoins de notre système. On distinguera parmi ces besoins les besoins fontionnels et non-fonctionnels de notre projet, définis dans les sections ci-dessous.

2.1 Besoins fonctionnels

Les besoins fonctionnels correspondent aux fonctionnalités attendues de l'application. Ils précisent quels comportements et quels services le logiciel devra fournir.

Supplanter l'OS du drone
Dans un premier temps, il faudra installer un système d'exploitation sur une clé USB selon une certaine démarche et ensuite modifier la séquence de boot du drone de manière à supplanter le système du drone par un Linux sur-mesure.
Test : Après avoir effectué la manipulation, vérifier que le drone boot bien sur la clé et que le système fonctionne (comportement cohérent du drone).

Retro-engineering
Nous écouterons le trafic entre le drone et l'application pour smartphone FreeFlight3 (utilisée pour contrôler le drone). Nous récupérerons les paquets et les analyserons dans le but de retrouver les spécifications du protocole permettant de communiquer avec le Bebop.
Test : Effectuer une attaque MITM en utilisant *arpspoof* et enregistrer les trames entre les deux parties, puis les analyser. Une fois le protocole identifié, tester l'envoi des commandes "forgées" au drone et vérifier son bon comportement.

Pilotage des drones
Les drones suivront un plan de vol en hauteur afin de recouvrir l'intégralité de la façade : ils devront se déplacer de bas en haut afin de recouvrir chacun une portion de la façade du bâtiment. Une interface graphique pourra être réalisée afin de faciliter le pilotage des drones et récupérer des informations en temps réel.
Test : Lancer le plan de vol sur plusieurs drones et constater leur bonne évolution dans l'espace en hauteur.

Définir des points communs dans les vidéos
Pour mener à bien la reconstitution par photogrammétrie, il est nécessaire que les vidéos prises par les drones durant leur vol aient des points en commun. Pour ce faire, une certaine distance entre eux sera définie.
Test : Lancer les deux vidéos et observer si elles contiennent des points similaires.

Récupération de vidéo

Il sera nécessaire de récupérer les vidéos prises par les drones et de la transmettre aux étudiants d'ISV afin de pouvoir réaliser par la suite la photogrammétrie.

Test : Récupérer sur le serveur *ftp* du Bebop le fichier vidéo voulu, le transmettre et vérifier que l'envoi et la réception se sont déroulés correctement.

2.2 Besoins non-fonctionnels

Les besoins non-fonctionnels définissent des qualités générales qui ne sont pas liés directement aux services spécifiques délivrés par le système, mais qui définissent des aspects qui engendreront une meilleure utilisation de l'application. Ces éléments s'appliquent généralement au système dans son ensemble.

Synchronisation entre les drones lors du recouvrement vidéo

On pourra penser à définir des points d'attente lors du vol (ligne, laser, ...), pour qu'un drone en avance puisse attendre son semblable. Ainsi, une meilleure synchronisation permettra de faciliter le travail à effectuer pour la modélisation 3D car les deux vidéos auront les mêmes séquences d'images.

Test : Lancer le plan de vol sur plusieurs drones et constater leur évolution dans l'espace en hauteur synchronisée, selon la solution choisie.

Convertir les vidéos

Les étudiants d'ISV utilisent la bibliothèque OpenCV et de ce fait nécessitent un certain format vidéo pour réaliser la photogrammétrie, lisible par celui-ci. Le format le plus optimum est AVI car il est mieux géré sur les différentes plateformes existantes. On pensera donc à convertir le format MP4 par défaut des vidéos du Bebop en AVI.

Test : Récupérer en entrée un fichier présent sur le Bebop au format MP4 en ayant pris soin de vérifier que la vidéo se lit correctement, et vérifier en sortie que le fichier obtenu est au format AVI et lisible également.

Récupération d'informations

Nous pourrons penser à récupérer des informations lors du vol des drones au travers d'une interface graphique dans le but de suivre leurs évolutions respectives pendant le vol.

Test : Vérifier le bon affichage en temps réel des informations sur l'interface graphique.

Spécificités du Parrot Bebop

Le but de ce chapitre est de rendre compte des différences et nouveautés apportées par le Parrot Bebop. Nous y détaillerons la démarche suivie pour la supplantation d'un Linux sur-mesure sur le drone, ainsi que les spécifications du nouveau protocole utilisé pour le piloter.

3.1 Particularités

Dans cette partie seront exposés des faits divers sur le Bebop constituant des informations à prendre en compte lors d'un travail à effectuer avec un tel drone. Parmi les informations pertinentes à retenir, on peut relever que :

- L'adresse IP par défaut du drone est *192.168.42.1*.
- Le drone dispose d'un serveur *ftp* sur lequel on peut se connecter sans avoir besoin de mentionner des informations de connexion. Ce serveur donne accès aux fichiers à partir de */data/ftp*.
- Le drone et le contrôleur interagissent par défaut via deux ports :

```
c2dport: 54321
d2cport: 43210
```

- Il est possible de se connecter en réseau avec le drone en le connectant depuis son port USB. Dans ce cas, son adresse par défaut devient *192.168.43.1* et pour ce faire il faut exécuter le script :

```
/ # /bin/usbnetwork.sh
```

- Le démon *dragon-prog* est lancé au démarrage et gère le système.
- Pour lancer le mode *Debug*, il faut exécuter le script :

```
/usr/bin/DragonDebug.sh
```

- Un SDK [1] a été mis en place pour le Parrot Bebop.

3.2 BebopXT

Le système d'exploitation du drone étant propriétaire et une mauvaise manipulation étant rédhibitoire, la mise en place d'un autre système pour supplanter ce dernier semble pertinent. Ce

1. urlhttps ://github.com/ARDroneSDK3

dernier sera pré-installé sur une clé USB. La démarche générale afin de mettre en place un tel système est principalement la même que pour l'ArDroneXT [2]. Néanmoins, quelques différences sont à noter :

- Le système du drone ne reconnaît pas le format *ext2*. De ce fait, lors de l'installation des 3 partitions sur la clé USB, il faut désormais formater le système de fichiers des partitions ARDSYS et ARDDEV en *ext3*.
- Contrairement à l'ArDrone2, le debootstrap n'a pas fonctionné en utilisant le système d'exploitation *Squeeze*, ce à quoi nous avons solutionné le problème en utilisant la version plus récente *Wheezy*.
- Lors de la manipulation délicate de création du script d'initialisation qui monte la partition ARDSYS (*xtStart*), et de l'appel à celui-ci dans le processus qui initialise le drone (*rcS*), la ligne de code suivante ne devra plus être intégrée dans le script directement (avec la commande *echo*) mais ajoutée à la fin du fichier :

```
# Ligne a ne plus inclure dans le script mais a ajouter
# manuellement dans le fichier rcS
echo '/etc/init.d/xtStart &' >> /etc/init.d/rcS
```

3.3 Initialisation de la communication

Pour initialiser la communication avec le Bebop, il est nécessaire d'envoyer à ce dernier un message au format JSON sur le port TCP 4444. La requête JSON prend la forme suivante :

```
{"controller_name":"toto","controller_type":"toto","d2c_port":43210}
```

Celle-ci détermine un nom, un type de contrôleur et le port que le drone utilisera pour communiquer avec le contrôleur. Pour maintenir la communication, il est nécessaire que le contrôleur envoie un message régulièrement au drone ; dans le cas contraire, ce dernier coupe la communication. Dans la section suivante, nous analyserons le protocole nécessaire pour communiquer avec le drone.

3.4 Protocole ArDrone 3

Parrot a mis en place un protocole dédié à la communication entre ses derniers drones (Bebop, Jumping Sumo et Rolling Spider), et son contrôleur. Il est intéressant de noter que le protocole utilise l'endianisme *little endian*, alors que le *big endian* est majoritairement utilisé pour les protocoles de transmission. Dans cette partie, nous expliquerons le principe de fonctionnement du protocole.

3.4.1 En-tête

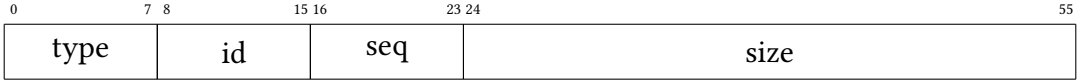

Figure 3.1 – En-tête Protocole Ar Drone 3

Tous les messages de communication entre le drone et le contrôleur utilisent la même en-tête. Celle-ci est codée sur 7 octets, contenant les champs *type*, *id* , *seq* et *size* que nous décrirons ci-après.

2. http://hal.archives-ouvertes.fr/hal-00916815

— *Type* : Ce champ, codé sur un octet, détermine le type du message. Celui-ci peut valoir quatre valeurs différentes, détaillées ci-dessous :
  1. Message d'acquittement
  2. Message d'envoi de données
  3. Message à faible latence
  4. Message d'envoi de données qui requiert un acquittement
— *Id* : ce champ, codé sur un octet, détermine le buffer employé pour le message, celui-ci est compris entre 10 et 127 selon le SDK. Néanmoins ce champ subit des modifications en fonction du type de message et de l'émetteur (contrôleur ou drone), sa valeur réelle est comprise entre 10 et 254.
— *Seq* : ce champ, codé sur un octet, détermine le numéro de séquence d'un message. Il existe plusieurs numéros de séquence dans le protocole, un par type.
— *Size* : ce champ, codé sur 4 octets, détermine la taille complète du message et non la taille du datagramme.

Dans la suite nous présentons la structure des paquets en fonction de leur type.

3.4.2 Message d'envoi de données avec ou sans acquittement

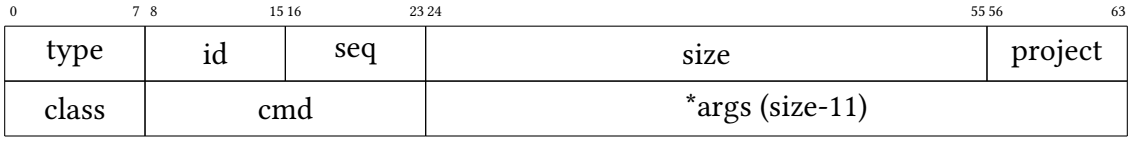

FIGURE 3.2 – Structure d'envoi de données avec un en-tête

Les messages d'envoi de données avec ou sans acquittement (type 2 et 4) disposent de la même structure de message. Celle-ci est codée au minimum sur 4 octets, contenant les champs *project*, *class*, *cmd* et occasionnellement *args*, que nous décrirons à la suite.

— *Project* : ce champ, codé sur un octet, détermine le projet auquel ce message est destiné. Il existe dans le SDK [3] huit projets référencés, deux projets pour chaque drone et deux projets communs à l'ensemble des drones. On note cependant à travers nos captures qu'il existe d'autres projets, non référencés. Chaque projet a un identifiant unique représenté dans ce champ.
— *Class* : ce champ, codé sur un octet, détermine la classe d'un projet utilisé auquel ce message est destiné. Chaque classe a un identifiant unique représenté dans ce champ.
— *Cmd* : ce champ, codé sur 2 octets, détermine la commande d'une classe d'un projet utilisé auquel ce message est destiné. Une commande peut comprendre des arguments ou pas. Chaque commande a un identifiant unique déterminé par sa déclaration dans sa classe.
— *Args* : ce champ, codé sur la taille complète du message moins 11, détermine les arguments d'une commande. Un argument peut avoir de multiples types de données : entier non signé et signé sur 8 à 64 octets, des enumérations codés sur 4 octets, des flottants à simple précision et double précision codés selon la norme IEEE-754 ou des chaines de caractères.

3.4.3 Message d'acquittement

Les messages d'acquittement font suite à des messages d'envoi avec acquittement (type DATA_WITH-ACK). L'unique champ de ce message, codé sur un octet, correspond au numéro de séquence du

3. `https://github.com/ARDroneSDK3/libARCommands/tree/master/Xml`

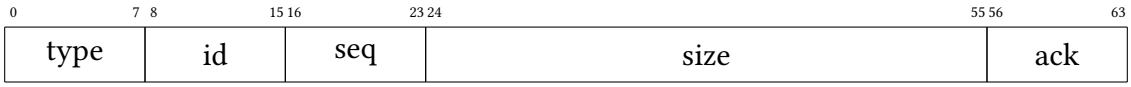

FIGURE 3.3 – Structure d'un acquittement avec un en-tête

message à acquitter.

3.4.4 Message à faible latence

Les messages à faible latence sont principalemement utilisés pour la transmission du flux vidéo. Ces messages disposent de deux structures : une structure d'envoi du flux vidéo du drone au contrôleur et une structure d'acquittement du flux vidéo du contrôleur au drone.

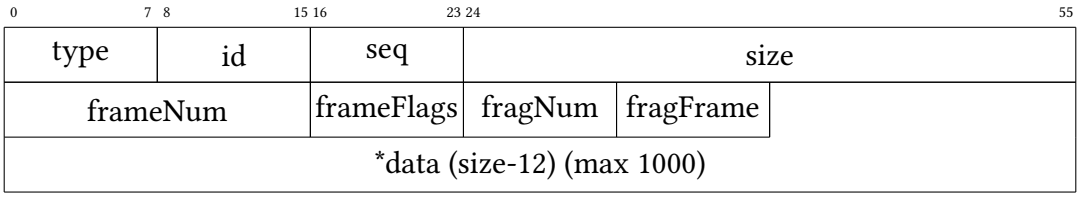

FIGURE 3.4 – Structure d'envoi du flux vidéo

La structure des messages d'envoi de flux vidéo est codée au minimum sur 6 octets. Celle-ci contient les champs *frameNumber*, *frameFlags*, *fragmentsNumber*, *fragmentsPerFrame* et *data*.

- *frameNumber* : ce champ, codé sur 2 octets, détermine le numéro de la trame vidéo.
- *frameFlags* : ce champ, codé sur un octet, détermine le flag de la trame. Nous n'avons cependant pas trouvé à quoi correspond chaque flag.
- *fragmentNumber* : ce champ, codé sur un octet, détermine le numéro de fragment d'une trame vidéo.
- *fragmentsPerFrame* : ce champ, codé sur un octet, détermine le nombre de fragments pour une trame vidéo. Une trame peut être fragmentée au maximum 128 fois.
- *data* : ce champ, codé sur la taille complète du message moins 12, détermine les données du flux vidéo. Le champ de données a pour taille maximale 1000 octets.

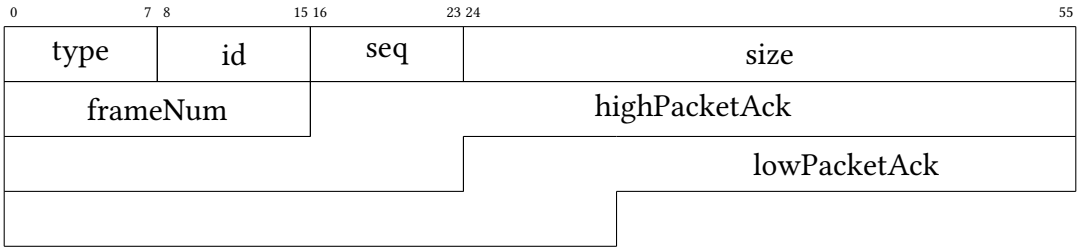

FIGURE 3.5 – Structure d'acquittement du flux vidéo

La structure des messages d'acquittement de flux vidéo est codée sur 131 octets. Celle-ci contient les champs *frameNumber*, *highPacketAck* et *lowPacketAck*.Les champs *highPacketAck* et *lowPacketAck* déterminent l'état d'avancement de l'acquittement d'une trame. Chaque bits de ces deux champs représentent un fragment d'une trame acquitté avec la valeur 1 ou non acquitté avec la valeur 0. Sachant qu'une trame ne contient pas toujours 128 fragments, une fonction utilisant le

poids de Hamming est mise en place pour acquitter les fragments par défaut non compris dans la trame.

— *frameNumber* : ce champ, codé sur 2 octets, détermine le numéro de la trame vidéo.
— *highPacketAck* : ce champ, codé sur 64 otctets, détermine l'acquittement des fragments 64 à 127.
— *lowPacketAck* : ce champ, codé sur 64 otctets, détermine l'acquittement des fragments 0 à 63.

3.4.5 Concaténation de message

Nous avons vu précédemment que le champ *size* de l'en-tête déterminait la taille d'un message et non celle du datagramme. Dès lors, il est possible pour le protocole ArDrone3 de concaténer plusieurs messages dans un même paquet.

3.4.6 Plugins Wireshark

Au cours du projet, nous avons effectué de nombreuses captures de paquets pour comprendre le fonctionnement du protcole ArDrone3. Pour mieux analyser ces captures, nous avons également codé un plugin minimaliste pour le logiciel Wireshark. Celui-ci met en avant la plupart des champs mentionnés auparavant.

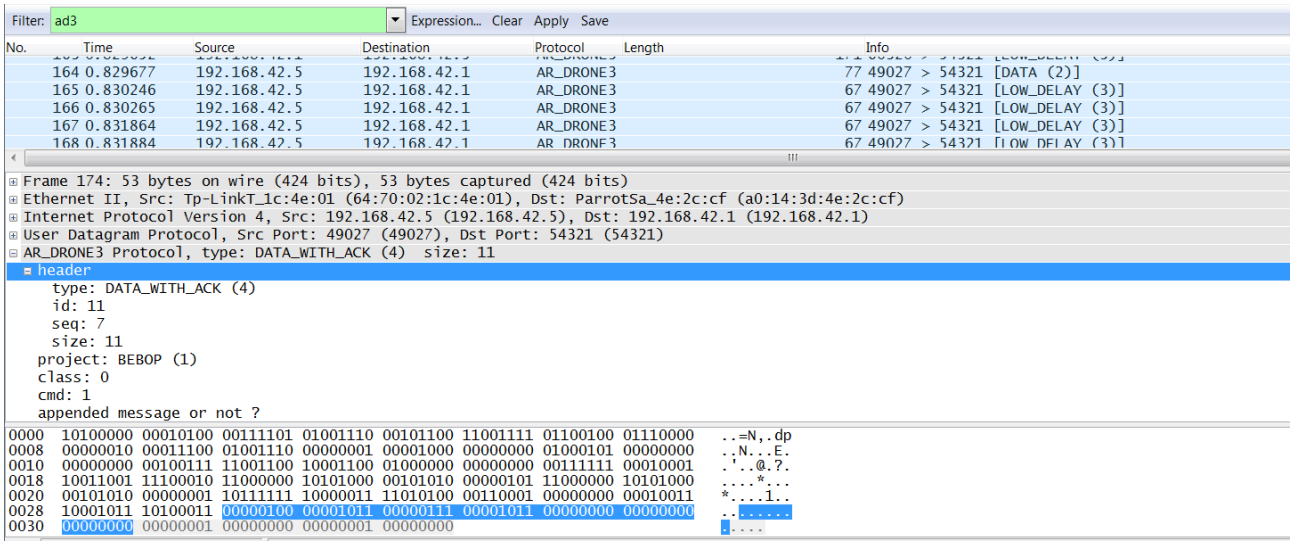

FIGURE 3.6 – Capture d'écran plugins Wireshark ArDrone3

Scénario d'utilisation

Nous présenterons ici un scénario d'utilisation de notre projet en trois phase (voir Fig. 4.1).

4.1 Initilisation

Dans un premier temps l'utilisateur lance notre programme. Ce dernier établit la connexion avec les deux drones, puis met à disposition de l'utilisateur une interface. A partir de cette interface l'utilisateur peut appréhender les informations de navigations recu et émis par chaque drone. Il peut également insérer des paramêtres nécessaires à la numérisation de la façade : l'altitude maximal, la vitesse verticale et l'angle panoramique. Une fois les paramêtres rentré, il peut lancer la procédure de numérisation en appuyant sur le bouton Launch et l'arréter en cas de problème via la bouton Stop.

4.2 Numérisation de la façade

On considère que l'utilisateur à préalablement placé les deux drones face à la façade à numériser.

- *En A* : Les deux drones sont aux sols convenablement placé par l'utilisateur.
- *De A à B* : Les deux drones entament leur décollage.
- *En B* : Une fois en vol stationnaire, les paramêtres de l'utilisateur sont envoyés et vérifiés. Puis on lance l'enregistrement vidéo.
- *De B à C* : Les deux drones entament leur ascencions jusqu'à l'altitude maximale.
- *En C* : On stop l'enregistrement, et enfin ordonne l'ordre d'aterissage.

4.3 Récupération et traitement de la vidéo

L'Utilisateur peut finalement utiliser le bouton Vidéo de l'interface pour récupérer le dernier enregistrement qui sera convertie du format mp4 au format avi, utilisable par le programme du groupe d'ISV.

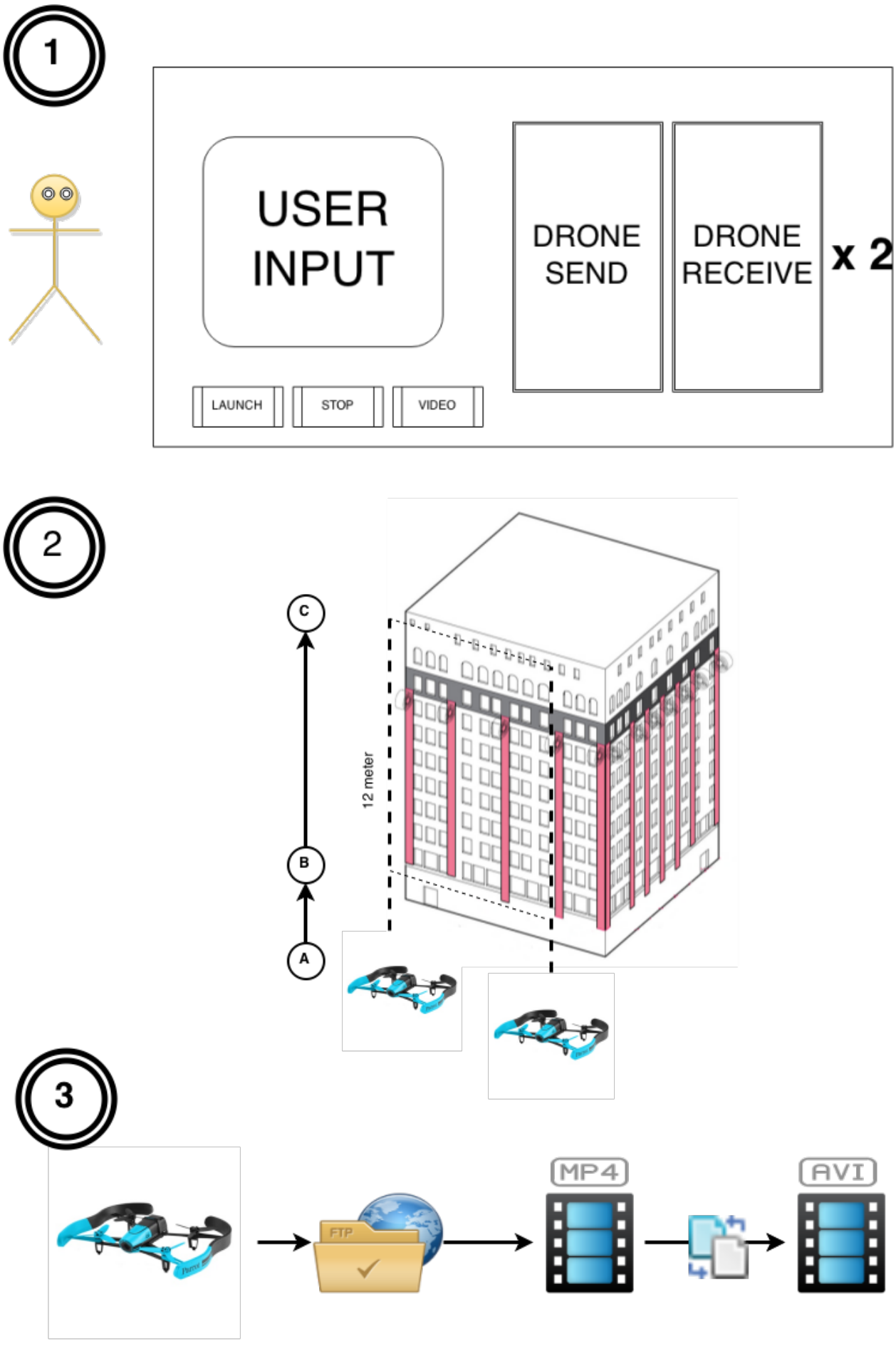

FIGURE 4.1 – Phase d'utilisation

Architecture

Nous présenterons ici le type d'architecture retenu dans le cadre de notre projet.

5.1 Modèle - Vue - Contrôleur

Nous avons choisi d'utiliser un modèle MVC pour notre implémentation :

Les contrôleurs abritent la logique métier de l'application. Ils se chargent de traiter les données par l'utilisation du modèle adéquat et choisissent la vue à utiliser pour l'affichage.

Les vues correspondent à des parties d'IHM, elles sont dédiées à la présentation des informations. Aucun traitement de donnée n'est effectué dans une vue.

Les modèles gèrent l'accès aux données. Ils fournissent une abstraction en mettant à disposition des méthodes pour récupérer/insérer les informations sous une forme facilement manipulable par les contrôleurs et les vues.

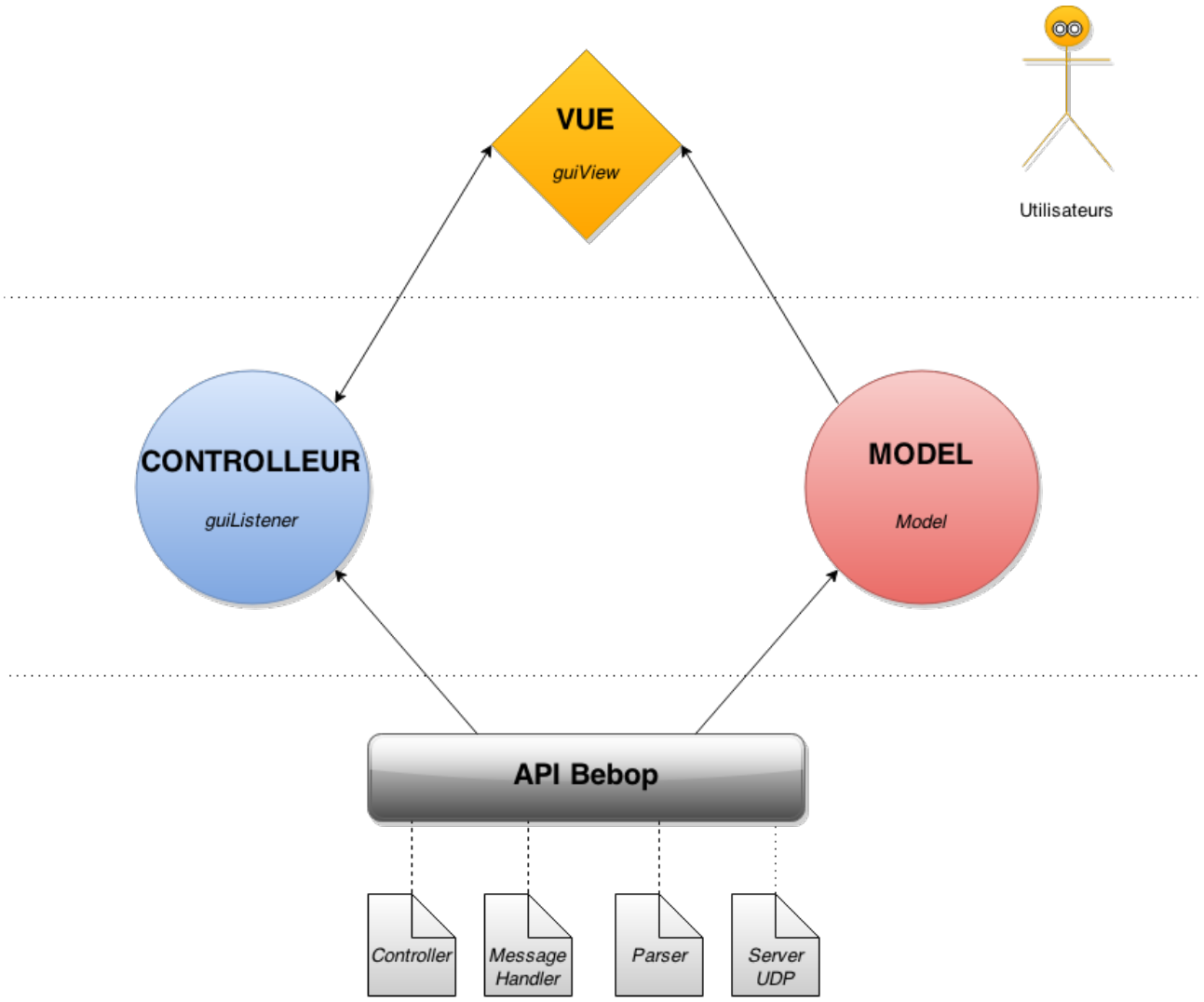

FIGURE 5.1 – Architecture MVC

Résultats

Nous présenterons dans cette partie les aboutissements de notre travail sur ce projet, en insistant particulièrement sur la réponse apportée aux différents besoins exposés précédemment.

6.1 Commandes

Lors de l'analyse des communications entre l'application FreeFlight 3 et le drone, nous avons pu observer la manière dont l'application envoie les commandes et ainsi reproduire ces commandes. A l'aide du SDK fourni par Parrot, nous avons pu forger en Java les commandes à envoyer pour interagir avec le drone et mener à bien sa mission. Comme nous souhaitions coder l'intégralité du projet en Java afin de pouvoir réutiliser quelques éléments développés lors de précédents projets (également en Java), il était plus simple pour nous de réécrire nos propres commandes, plutôt que d'utiliser le SDK tel quel. Parmi toutes les commandes disponibles, nous nous sommes limités à celles qui nous sont utiles, entre autres :

setTime & setDate : Nécessaire pour l'enregistrement des vidéos.

set max altitude & speed : Définit la vitesse maximale et l'altitude maximale.

takeoff : Permet de faire décoller le drone.

landing : Permet de faire attérir le drone.

emergency : Envoi un signal d'urgence au drone.

gaz : Permet de faire monter/descendre le drone.

start & stop recording : Démarre/arrête l'enregistrement vidéo.

Pour forger une commande, on utilise l'objet *Command* dont les attributs correspondent aux champs d'un message. Nous avons utilisé les champs suivants :

- byte _type
- byte _id
- byte _seq
- int _size
- byte _project
- byte _class
- short _cmd ;
- byte[] _arg ;

On définit pour chacune des commandes les attributs ci-dessus avec les valeurs correspondantes trouvées dans le SDK. Par exemple, pour la commande **takeoff** :

```
public Command takeoff(byte seq){
        Command cmd_takeoff = new Command(TYPE_DATA_WITH_ACK,
                11, seq, BEBOP, PILOTING, TAKE_OFF);
        cmd_takeoff._size = 11;
        _cm.addText("take off");
        return cmd_takeoff;
}
```

Les différentes constantes sont définies afin de faciliter la création de commandes en concordance avec le SDK. On précise ensuite la taille du paquet : toutes les commandes ont une taille minimale de 11 octets. Finalement on utilise la variable *_cm* représentant une instance de *ConsoleModel* afin d'afficher sur l'interface utilisateur un message explicitant la création et l'envoi du message (cf. 5.3)IHM). De plus, ce message est enregistré dans un fichier de log.

Avant d'envoyer une commande, on utilise la méthode **commandToByteArray()** qui convertit l'instance de *Command* en un tableau d'octets qui sera envoyé *via* le socket UDP. Chaque attribut de l'objet est ajouté à la bonne position dans le tableau en respectant les formats des paquets du protocole présenté plus haut. Dans le cas d'une commande prenant en argument une valeur de type *float* ou *double*, ces valeurs doivent être converties en IEEE-754. Prenons l'exemple de la méthode **doubleToIeee754(double value)** :

```
public static byte[] doubleToIeee754 (double value){
        long holder = 0;
        holder = Double.doubleToRawLongBits(value);
        ByteBuffer buffer = ByteBuffer.allocate(Long.SIZE);
        buffer.order(ByteOrder.LITTLE_ENDIAN);
        buffer.putLong(holder);
        return buffer.array();
}
```

On appelle la méthode **doubleToRawLongBits(value)** afin de convertir la variable *value* en IEEE-754 que l'on stocke dans une variable temporaire (*long holder*). L'utilisation d'un *long* permet de conserver le format IEEE-754. On utilise ensuite un *ByteBuffer* en spéficiant l'ordre souhaité (*little endian*) et l'on y insère la valeur de la variable *holder*. Le tableau de d'octets retourné par la méthode sera ensuite ajouté aux arguments de la commande.

6.2 Parser

Lors de l'analyse des communications entre l'application FreeFlight 3 et le drone, nous avons pu obverser la manière dont le drone répond à l'application et ainsi comprendre son comportement. A l'aide du SDK fourni par Parrot, nous avons pu mettre en place en Java un *parser* pour récupérer et analyser les données de navigation du drone afin de mener à bien sa mission. Parmi tous les messages "parsés", nous nous sommes limités à ceux qui nous sont utiles, à savoir :

getMaxAltitude & getCurrentAltitude : Permet de récupérer l'altitude maximale et l'altitude courante du drone.

getMaxSpeed & getCurrentSpeed : Permet de récupérer la vitesse verticale maximale et la vitesse courante du drone.

getDroneState : Permet de récupérer le statut actuel du drone.

getMediaRecordState : Permet de récupérer le statut de l'enregistrement de la vidéo.

Pour *parser* un message, on utilise l'objet *Parser* dont les attributs correspondent aux champs d'un message. On parse les champs suivants :

- byte type
- byte seq
- int size
- byte project
- byte class
- short cmd
- ByteBuffer args

On définit dans un switch l'ensemble des *type*, *projet*, *class* et *cmd* nécessaires de "parser". Pour récupérer l'argument d'une commande, on utilise la valeur de *size* et la taille constante d'une structure de message d'envoi de données. Par exemple pour le message *getDroneState* :

```
case DATA:
project = packetAD3[offset+7];
switch(project){
  case PROJECT_BEBOP:
  classs = packetAD3[offset+8];
  cmd = packetAD3[offset+9];
  switch(classs){
    case CLASS_PILOTING_STATE:
        switch(cmd){
          case CMD_FLYING_STATE_CHANGED:
          args = ByteBuffer.wrap(packetAD3, offset+11, size-11);
          _navData.setDroneState(args);
          break;
```

Les différentes constantes sont définies afin de faciliter le "parsing" des messages. Les arguments *size* et *offset* nous permettent de "parser" les éventuels messages concaténés. De plus, lorsque nous recevons un message de type envoi avec acquittement, on se charge d'acquitter le message. Une fois l'argument récupéré, ce dernier est transmis à l'objet *NavData* qui se charge de traiter l'information comprise dans l'argument afin d'afficher sur l'interface utilisateur les données de navigation actuelles du drone (cf. 5.3).

Certains arguments de commande, notamment l'altitude et la vitesse, retournent des données au format IEEE-754. Pour convertir ces données en valeurs pertinentes pour l'utilisateur, nous utilisons une fonction de conversion de ce type.

```
private double ieee754ToDouble(ByteBuffer args){
        long holder = 0;
        args.order(ByteOrder.LITTLE_ENDIAN);
        holder = args.getLong();
        return Double.longBitsToDouble(holder);
}
```

L'utilisation d'un *long holder* permet de contenir les 64 bits de données d'un flottant à double précisions sous le format IEEE-754. L'objet *ByteBuffer* nous permet de préciser l'endianisme de la trame (*little endian*) et de récupérer un *long*. Enfin, la classe *Double* se charge de convertir notre *holder* en *double*. Une fonction similaire existe pour les flottants.

6.3 IHM

L'interface homme-machine se présente de la sorte :

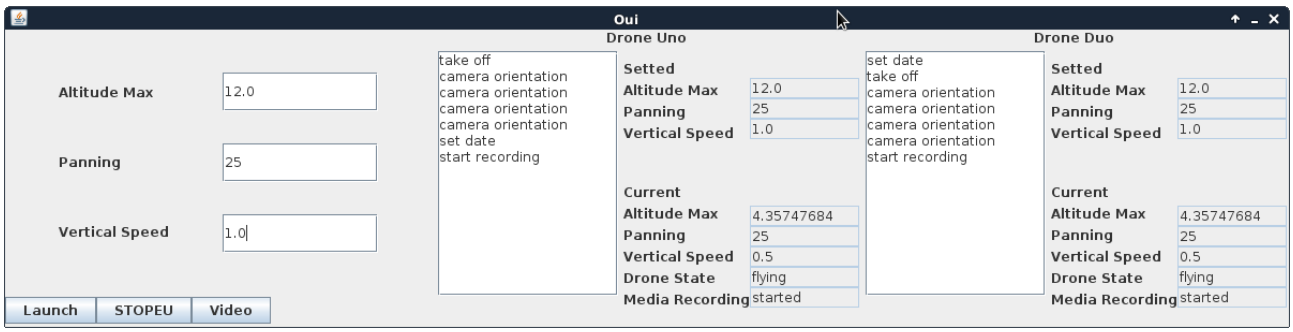

FIGURE 6.1 – Capture d'écran de l'IHM

Après avoir exécuté le programme, l'utilisateur peut saisir trois informations en fonction de la façade à parcourir :

Altitude Max (m) : L'altitude maximale du drone, typiquement la hauteur de la façade.

Panning : Le degré de la caméra sur le plan horizontal.

Vertical Speed (m/s) : La vitesse de montée du drone.

Comme expliqué précédemment, on retrouve les messages de *log* dans les deux fenêtres à chaque envoi de commande. De plus, les deux drones nous renvoient en temps réel les informations relatives à leur vol (altitude, vitesse, en cours d'enregistrement ou non,...). Enfin, le bouton **Launch** permet de démarrer la mission, le bouton **Stop** l'arrête, et le bouton **Video** démarre la récupération de la vidéo et sa conversion en AVI.

6.4 Système de numérisation de façade automatisé

L'utilisation de deux drones implique qu'ils soient tous deux synchronisés afin de fournir des vidéos exploitables. Une idée de synchronisme proposée avait été d'utiliser un laser. Toutefois, nous avons favorisé une approche plus naïve, en considérant leur vitesse de déplacement identique. Leur programme de vol se décompose en plusieurs phases :

Takeoff Les deux drones décollent et attendent d'être tous deux dans le même état "hovering".

Config Configuration de la panoramique de la caméra, de l'altitude et vitesse maximum.

MediaStarted Démarrage de l'enregistrement.

Flying Phase de montée des drones, jusqu'à atteindre la hauteur saisie par l'utilisateur.

MediaStopped Arrêt de l'enregistrement.

Lors de chacune de ces phases, les *threads* relatifs aux drones vont boucler après avoir effectuer chaque phase afin d'attendre si besoin que leur homologue termine sa phase en cours.

6.5 Récupération de vidéo et conversion

Dans le but de numériser la façade d'un bâtiment donné, les drones ont pour objectif de prendre des vidéos de la façade (une portion de bâtiment par drone). Une fois ces vidéos tournées, elles sont

stockées directement dans la mémoire vive du drone. Ces vidéos sont par défaut enregistrées au format MP4, alors que le format attendu par nos collègues d'ISV est le format AVI (car mieux géré sur toutes les plateformes). L'objectif ici est donc d'automatiser la récupération et la conversion de ces vidéos.

6.5.1 Récupération de vidéo

Le Bebop dispose d'un serveur FTP sur lequel nous pourrons récupérer ladite vidéo pour chaque drone :

```
user@machine# ftp 192.168.42.1 21

#entrer un username
username : toto

# repertoire contenant les videos
ftp> cd internal_000/Bebop_Drone/media/

# recuperation de la video voulue
ftp> get <nom_de_la_video>

# suppression de la video pour desencombrer la memoire
ftp> delete <nom_de_la_video>
```

Pour automatiser cette étape, nous avons utilisé le langage Python et plus particulièrement le module Pexpect afin de créer un processus fils simulant les commandes à effectuer pour récupérer la vidéo sur un drone. Ce script Python sera ensuite exécuté depuis l'interface graphique Java décrite précédemment.

6.5.2 Conversion de vidéo

La conversion du format MP4 en AVI s'effectue à l'aide de la solution FFMPEG, destinée au traitement de flux audio et vidéo. Le principe général de l'algorithme mis en place et de récupérer toutes les images de la vidéo à l'aide d'un objet de type *FFmpegFrameGrabber* et de les enregistrer à l'aide d'un objet de type *FFmpegFrameRecorder*. De fait, pour chaque image "saisie" par le grabber, nous l'enregistrons à l'aide du recorder au format désiré, en l'occurence le format AVI (format lisible par OpenCV).

Difficultés rencontrées

Ce chapitre est destiné à rendre compte des problèmes rencontrés lors du projet et qui ont gêné le bon déroulement de ce dernier.

7.1 Mise à jour du firmware

Plus ou moins régulièrement, Parrot déployait une mise à jour pour le Bebop, rendant impossible son utilisation avec notre OS "maison". Pour chaque nouvelle mise à jour, la manipulation fastidieuse suivante nous a ralenti :

1. Télécharger le fichier de mise à jour sur le site de Parrot
2. Le copier sur une clé USB formatée en FAT32
3. Brancher la clé sur le drone à l'aide d'un câble spécial (USB femelle, mini-USB mâle)
4. Démarrer le drone et attendre qu'il redémarre tout seul

Cette manipulation a pour conséquence de réinitialiser le drone et ainsi d'effacer notamment notre script d'initialisation (*xtstart*).

7.2 Défaillance du drone

Durant la dernière semaine de développement, nous avons rencontré un problème insolvable avec le drone : à chaque allumage du drone, son voyant d'état (normallement vert fixe) est rouge et clignote. En utilisant l'application FreeFlight 3 sur un smartphone OnePlus One, aucun problème n'est détecté : on peut se connecter au drone, on obtient bien le retour vidéo, on peut déplacer la caméra, toutefois il est impossible de le faire décoller. En utilisant l'application sur un Iphone, nous obtenons le message *"Motor problem (1), go to motor settings for more information"*, bien qu'aucune erreur ne soit affichée sur la page dédiée aux moteurs. Après de nombreuses recherches et tentatives (hard reset, réinstallation du firmware), il semblerait qu'il faille le renvoyer au S.A.V.

Conclusions et perspectives

En définitive, ce projet de modélisation de façade en 3D à l'aide de drones nous a permis de travailler avec le nouveau drone Bebop et d'analyser ses différentes spécificités, notamment au niveau du nouveau protocole mis en place par Parrot. Notre application permet donc de contrôler le drone à l'aide des différentes commandes nécessaires pour le recouvrement vidéo de façades et de récupérer ces vidéos afin de les transmettre à nos collègues qui réalisent la photogrammétrie.

En guise d'améliorations, on pourrait penser à mettre en place un retour du flux vidéo de chaque drone en temps réel, pour que l'utilisateur soit assuré de sa prise vidéo. Il serait également intéressant de mettre en place un système permettant à l'utilisateur de traiter des façades nécessitant plusieurs prise de vue des drônes, en rajoutant de la synchronisation sur des mouvements latéraux. Voire calculer par un algorithme le nombre de traitements nécessaires en fonction de la superficie de la facade et la distance focale des drones.