Projet : DRONE AUTOPILOT

Cahier des charges

Université Pierre et Marie Curie

Alex Archambault
Yoann Ghigoff
Nicolas Salleron
Kévin Vu-Saintonge

Tuteur : M. Fabrice Kordon

-24 novembre 2017

Table des matières

Chapitre 1

Introduction

Utilisés depuis les années 1990 dans le cadre militaire, les drones font aujourd'hui partie de notre vie de tous les jours. Des sociétés comme Parrot les ont popularisés au sein du grand public. Les smartphones étant de plus en plus répandus et performants, les applications peuvent exploiter les multiples capacités de leurs capteurs embarqués. Les progrès de la miniaturisation permettent même l'utilisation de ces technologies dans de plus petits objets connectés telles les montres.
Les drones sont très souvent utilisés, notamment dans les domaines artistiques comme la danse. Des émissions en témoignent : "La France a un incroyable talent" (25/10/2016) et "Britain's got talent", outre-Manche.
À Las Vegas, durant le CES 2016, la société Parrot a réalisé le "Drone dance" dans lequel les drones Parrot "dansent" de manière autonome sur une chorégraphie pré-enregistrée. A Londres la même année, la société BeTomorrow a présenté le "Flying Oreo show". Nous observons un réel intérêt du grand public pour les drones et la démonstration de leurs capacités à travers le spectacle et la danse.

1.1 Contexte

Notre projet d'"*Ingénierie Dirigée par les modèles*" est inscrit dans le domaine de la danse. Un danseur doit être capable de préparer la chorégraphie puis de l'exécuter sur le terrain de son choix. L'utilisateur utilisera pour ce faire un langage de programmation spécifique que nous allons réaliser afin de préparer ses danses simplement.

1.2 Objectifs

L'objectif est de permettre à un danseur de réaliser une chorégraphie. Pour ce faire notre équipe produira un langage de programmation simple et compréhensible pour l'utilisateur.
Ce dernier devra par la suite l'utiliser de manière autonome afin de préparer en amont la chorégraphie de son drone. Une fois la chorégraphie validée et compiler par notre logiciel frontal, l'utilisateur doit pouvoir lancer l'application générée par notre frontal. Cette application enverra différents ordres au drone et réalisera la chorégraphie.

1.3 Langage de contrôle de drone

Dans le but de rendre accessible le pilotage de drone aux artistes chorégraphes du futur, nous implémenterons un langage dédié permttant de décrire un scénario de pilotage prédéfini sous la forme d'un script.
Dans cette partie nous présentons les commandes de base de ce langage ainsi que les actions réalisées par un drone pour chacune d'elles.

1.3.1 Décoller

Syntaxe : *decoller()*

Cette commande permettra à un drone de décoller. Elle devra obligatoirement être la première commande lors de l'exécution d'un script du langage et ne pourra pas être à nouveau exécutée si le drone n'a pas atterri. Il est possible de décoller après chaque atterrissage.

1.3.2 Atterrir

Syntaxe : *atterrir()*

Cette commande permettra à un drone d'atterrir. Elle devra obligatoirement être la dernière commande lors de l'exécution d'un script du langage et ne pourra pas être à nouveau exécutée si le drone n'a pas décoller. Il est possible d'atterrir après chaque décollage.

1.3.3 Monter

Syntaxe : *monter(durée : réel)*

Cette commande permettra à un drone de monter en altitude. Elle ne pourra être exécutée que si un drone a décollé et n'a pas atterri. Le paramètre *durée* donnera le temps en secondes durant lequel le drone s'élève.

1.3.4 Descendre

Syntaxe : *descendre(durée : réel)*

Cette commande permettra à un drone de descendre en altitude. Elle ne pourra être exécutée que si un drone a décollé et n'a pas atterri. Le paramètre *durée* donnera le temps en secondes durant lequel le drone décline.

1.3.5 Avancer

Syntaxe : *avancer(durée : réel)*

Cette commande permettra à un drone de se déplacer sur un axe horizontal dans la direction vers laquelle sa face avant est tournée. Elle ne pourra être exécutée que si un drone a décollé et n'a pas atterri. Le paramètre *durée* donnera le temps en secondes durant lequel le drone doit se déplacer.

1.3.6 Reculer

Syntaxe : *reculer(durée : réel)*

Cette commande permettra à un drone de se déplacer sur un axe horizontal dans la direction opposée à celle vers laquelle est tournée sa face avant. Elle ne pourra être exécutée que si un drone a décollé et n'a pas atterri. Le paramètre *durée* donnera le temps en secondes durant lequel le drone doit se déplacer.

1.3.7 Gauche

Syntaxe : *gauche(durée : réel)*

Cette commande permettra à un drone de se déplacer sur un axe horizontal dans la direction à gauche de celle vers laquelle est tournée sa face avant. Elle ne pourra être exécutée que si un drone a décollé et n'a pas atterri. Le paramètre *durée* donnera le temps en secondes durant lequel le drone doit se déplacer.

1.3.8 Droite

Syntaxe : *droite(durée : réel)*

Cette commande permettra à un drone de se déplacer sur un axe horizontal dans la direction à droite de celle vers laquelle est tournée sa face avant. Elle ne pourra être exécutée que si un drone a décollé et n'a pas atterri. Le paramètre *durée* donnera le temps en secondes durant lequel le drone doit se déplacer.

1.3.9 Pause

Syntaxe : *pause(durée : réel)*

Cette commande permettra à un drone de s'arrêter en position stationnaire pendant une certaine durée. Elle ne pourra être exécutée que si un drone a décollé et n'a pas atterri. Le paramètre *durée* donnera le temps en secondes durant lequel le drone doit s'arrêter.

1.4 Editeur du langage

1.4.1 Description

Afin de faciliter l'écriture des scénarios de pilotage, un éditeur textuel sera fourni au client.
Celui-ci permettra entre autre : l'auto-completion du langage décrit précédemment, la détection de fautes syntaxiques ainsi que les erreurs de cohérence dans les scénarios.
Une erreur de cohérence simple peut être le fait de commander au drone d'avancer avant que celui-ci n'ai décollé.

1.4.2 Tests de validation de l'éditeur du langage

Ci-dessous une liste de tests permettant au client de valider la recette de l'éditeur du langage.

Pour le scénario suivant :

```
decoller()
monter(1.0, 20)
atterrir()
```

Résultat attendu : Aucune erreur n'est détectée par l'éditeur.

Pour le scénario suivant :

```
decoller()
monter(1.0, 20)
avancer(1.0, 20)
reculer(1.0, -20)
atterrir()
```

Résultat attendu : Aucune erreur n'est détectée par l'éditeur.

Pour le scénario suivant :

```
decoller()
blabalbla()
atterrir()
```

Résultat attendu : Une erreur est détéctée sur la commande *blabalbla*.

Pour le scénario suivant :

```
decoller()
monter("hello")
atterrir()
```

Résultat attendu : Une erreur est détéctée sur la commande *monter*.

Pour le scénario suivant :

```
decoller()
atterrir()
monter(1.0, 20)
```

Résultat attendu : Une erreur est détéctée sur la commande *monter*.

Pour le scénario suivant :

```
monter(1.0, 20)
decoller()
atterrir()
```

Résultat attendu : Une erreur est détéctée sur la commande *monter*.

Pour le scénario suivant :

```
decoller()
decoller()
```

Résultat attendu : Une erreur est détéctée sur la deuxième commande *decoller*.

Pour le scénario suivant :

decoller()
monter(1.0, 20)
atterrir()
atterrir()

Résultat attendu : Une erreur est détéctée sur la deuxième commande *atterrir*.