



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

1	Introduction	2
1.1	Contexte	2
1.2	Objectifs	2
2	Le langage de pilotage du drone	3
2.1	Description	3
2.2	Description de l'éditeur du langage	3
2.3	Écriture de la chorégraphie par l'utilisateur	3
2.4	Déploiement	3
3	Fonctionnement du drone	4
3.1	Décollage - Taking off	4
3.2	Altitude - Gaz	4
3.3	Mouvements horizontales - Roll	4
3.4	Mouvements de profondeur - Pitch	4
3.5	Pause - Stop	4
3.5.1	Atterrissage - Landing	5
4	Les commandes à implémenter	6
4.1	Décoller	6
4.2	Atterrir	6
4.3	Monter	6
4.4	Descendre	6
4.5	Avancer	7
4.6	Reculer	7
4.7	Gauche	7
4.8	Droite	7
4.9	Pause	7
4.10	Exemple d'une exécution	8
5	Tests de validation	9
5.0.1	Contrôler drone depuis l'ordinateur (via le programme utilisateur)	9
5.0.2	Tests de validation de l'éditeur du langage	12
6	Étapes du projet	14
4.0	Tranche 0	14
4.1	Tranche 1	14
4.2	Tranche 2	14
4.3	Tranche 3	14

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.

Chapitre 2

Le langage de pilotage du drone

2.1 Description

Notre solution sera un langage avec une syntaxe textuelle. Ce langage sera généré à partir d'un modèle qui respectera un méta-modèle. Une extension de fichier spécifique au langage sera utilisée.

2.2 Description de l'éditeur du langage

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'ait décollé.

2.3 Écriture de la chorégraphie par l'utilisateur

L'utilisateur écrit sa chorégraphie par une suite d'actions séparées par un retour à la ligne. La liste des actions disponibles et leurs paramètres sont listés dans la partie "commandes à implémenter?". Une chorégraphie commencera forcément par un décollage et finira par un atterrissage.

2.4 Déploiement

Pour que l'application fonctionne un certains nombre de pré-conditions d'utilisation :

- Le drone doit être allumé par l'utilisateur.
- Le drone doit être connecté via Wi-Fi à l'ordinateur exécutant le programme réalisé par l'utilisateur.
- Le drone doit être utilisé dans un endroit sans un vent de trop grande envergure.
- Le programme utilisateur doit être réalisé sous le logiciel Eclipse Oxygen version 4.7.0.
- L'ordinateur exécutant le programme réalisé par l'utilisateur doit avoir installé Java version 1.8.

Des fichiers et documents aideront pour son déploiement.

Chapitre 3

Fonctionnement du drone

3.1 Décollage - Taking off

Le drone est capable de décoller en faisant tourner ses rotors. Le décollage se fait en un temps fini, durant ce laps de temps le drone n'interprète pas les commandes qui lui sont envoyés (il les effectuera quand il sera dans un état stable).

Il correspond à l'instruction suivante dans le projet :

- **decoller**

3.2 Altitude - Gaz

Le drone est capable de se déplacer verticalement (axe z), c'est à dire qu'il peut élever ou réduire son altitude avec plus ou moins de vitesse en fonction de la vitesse de rotation des rotors (donc des moteurs).

-Il correspond aux instructions suivantes dans le projet :

- **monter**
- **descendre**

3.3 Mouvements horizontales - Roll

Le drone est capable de se déplacer horizontalement (axe x), pour se faire le drone va s'incliner vers la gauche ou la droite d'un certain angle.

Il correspond aux instructions suivantes dans le projet :

- **gauche**
- **droite**

3.4 Mouvements de profondeur - Pitch

Le drone est capable de se déplacer en profondeur (axe y), pour se faire le drone va s'incliner en avant ou en arrière d'un certain angle.

Il correspond aux instructions suivantes dans le projet :

- **avancer**
- **reculer**

3.5 Pause - Stop

Le drone est capable de faire du surplace (stabilisation).

Il correspond à l'instruction suivante dans le projet :

- **pause**

3.5.1 Atterrissage - Landing

Le drone est capable d'atterrir en ralentissant ses rotors. L'atterrissage se fait en un temps fini, durant l'atterrissage le drone n'interprète pas les commandes qui lui sont envoyés. Une fois au sol, il ne peut plus exécuter d'instruction sauf celle de décollage.

Il correspond à l'instruction suivante dans le projet :

- **atterrir**

Chapitre 4

Les commandes à implémenter

Dans le but de rendre accessible le pilotage de drone aux artistes chorégraphes du futur, nous implémenterons un langage dédié permettant 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.

4.1 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.

4.2 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.

4.3 Monter

Syntaxe : **monter(durée : réel, vitesse_vertical : entier)**

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. Le paramètre **vitesse_vertical** est un entier positif compris entre 1 et 100. Il indique la vitesse d'élévation du drone.

4.4 Descendre

Syntaxe : **descendre(durée : réel, vitesse_vertical : entier)**

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. Le paramètre **vitesse_vertical** est un entier négatif compris entre -100 et -1. Il indique la vitesse d'élévation du drone.

4.5 Avancer

Syntaxe : **avancer**(durée : réel, vitesse_deplacement : entier)

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. Le paramètre **vitesse_deplacement** est un entier négatif compris entre 1 et 100. Il indique la vitesse de déplacement du drone.

4.6 Reculer

Syntaxe : **reculer**(durée : réel, vitesse_deplacement : entier)

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. Le paramètre **vitesse_deplacement** est un entier négatif compris entre -100 et -1. Il indique la vitesse de déplacement du drone.

4.7 Gauche

Syntaxe : **gauche**(durée : réel, vitesse_deplacement : entier)

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. Le paramètre **vitesse_deplacement** est un entier négatif compris entre -100 et -1. Il indique la vitesse de déplacement du drone.

4.8 Droite

Syntaxe : **droite**(durée : réel, vitesse_deplacement : entier)

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. Le paramètre **vitesse_deplacement** est un entier négatif compris entre 1 et 100. Il indique la vitesse de déplacement du drone.

4.9 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.

4.10 Exemple d'une exécution

Voici un exemple type de programme que l'utilisateur peut faire :

```
decoller()
monter(2, 10)
avancer(2, 20)
droite(2, 15)
atterrir()
```

FIGURE 4.1 – Exemple de programme utilisateur

Le programme utilisateur sera ensuite inspecté, si ce dernier est syntaxiquement correct, l'utilisateur sera capable d'exécuter son programme. Pour se faire, son programme sera copié et cette copie sera transformée en programme JAVA, ce dernier sera ensuite exécuté ce qui demandera au drone d'effectuer les instructions qui ont été renseignées par l'utilisateur dans son programme.

4.11 Editeur du langage

4.11.1 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'ait décollé.

4.11.2 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 détectée sur la commande *blabalbla*.

Pour le scénario suivant :

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

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

Pour le scénario suivant :

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

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

Pour le scénario suivant :

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

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

Pour le scénario suivant :

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

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

Pour le scénario suivant :

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

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

Chapitre 5

Tests de validation

5.0.1 Contrôler drone depuis l'ordinateur (via le programme utilisateur)

Décollage

Titre :	Décollage du drone		
Tranche :	TODO		
Date :	24 novembre 2017		
Description :	Le drone est au sol, prêt à décoller		
Précondition :	L'ordinateur est connecté au drone. L'utilisateur a déjà réalisé son application ne contenant que l'instruction de décollage.		
Scénario	Utilisateur : 1. Le testeur lance l'application compilée par l'environnement de développement fourni. Elle ne contient que l'instruction de décollage.	Application : 2. Le système fait décoller le drone. 4. Le système le place à une hauteur minimale de fonctionnement. 6. Le système enclenche le mode de vol stationnaire	Drone : 3. Le drone décolle. 5. Le drone se place à la hauteur minimale de fonctionnement.
Postcondition :	Le drone n'est plus au sol et est maintenant dans le mode de vol "Stationnaire"		
Validation :	TV01 Décollage		
Contexte :	L'ordinateur est connecté au drone et l'utilisateur a compilé son application ne contenant que l'instruction de décollage.		
Entrée :	Aucune.		
Scénario :	1. L'utilisateur lance son application.		
Moyen de vérification :	Visuel, le drone est à sa hauteur minimale de fonctionnement.		

Atterrissage

Titre :	Atterrissage du drone		
Tranche :	TODO		
Date :	24 novembre 2017		
Description :	Le drone est en mode stationnaire, prêt à atterrir.		
Précondition :	L'ordinateur est connecté au drone. Le drone a décollé (et est en mode stationnaire). L'utilisateur a déjà réalisé son application ne contenant que l'instruction de d'Atterrissage.		
Scénario	Utilisateur : 1. Le testeur lance l'application compilée par l'environnement de développement fourni. Elle ne contient que l'instruction de d'atterrissage.	Application : 2. Le système entre en phase d'atterrissage. 3. Le système descend lentement le drone vers le sol. 5. Au sol, le système coupe le fonctionnement des hélices.	Drone : 4. Le drone descend vers le sol. 6. Le drone est au sol, les hélices ne bougent plus.
Postcondition :	Le drone est au sol.		
Validation :	TV02 Atterrissage		
Contexte :	L'ordinateur est connecté au drone et l'utilisateur a compilé son application ne contenant que l'instruction de d'atterrissage.		
Entrée :	Aucune.		
Scénario :	1. L'utilisateur lance son application.		
Moyen de vérification :	Visuel, le drone est au sol, les hélices sont en arrêt de fonctionnement.		

Mouvements sur le plan horizontal

Titre :	Mouvement sur le plan horizontal (axe X ou Y)		
Tranche :	TODO		
Date :	24 novembre 2017		
Description :	Le drone réagit aux commandes définies par l'utilisateur dans son application.		
Précondition :	L'ordinateur est connecté au drone. Le drone a décollé. L'utilisateur à déjà réaliser son application ne contenant que des instructions de mouvements sur le plan horizontal		
Scénario	Utilisateur : 1. Le testeur lance l'application compilée par l'environnement de développement fourni. Elle ne contient que des instructions de déplacement sur les axes X et Y.	Application : 2. Le système donne des ordres au drone pour le déplacement.	Drone : 3. Le drone se déplace de gauche à droite sur l'axe X (ou Y).
Postcondition :	Le drone a changé de position.		
Validation :	TV03 Mouvement horizontal (axe X ou Y)		
Contexte :	L'ordinateur est connecté au drone. Le drone a décollé et est en mode stationnaire. L'utilisateur à compiler son application ne contenant que des instructions de déplacement sur les axes X et/ou Y.		
Entrée :	aucune		
Scénario :	1. L'utilisateur lance son application.		
Moyen de vérification :	Visuel, le drone a changé de position.		

Mouvement sur le plan vertical

Titre :	Mouvement sur le plan vertical (axe Z)		
Tranche :	TODO		
Date :	24 novembre 2017		
Description :	Le drone réagit aux actions de l'utilisateur sur le plan vertical		
Précondition :	L'ordinateur est connecté au drone. Le drone a décollé. L'utilisateur à déjà réaliser son application ne contenant que des instructions de déplacement sur l'axe Z		
Scénario	Utilisateur : 1. Le testeur lance l'application compilée par l'environnement de développement fourni. Elle ne contient que des instructions de déplacement sur l'axe Z.	Application : 2. Le système donne des ordres au drone pour le déplacement.	Drone : 3. Le drone se déplace de gauche à droite sur l'axe Z.
Postcondition :	Le drone a changé de position.		

Validation :	TV04 Mouvement vertical
Contexte :	Le mobile est connecté au drone. Le drone a décollé et est en mode stationnaire. L'utilisateur à compiler son application ne contenant que des instructions de déplacement sur l'axe Z.
Entrée :	aucune
Scénario :	1. L'utilisateur lance son application.
Moyen de vérification :	Visuel, le drone a changé de position.

5.0.2 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 détectée sur la commande *blabalbla*.

Pour le scénario suivant :

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

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

Pour le scénario suivant :

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

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

Pour le scénario suivant :

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

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

Pour le scénario suivant :

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

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

Pour le scénario suivant :

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

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

Chapitre 6

Étapes du projet

À propos des délais de livraison, nous avons opté pour une livraison en tranches à intervalles réguliers. Le client aura donc un état d'avancement continu du projet. Ainsi, avec son accord, nous livrerons dans un délai maximum les tranches suivantes :

4.0 Tranche 0 : 28 novembre 2017

Tout d'abord, nous livrerons un cahier des charges au client pour formaliser notre compréhension du projet et de ses modalités. Nous exposerons l'analyse des besoins, les réponses apportées, les tests que pourra effectuer le client afin de tester le fonctionnement du programme et nous décrirons les modalités finales du projet.

4.1 Tranche 1 : 14 décembre 2017

Nous spécifions un métamodèle pour développer un langage compréhensible pour l'utilisateur afin qu'il puisse envoyer des commandes au drone. Ce dernier pourra exécuter les mouvements grâce à des appels de fonction du type (Décoller, Atterrir, Avancer, Monter...) spécifiés dans la partie Commandes.

Nous livrerons un éditeur textuel basé sur XText spécifique au langage afin de pouvoir déterminer un enchaînement de mouvements qui définira une chorégraphie. L'utilisateur pourra grâce à cet éditeur appelé les instructions qui sont reconnues par l'éditeur

4.2 Tranche 2 : 12 janvier 2018

La tranche 2 consistera à produire un programme pour que le drone puisse exécuter un scénario. Le langage cible JAVA du programme sera généré à partir du DSL spécifié pour le scénario. L'ensemble des instructions sera lié avec des appels de fonctions de l'API du drone Parrot. Le code ainsi généré sera ensuite compilé afin de produire un programme exécutable par le drone.

4.3 Tranche 3 : 25 janvier 2018

Cette dernière étape consistera à une livraison de la distribution permettant d'exécuter une chorégraphie par l'utilisateur et sa mise en place sur un système de déploiement avec l'installation des plugins pour le DSL. Le serveur sera lié au drone qui exécutera le scénario. Enfin une démonstration avec le drone illustrera le fonctionnement de la distribution.