Projet : DRONE AUTOPILOT

Cahier des charges

Université Pierre et Marie Curie

Alex Archambault
Yoann Ghigoff
Nicolas Salleron
Kévin Vu-Saintonge

Tuteur : M. Fabrice Kordon

-10 décembre 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 de l'éditeur du langage . . . 3

2.2 Écriture de la chorégraphie par l'utilisateur . . . 3

2.3 Conditions de fonctionnement . . . 3

3 Fonctionnement du drone **4**

3.1 Décollage . . . 4

3.2 Altitude - Gaz . . . 4

3.3 Mouvements horizontaux - Roll . . . 4

3.4 Mouvements horizontaux - Pitch . . . 4

3.5 Pause . . . 5

3.6 Rotation . . . 5

3.7 Atterrissage . . . 5

3.8 Camera . . . 5

4 Les commandes du DSL **6**

4.1 Prologue . . . 6

4.1.1 Pourcentage vitesse max ? . . . 6

4.1.2 vitesse_deplacement max . . . 6

4.1.3 vitesse_rotation max . . . 6

4.1.4 Hauteux max . . . 6

4.1.5 Eloignement max . . . 6

4.2 Les fonctions basiques . . . 6

4.2.1 Décoller . . . 6

4.2.2 Atterrir . . . 6

4.2.3 Monter . . . 7

4.2.4 Descendre . . . 7

4.2.5 Avancer . . . 7

4.2.6 Reculer . . . 7

4.2.7 Gauche . . . 7

4.2.8 Droite . . . 8

4.2.9 Pause . . . 8

4.2.10 RotationGauche 8
4.2.11 RotationDroite 8
4.2.12 Exemple d'une exécution 9
4.3 Fonction de composition 9
4.3.1 Création de composition sequentielle 9
4.3.2 Bilbiothèque de fonctions 9
4.3.3 Fonction de composition parallèle 10

5 Tests de validation **11**
5.1 Validation tranche 1 - (Validation de l'éditeur du langage) 11
5.2 Validation tranche 3 - (Contrôler drone depuis l'ordinateur via le programme utilisateur) 15

6 Étapes du projet **19**
6.1 Tranche 0 : 28 novembre 2017 19
6.2 Tranche 1 : 14 décembre 2017 19
6.3 Tranche 2 : 12 janvier 2018 19
6.4 Tranche 3 : 25 janvier 2018 19

7 Glossaire **20**

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

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.

2.1 Description de l'éditeur du langage

Afin de faciliter l'écriture des scénarios de pilotage, un éditeur textuel sera fourni au client. Cet éditeur aura les fonctionnalités suivantes :

- l'auto-completion du langage (décrit dans la partie "commandes à implémenter")
- la détection de fautes syntaxiques.
- la détection des erreurs de cohérence dans les scénarios. Ex : Une erreur de cohérence simple peut être le fait de commander au drone d'avancer avant que celui-ci n'ai décollé.
- TODO des "bullets" avec d'autres exemples

2.2 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.

2.3 Conditions de fonctionnement

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

- Le drone utilisé est un drone à hélices, le langage ne supportera pas les drones type avion.
- 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 à son déploiement et son utilisation.

Chapitre 3

Fonctionnement du drone

Nous considérons par drone les objets connectés volants, type aéronef à voilure tournante comportant quatre rotors pour sa sustentation.

3.1 Décollage

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 horizontaux - 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 horizontaux - 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

Le drone est capable de faire du surplace (stabilisation).
Il correspond à l'instruction suivante dans le projet :

— **pause**

3.6 Rotation

Le drone est capable d'effectuer une rotation sur le plan horizontal sans modifier sa hauteur.
Il correspond aux instructions suivantes dans le projet :

— **rotationGauche**
— **rotationDroite**

3.7 Atterrissage

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**

3.8 Camera

Nous avons fait le choix de ne pas nous occuper de la caméra, car certains drones n'en possèdent pas. Il n'est pas standard qu'un drone possède une caméra.

Chapitre 4

Les commandes du DSL

4.1 Prologue

4.1.1 Pourcentage vitesse max ?

4.1.2 vitesse_deplacement max

4.1.3 vitesse_rotation max

4.1.4 Hauteux max

4.1.5 Eloignement max

4.2 Les fonctions basiques

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. Nous avons décidé de séparer les mouvements horizontaux et verticaux en 6 fonctions distinctes (Monter, Descendre, Avancer, Reculer, Gauche, Droite) afin de rendre l'utilisation du langage plus implicite pour l'utilisateur. L'unité du paramètre durée est en Seconde.
La vitesse verticale étant dépendante du drone, nous avons fait le choix d'exprimer ce paramètre sous la forme d'une unité en pourcentage de la vitesse maximale du drone. L'inclinaison étant également dépendant du drone cible, nous avons fait le choix de l'exprimer également en pourcentage d'inclinaison par rapport à l'inclinaison maximale disponible.

4.2.1 Décoller

Syntaxe : **decoller()**

Cette commande permettra à un drone de décoller. Elle devra obligatoirement être la première commande exécutée dans 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.

4.2.2 Atterrir

Syntaxe : **atterrir()**

Cette commande permettra à un drone d'atterrir. Elle devra obligatoirement être la dernière commande exécutée dans 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.

4.2.3 Monter

Syntaxe : **monter(durée : Seconde, vitesse_verticale : Pourcentage)**

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'élèvation du drone.

4.2.4 Descendre

Syntaxe : **descendre(durée : Seconde, vitesse_verticale : Pourcentage)**

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 positif compris entre 1 et 100. Il indique la vitesse d'élèvation du drone.

4.2.5 Avancer

Syntaxe : **avancer(durée : Seconde, vitesse_deplacement : Pourcentage)**

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 positif compris entre 1 et 100. Il indique la vitesse de déplacement du drone.

4.2.6 Reculer

Syntaxe : **reculer(durée : Seconde, vitesse_deplacement : Pourcentage))**

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 positif compris entre 1 et 100. Il indique la vitesse de déplacement du drone.

4.2.7 Gauche

Syntaxe : **gauche(durée : Seconde, vitesse_deplacement : Pourcentage))**

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 positif compris entre 1 et 100. Il indique la vitesse de déplacement du drone.

4.2.8 Droite

Syntaxe : **droite(durée : Seconde, vitesse_deplacement : Pourcentage))**

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 positif compris entre 1 et 100. Il indique la vitesse de déplacement du drone.

4.2.9 Pause

Syntaxe : **pause(durée : Seconde)**

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.

4.2.10 RotationGauche

Syntaxe : **rotation_gauche(durée : Seconde, vitesse_rotation : Pourcentage)**

TODO
TODO KORDON veut l'angle ?
TODO
Cette commande permettra à un drone de tourner sur lui même dans le sens inverse des aiguilles d'une montre. 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 tourner Le paramètre **vitesse_rotation** est un entier positif compris entre 1 et 100. Il indique la vitesse de rotation du drone.

4.2.11 RotationDroite

Syntaxe : **rotation_droite(durée : Seconde, vitesse_rotation : Pourcentage)**

TODO
TODO KORDON veut l'angle ?
TODO
Cette commande permettra à un drone de tourner sur lui même dans le sens des aiguilles d'une montre. 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 tourner Le paramètre **vitesse_rotation** est un entier positif compris entre 1 et 100. Il indique la vitesse de rotation du drone.

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

FIGURE 4.1 – Exemple de programme utilisateur

4.2.12 Exemple d'une exécution

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

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.

4.3 Fonction de composition

Notre langage impose des contraintes sur les fonctions. D'abord, la première fonction du fichier à exécuter s'appelle "main", elle est la première fonction exécutée par le drone. De plus, les fonctions ne peuvent pas être récursive, c'est à dire, qu'une fonction ne peut s'appeller elle même.

4.3.1 Création de composition sequentielle

En plus des commandes de base, notre langage permet à l'utilisateur de définir ses propres compositions qui sont des suites d'instructions à exécuter par le drone.

Exemple :

```
func _nom_de_la_fonction_() {
    monter(2, 10)
    avancer(2, 20)
    droite(2, 15)
}
```

4.3.2 Bilbiothèque de fonctions

Une fois que l'utilisateur a défini ses fonctions de composition et les a sauvegardées dans un fichier, celui-ci aura la possibilité de les référencer depuis un autre fichier au moyen d'un système d'import. Ainsi l'uilisateur pourra appeler dans un fichier une fonction définie dans un autre fichier.
Ce système permettra à l'utilisateur de créer ses propres bilbiothèques de fonctions.
Afin de pouvoir appeler les fonctions définies dans un fichier, ce dernier devra se trouver dans le même répertoire que celui contenant le fichier appelant les fonctions.

4.3.3 Fonction de composition parallèle

Notre langage intégrera un mécanisme de composition d'exécution de commandes en parallèle. Il permettra par exemple de monter et d'avancer en même temps.
Ce mécanisme sera implémenté dans notre langage par le biais du symbole '*&*'.
La parallélisation ne sera disponible que sur les commandes de bases. Elle n'est pas disponible pour paralléliser deux fonctions.

De plus on ne pourra pas composer deux commandes opposées en parallèle par exemple :

- Monter(1,10) & Descendre(1,10)
- Avancer(1,10) & Reculer(1,10)
- Gauche(1,10) & Droite(1,10)
- RotationGauche(1,10) & RotationDroite(1,10)

Chapitre 5

Tests de validation

5.1 Validation tranche 1 - (Validation de l'éditeur du langage)

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

Validation :	TVEDT-01
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	monter(1.0, 20)
	atterrir()
	}

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

Validation :	TVEDT-02
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	monter(1.0, 20)
	avancer(1.0, 20)
	reculer(1.0, 20)
	atterrir()
	}

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

Validation :	TVEDT-03
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	blabalbla()
	atterrir()
	}

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

Validation :	TVEDT-04
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	monter("hello")
	atterrir()
	}

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

Validation :	TVEDT-05
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	atterrir()
	monter(1.0, 20)
	}

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

Validation :	TVEDT-06
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	monter(1.0, 20)
	decoller()
	atterrir()
	}

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

Validation :	TVEDT-07
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	decoller()
	}

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

Validation :	TVEDT-08
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	monter(1.0, 20)
	atterrir()
	atterrir()
	}

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

Validation :	TVEDT-09
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	aller_retour()
	atterrir()
	}
	func aller_retour() {
	avancer(1.0, 20)
	reculer(1.0, 20)
	}

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

Validation :	TVEDT-10
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	foo()
	atterrir()
	}
	func bar() {
	avancer(1.0, 20)
	}

Résultat attendu :	Une erreur est détéctée sur l'appel à la fonction *foo*.

Validation :	TVEDT-11
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	monter(1.0, 20) & avancer(1.0, 20)
	atterrir()
	}

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

Validation :	TVEDT-12
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune

Scénario :	func main {
	prologue {
	}
	decoller()
	monter(1.0, 20) & descendre(1.0, 20)
	atterrir()
	}

Résultat attendu :	Une erreur est détéctée sur l'appel de la composition parallèle des fonctions *monter* et *descendre*.

TODO - Faire des tests plus méchants (fct composition séquence, etc)

5.2 Validation tranche 3 - (Contrôler drone depuis l'ordinateur via le programme utilisateur)

Décollage

Titre :	Décollage du drone		
Tranche :	2		
Date :	10 décembre 2017		
Description :	Le drone est au sol, prêt à décoller		
Précondition :	L'ordinateur est connecté au drone. L'utilisateur à 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 à compiler 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 :	2		
Date :	10 décembre 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 à déjà réalisé son application ne contenant que l'instruction de d'Atterrisage.		
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 Atterrisage
Contexte :	L'ordinateur est connecté au drone et l'utilisateur à compiler 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 :	2		
Date :	10 décembre 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 réalisé 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 :	2		
Date :	10 décembre 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 compiler 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 :	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 l'axe Z.
Entrée :	aucune
Scénario :	1. L'utilisateur lance son application.
Moyen de vérification :	Visuel, le drone a changé de position.

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 :

6.1 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.

6.2 Tranche 1 : 14 décembre 2017

Nous spécifirons 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 enchainement de mouvements qui définira une chorégraphie. L'utilisateur pourra grâce à cet éditeur appelé les instructions qui sont reconnues par l'éditeur. Enfin la validation de cette tranche se fera avec les tests de validation TVEDT-01 à TVEDT-08

6.3 Tranche 2 : 12 janvier 2018

La tranche 2 consistera à produire un programme pour que le drone puisse éxecuter 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. Enfin la validation de cette tranche se fera avec les tests de validation TV-01 à TV-04

6.4 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.

Chapitre 7

Glossaire

méta-modèle TODO