



PIE - 2017

Simulation de mission d'un drone d'inspection d'éolienne

Auteurs :

M. Mohammed Wassim BENCHEKROUN
M. Adrien DE-JAUREGUIBERRY
M. Matthieu DROUARD
M. Brendan JEHANNIN
M. Noha LOIZON
M. Clément POIRIER



14 mars 2017

Table des matières

Résumé	2
Introduction	3
I Gestion de projet et organisation	4
1 Introduction	5
2 Définition du projet	6
3 Risques et opportunités :	9
4 Organisation du travail	11
II Travaux et réalisations techniques	14
1 Présentation du simulateur	15
2 Aérodynamique du stork	21
3 Simulation du Cardan	25
4 Simulation de la motorisation	32
5 Autopilote et interfaçage PX4	34
Conclusion	37

Résumé

OBJECTIF : Développer une simulation réaliste du Stork en fonction du plan de vol et des paramètres d'entrée pour tester des scénarios de mission et quantifier le taux de réussite.

RÉALISATION :

Introduction

Le champs d'application des drones s'élargit de plus en plus. Ils permettent à l'homme de s'affranchir de tâches qui sont dangereuses, pénibles ou simplement répétitives. L'entreprise Sterblue, commanditaire de ce projet, applique cette technologie à l'inspection et à la maintenance de grandes structures comme les éoliennes et les pylônes électriques. Jusqu'à aujourd'hui, ces inspections se faisaient par des équipes humaines. C'est une tâche complexe et dangereuse car le nombre d'éléments à inspecter est très important et ceux-ci sont difficiles d'accès. L'utilisation de drone pour le suivi des infrastructures offre alors plusieurs avantages : l'inspection est plus rapide et moins coûteuse. Le drôle peut aussi, en parallèle, constituer une base de donnée pour aider à la maintenance sur de grandes échelles de temps. Cette activité n'est pas sans risque pour l'engin volant : les environnements ne sont pas dégagés. Le drone doit donc pouvoir effectuer sa mission en évitant les obstacles. Il n'est pas non plus souhaitable que le drone collisionne l'infrastructure inspectée. Il est ainsi important pour Sterblue de pouvoir démontrer la fiabilité et la robustesse de ses outils à ses clients. C'est dans ce cadre que cette entreprise nous a demandé de développer un simulateur.

Pour bien comprendre ce que doit pouvoir faire ce simulateur, il faut détailler les tâches effectuées par le drone ainsi que les contraintes qui s'exercent sur la mission. Pour ce qui est des tâches, un travail d'inspection se déroule en plusieurs temps. Le drone commence par décoller de sa base. Puis il se rend à différents points de passages. Il doit ensuite orienter une caméra vers la zone à inspecter et prendre des photos. Pendant les prises de vue, le drone doit être immobile. La mission se déroule ainsi entre les différents points de passage puis le drone retourne se poser à sa base. Sterblue cherche donc à effectuer toutes ces tâches dans un environnement qui peut être difficile par la présence d'obstacles et de vent. Nous allons développer un simulateur pour montrer les conditions qui peuvent mener à une réussite ou à un échec de la mission.

Il convient maintenant de s'intéresser au drone en lui-même. Le drone de Sterblue est un drone convertible nommé Stork. Il combine donc deux modes de vol : le vol de croisière en mode avion et le vol stationnaire en mode quadrirotor. Le passage de l'un à l'autre se fait par la rotation des rotors qui se trouvent en bout d'ailes. Le drone embarque un ensemble de capteurs ainsi qu'un autolopilote et une caméra dirigée par un cardan trois axes. Le drone est alimenté par une batterie. C'est autour du Stork que nous avons développé le simulateur.

Le simulateur s'intéresse à l'évolution du drone dans son environnement. La physique du drone en mode avion et en mode quadrirotor est donc simulée. Les fonctions essentielles pour le suivi de la mission comme l'orientation de la caméra et la prise de vue sont aussi prises en charge. La consommation de la batterie est aussi simulée car c'est une cause de panne. Comme il sera détaillé plus tard, notre simulateur s'interface avec les outils de commande et de décision du vrai drone. Notre charge de travail sur les commandes concerne donc l'interface avec ces systèmes.

Au cours de ce travail, nous avons travaillé avec les outils Morse et Blender. Blender est un logiciel de rendu 3D qui gère aussi l'évolution physique des objets (application de force, calcul de trajectoire inertielles) via ce qui est appelé un Game Engine. Morse est une surcouche de Blender, développée par le LAAS, qui permet de simuler des robots. Ces deux outils sont open source et nous pouvons donc développer une application spécifique en se basant dessus.

Dans ce rapport, nous détaillons notre travail sur le sujet de sa définition avec le client à sa réalisation. Nous commençons ainsi par expliciter les objectifs de notre simulateur et les livrables à fournir. Nous présenterons ensuite notre organisation de travail ainsi que le plan autour duquel le simulateur s'est construit. Puis nous détaillerons des points spécifiques sur lesquels un effort important a été nécessaire. Enfin, nous dresserons un bilan de notre travail au regard des objectifs et des exigences qui ont été initialement fixés.

Première partie

Gestion de projet et organisation

Chapitre 1

Introduction

Notre équipe est constituée de six ingénieurs issus de filières diverses. Nous avons bénéficié de l'encadrement de notre tuteur projet Mr. Guigou qui nous a guidé lors de notre progression en terme de gestion et d'organisation. Du côté de notre client Sterblue, notre contact principal était Vincent Lecrubier qui nous a introduit au projet et aux besoins exprimés par l'entreprise. Il a aussi été notre interlocuteur privilégié au sein de Sterblue.

L'entreprise Sterblue utilise des drones pour réaliser l'inspection d'éoliennes et de pylones électriques. Les risques d'échec de la mission sont multiples : vent trop important, panne de batterie, collision. Les environnements sont exigeants car l'élément à inspecter peut être proche d'habitations ou d'arbres. Notre client a donc exprimé le besoin de réaliser des simulations du drone Stork lors de la réalisation d'une mission d'inspection. Le but est de mesurer le taux d'échec des missions. Pour répondre à ce besoin, nous avons donc développé un simulateur pour le drone Stork et son environnement.

Afin d'aboutir à ce résultat, Sterblue nous a imposé les outils et logiciels à utiliser. La réalisation du produit s'effectue essentiellement à travers le logiciel de simulation MORSE qui est couplé à Blender, le langage de programmation étant Python. Blender est un logiciel modélisation 3D qui intègre notamment un Game Engine, c'est à dire la gestion des forces physiques sur les objets. Morse en une surcouche de Blender pour la simulation de robots. Ce module important est un projet open-source du LAAS. La programmation dans cet environnement est donc complexe. Pour pouvoir travailler sur plusieurs versions et en parallèle, Sterblue a créé en ligne un gestionnaire de versions Git sur lequel les fichiers exploités par la simulation sont enregistrés et sauvegardés.

Une communication efficace a été essentielle tout au long du déroulement de notre projet. Nous avons eu recours, conformément au souhait de Sterblue, à la plate-forme en ligne **Mattermost**. Cet outil de chat a permis de faciliter considérablement nos échanges, non seulement avec notre référent entreprise Vincent Lecrubier, mais également avec le reste des membres de l'entreprise. Nous avons ainsi pu échanger sur l'avancé du projet, les difficultés rencontrées et nous avons pu bénéficier de leur aide précieuse.

Le besoin initial exprimé par notre client était difficile à cadrer et à bien définir en raison de son ambiguïté. Suivant les conseils de notre tuteur projet, il a été primordial de cerner le besoin et de pouvoir le quantifier. Ainsi le besoin que notre équipe a reformulé et à validé auprès du client est le suivant :

Objectif : développer une simulation réaliste du Stork en fonction du plan de vol et des paramètres d'entrée pour tester des scénarios de mission et quantifier le taux de réussite.

Cette objectif est ambitieux et il est vite apparu qu'il allait falloir définir quels points sont importants pour l'atteindre. Le niveau de détail de ces points allait aussi devoir être défini de manière précise. Pour cela, nous avons adopté un processus itératif. Des retours fréquents ont été faits au client afin de définir avec lui les aspects importants de la simulation.

Chapitre 2

Définition du projet

2.1 Organisation générale :

2.2 Contraintes imposées :

- Contraintes en terme de ressources utilisées :
 - MORSE et Blender.
 - Gestionnaire de versions GIT.
 - Interfacage avec Perception¹.

2.3 Exigences client :

- Exigences spécifiques du point de vue SIMULATION :
 - Simple
 - Réaliste pour estimer le taux d'erreur. Devra inclure :
 - La dynamique de vol en mode multi rotor.
 - La dynamique de vol en mode avion.
 - La transition (multirotor → Avion)
 - Réaliser le lien avec perception.
 - Prendre des photos et les sauvegarder au format jpg
 - Réutilisable : faciliter l'utilisation par de nouveaux utilisateurs
 - Documentation complète.
 - Architecture modulaire.

2.4 PBS :

Pour une meilleure vision du produit final, on procède à la décomposition du besoin client en terme de sous composants constituteurs. Le diagramme ci dessous résume la structure développée.

1. Sterblue a développé un système qui fait le lien entre une mission et les tâches nécessaires pour la réaliser. Ce module, Perception, transforme une mission en tâches unitaires. Ce sont ces tâches qui sont envoyées en entrée de notre simulation.

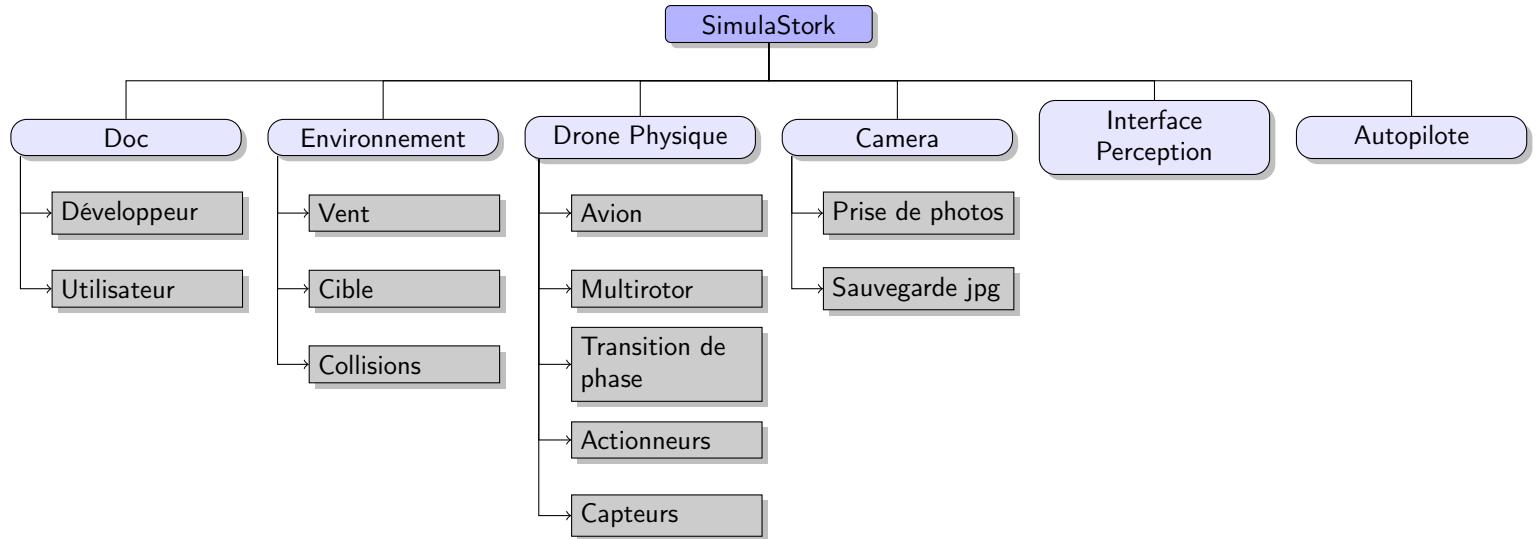


FIGURE 2.1 – PBS Simulastork

Bien évidemment des changements de structure du produit ont eu lieu tout au long du projet. La dynamique Avion et Multirotor ont été couplées et ne sont donc plus indépendantes. La transition de phase entre ces deux modes n'a donc plus été nécessaire. On cite également la modification de l'autopilote qui a été remplacée par un interfacage PX4, les détails seront donnés plus loin. La modélisation du vent dans l'interface n'a pas été jugée prioritaire, c'est pourquoi elle n'a pas été implémentée. Enfin, des caméras latérales ont été ajoutés pour répondre à un besoin spécifique de Sterblue.

La décomposition finale est résumé dans le diagramme ci dessous. Les rectangle rouge indiquent des composants supprimés, en vert on indique les composants ajoutés et en orange on indique les composants qui ont été conservés mais ont été sujets à des modifications.

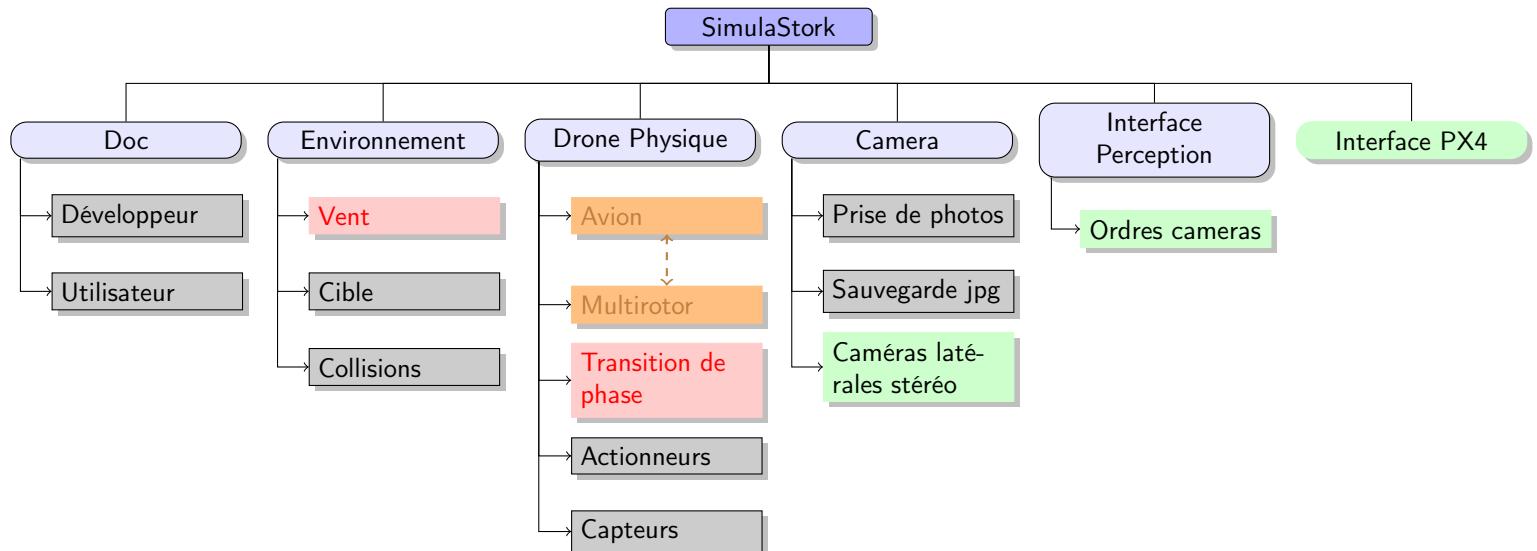


FIGURE 2.2 – PBS Simulastork final

2.5 WBS :

Une première décomposition en tâche élémentaire du travail à effectuer a été réalisée, celle ci est résumée dans le diagramme WBS suivant :

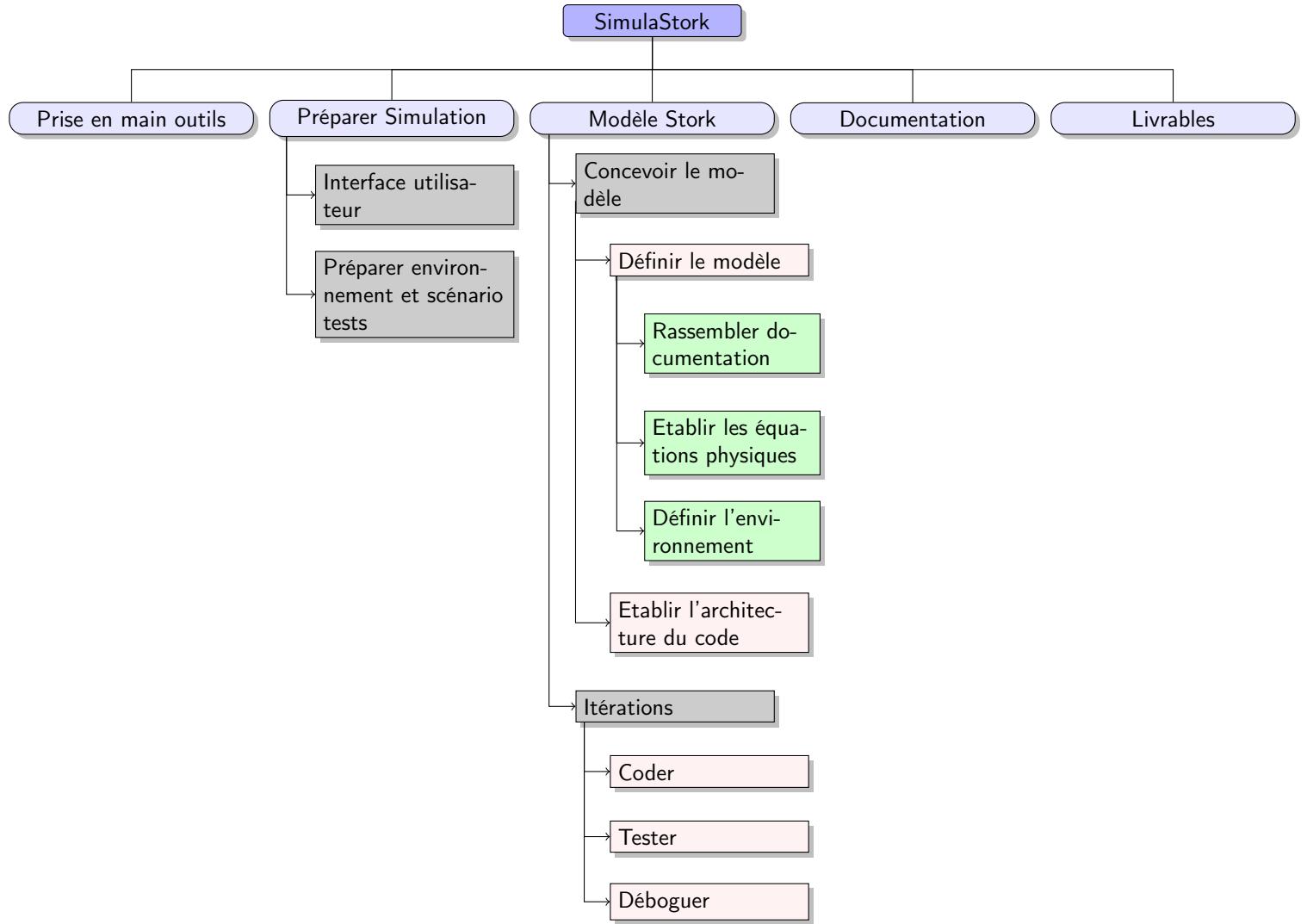


FIGURE 2.3 – WBS Simulastork

La sous tâche nommée *Itération* a été répétée deux fois. L'organisation relative entre ces différentes tâches est représenté dans le diagramme de logique de développement suivant :

Chapitre 3

Risques et opportunités :

3.1 Risques :

Afin de se préparer aux imprévus, une étude des risques a été effectuée. Cependant, certains risques initialement non identifiés ont eu lieu.

3.1.1 Risques initiaux :

Le tableau ci dessous résume les risques identifiés en début de projet.

TABLE 3.1 – Risques initiaux

Risque	Réponse apportée
1) Impossibilité d'atteindre un niveau de réalisme suffisant pour l'estimation du taux d'échec.	Développement itératif du simulateur
2) Mauvais fonctionnement des outils de travail.	Installation des outils au centre informatique de Supaero.
3) Absence de fonctionnalités essentielles dans Morse.	Réduire la complexité du modèle.
4) Obtention tardive des données nécessaires.	Communication avec le client.

TABLE 3.2 – Impact des risques :

Haute		1	
Moyenne			2
Basse		4	3
Proba			
Impact	Faible	Moyen	Fort

Compte tenu du temps imparti, l'impossibilité d'atteindre un niveau de réalisme suffisant nous semble très probable. Néanmoins, par un développement en plusieurs étapes avec revue de la part du client, l'impact sur le projet est modéré. En effet, le client aura l'opportunité de définir ses priorités au fur et à mesure de l'avancement.

Le risque de non-fonctionnement des outils informatiques était très probable initialement. L'installation de ces outils au centre informatique a réduit la probabilité de ce risque. Celui-ci aura quand même un impact important si il se produit car il nous rendrait dans l'incapacité de travailler.

3.1.2 Evolution des risques :

1. Impossibilité d'atteindre un niveau de réalisme suffisant pour l'estimation du taux d'échec.

— **Evolution du risque :**

ce risque a été spécifié. Notre nouvelle maîtrise de Morse nous permet de dire que la simulation de l'aérodynamique est réalisable avec le niveau de détail désiré. Cependant, l'interfaçage avec l'autopilote reste un risque important.

— **Nouvelle réponse :**

après l'apport des petites améliorations désirées par Sterblue sur la caméra et le cardan.

2. Mauvais fonctionnement des outils de travail.

— **Évolution du risque :**

la réponse apportée a permis d'éviter le risque sur les outils locaux. Nous n'avions pas prévu la perte des outils en ligne. Lorsque cela est arrivé, cela nous a bloqué et a induit un retard.

3. Absence de fonctionnalités essentielles dans Morse.

— **Évolution du risque :**

avec notre nouvelle maîtrise de Morse, nous pouvons réduire la probabilité de ce risque. Il nous semble possible de l'utiliser pour faire une modélisation correcte. Il semble que Morse ne permet pas de spécifier une matrice d'inertie. Cela a un impact faible pour le moment mais limitera la complexification du modèle par Sterblue.

4. Obtention tardive des données nécessaires.

— **Évolution du risque :**

la communication s'effectue bien et le retour de Sterblue sur toutes les questions est rapide.

TABLE 3.3 – Évolution des risques

Haute		1	
Moyenne			
Basse		(4)	2, (3)
Proba Impact	Faible	Moyen	Fort

En plus de l'évolution des risques initiaux, un nouveau risque non identifié a eu lieu et a été source de retard :

Indisponibilité des outils en ligne entre le 15 et le 27 décembre. Durant cette période, le dépôt git n'était plus disponible. Nous n'avons donc pas pu fusionner nos travaux respectifs durant cette période. La mise en commun et la vérification du bon fonctionnement du simulateur a été retardé jusqu'au début janvier.

3.2 Opportunités

Les opportunités suivantes ont été identifiées :

- Existence de modules proches de nos besoins déjà développés par la communauté.
- Possibilité de cours et de contacts avec le LAAS, qui a développé Morse.

Au final, aucune de ces deux opportunités ne s'est présenté. Notre utilisation de Morse pour simuler un drone est apparemment originale. Pour les contacts avec le LAAS, nous avons poser des questions dont les réponses ont été peu utiles.

Chapitre 4

Organisation du travail

4.1 Répartition du travail

Une partie des tâches ont été effectués par l'ensemble du groupe : l'installation des outils de travail et la prise en main des outils. Ensuite, les tâches ont été réparties selon les affinités et les compétences de chacun :

- Mohammed : modélisation de l'aérodynamique
- Adrien : chef de projet, gestion des différentes versions et de la fusion des travaux, modélisation de l'environnement
- Mathieu : modélisation des caméras
- Brendan : modélisation du cardan
- Noha : Gestion du contrôleur, interface entre la simulation et l'autopilote PX4
- Clément : modélisation de la batterie et des moteurs

4.2 Planning

Au début du projet, nous ne connaissions pas la difficulté de chacune des tâches. Il a ainsi fallu mettre en place une logique de développement basé sur des itérations successives de version du simulateur. En effet, de cette manière, nous pouvions avancer sur le sujet tout en redéfinissant régulièrement les priorités avec notre clients. Nous avons ainsi pu converger sur une version en un temps raisonnable et qui contente Sterblue. Notre logique de développement a suivi le schéma suivant :

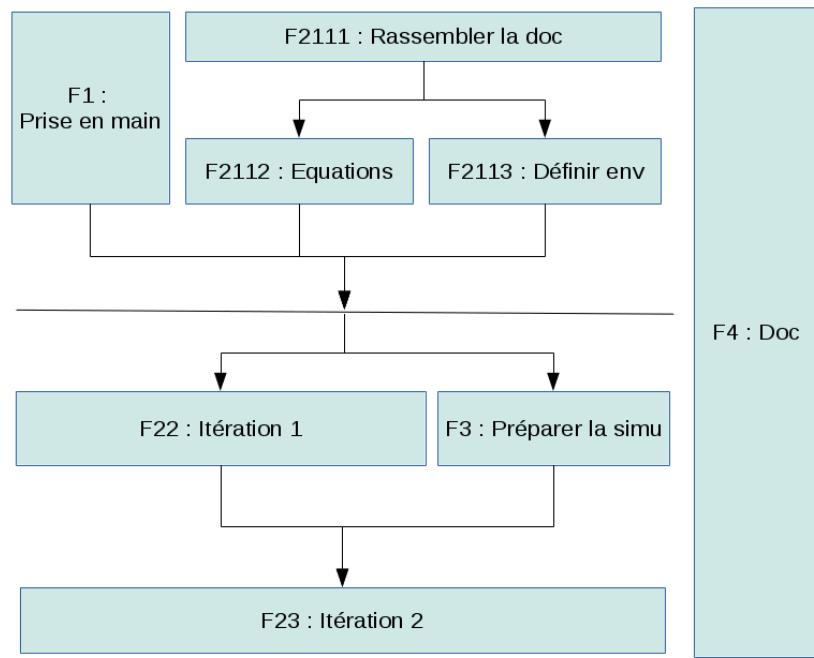


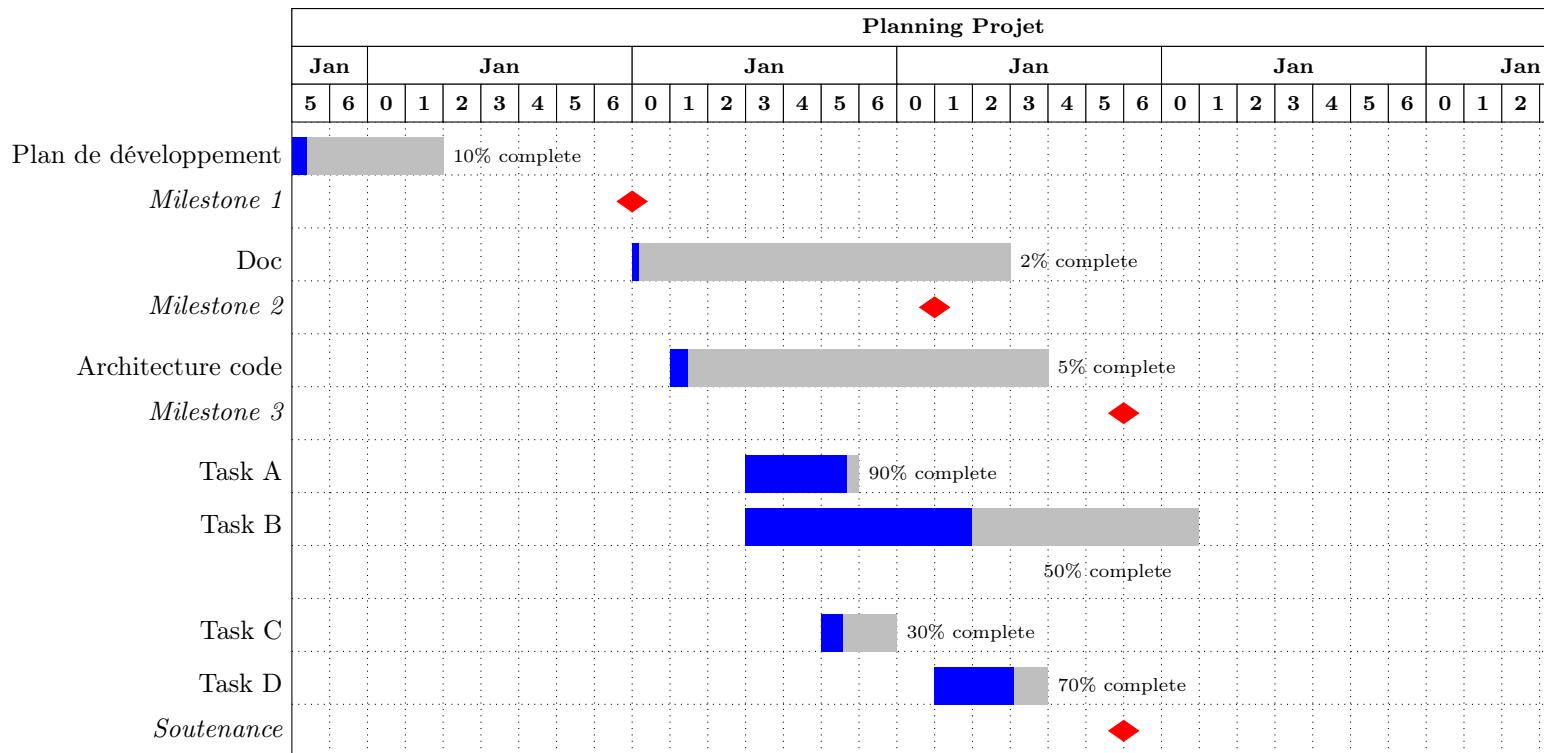
FIGURE 4.1 – Logique de développement

Initialement, nous nous étions fixé un certain nombre de jalons :

- 2 novembre : Plan de développement - Répartition des tâches
- 15 novembre : Documentation achevée
- 30 novembre : Architecture du code
- 17 décembre : Première simulation fonctionnelle
- 23 février : Deuxième version du simulateur
- 24 mars : Soutenance et remise des livrables

Un imprévu est arrivé : entre le 15 et le 27 décembre, les outils en ligne de Sterblue n'ont pas fonctionné. Nous n'avions plus accès au dépôt Git, ce qui a empêché la fusion de nos travaux pour la première itération. L'arrivée des fêtes de fin d'années ensuite a encore ralenti le rendu de la première version du simulateur. Au final, en prenant ce retard en compte nous avons mis à jour le planning de la façon suivante :

Jalon	Date initiale	Nouvelle date
Plan de développement - Répartition des tâches	2 novembre	2 novembre
Prise en main de Morse	15 novembre	15 novembre
Architecture du code	30 novembre	30 novembre
Première simulation fonctionnelle	17 décembre	8 janvier
Retour sur la première itération		20 janvier 2017
Deuxième version du simulateur	23 février	10 mars
Soutenance et remise des livrables	24 mars	13 ou 17 mars



Deuxième partie

Travaux et réalisations techniques

Chapitre 1

Présentation du simulateur

1.1 Principes de fonctionnement

1.1.1 Schématisation globale

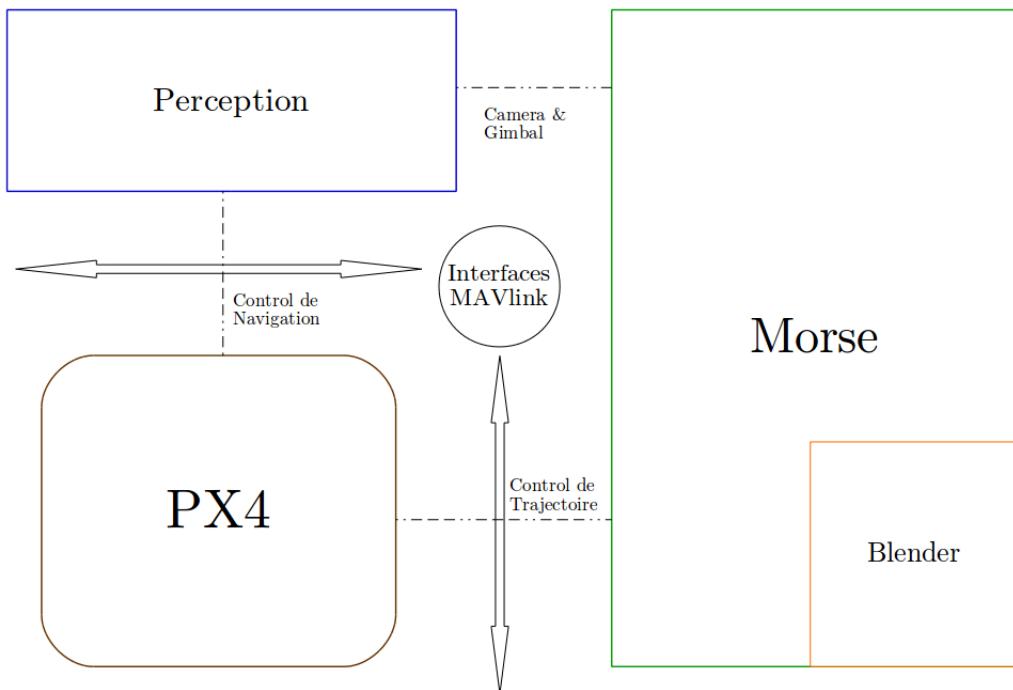


FIGURE 1.1 – Schéma global du simulateur

Le simulateur consiste en un interfaçage entre trois sous programmes :

- Le module Perception : Développé par Sterblue, ce logiciel sert en pratique de station au sol pour communiquer avec le Stork. Ici il joue le rôle du "maître" de la simulation. C'est d'ici qu'émanent tous les ordres de plus haut niveau tels que les trajectoires par WayPoints ou les prises de photos.
- Le PX4 : Simulé ici, c'est en pratique un microcontrôleur avec un OS temps réel qui génère des lois de commandes pour les actionneurs. Il a l'avantage de communiquer par protocole MAVlink donc peut être inséré dans la simulation en SITL sans modification du micrologiciel.
- Morse : Développé par le LAAS, il sert de surcouche pour exploiter le logiciel blender et son moteur physique. Le principal intérêt est de pouvoir assembler des actionneurs et capteurs

dans un véhicule et le faire se mouvoir dans un environnement Blender. Il offre également par défaut une interface Socket assez simple mais sans garantie de synchronisme.

1.1.2 Fonctionnement de Morse

Morse a été construit en deux parties : un Builder qui permet la mise en place et l'initialisation de l'architecture et une partie réservée à la boucle de simulation. C'est deux parties ont des fichiers sources bien distincts qu'il ne faut pas confondre, les premiers étant dans les dossier "builder".

Le builder utilise l'api *bpy* pour manipuler les data de blender hors simulation et ainsi construire l'environnement et les robots. En revanche, la partie boucle de simulation exploite l'api *bge* utilisée généralement dans les scripts python du Blender Game Engine et qui permet d'agir sur la simulation en temps réel.

Builder - Phase d'initialisation

Avant de lancer la simulation, Morse utilise le builder pour instancier les objets dont il a besoin et modifier le fichier blender de la simulation. Cette étape est nécessaire car impossible à réaliser une fois la simulation lancée.

Boucle de simulation

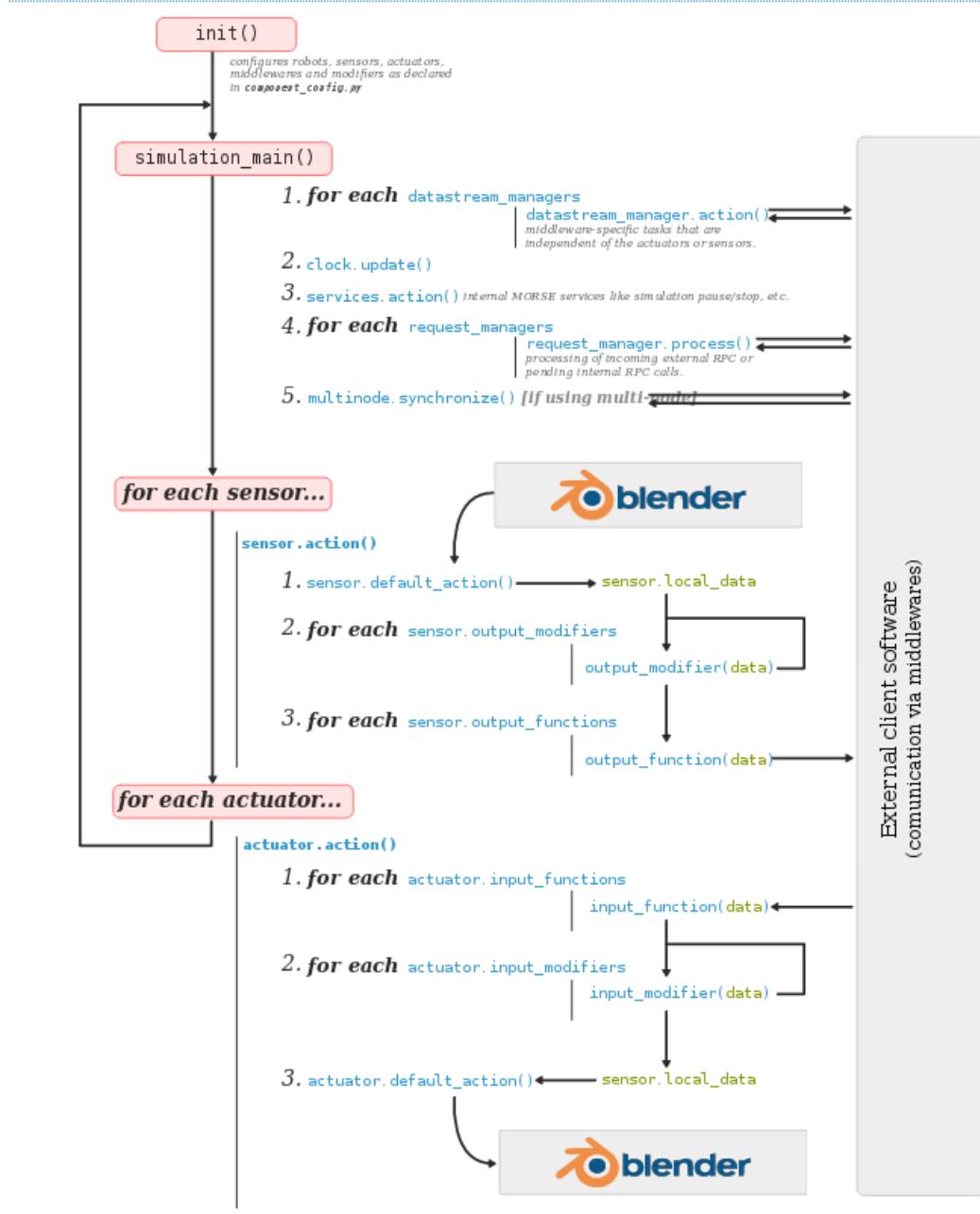


FIGURE 1.2 – Boucle de simulation dans Morse (source : www.openrobots.org)

Morse a des concepts d'Actuator et de Sensor permettant respectivement de lire des données de la simulation et d'agir sur la simulation. Après une phase d'initialisation où Morse construit les objets (dont les robots), la simulation entre dans cette boucle de fonctionnement :

- Morse met à jour l'horloge et effectue les services demandés.
- Chaque capteur (sensor) lit des informations sur l'état actuel de la simulation depuis l'api de Blender.
- Chaque actionneur (actuator) modifie des données de la simulation en vue de la prochaine boucle de calcul du Blender game engine.

1.2 Architecture du Simulastork (Morse)

1.2.1 Schéma UML

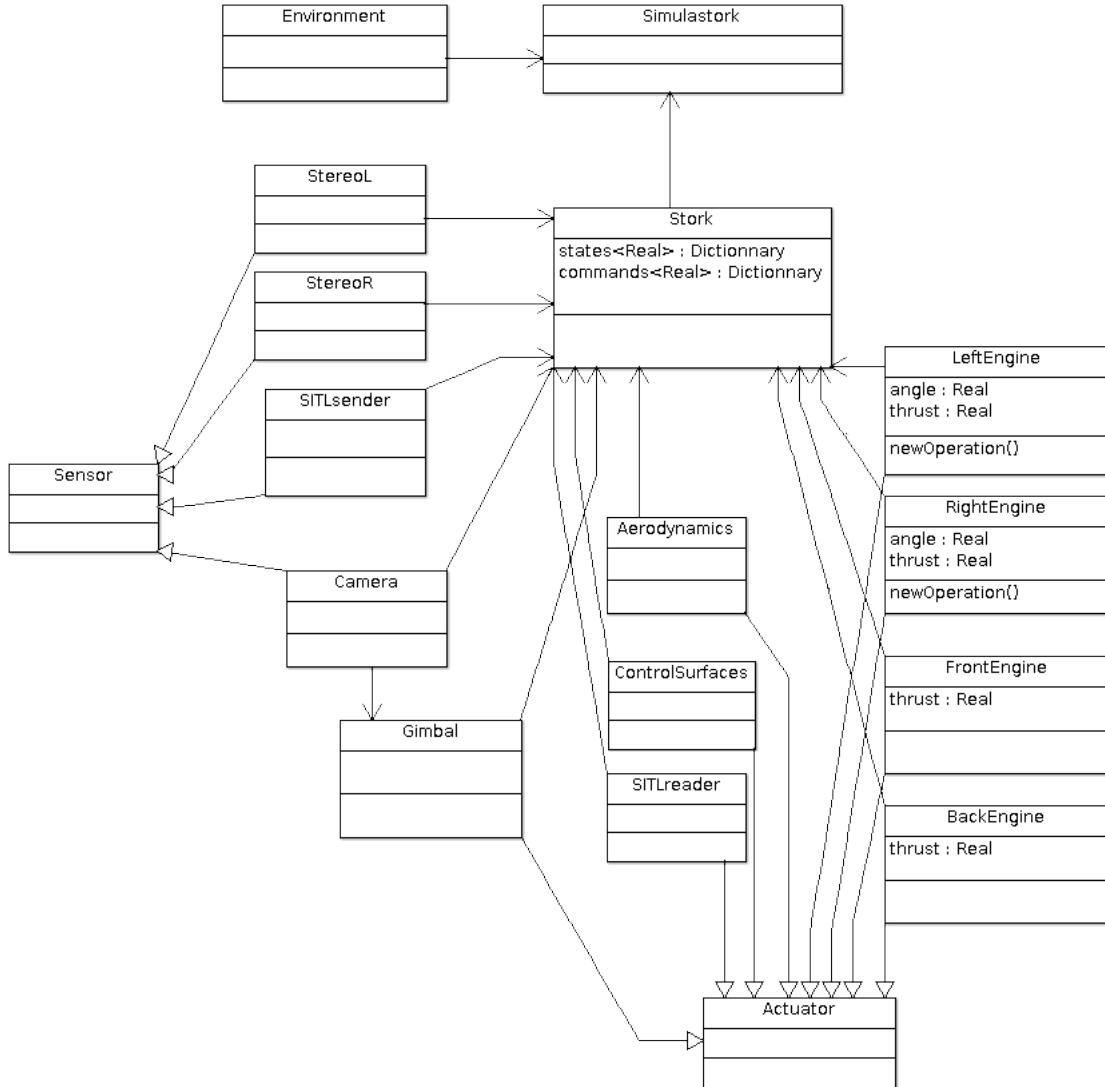


FIGURE 1.3 – Diagramme UML du Simulastork simplifié

L'architecture du logiciel est telle que sur le schéma UML. La simulation possède un robot "Stork". Ce robot possède 7 actionneurs et 5 capteurs. De plus, la simulation dispose également d'un environnement Blender qu'elle mettra en interaction avec tous ses robots.

Nous avons séparé en plusieurs blocs les différents composants du drone pour rendre l'architecture la plus naturelle et compréhensible possible, nous avons donc :

- 4 moteurs qui appliquent chacun une force et mettent à jour la vue 3d
- un actionneur Aerodynamics qui calcule les forces aérodynamiques
- un actionneur ControlSurfaces qui met à jour les positions des gouvernes
- des STIL reader et sender qui gèrent la communication avec le px4
- trois cameras dont une déplaçable par le gimbal

1.2.2 Communication entre les blocs

Les différents modules décrits précédemment ont besoin de s'échanger des données pour fonctionner correctement. Nous avons fait le choix pour la plupart d'entre eux de rassembler les échanges de données dans l'instance de l'objet Srork. Celui-ci dispose par exemple d'un dictionnaire de commandes qui est mis à jour soit par le SITLreader, soit par le service de communication avec joystick, et qui est lu par la suite par les autres actionneurs pour calculer les efforts résultants.

1.3 Fonctionnalités de la version finale

Dans notre livrable final, nous fournissons un simulateur ayant toutes les fonctionnalités suivantes :

- **Simulation de vol d'un drone de type Stork dans un environnement Blender :** Les collisions avec l'environnement sont gérées. Les calculs de dynamique du vol sont générés par contre l'inertie du drone est approximative et non réglable par faute de fonctionnalités suffisantes dans le Blender game engine.
- **Simulation du cardan, support de caméra frontale :** La simulation utilise l'affichage auxiliaire de Morse pour montrer le pointage de la caméra principale. Celui-ci est géré suivant un ordre envoyé par Perception ou en appelant un script.
- **Prise de photos de la caméra frontale :** Le simulateur autorise l'enregistrement instantané d'une image de la caméra frontale en vue d'un traitement postérieur. L'ordre est envoyé soit par Perception, soit en appelant un script.
- **Génération automatique d'un environnement de simulation avec les APIs google :** Il est possible de générer à partir d'un template de base pré-construit un nouvel environnement 3D focalisé sur une coordonnée géographique. Le script télé-chargera des données d'altitude et de photos satellite et les assemblera pour former un fichier blender exploitable par Morse.
- **Intégration d'un PX4 en SITL :** L'interfaçage MAVlink du simulateur autorise l'insertion d'un firmware de type PX4 au sein de la simulation. Ainsi, la partie Perception/PX4 est quasi identique à la version utilisée dans la réalité.
- **Visualisation 3D temps réel :** En supposant la machine suffisamment puissante, la simulation produira une visualisation en temps réel avec suivi du drone dans l'environnement.
- **Personalisation du Stork :** La fichier de configuration à la racine du simulateur permet en le modifiant une paramétrisation détaillée du drone.
- **Contrôle du Stork via un joystick :** Un script paramétrable est également fourni pour pouvoir contrôler le Stork dans la simulation avec un joystick.

1.3.1 Création d'un environnement blender

Le script de génération d'environnement de simulation fonctionne en utilisant l'api *bpy* de Blender. Il fonctionne en exécutant la procédure suivante :

- Lecture des paramètres d'entrée : L'utilisateur doit saisir les paramètres du générateur (position géographique de la scène, taille de carte, résolution de la carte...)
- Ouverture du logiciel Blender en tâche de fond et en mode exécution de script python.
- Chargement du fichier template de base : Ce fichier contient tous les éléments requis au minimum pour la simulation (sphère de ciel, éolienne, sol vierge). L'éolienne se trouve dans un calque différent et devra être placée à la main. Le sol consiste en un plan carré trivial de 2m de côté sans texture et est placé à l'origine des axes.
- Téléchargement de la carte : Le script va utiliser l'api StaticMap de Google pour télécharger l'image satellite de la scène en petits tiles assemblés par la suite. Il sera éventuellement nécessaire de fournir une clé d'api car elle n'est pas tout à fait gratuite.
- Traitement de la texture du sol : Une fois l'image récupérée, elle est enregistrée dans le dossier textures de l'environnement et assignée au plan du sol.

- Téléchargement des altitudes : Le script utilise cette fois l'api Elevations de Google pour récupérer des listes d'altitudes suivant la précision demandée. Une autre clé pourra également être utilisée.
- Manipulation des vertex du sol : Le plan du sol est ensuite mis à l'échelle pour subdivisé jusqu'à atteindre la précision nécessaire pour les données d'altitudes. Ensuite, ces dernières sont exploitées pour déplacer verticalement chacun des vertex du plan.
- Enregistrement du fichier Blender : Le modèle est finalement enregistré dans le dossier de l'environnement.

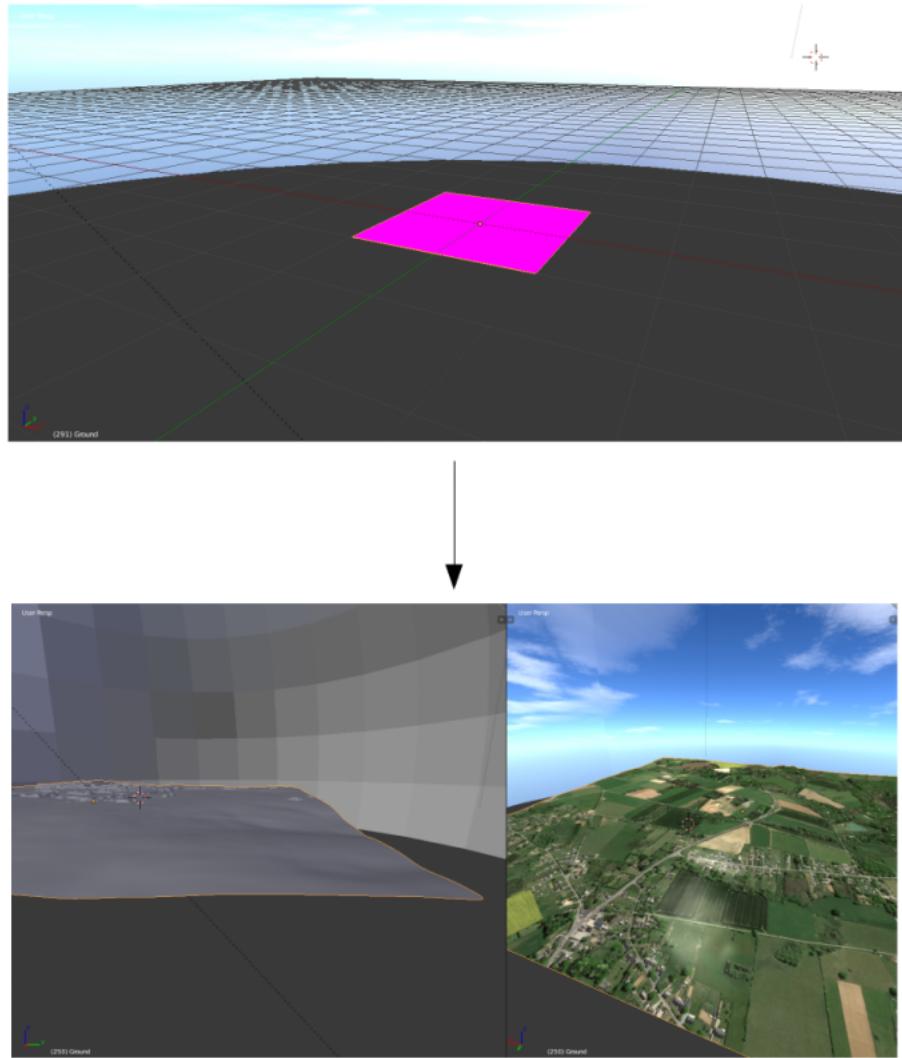


FIGURE 1.4 – Génération d'un environnement depuis le fichier vierge

Une fois généré, le fichier doit être sélectionné dans la simulation (default.py) pour être chargé au lancement.

Chapitre 2

Aérodynamique du stork

La modélisation de l'aérodynamique du stork est essentielle pour le réalisme de la simulation. Ceci dit, les performances aérodynamiques du stork, notamment les coefficients aérodynamiques, sont requis pour la réalisation de cette tâche.

Cependant, notre client Sterblue n'ont pas encore effectué de test de vol qui leur permettrait d'obtenir les paramètres aérodynamiques du stork. Par conséquent, et suite à nos échanges, il a été décidé de se baser sur les performances aérodynamiques d'un drone similaire au stork connu sous le nom de *Skywalker* 2.1, et ce pour éviter de baser les résultats du simulateur sur des données théoriques.



FIGURE 2.1 – Skywalker

Une fois les paramètres et coefficients aérodynamiques obtenus l'implémentation de la dynamique de vol s'effectue à travers les équations classiques de mécanique du vol représentées ci dessous :

$$\begin{cases} C_L = C_{L0} + C_{L\alpha} \cdot \alpha & ; \quad P = \frac{1}{2}\rho(h)V_a^2S \cdot C_L \\ C_D = C_{D0} + C_{D1} \cdot \alpha + C_{D2} \cdot \alpha^2 & ; \quad T = \frac{1}{2}\rho(h)V_a^2S \cdot C_D \\ C_Y = C_{Y\beta} \cdot \beta + C_{Y\delta r} \cdot \delta_r & ; \quad Y = \frac{1}{2}\rho(h)V_a^2S \cdot C_Y \end{cases}$$

FIGURE 2.2 – Coefficient des forces aérodynamique

$$\left\{ \begin{array}{l} C_l = C_{l\beta} \cdot \beta + \frac{b}{2 \cdot V_a} [C_{lp} \cdot p + C_{lr} \cdot r] + C_{l\delta_p} \cdot \delta_p ; \quad L = \frac{1}{2} \rho(h) V_a^2 S \cdot b \cdot C_l \\ C_m = C_{m0} + C_{m\alpha} \cdot \alpha + \frac{\bar{c}}{2 \cdot V_a} \cdot C_{mq} \cdot q + C_{m\delta_q} \cdot \delta_q ; \quad M = \frac{1}{2} \rho(h) V_a^2 S \cdot \bar{c} \cdot C_m \\ C_n = C_{n\beta} \cdot \beta + \frac{b}{2 \cdot V_a} [C_{np} \cdot p + C_{nr} \cdot r] + C_{n\delta_r} \cdot \delta_r ; \quad N = \frac{1}{2} \rho(h) V_a^2 S \cdot b \cdot C_n \end{array} \right.$$

FIGURE 2.3 – Coefficient des moments aérodynamique

Pour la première itération les variables δ_p , δ_q , δ_r , qui représentent respectivement les braquages des ailerons, de la gouverne de profondeur et celle de direction, n'ont pas été prises en compte. Par contre dans la deuxième itérations ces paramètres ont été modélisés et les surfaces de contrôles qui leur sont associées sont devenues mobile au niveau du stork.

Un modèle d'atmosphère standard a été utilisé pour représenter les variations des propriétés de l'air en fonction de l'altitude. On suppose que notre drone vol exclusivement au sein de la troposphère, c'est à dire à des altitudes inférieures à 11 km au dessus du niveau de la mer.

$$\rho_0(1 - 22.6 h 10^{-6})^{4.26}$$

2.1 Différents repères mis en jeu :

L'application des forces et moments aérodynamiques dépend du repère adopté, pour cela nous explicitons les différents repères mis en jeu.

2.1.1 Repère avion :

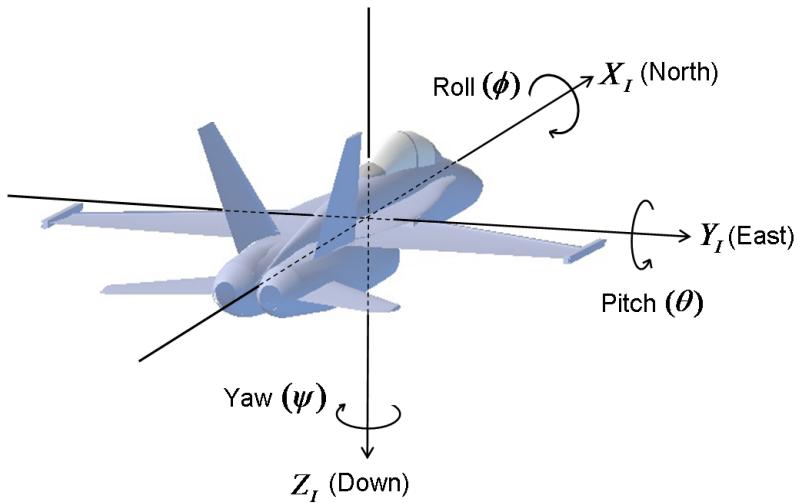


FIGURE 2.4 – Repère avion

L'origine du repère est le centre de gravité G de l'appareil.

- \vec{x}_b : Aligné avec l'axe de symétrie du fuselage et dirigé vers le nez de l'avion.
- \vec{y}_b : Dirigé selon l'aile droite.
- \vec{z}_b : Dirigé vers le bas afin que le repère soit direct.

2.1.2 Repère aérodynamique :

Ce repère est obtenue en effectuant une rotation autour de l'axe \vec{y}_b d'un angle α représentant l'incidence, suivi d'une rotation autour de l'axe \vec{z}_a d'un angle β représentant l'angle de dérapage. Le centre du repère est toujours le centre de gravité G .

- \vec{x}_a : Colinéaire avec la vitesse aérodynamique \vec{V}_a et dirigé vers le nez de l'avion.
- \vec{y}_a : Rotation de \vec{y}_b par l'angle de dérapage β .
- \vec{z}_a : Rotation de \vec{z}_b par l'angle de dérapage α .

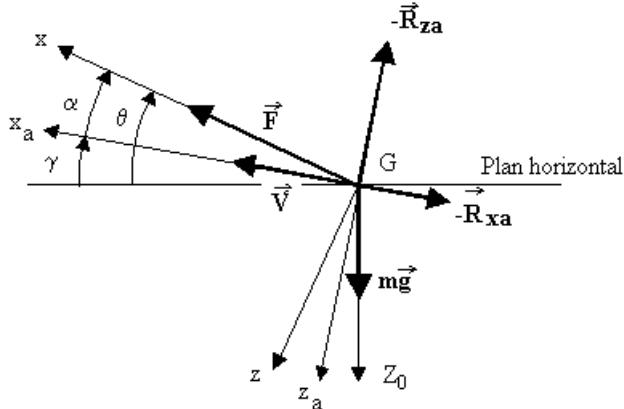


FIGURE 2.5 – Repère aérodynamique

2.1.3 Repère Stork sous blender :

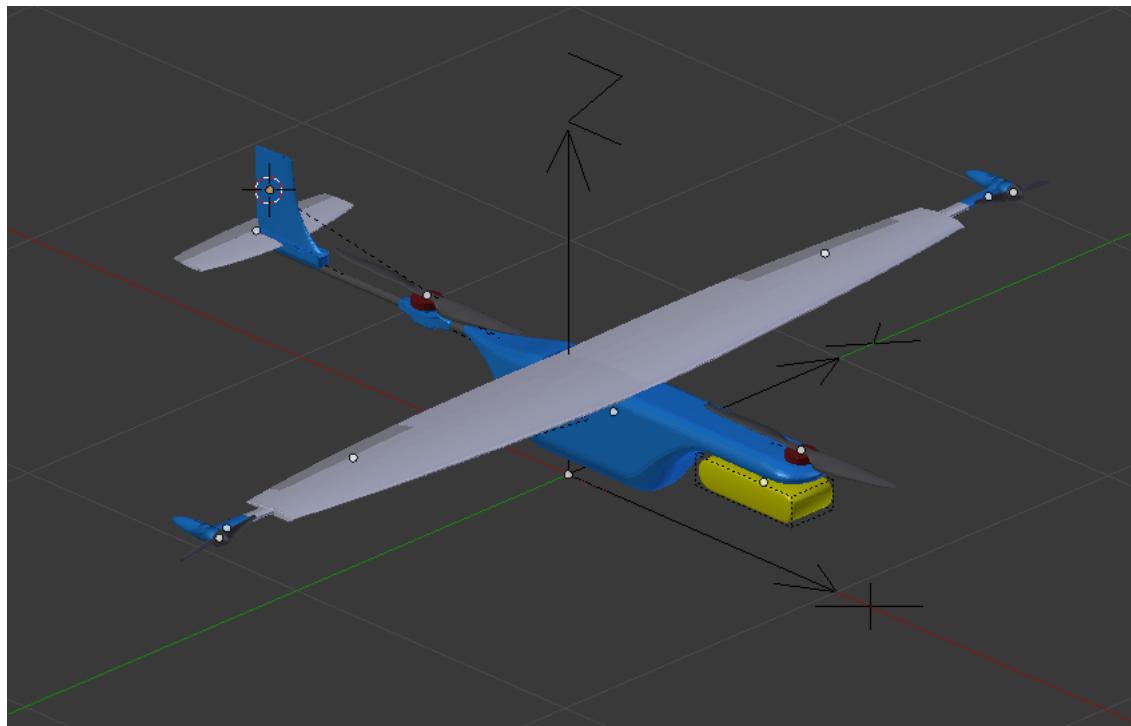


FIGURE 2.6 – Repère stork

Le centre du repère local du stork est le centre de gravité G .

- \vec{x}_{stork} : Colinéaire au fuselage et dirigé vers le nez de l'avion.
- \vec{y}_{stork} : Dirigé selon l'aile gauche.
- \vec{z}_{stork} : Dirigé vers le haut en assurant un repère direct.

2.1.4 Simulation des forces et moments :

Trois forces sont mises en jeu, dont les coordonnées selon le repère aérodynamique sont :

- Portance : $\vec{L} (0, 0, -L)_a$
- Traînée : $\vec{D} (\text{sign}(V_x) D, 0, 0)_a$
- Force latérale : $\vec{F}_{lat} (0, F_{lat}, 0)_a$

La traînée, ou plus généralement la force axiale, dépend du sens de la vitesse axiale de l'objet. Pour appliquer ces forces au stork les transformations de repère suivante sont effectuées :

$$\begin{aligned}\vec{L}_{stork} &= -(\sin(\alpha) \vec{L}_a + \cos(\alpha) \vec{D}_a) \\ \vec{D}_{stork} &= \cos(\alpha) \vec{D}_a - \sin(\alpha) \vec{L}_a \\ (\vec{F}_{lat})_{stork} &= -(\vec{F}_{lat})_a\end{aligned}$$

Les mêmes transformations sont appliquées au moment de roulis, tangage et de lacet :

$$\begin{aligned}(\vec{\mathcal{M}}_{roll})_{stork} &= \cos(\alpha) (\vec{\mathcal{M}}_{roll})_a - \sin(\alpha) (\vec{\mathcal{M}}_{yaw})_a \\ (\vec{\mathcal{M}}_{pitch})_{stork} &= -(\vec{\mathcal{M}}_{pitch})_a \\ (\vec{\mathcal{M}}_{yaw})_{stork} &= -(\cos(\alpha) (\vec{\mathcal{M}}_{yaw})_a + \sin(\alpha) (\vec{\mathcal{M}}_{roll})_a)\end{aligned}$$

Chapitre 3

Simulation du Cardan

3.1 Présentation

Le Stork est équipé d'une caméra pour prendre des photos de la cible inspectée. Le pointage de cette caméra est effectué par un cardan (ou gimbal en anglais). Le cardan utilisé sur le drone est commandé selon trois axes. Du point de vue mission, le cardan a deux modes de fonctionnement : commande en angle ou vers une cible. Dans le premier mode, deux angles sont donnés dans un repère NED (North, East, Down). Dans le second mode, la mission donne les coordonnées de l'objectif au contrôleur qui calcule les angles à appliquer en fonction de la position du Stork. Cette opération est effectuée par l'autopilote. Ainsi, les commandes du cardan ne sont que les deux angles du repère NED. L'actuateur calcule les angles à appliquer sur chacun de ces axes avec les données d'une centrale inertielle et d'un GPS indépendants.



FIGURE 3.1 – Exemple de cardan.

L'objectif de l'actuateur simulé est de reproduire les comportements observés avec un jeu de paramètres simples à régler. Les comportements à reproduire sont :

- Le pointage de la caméra vers la direction commandée dans le repère NED.
- Un retard sur le suivi de la commande et dans la correction des mouvements du drone.
- Un dépassement visible sur une commande échelon.
- Une limitation du nombre de tours possibles sur chaque axe.
- Une limitation en vitesse angulaire sur chaque axe.

Dans la première version du simulateur, seul le premier comportement était reproduit. Le cardan se contentait de calculer les angles à appliquer pour pointer la caméra dans la direction

voulue et le pointage était fait de manière instantanée. Cette implémentation simple a permis de poser l'architecture du code et de résoudre le premier point difficile de cette partie : la gestion des différents repères dans Morse et dans Blender. Pour la seconde itération, la dynamique a été rajoutée. Dans un premier temps, la vitesse a été limitée ainsi que le nombre de tours. Puis, le retard et le dépassement observés en réalité ont été simulés avec un contrôle d'ordre 2. Ce contrôle est simple à mettre en œuvre et à régler et il fait apparaître ces deux comportements. C'est pourquoi il a été choisi.

3.2 Principe de fonctionnement

Pour simuler le cardan, une nouvelle classe héritant de la classe Actuator a été créée. Les instances de Actuator représentent des actuateurs et agissent donc sur les autres objets de la simulation. Celui-ci agit sur l'objet Camera qui représente la caméra du Stork. On y a accès à travers la classe Stork qui simule le drone.

Les commandes de la Gimbal sont données par deux angles dans le repère NED (North East Down). Dans la simulation, les angles commandés sont donnés dans le repère XYZ de Blender. La rotation de la caméra du Stork se fait par contre par l'application d'une matrice de rotation dans le repère local du Stork. Les repères utilisés sont définis ci-dessous. Comme on peut le voir ci-dessous, le rôle du cardan est d'aligner l'axe z du repère de la caméra avec la direction de visée donnée dans le repère Blender. Pour ce faire, les angles d'Euler correspondant à cette direction sont calculés pour former une matrice de rotation qui est appliquée à la caméra dans le repère du Stork.

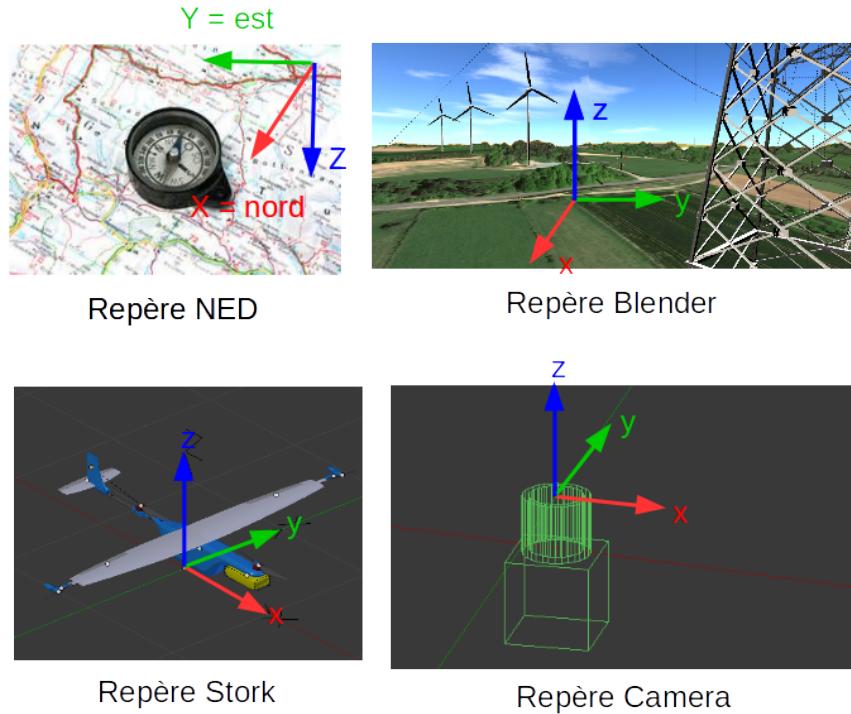


FIGURE 3.2 – Les différents repères mis en jeu dans le contrôle du cardan.

3.3 Contrôle

Le contrôle du cardan suit les étapes suivantes :

1. **Lecture des commandes.** A chaque itération, les commandes dans le repère Blender sont actualisées.
2. **Calcul des nouveaux angles.** Il faut commencer par changer les commandes de repère. On obtient alors les commandes comme des angles d'Euler dans le repère local de la caméra.

L'application de la matrice de rotation correspondante à ces angles d'Euler placera la caméra dans la bonne direction. Ensuite, les commandes sont corrigées pour être atteignables : si le cardan ne peut pas respecter la commande à cause des limitations d'angle, alors on ajoute ou on enlève 2π pour obtenir une commande acceptable. Le filtre du second ordre est ensuite appliqué sur ces commandes pour calculer les nouveaux angles à appliquer.

3. **Limitation des angles et des vitesses sur chaque axe.** Les nouveaux angles ne respectent peut-être pas les limitations de vitesse et d'angle. Cette étape modifie les angles appliqués pour prendre en compte ces contraintes.
4. **Application des commandes.** La nouvelle rotation est ensuite appliquée à la caméra.

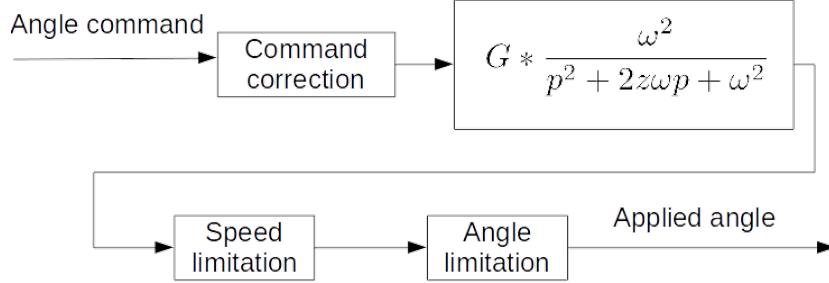


FIGURE 3.3 – Calcul des angles appliqués à la caméra à partir des commandes dans le repère Stork.

Le contrôle d'ordre 2 est discrétisé avec l'approximation de Tucson afin d'être appliqué dans la simulation. On obtient alors un filtre du second ordre de la forme suivante :

$$a_2 y_n + a_1 y_{n-1} + a_0 y_{n-2} = b_2 c_n + b_1 c_{n-1} + b_0 c_{n-2}$$

Avec y l'angle calculé autour d'un axe correspondant à la commande c . Chacun des axes est indépendant. Les coefficients peuvent être calculés en fonction du gain G , de la pulsation ω , de l'amortissement z et du pas d'échantillonnage t_s .

$$\begin{aligned} a_2 &= 1 + z \omega t_s + \frac{t_s^2 \omega^2}{4} \\ a_1 &= \frac{t_s^2 \omega^2}{2} - 2 \\ a_0 &= 1 - z \omega t_s + \frac{t_s^2 \omega^2}{4} \\ b_2 &= \frac{G t_s^2 \omega^2}{4} \\ b_1 &= \frac{G t_s^2 \omega^2}{2} \\ b_0 &= \frac{G t_s^2 \omega^2}{4} \end{aligned}$$

Ce modèle a été vérifié sur des entrées en échelon. On vérifie alors qu'il est très proche du système d'ordre 2 continu discrétisé.

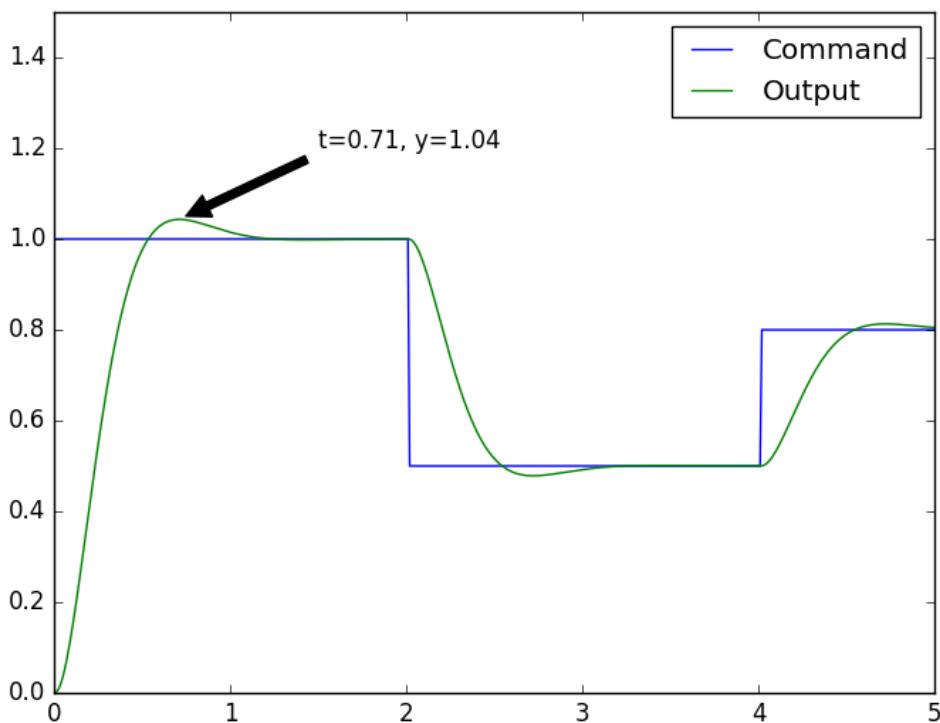


FIGURE 3.4 – Réponse du second ordre discréte à différents échelons. Paramètres : $G = 1$, $z = \frac{\sqrt{2}}{2}$, $\omega = 2\pi$. Le second ordre continu doit avoir un dépassement de 4% obtenue pour $t = \frac{\sqrt{2}}{2} = 0.7$. Les valeurs du second ordre discréte sont très proches.

3.4 Evaluation des résultats

Un module a été codé pour récupérer les commandes et les angles dans le repère Stork. Nous pouvons ainsi tracer et analyser les performances du cardan. On rappelle les comportements qui sont désirés : pointage correct, retard dans le suivi, dépassement, limitation de la vitesse et des angles. Trois scénarios de test unitaires ont été réalisés :

1. Le drone est immobile et on commande au cardan de pointer la caméra vers les axes X, Y puis Z de Blender. La caméra pointe ensuite vers le haut de l'éolienne. On vérifie que la caméra pointe bien dans la bonne direction dans l'interface de simulation.
2. La caméra pointe dans la direction de l'axe X du repère Blender. Le Stork a un mouvement de rotation autour de l'axe Z du repère Blender (rotation horizontale). Ce test permet de vérifier le comportement du cardan lorsqu'il atteint les angles limites. On vérifie aussi qu'il n'y a pas de singularités liées aux ambiguïtés des angles d'Euler.
3. Le Stork fait un mouvement non contrôlé (il chute de sa plate-forme). On vérifie alors les points voulus sur tous les axes.

Ces scénarios ont été réalisés avec les paramètres suivants :

- **Axe de roulis.** Nombre de tours maximal : 1. Vitesse maximale : 90 degrés/secondes.
- **Axe de tangage.** Nombre de tours maximal : 1. Vitesse maximale : 100 degrés/secondes.
- **Axe de lacet.** Nombre de tours maximal : 1. Vitesse maximale : 110 degrés/secondes.
- **Loi de contrôle.** Gain : 1. Amortissement : $\sqrt{2}/2$. Pulsation : 10 rad/secondes.

3.4.1 Scénario 1

Dans ce scénario, le Stork est immobile sur sa plateforme et la caméra doit pointer en direction des axes X, Y puis Z de Blender. La caméra est ensuite pointée vers le haut de l'éolienne. On vérifie

que les pointages sont corrects dans l'interface graphique de la simulation. Sur la figure de gauche, on observe que la caméra pointe bien vers le haut de l'éolienne. La vue de la caméra est en haut à droite de l'image. Les comportements du cardan sont bien ceux attendus (limitation en vitesse, retard et dépassement sur les commandes en échelon). On observe une légère oscillation sur l'axe de tangage qui est compensée avec un retard par la commande.

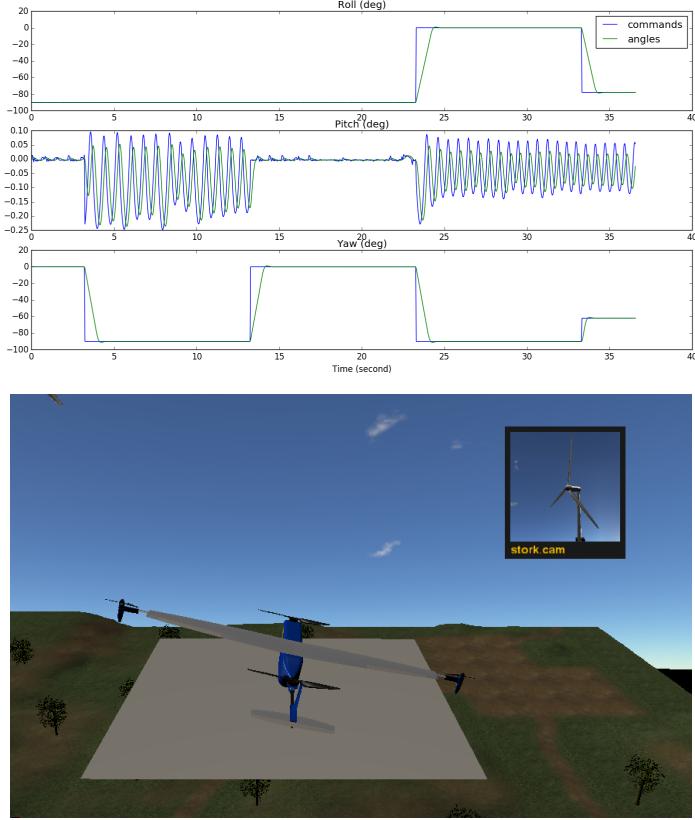


FIGURE 3.5 – Haut : pointage vers quatre directions différentes. Bas : vérification visuelle du pointage correct vers le haut de l'éolienne.

3.4.2 Scénario 2

La caméra pointe vers l'axe X du repère Blender puis on applique une rotation autour de l'axe z au Stork. On observe que le cardan compense bien cette rotation avec un retard. Lorsque le cardan arrive en buté (après un tour), la commande est recalculée pour avoir un angle atteignable et la caméra tourne dans l'autre sens pour repointer dans la bonne direction.

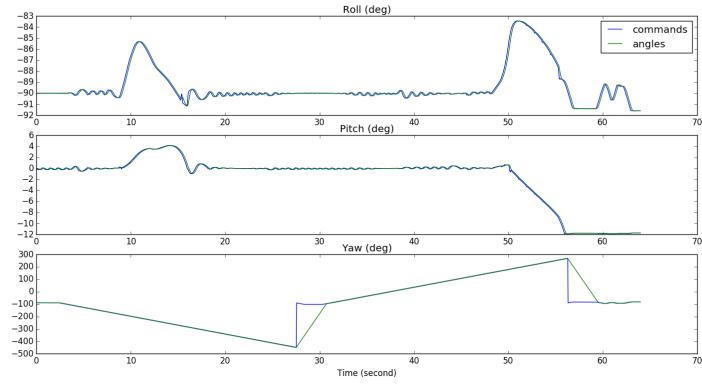


FIGURE 3.6 – Essai avec une rotation du Stork. Au temps 28s et 56s, le cardan arrive en buté sur l'axe de lacet. De nouvelles commandes sont recalculées pour atteindre l'objectif.

3.4.3 Scénario 3

Dans ce scénario, le Stork tombe de sa plate-forme. Les commandes sont donc plus complexes. On vérifie bien les comportements désirés et que la caméra pointe bien dans la bonne direction après la chute.

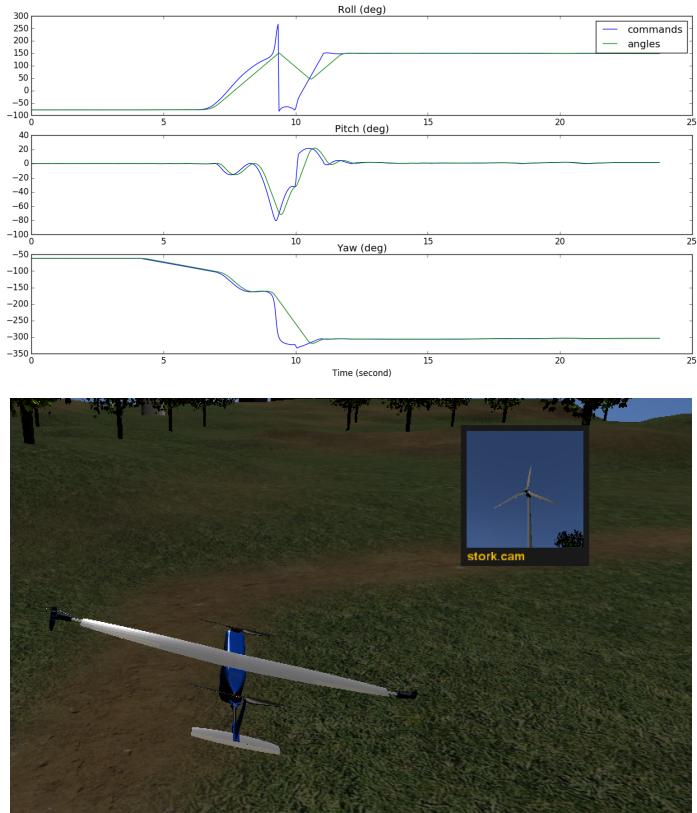


FIGURE 3.7 – Haut : pointage vers le haut de l'éolienne pendant une chute. Bas : vérification visuelle du pointage correct vers le haut de l'éolienne à la fin de la chute

3.4.4 Conclusion

L'ensemble des comportements identifiés pour la fonction de pointage de la caméra sont réalisés par un système simple et paramétrable. Cela permettra à notre de trouver les bons paramètres pour

simuler correctement le comportement réel du cardan. Une différence importante doit néanmoins être mentionnée : la trajectoire de la direction de visée n'est pas conforme à celle observée en réalité. Avec le vrai cardan, des coordonnées sphériques sont utilisés et le chemin le plus court entre deux direction est suivi. Au contraire, dans cette implémentation du cardan, la dynamique est contrôlé sur chacun des angles d'Euler séparément. Cela ne pose pas de problèmes pour l'atteinte de l'objectif car la caméra n'est utilisée qu'une fois la direction stabilisée. La trajectoire entre deux directions n'est donc pas un élément important. Ce point a été soulevé avec notre client qui nous a précisé que les paramètres importants sont le retard dans le pointage et un dépassement transitoire en cas de changement de direction important. Ces deux comportements sont simulés et sont réglables avec les différents paramètres de l'actuateur.

Chapitre 4

Simulation de la motorisation

4.1 Presentation du Stork

Le Stork est un drone atypique, il doit répondre à un cahier des charges très exigeant, en terme de manœuvrabilité afin d'être capable de détecter avec précision des anomalies sur des éoliennes, ainsi qu'en terme d'endurance pour pouvoir voler jusqu'à l'éolienne ciblé en autonomie depuis la base. Pour satisfaire ces exigences, le Stork est capable de deux modes de vol différents, un mode stationnaire de type quadri-rotor, et un mode à vitesse plus élevée de type voilure fixe ou avion. Pour se faire, le Stork dispose de quatre moteurs, deux longitudinaux, c'est-à-dire dans l'axe du fuselage, et deux latéraux aux extrémités des voilures. Les deux moteurs latéraux sont orientables, afin de pouvoir s'adapter au mode de vol.

En effet, en mode stationnaire, les quatre moteurs sont en marche, de manière contra-rotative comme un quadri-rotor, et leurs axes de rotations sont selon la verticale du Stork. En mode avion, les deux rotors longitudinaux sont fixe dans l'axe du fuselage, et les moteurs latéraux s'orientent selon l'horizontale du Stork, vers l'avant afin de jouer le rôle de turbopropulseur.

La figure suivante illustre les positions des différents moteurs sur le Stork.

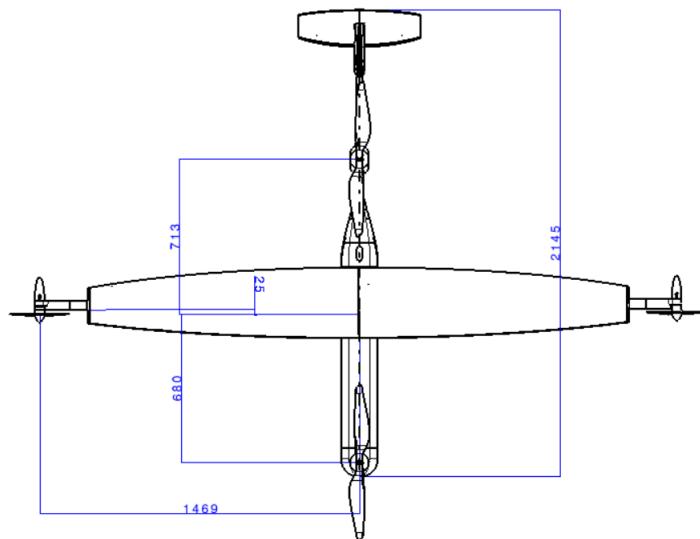


FIGURE 4.1 – Plan du Stork

4.2 Simulation des moteurs

Chaque moteur a été simulé comme une Force et un Moment, appliqués au centre de gravité du Stork.

4.2.1 Moteurs Longitudineaux

Les moteurs longitudinaux ne fonctionnent qu'en mode stationnaire, ils ont une puissance de 250W et une poussée unitaire de 30N. En utilisant ces paramètres, ainsi que la distance des moteurs au centre de gravité, et la vitesse de rotation des hélices, nous avons pu déterminer la force, et le moment à appliquer au centre de gravité du Stork, en fonction de la commande du contrôleur.

De plus, nous avons utilisé la vitesse de rotation des hélices afin d'animer les moteurs visuellement dans la simulation, dans le but de rendre la simulation visuellement plus réaliste.

4.2.2 Moteurs Latéraux

Les moteurs latéraux sont implémentés de la même manière que les moteurs longitudinaux, à l'exception près qu'ils sont orientables. Ils ont une puissance de 250W et une poussée unitaire de 10N en mode stationnaire, et une puissance de 125W pour une poussée unitaire de 5N en mode avion. Nous avons donc défini un angle d'orientation du moteur, que nous avons utilisé pour assurer la transition entre les deux modes. Nous avons fait passer la force et le moment d'un moteur latéral dans une matrice de rotation d'angle l'angle d'orientation du moteur, afin que la force et le moment appliqués au centre de gravité suivent l'orientation du moteur.

4.3 Consommation de la batterie

Le Stork dispose d'une batterie de 17600 mAh. Avec la puissance des moteurs en W, nous avons pu déterminer un taux de décharge de la batterie en mA/s en fonction du domaine de vol, c'est à dire du mode de vol et de la vitesse du Stork. Ainsi au début de la simulation, la batterie est complètement chargée. Elle se décharge à mesure que le drone vol, avec un taux adapté à la commande en poussée des moteurs et au mode de vol. Lorsque la batterie est complètement déchargée, on force les commandes des moteurs à zéro, afin de provoquer la chute libre de Stork dans la simulation.

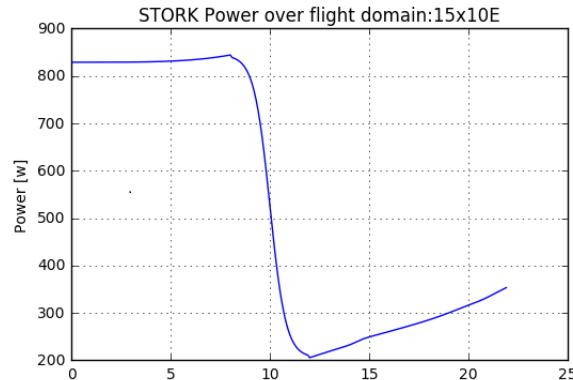


FIGURE 4.2 – Puissance demandée pour les différents modes de vol

Dans le mode stationnaire, nous avons implémenté deux taux possibles de décharge qui correspondent à une puissance demandée respectivement de 850W et 950W des moteurs en fonction de leurs commandes. En mode avion, nous avons approximé le taux de décharge de la batterie par une fonction affine, correspondant à la pente visible sur la figure précédente pour les vitesses supérieures à 12m/s.

Chapitre 5

Autopilote et interfaçage PX4

5.1 Première approche : Implantation d'un Controller

La première approche consistait à implémenter un contrôle d'attitude du stork de sorte qu'en fonction de la destination (Waypoint) à suivre. Un Waypoint classique tel qu'il était implémenté par défaut pour un quadrotor se contenter de contrôler le drone en vitesse sans tenir compte des puissances moteurs et de l'aérodynamique.

Pour la première itération, nous avons donc imposé un *controller* basé sur le contrôle d'attitude du Stork en mode quadrotor, avec possibilité de modifier l'inclinaison des moteurs latéraux à une certaine vitesse. Le *main* du *controller* fait appel à un PID :

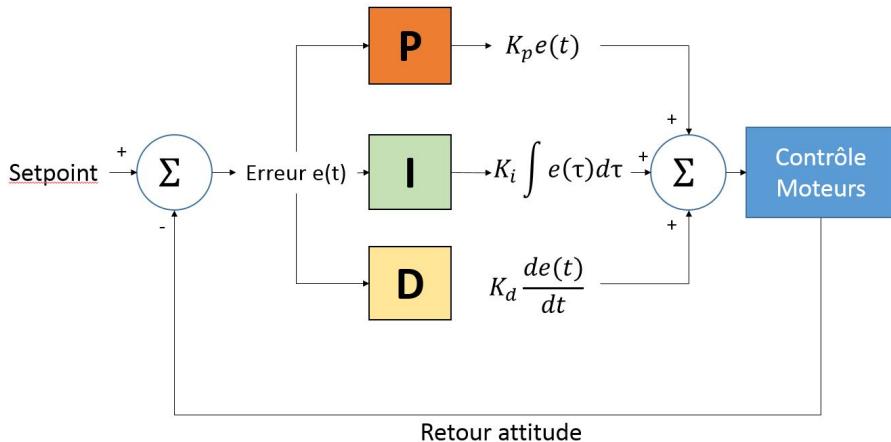


FIGURE 5.1 – Boucle de contrôle PID, dans le bloc *Contrôle Moteurs* le calcul des puissances moteurs

Le PID classique calcule la force et le couple à appliquer sur le drone afin de maintenir son attitude. Au lieu de directement appliquer ces dernières à l'objet *Stork*, nous déterminons la puissance des moteurs nécessaire pour parvenir au même résultat en :

- Calculant les rapports de poussée entre les moteurs pour obtenir le couple nécessaire calculé (en prenant en compte les dimensions du drone)
- Répartissant la poussée nécessaire calculée sur les quatre moteurs

De même que le gimbal, les coordonnées du waypoint ainsi que le contrôle de la puissance des moteurs se fait via la bibliothèque *commands* écrite dans *stork*. Les puissances sont des variables réelles allant de 0 à 1, représentant la proportion par rapport à la puissance maximale des moteurs.

Malheureusement, cette démarche s'est avérée peu efficace dans le cadre d'un drone type VTOL comme le Stork : une fois lancé, le drone avait tendance à faire de grands loopings en tentant de se diriger vers son objectif. Le contrôle d'attitude en tant que "quadrotor" couplé à la portance de la voilure aurait demandé beaucoup de paramétrage et plusieurs boucles de contrôle différentes en fonction des cas d'utilisation.

5.2 Seconde approche : Interfaçage avec l'autopilote PX4

En réponse à cette difficulté, il a été décidé pour la deuxième itération du projet de repasser sur la configuration pensée initialement, c'est à dire d'effectuer une simulation de type SITL sur morse via un autopilote PX4. Ce genre de simulation a pour but de modéliser aussi fidèlement que possible le comportement d'un autopilote. Le programme faisant tourner ce dernier est appelé Firmware.

Lorqu'exécuté, le Firmware attendra de recevoir les mesures des capteurs du drone sous forme de messages MAVLink, et en fonction de la commande envoyée par l'utilisateur, calcule les signaux à envoyer aux actuateurs, sous forme de messages MAVLink également. Ce message est interprété par le simulateur qui met à jour les variables d'état de l'aéronef, puis celui-ci renvoie les nouvelles mesures de capteurs du drone, signalant par la même qu'une nouvelle boucle de simulation peut être lancée.

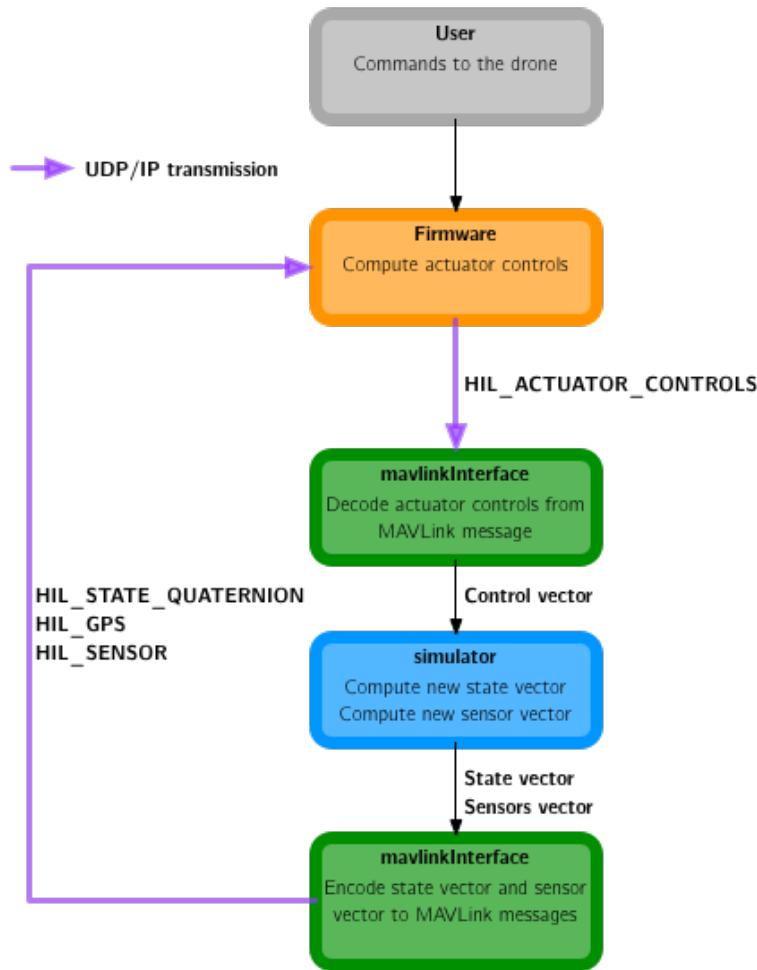


FIGURE 5.2 – Modèle de contrôle SITL

Morse ne prenant pas en charge les messages MAVLink, un script de traduction nommé *mavlinkInterface* nous a été fourni. Notre travail est donc d'interfacer Morse avec le Firmware. Cela comprend l'interprétation des vecteurs de contrôle et le renvoi des valeurs de capteurs, mais également la synchronicité des deux entités afin que la boucle d'ensemble fonctionne à une seule et même fréquence. Un script *simulator* a été fourni, simulant un déplacement vertical du drone en réponse aux ordres donnés par l'utilisateur via l'autopilote.

La solution retenue pour l'interfaçage a été d'ajouter au stork un actuateur *sitlReader* et un capteur *sitlSender* :

- sitlReader bloque la simulation jusqu'à ce qu'a ce que la réception d'un message MAVLink soit détectée au niveau du simulateur.
- sitlSender attend la fin de la simulation (calcul des variables d'état) et donne le signal autorisant à envoyer les nouvelles variables au Firmware.

Cette solution permet à la fois l'interfaçage et la synchronisation de la boucle de simulation, étant donné qu'une simulation ne peut être lancée que sous réception d'un message MAVLink et qu'il en est de même pour le Firmware.

Conclusion