Projet Ingénierie et Entrepreneuriat

SA 006

Rapport de projet

Téléopération intelligente pour micro-drones complexes

Auteurs :
Andrea Brugnoli
Diane Bury
Alexis Nicolin
Paolo Panicucci
Steve Ravalisse
Bertrand Sudérie

Encadrants :
Matthieu Geisert
Fabrice Martinez
Steve Tonneau

Version du
23 mars 2017

Préface

Ce rapport représente le travail réalisé dans le cadre du **Projet Entrepreneuriat et Ingénierie** portant sur la *Téléopération Intelligente pour Micro-drone Complexe*. Ce document décrira les aspects de gestion de projet ainsi que les réalisations techniques du projet. Une documentation technique est également disponible séparément [1].

Abstract — Ce document détaille nos réalisations dans le cadre de notre **Projet Entrepreneuriat et Ingénierie** portant sur la *Téléopération Intelligente pour Micro-drone Complexe*. Ce projet, mandaté par le laboratoire du CNRS-LAAS et confié à une équipe d'étudiants de l'ISAE-Supaéro, a été réalisé avec l'aide de deux tuteurs du LAAS et d'un tuteur en gestion de projet (Airbus). Ce rapport donne la description du projet ainsi qu'une analyse de gestion de projet et un planning du travail de l'équipe. Les différentes tâches réalisées au cours du projet seront présentées, ainsi que les résultats obtenus, accompagnés des instructions nécessaires à l'installation et à l'utilisation du simulateur.

Mots clés : gestion de projet, entrepreneuriat, drone, téléopération, contrôle optimal, LAAS-CNRS

Table des matières

Table des figures

Liste des sigles et acronymes

ROS	*Robot Operating System*
WBS	*Work Breakdown Structure*
PIE	*Projet Ingénierie et d'Entreprise*

Chapitre 1

Présentation du Projet

Le projet proposé consiste à améliorer un système de téléopération à contrôle optimal ainsi que sa simulation. Ceci s'inscrit dans la continuité d'un projet réalisé l'année précédente (2016) [2], qui avait abouti sur le code du système de téléopération existant. Ce code C++ utilisait l'outil de simulation Geppetto, développé par le LAAS.

Le principe de la téléopération intelligente est de pouvoir commander un drone volant (avec une manette ou le clavier) qui évolue dans un espace contenant des obstacles (par exemple, des piliers). Le contrôle optimal doit permettre à l'utilisateur de piloter le drone à l'aide de commandes simples (avant, arrière, gauche, droite, monter, descendre...), ce tout en évitant les obstacles. Le contrôle optimal est un système prenant en entrée des commandes en vitesses linéaires (ainsi que l'état du drone et l'environnement) et calculant les commandes moteurs à donner au drone pour effectuer le mouvement demandé. Nous considérons l'environnement connu a priori, c'est-à-dire que le contrôle optimal ne reposera pas sur des données issues de capteurs pour déterminer les positions des obstacles.

La motivation derrière ce nouveau projet était double :

— améliorer le modèle de contrôle optimal : raffiner le modèle du drone, son estimateur, implémenter de nouveaux types d'obstacles...

— améliorer la qualité du code : le code du projet de l'année précédente consistait en effet en un bloc fonctionnel mais peu portable ou modulable ; une refonte du code semblait alors nécessaire.

Chapitre 2

Gestion de projet

2.1 Analyse du projet

Il est nécessaire de noter que notre projet ne débute pas de zéro, mais s'inscrit au contraire en continuité du travail effectué l'année précédente par un autre groupe. Le programme livré doit permettre à un utilisateur sans expérience partculière en pilotage de drone de pouvoir manœuvrer le drone à l'aide de commandes simples.

2.1.1 Exigences et spécifications

Les exigences du projet ont été définies en accord avec les donneurs d'ordre du LAAS et après délibération au sein de l'équipe. Les débats se sont notamment attachés à estimer le travail réalisable compte tenu des ressources disponibles et du temps imparti.

1. Système général

a. Le système doit être implémenté en utilisant un logiciel de gestion de versions collaboratif distribué.

b. Les différentes parties du logiciel du système doivent être séparées du point de vue du code et placées chacune sur une branche différente au sein du gestionnaire de versions.

c. Le merging d'une branche de développement avec la branche principale devra être approuvée par un membre non impliqué dans le développement de cette branche.

d. Le système doit être fourni avec son code source et une documentation technique

2. Documentation technique

a. La documentation doit détailler l'installation du logiciel.

b. La documentation doit détailler les librairies externes utilisées avec le numéro de la version nécessitée.

c. La documentation doit contenir les instructions d'utilisation du logiciel.

3. Simulateur

a. Le simulateur doit opérer sous un environnement Linux.

b. Le simulateur doit s'accompagner d'un visualisateur temps réel.

c. Le simulateur doit simuler un drone à hélices commandable.

d. Le simulateur doit simuler le drone dans le cas nominal (i.e. sans panne).

e. Le simulateur doit utiliser un modèle réaliste du drone (propulsion...) et de l'environnement (aérodynamique...).

f. Le simulateur doit simuler des obstacles.

g. Le simulateur doit comporter un modèle aérodynamique intégrant un modèle de vent.

h. Le simulateur doit utiliser le contrôle optimal.

4. Contrôle optimal

a. Le contrôle optimal doit assister en temps réel la téléopération du drone par l'utilisateur.

b. Le contrôle optimal doit s'accompagner d'un module d'interface utilisateur.

c. Le contrôle optimal doit transformer des commandes simples provenant de l'interface utilisateur en commandes d'entrée du drone.

d. Le contrôle optimal doit empêcher le drone d'entrer en collision avec les objets de l'environnement simulé.

e. Le contrôle optimal devra pouvoir s'exécuter sur un PC de bureau équipé d'un processeur i7 de de 3ème génération ou plus récent, à une fréquence de 10Hz.

5. Interface utilisateur

a. L'interface utilisateur doit accepter des commandes utilisateurs provenant d'un clavier d'ordinateur ou d'une manette, ce en temps réel.

2.1.2 Périmètre du projet

Le projet a pour objectif de se rapprocher d'une utilisation du contrôle optimal par un drone réel dans un environnement réel grâce à la modularité du logiciel et de ses composants, ce que ne permettait pas le projet de l'année précédente dans son état final. Cela implique que des améliorations doivent être faites sur cette base.

Par ailleurs, les limites suivantes doivent être énoncées :

— L'environnement est supposé connu du drone indépendamment de ses capteurs.
— L'environnement est composé de cylindres et d'ellipses ; des objets plus variées seront approximés par des ellipses
— Les paramètres du drones sont supposés connus et utilisés pour régler le contrôle optimal
— Même si les modules ROS permettent de porter facilement le contrôle optimal à un drone réel, les limitations des calculateurs embarqués ne permette pas actuellement de faire tourner le contrôle optimal sur un drone

2.2 Livrables du projet

Les livrables du projet sont les suivants :

— Un **rapport de gestion de projet**, qui doit détailler notre plan de développement, notre analyse du projet ainsi que nos méthodes de travail, et être mis à jour au cours du projet.
— Le **rapport final du projet**, qui synthétise aussi bien notre gestion du projet, nos réalisations technique et les résultats obtenus. Il rend également compte des difficultés que nous avons rencontrées et de la manière dont nous avons surmonté ces problèmes.
— Le **système** sur lequel porte le projet (composé de son code source) : un logiciel (donné sous forme de code source) qui permet de faire une téléopération intelligente d'un micro-drone avec une loi de commande à contrôle optimal.
— Une **documentation technique** servant de manuel d'installation et d'utilisation du logiciel.

2.3 Répartition des tâches

A travers l'analyse des tâches à réaliser, nous avons pu établir l'organisation générale de notre projet. Notre projet a évolué au cours de sa réalisation, et certaine tâches ont été

supprimées, modifiées ou ajoutées depuis la plannification initiale. Dans le tableau suivant sont récapitulés les tâches et leurs charges de travail prévisionnelles telles qu'ellles étaient après une premièe actualisation.

TABLE 2.1 – Charges de travail globales du projet

	Tâches	**Durée**
	Réalisation du projet	**650h**
1.	Gestion de projet	180h
2.	Migration du code vers ROS	120h
3.	Amélioration du modèle du contrôle optimal	160h
4.	Amélioration de la gestion de l'environnement	80h
5.	Intégration et tests	110h
	Marge (10%)	**65h**
	Charge de travail totale	**715h**
	Charge de travail par personne	**120h**

Nous avons détaillé chaque tâche en sous-tâches, auxquelles nous avons assigné des estimations de charge de travail. Nous disposions au sein du projet de 80 heures de travail par personne (créneaux horaires bloqués pour le projet) et nous estimions le travail en dehors de ces horaires entre 1h30 et 2h par semaine et par personne. Nous disposions donc au total d'environ 675h de travail (environ 112h par personne), soit 96 jours-hommes.

Ci-dessous le détail des tâches et sous-tâches :

1. Gestion de projet
   a. Analyse du projet
   b. Analyse de la charge de travail
      — Définition du WBS
      — Définition des charges de travail par partie
   c. Analyse des risques
      — Analyse des risques

   - — Définition du plan d'action
  d. Définition des méthodes de travail
   - — Définition des méthodes de communication et de partage de fichiers
   - — Définition des conventions de travail
  e. Définition du planning
  f. Gestion des livrables
2. Migration du code sous ROS
  a. Définition de l'architecture logicielle
  b. Implémentation des noeuds
  c. Choix et intégration d'un modèle de drone sous ROS
  d. Mise en place des communications entre noeuds
   - — Mise en place des topics
   - — Mise en place des services
  e. Intégration dans le simulateur et tests
   - — Ecriture de launch files
   - — Test du modèle de drone
3. Amélioration du modèle du contrôle optimal
  a. Prise en main de la problématique
   - — Etude bibliographique
   - — Familiarisation avec ACADO
  b. Création d'un modèle dynamique complet du drone
   - — Mise en relation avec le modèle du drone
   - — Ecriture du modèle dynamique et simulation
   - — Ajout d'éléments de différenciation simulation/analyse
  c. Ajout d'un estimateur d'état
   - — Choix du système de mesure de position
   - — Choix des paramètres et états à estimer
   - — Evaluation des performances
4. Amélioration de la gestion de l'environnement
  a. Choix et définition des obstacles à modéliser
  b. Intégration des obstacles dans le simulateur
  c. Intégration des obstacles dans le contrôle optimal
  d. Implémentation d'une librairie de gestion de l'environnement

e. Implémentation de l'approximation d'obstacles complexes par des ellipses
   — Etude bibliographique et choix d'un algorithme
   — Implémentation de l'algorithme

5. Intégration et tests
   a. Intégration progressive du contrôle optimal dans le simulateur
   b. Tests sans obstacles
   c. Tests avec obstacles
   d. Acquisition de résultats
   e. Intégration progressive du contrôle optimal dans le simulateur
      — Acquisition de données de trajectoires
      — Analyse des données de trajectoires

2.4 Présentation de l'équipe et des règles de travail

2.4.1 Présentation de l'équipe

Le projet a été conduit par 6 étudiants de 3ème année à l'ISAE-SUPAERO, dont les différentes spécialisations ont mis à disposition des compétences multiples et transverses.

2.4.2 Règles de travail et conventions

Répartition du travail

En plus de devoir être relue et révisée par plusieurs personnes, chaque tâche nécessite plusieurs compétences transverses pour être menée à bien et donc souvent plusieurs personnes assignées dessus. Nous avons réparti les membres de l'équipe au sein de plusieurs tâches, la tâche en gras étant la tâche principal de chacun-e. Il est également conseillé à tous les membres de l'équipe de demander de l'aide aux autres membres quand cela est nécessaire afin d'éviter un blocage inutile.

— Andrea Brugnoli : **contrôle optimal**
— Diane Bury : **code ROS**, gestion de l'environnement
— Alexis Nicolin : **code ROS**, contrôle optimal, gestion de l'environnement
— Paolo Panicucci : **contrôle optimal**
— Bertrand Sudérie : **code ROS** (partie interface utilisateur et intégration du modèle)
— Steve Ravalisse : **gestion de l'environnement**, code ROS

Chaque membre de l'équipe posédait également une charge de travail de gestion de projet et d'intégration et de tests, et de documentation du code.

TABLE 2.2

Nom	Domaine d'application	Filière d'expertise
Andrea BRUGNOLI	Modélisation des systèmes complexes (SXS)	Signaux et systèmes (SISY) – Parcours : Automatique avancée + Automatique appliquée
Diane BURY **Chef de projet**	Systèmes autonomes (SA) – Parcours : Robots	Informatique, télécommunications et réseaux (ITR) – Parcours : Informatique
Alexis NICOLIN	Systèmes autonomes (SA) – Parcours : Robots	Informatique, télécommunications et réseaux (ITR) – Parcours : Informatique
Paolo PANICUCCI	Modélisation des systèmes complexes (SXS)	Signaux et systèmes (SISY) – Parcours : Automatique avancée + Automatique appliquée
Steve RAVALISSE	Systèmes autonomes (SA) – Parcours : Robots	Signaux et systèmes (SISY) – Parcours : Circuits et antennes + Traitement d'image
Bertrand SUDÉRIE	Systèmes autonomes (SA) – Parcours : Drones	Informatique, télécommunications et réseaux (ITR) – Parcours : Informatique

Moyens de communications

Nous avons mis en place un certain nombre de moyens de communication afin d'assurer un contact permanent entre les membres de l'équipe et les encadrants.

Un groupe WhatsApp permet aux membres de l'équipe de convenir rapidement de rendez-vous.

Un compte Slack a été créé pour le projet. Il permet aux membres de l'équipe d'échanger des informations relatives au projet via plusieurs fils thématiques. Les donneurs d'ordres du CNRS-LAAS sont notamment abonnés à un fil particulier qui permet de poster des comptes-rendus et éventuellement des questions. Ce procédé nous fait gagner en efficacité par rapport au mail et permet de centraliser notre travail.

Bien qu'un dépôt EduForge soit disponible et contienne le travail effectué sur ce projet l'an dernier, nous avons décidé d'utiliser un dépôt Git en tant que gestionnaire de version pour notre code. Ce dernier est en effet plus flexible et pratique que l'EduForge mis à disposition par l'ISAE.

Conventions

Afin d'homogénéiser notre travail, notre code et nos documents, nous avons choisi certaines conventions que vont devoir suivre tous les membres de l'équipe.

Documents écrits

— Les rapports livrables seront rédigés en français
— Les rapports seront rédigés en utilisant LaTeX
— La documentation du code sera écrite en anglais

Code

— Le code devra être correctement commenté
— Le code devra être commenté en anglais
— Le code devra être correctement indenté (par incrément d'une tabulation) de manière à faire ressortir clairement sa structure

Communication

— L'équipe doit utiliser un SLACK avec différentes channels pour la communication entre membres de l'équipes ainsi qu'avec les encadrants du LAAS.
— L'équipe peut éventuellement échanger rapidement des informations sur le groupe WhatsApp. Cependant, le Slack doit être privilégié.

Gestion de projet

— L'équipe se réunira au moins une fois par semaine dans le créneau dédié
— A chaque réunion, chaque membre fera un compte-rendu de son travail depuis la dernière réunion
— L'équipe utilisera le repo Git existant du projet pour la gestion de version
— Chaque membre de l'équipe notera ses heures de travail sur chaque tâches et les partagera avec l'équipe sur le SLACK

2.5 Analyse des risques et des plan d'action

2.5.1 Analyse des risques

Dans cette partie nous allons énumérer les risques auxquels nous pourrions être confrontés pendant le projet et établir des plans d'action / de prévention pour limiter les impacts si ces risques venaient à se concrétiser.

Nous avons listé les principaux risques puis établi un plan d'action pour chacun d'eux.

Risque	Probabilité	Gravité	Criticité (C=PxG)
Problème de prise en main de ROS	30 %	3/4	23 %
Difficultés dans le choix du modèle de drone	40 %	2/4	20 %
Problème de prise en main du modèle de drone	50 %	2/4	25 %
Problème d'interfaçage du modèle de drone avec le contrôle optimal	75 %	3/4	56 %
Problèmes liés au dépendances logicielles	60 %	4/4	60 %

FIGURE 2.1 – Analyse de risques : risques principaux

2.5.2 Plan d'action

Risque	Action préventive
Problème de prise en main de ROS	Formation ROS pour tous les membres de l'équipe
Difficultés dans le choix du modèle de drone	
Problème de prise en main du modèle de drone	Choix dans la mesure du possible d'un modèle modulable - ie simplifiable
Problème d'interfaçage du modèle de drone avec le contrôle optimal	idem
Problèmes liés au dépendances logicielles	Limiter les dépendances externes Consigner les versions utilisées Sauvegarder le code source quand c'est possible

FIGURE 2.2 – Analyse de risques : mesures préventives pour les risques principaux

2.6 Planning

Un planning a été réalisé en début de projet sous forme d'un diagramme de Gantt sur Microsoft Project.

Malheureusement, il nous a été difficile de maintenir ce diagramme à jour compte tenu des grands changements d'orientation survenus en cours de projet. Nous n'avons ainsi pas été en mesure d'exploiter entièrement le potentiel de cet outil.

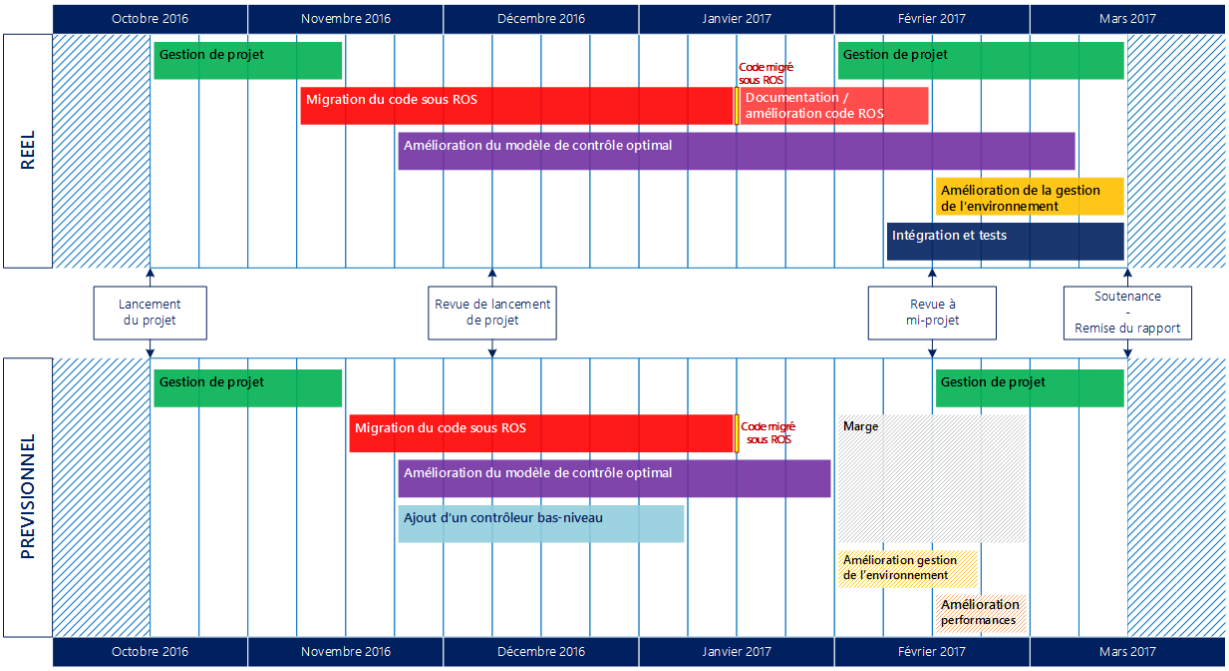

FIGURE 2.3 – Frise temporelle représentant le planning de travail

2.7 Conclusion sur la gestion de projet

Ce projet a été pour l'équipe l'occasion de découvrir les méthodes de gestion de projet. Bien que notre manque d'expérience ainsi que notre réticence initiale aient pu nous ralentir et nuire à l'efficacité de ces méthodes, il nous est néanmoins peu à peu apparu qu'elles étaient bénéfiques à la conduite du projet dans la mesure où la gestion de projet impose une constante remise en question et réévaluation de l'état du projet. Il nous est apparu également au cours du projet qu'il est important de définir la juste granularité à laquelle nous devons descendre dans les analyse de gestion de projet.

2.7.1 Bilan sur les charges de travail

Nous avions affecté un volume horaire très important à la gestion de projet (180h : 28% du projet environ). Nous pouvons dire a posteriori que nous avons dépassé la durée annoncée (le nombre total se situant autour de 220h). Ce dépassement s'explique par le fait que nous avons en parallèle fait un apprentissage et une mise en pratique de la gestion de projet : nous n'étions que très peu familiers avec les notions et nous avons pris plus de temps que prévu pour les appliquer.

La partie Amélioration du modèle du contrôle optimal a pris moins de temps que prévu (plutôt 120h que les 160h prévues), mais une partie de ces heures a été rebasculée sur la tâche 5 (Intégration et tests), qui a donc pris autour de 150h au lieu des 110h prévues. Les charges réelles pour les tâches 2 (migration du code) et 4 (amélioration de l'environnement) ont été plutôt proches des charges prévues. Cependant, après la migration complète sous ROS, nous avons continué à travailler à l'amélioration et à la documentation du code sur environ 20h.

Au total, nous chiffrons le temps travaillé à environ 710h, ce qui correspond à la charge de travail totale prévue en comptant la marge de 10%.

Chapitre 3

Architecture du code

3.1 Présentation du code existant

Le code existant consiste en un projet C++ compilé par *cmake*. Les différentes parties du logiciel (contrôle optimal, simulation, interface utilisateur) sont regroupées au sein d'un seul et même module, ce qui nuit à la modularité et rend le code difficilement modifiable et maintenable.

De plus, le code existant n'implémente pas une véritable simulation. En effet, il n'utilise pas de moteurs physique pour simuler le comportement hardware du drone (moteurs, hélice, ...) et ne comporte pas d'estimation de l'état du drone via une simulation de capteurs (le code reprend directement les valeurs exactes).

Les instructions d'installation ne permettent pas une installation rapide et sans accrocs. En effet, les versions des logiciels requis ne sont pas indiquées, ce qui crée des problèmes de compatibilité.

3.2 Migration sous ROS

3.2.1 Motivations du passage sous ROS

Les problèmes présents dans le code du projet 2016 nous ont poussés à modifier en profondeur le code, avec comme objectifs l'obtention d'un code modulaire et d'un logiciel facilement installable, ainsi que l'adoption d'un outil de simulation/visualisation plus simple à intaller et à utiliser. Nous avons donc décidé de **migrer le code sous ROS**, le Robot Operating System.

Les motivations derrière le passage à ROS sont les suivantes :

— L'existence d'une grande communauté de soutien derrière ROS
— La popularité de ROS et sa facilité d'installation
— La facilité de construire et gérer une application modulaire ainsi que la communication entre les modules
— L'existence de nombreux packages ROS déjà développés et testés qui implémentent de nombreuses fonctions utiles
— La compatibilité de ROS avec les langages de programmation les plus utilisés (dont le C++ que nous avons décidé d'utiliser pour le projet) et la facilité d'intégration du code existant dans un noeud ROS

ROS dispose d'un outil de compilation : *catkin*.

3.2.2 Architecture du code

ROS permet de créer facilement des modules (des noeuds ROS) qui s'échangent des messages ou des services. Notre programme se divise selon les modules suivants, regroupés sous un meta-package appelé *intel_teleop* :

— Noeud de calcul de **contrôle optimal**
— Noeud d'**interface utilisateur** pour la réception des commandes utilisateurs
— Noeud de **génération d'environnement**

Le simulateur/visualisateur de ROS, *Gazebo*, est un noeud indépendant qui interagit avec les noeuds de l'application. *Gazebo* charge le modèle du drone sous la forme d'un fichier *URDF* qui décrit les caractéristiques physiques du drone, accompagné de plugins qui décrivent son comportement dynamique. Le noeud d'interface utilisateur récupère les commandes (en provenance du clavier ou d'une manette) de l'utilisateur et les envoie au noeud de contrôle optimal. Ce dernier calcule alors les commandes moteurs à envoyer au drone et les envoie au noeud *Gazebo*. Le noeud de génération de l'environnement sert à créer l'environnement en le peuplant d'obstacles qui sont intégrés ensuite dans le calcul de contrôle optimal et dans le simulateur.

3.3 Choix du modèle de drone

3.3.1 Choix du package Hector Quadrotor

Nous avons décidé de chercher un package de drone déjà existant sous ROS, et avons choisi le package **Hector Quadrotor** [3]. Ce package assez complet comprend la modéli-

sation physique d'un drone à 4 hélices et de nombreux plugins qui permettent de simuler son comportement bas-niveau. Ce modèle est dirigeable en commande moteurs : nous lui envoyons des commandes PWM pour chaque moteur d'hélice, et le simulateur calcule la position réelle du drone puis l'affiche. Il est également possible d'ajouter une valeur de vent, qui viendra perturber le vol du drone. Le modèle comporte aussi des capteurs simulés de localisation du drone, dont les données seront utilisées par notre noeud de calcul de contrôle optimal (baromètre, gyromètre, accéléromètre) afin de calculer les commandes PWM. Rappelons que ces capteurs permettent simplement de déterminer la position du drone et non de détecter les obstacles à proximité (leur position est connue a priori).

En utilisant Hector Quadrotor, nous pouvons simuler entièrement le drone, en lui donnant les mêmes entrées et en récupérant les mêmes données de capteurs qu'avec un drone réel. C'est un pas en avant réalisé dans la direction de l'utilisation de notre algorithme sur un vrai drone. Si nous disposions d'un drone avec des capacités de calcul suffisamment élevées et doté d'un système avec ROS, nous pourrions faire tourner notre noeud de contrôle optimal sur le drone. Malheureusement, le contrôle optimal demande beaucoup de ressources, ce qui rend son utilisation inenvisageable en conditions réelles pour l'instant.

3.3.2 Installation, test et validation du package Hector Quadrotor

Les instructions de téléchargement et d'installation d'Hector Quadrotor sont disponibles dans la documentation technique. Une fois l'installation et la compilation effectuée, nous avons pu tester le modèle en utilisant les démos fournies. Deux modes de commandes sont possibles : commandes en vitesse ou commandes moteurs. Les commandes vitesses restent un niveau au-dessus des commandes moteurs, le modèle d'Hector ne convertissant pas ces commandes vitesses en commandes moteurs, mais déplaçant directement le drone selon la vitesse donnée (ce qui ne serait pas reproductible sur un drone réel hors simulation). Le drone peut être déplacé en commandes moteurs, mais il est bien entendu extrêmement difficile de le guider. Le contrôle optimal entre alors en jeu pour résoudre ce problème.

Chapitre 4

Amélioration du modèle et introduction d'un estimateur

4.1 Amélioration du modèle

L'amélioration du modèle a été atteinte grâce à l'introduction du paquet ROS Hector Quadrotor [4], qui comporte un modèle complet des effets aérodynamiques et des forces propulsives. Cela représente le modèle de simulation, qui essaye de reproduire dans les moindres détails toutes le phénomènes auxquels le drone est soumis. Pour implémenter le contrôle, le modèle utilisé est beaucoup plus simple et ne tient pas en compte des effets aérodynamiques et propulsifs. Les variables d'intérêt sont la position vitesse et attitude du drone. La matrice d'orientation est paramètrisée à travers les angles d'Euler 3-2-1 (Yaw ψ- Pitch θ- Roll ϕ). La matrice de rotation/orientation pour passer du repère drone au repère inertiel $R_{\mathcal{B}\to\mathcal{I}}$ est donc la suivante :

$$
\begin{aligned}
R_{\mathcal{B}\to\mathcal{I}} &= R_z(\psi)\, R_y(\theta)\, R_x(\phi) = \\
&= \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} = \\
&= \begin{bmatrix} \cos(\psi)\cos(\theta) & \cos(\psi)\sin(\theta)\sin(\phi) - \sin(\psi)\cos(\phi) & \cos(\psi)\sin(\theta)\cos(\phi) + \sin(\psi)\sin(\phi) \\ \sin(\psi)\cos(\theta) & \sin(\psi)\sin(\theta)\sin(\phi) + \cos(\psi)\cos(\phi) & \sin(\psi)\sin(\theta)\cos(\phi) - \cos(\psi)\sin(\phi) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}
\end{aligned}
\tag{4.1}
$$

Les équations du modèle différentiel associées à la dynamique du drone seront alors :

$$f_{\text{dyn}} := \begin{cases} {}^{\mathcal{I}}\dot{\boldsymbol{x}} =^{\mathcal{I}} \boldsymbol{v} \\ \dot{R}_{\mathcal{B}\to\mathcal{I}} = R_{\mathcal{B}\to\mathcal{I}}[{}^{\mathcal{B}}\boldsymbol{\omega}\times] \\ {}^{\mathcal{I}}\dot{\boldsymbol{v}} +^{\mathcal{I}} \boldsymbol{g} = R_{\mathcal{B}\to\mathcal{I}}\,{}^{\mathcal{B}}\mathbf{F}_p \\ {}^{\mathcal{B}}\mathbf{J}\,{}^{\mathcal{B}}\dot{\boldsymbol{\omega}} +^{\mathcal{B}} \boldsymbol{\omega} \times^{\mathcal{B}} \mathbf{J}\,{}^{\mathcal{B}}\boldsymbol{\omega} = {}^{\mathcal{B}}\mathbf{M}_p \end{cases} \tag{4.2}$$

Les deux premières équations représentent la cinématique du système, i.e le lien entre les différentes grandeurs, notamment la vitesse angulaire et la matrice d'orientation. Les deux dernières équations représentent la dynamique du système. Chaque grandeur vectorielle est définie dans un certain repère, qui est noté en exposant à sa gauche. Les forces propulsives dans le repère corps ${}^{\mathcal{B}}\mathbf{F}_p$,${}^{\mathcal{B}}\mathbf{M}_p$ sont modélisées de la façon suivante :

$${}^{\mathcal{B}}\mathbf{F}_p = \begin{pmatrix} 0 \\ 0 \\ -\frac{C_f}{m}\sum_{i=1}^{4}\omega_{m,i}^2 \end{pmatrix} \qquad {}^{\mathcal{B}}\mathbf{M}_p = \begin{pmatrix} d\,C_f(\omega_{m,4}^2 - \omega_{m,2}^2) \\ d\,C_f(\omega_{m,1}^2 - \omega_{m,3}^2) \\ c\,(-\omega_{m,1}^2 + \omega_{m,2}^2 - \omega_{m,3}^2 + \omega_{m,4}^2) \end{pmatrix} \tag{4.3}$$

où d est la distance entre deux hélices opposées, et C_f et c sont des coefficients qui proviennent de la modélisation aérodynamique des efforts. Ils seront calculés sur la base du modèle de drone de Hector.

On a introduit un modèle purement cinématique pour pouvoir donner comme commandes les vitesses linéaires et angulaires du drone. Les équations qui le modélisent sont les suivantes :

$$f_{\text{cyn}} := \begin{cases} {}^{\mathcal{I}}\dot{\boldsymbol{x}} =^{\mathcal{I}} \boldsymbol{u}_v \\ \dot{R}_{\mathcal{B}\to\mathcal{I}} = R_{\mathcal{B}\to\mathcal{I}}[{}^{\mathcal{B}}\boldsymbol{u}_\omega\times] \end{cases} \tag{4.4}$$

La commande ici est constituée des 6 variables $\boldsymbol{u}_v = [u_{v_x}, u_{v_y}, u_{v_z}]^T$, $\boldsymbol{u}_\omega = [u_p, u_q, u_r]^T$.

4.2 Commande Optimale

La commande optimale consiste en la minimisation d'un critère quadratique et utilise les paradigmes du "model predicitive control". On ira donc minimiser un critère du type :

$$\mathcal{L} = \int_{t_{in}}^{t_{fin}} (\mathbf{z} - \mathbf{r}_z)^T Q(\mathbf{z} - \mathbf{r}_z) + (\mathbf{u} - \mathbf{r}_u)^T R(\mathbf{u} - \mathbf{r}_u) dt \tag{4.5}$$

où $\mathbf{z}$ sont les états contrôlés $\mathbf{z} = [v_x\ v_y\ v_z\ p\ q\ r]^T$; le vecteur $\mathbf{r} = [\mathbf{r}_x^T\ \mathbf{r}_u^T]^T$ est la référence à suivre. Dans notre problème, nous n'aurons qu'une composante $\mathbf{r}_x$, qui modélise le fait que

la variable d'état vitesse linéaire **v** doit rester proche de la consigne fournie par l'utilisateur. Les contraintes à imposer dans ce problème comprennent :

— des limitations sur la vitesse angulaire des hélices ;

— une limitation sur l'assiette du drone pour éviter les singularités lié à la paramétrisation selon les angles d'Euler ;

— des contraintes géométriques pour empêcher les collisions avec des obstacles.

Cela se résume en un système d'inégalités à respecter tout le long du vol. Si on prend des cylindres à bases ellipsoïdales de hauteurs infinies et des ellipses, les contraintes deviennent :

$$c_{\text{systeme}} := \begin{cases} \underline{u}_i \leq u_i \geq \bar{u}_i; & \forall i \in [1,2,3,4] \\ \underline{\theta} \leq \theta \geq \bar{\theta}; \end{cases} \tag{4.6}$$

$$c_{\text{envir}} = \begin{cases} \left(\frac{x-x_c}{ac_i}\right)^2 + \left(\frac{y-y_c}{bc_i}\right)^2 \geq 1 & \forall \mathcal{C}_i \,(\text{Cylindre}) \in \mathcal{W}\,(\text{Monde}) \\ [\mathbf{x} - \mathbf{x}_{c,\mathcal{E}_i}]^T A_{\mathcal{E}_i} [\mathbf{x} - \mathbf{x}_{c,\mathcal{E}_i}] \geq 1 & \forall \mathcal{E}_i \,(\text{Ellipse}) \in \mathcal{W}\,(\text{Monde}) \end{cases} \tag{4.7}$$

où on a choisi $\underline{u}_i = 16\,rad/s$, $\bar{u}_i = 150\,rad/s$ et $\bar{\theta} = -\underline{\theta} = 1rad$.

4.3 Introduction d'un estimateur

La grande différence avec le modèle de drone de l'année précédente est l'introduction des capteurs et d'un estimateur pour trouver la valeur des variables d'état à partir des mesures obtenues. Dans une première partie de cette section, on analyse les capteurs qui ont été introduits dans le modèle puis, dans une deuxième partie, l'estimateur choisi. Le choix des capteurs est lié d'une part à leur utilisation dans le domaine des drones et d'autre part à leur simplicité d'implémentation au niveau du code. Les capteurs implémentés sont les **accéléromètres**, **les gyromètres** et le **baromètre**.
Une suite possible pour ce projet pourrait être l'introduction de capteurs plus complexes tels que le GPS dans le modèle.

4.3.1 Les accéléromètres

L'accéléromètre est un capteur qui mesure l'accélération du drone dans le repère lié au drone. Il faut remarquer que la valeur mesurée ne tient pas compte de l'accélération gravitationnelle agissant sur le drone, car le capteur est lui-même soumis à cette accélération. Les équations qui schématisent ce dispositif sont :

$$^{\mathcal{B}}\boldsymbol{a}_m = R_{\mathcal{B}\to\mathcal{I}}\,(^{\mathcal{I}}\boldsymbol{a} - {}^{\mathcal{I}}\boldsymbol{g}) \tag{4.8}$$

4.3.2 Les gyromètres

Le gyromètre est un capteur qui mesure la vitesse angulaire dans le repère fixé à la structure du drone. Les équations qui modélisent ce capteur sont :

$$\omega_{\text{mes}} = \begin{pmatrix} p \\ q \\ r \end{pmatrix} \tag{4.9}$$

4.3.3 Le baromètre

Le baromètre permet de calculer la pression pour une altitude inférieure à 11 km. La pression est évaluée selon le modèle standard de l'atmosphère.

Chapitre 5

Modélisation de l'environnement

L'environnement modélisé doit simuler des obstacles comme en rencontrerait un drone en environnement réel. Dans le projet 2016, seuls des cylindres (base cerculaire, orienté dans n'importe quelle direction) étaient modélisés et pris en compte dans le contrôle optimal.

5.1 Représentation des obstacles

Nous avons dû décider quelles formes seraient utilisées et modélisées. Afin de progresser vers une application utilisable sur un système réel, nous avons voulu élargir les types d'obstacles modélisables.

Nous avons choisi de modéliser des **cylindres** (qui étaient déjà modélisés dans le projet 2016) et des **ellipsoïdes**. Les cylindres peuvent être de bases elliptiques, et d'orientation quelconque dans l'espace. Les ellipsoïdes permettent quant à elles d'approximer des formes plus complètes (voir partie 5.2).

5.1.1 Intégration des obstacles en contraintes de contrôle optimal

Les obstacles sont représentées dans le contrôle optimal en tant que contraintes à ajouter :

$$c_{\text{envir}} = \begin{cases} \left(\frac{x-x_c}{a_{\mathcal{C}_i}}\right)^2 + \left(\frac{y-y_c}{b_{\mathcal{C}_i}}\right)^2 \geq 1 & \forall \mathcal{C}_i \,(\text{Cylindre}) \in \mathcal{W}\,(\text{Monde}) \\ [\mathbf{x} - \mathbf{x}_{c,\mathcal{E}_i}]^T A_{\mathcal{E}_i} [\mathbf{x} - \mathbf{x}_{c,\mathcal{E}_i}] \geq 1 & \forall \mathcal{E}_i \,(\text{Ellipse}) \in \mathcal{W}\,(\text{Monde}) \end{cases}$$

5.1.2 Intégration des obstacles dans le simulateur

Il existe une primitive de modèle de cylindre sous *Gazebo*, le simulateur/visualisateur de ROS. Il suffit d'inscrire dans un fichier XML particulier les paramètres de ce cylindre et de le charger sous *Gazebo* pour charger la forme. Cela peut se faire au run-time (une fois le logiciel lancé) ou à l'initialisation (l'environnement est fixé et fourni au simulateur au début de la simulation).

Cependant, il n'existe pas de primitive pour les ellipses : il faudrait, afin de les afficher dans Gazebo, les créer sous la forme d'un fichier de mesh (en utilisant, par exemple, un logiciel de CAO 3D comme Blender ou Solidworks et en les exportant) puis les importer sous *Gazebo*. Cette méthode peut être utilisée au cas par cas à la main, mais est difficile à automatiser. Nous avons ainsi décidé de ne pas faire apparaître les ellipses sous *Gazebo*, mais uniquement de les implémenter dans le contrôle optimal. Notre but étant d'approximer des objects complexes (une statue par exemple) par une ellipse, nous pouvons nous contenter d'afficher en visuel l'objet complexe (la statue) tout en prenant en compte dans le contrôle l'ellipse approximant l'objet (l'ellipse optimale entourant la statue).

5.2 Élargissement de l'environnement : approximation d'obstacles complexes par des ellipsoïdes

5.2.1 Principe

Étant donné que nous avons choisi de modéliser uniquement des ellipses et des cylindres comme modèles d'obstacles, nous avons décidé de permettre l'approximation d'autres formes (parallélépipède, cube, prisme, ...) par des ellipsoïdes. Le programme est basé sur l'algorithme *Khachiyan* [5]. L'algorithme permet d'obtenir l'ellipse de plus petit volume enveloppant un nuage de points dans un espace de dimension d finie quelconque ($d = 3$ dans notre cas). Les paramètres d'entrées de cette fonction sont un nuage de point (ex : les extrémité de l'objet à envelopper) et une tolérance (distance maximum acceptée entre les points extrêmes et la surface de l'ellipse). Nous avons généré un code en C++ grâce à *Matlab coder*, à partir d'une version de cet algorithme implémenté sous Matlab. Cette fonction s'appelle "Minimum Volume Enclosing Ellipsoid". Nous avons ensuite écrit une fonction interface permettant d'appeler le code C++ généré avec des objets classiques de C++.

Les figures suivantes illustrent l'utilisation de la fonction sur différentes formes d'obstacles : un nuage de point à gauche et une forme complexe maillée par un réseau de points à droite.

Cet algorithme permet donc d'envelopper une forme 3D dès lors que l'on possède les

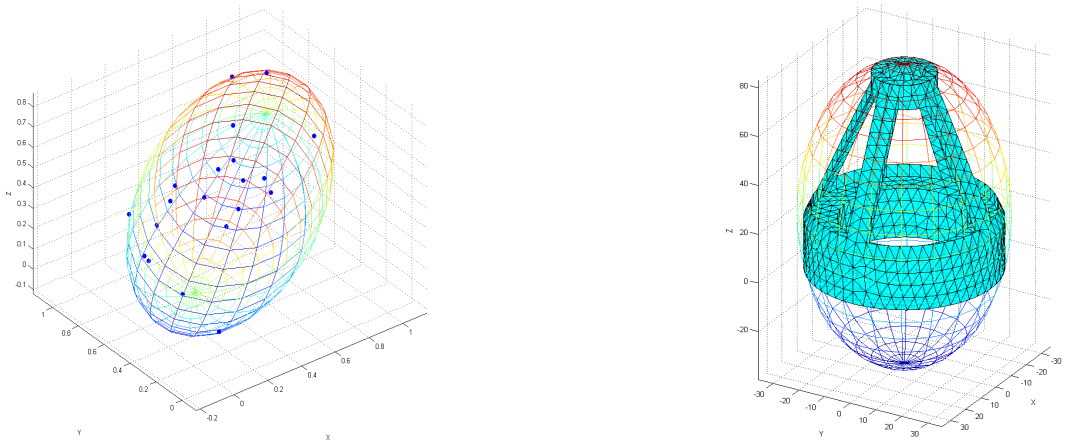

FIGURE 5.1 – Nuage de point et forme complexe enveloppés par des ellipses de volume minimal

coordonnées des points extrêmes de l'objet. Cette approche a néanmoins des limitations. Lorsque la forme à envelopper dispose de sous-formes à "membres" ou bien de trous, approximer celle-ci avec une ellipse n'est plus pertinent, car l'ellipse approximera trop grossièrement l'objet. Un grand volume autour de l'objet ne sera ainsi pas atteignable même si visuellement l'espace est vide (exemple : une ellipse enveloppant l'Arc de Triomphe bloquera toute la partie intérieure sous l'arc). On peut aussi voir sur la Figure 5.1 que le volume entre les poutres de l'objet de droite est compris dans l'ellipse donc non atteignable. On peut ainsi améliorer cette modélisation en découpant l'objet à envelopper en plusieurs parties (grace à des algorithmes de décomposition convexes par exemple). Pour des objets plus complexes, cela reviendra à utiliser plusieurs ellipses et non une seule.

5.2.2 Application à notre projet

Afin d'utiliser cet algorithme dans notre projet, nous avons réalisé :

— la génération d'une version C++ du code à partir du code Matlab
— l'interfaçage du code généré avec les objets C++ standard
— l'intégration de l'ensemble dans un noeud ROS

Afin de pouvoir calculer l'ellipse entourant un objet, il nous faut :

— obtenir un fichier représentant la forme 3D
— extraire de ce fichier un nuage de point
— fournir ce nuage de point à l'algorithme
— récupérer l'équation de l'ellipse

L'extraction de points depuis le fichier de l'objet peut se faire de plusieurs manières. Nous avons choisi, par gain de temps, d'extraire des points pertinents (les points extrêmes) à la main. Nous avons envisagé une automatisation de cette extraction, en échantillonant les points de l'objet selon une distribution particulière. Cette idée n'a pas été implémentée, mais serait intéressante à réaliser dans le futur. Ce processus d'extraction des points et de calcul de l'ellipse n'est à faire qu'une seule fois par objet : les calculs n'ont pas de besoin temps réel. Avec quelques dizaines de points, le calcul de l'ellipse est quasi-instantanné, mais peut monter à plusieurs secondes s'il y a plusieurs millions de points.

Une fois l'équation de l'ellipse obtenue, elle peut être directement intégrée dans le contrôle optimal en tant que contrainte. A ce stade, si l'on charge le fichier original de l'objet, on verra le drone éviter l'objet. Faire apparaître l'ellipse sous Gazebo requiert des étapes supplémentaires, car il n'y a pas de manière simple de créer une ellipse sous *Gazebo*. Il faut donc passer par un autre logiciel. Les étapes sont les suivantes :

— calculer une pramétrisation de l'ellipse à partir de son équation

— sous Blender (logiciel de 3D gratuit), créer une ellipse à partir de cette paramétrisation

— exporter à partir de Blender un fichier compatible avec Gazebo décrivant l'ellipse (extension .dae)

— importer le fichier sous Gazebo et régler si nécessaire la transparence

On obtient ainsi une ellipse entourant l'objet, afin de visualiser l'obstacle réellement pris en compte par le drone. Nous avons suivi toutes ces étapes sur l'exemple suivant (fig.5.2).

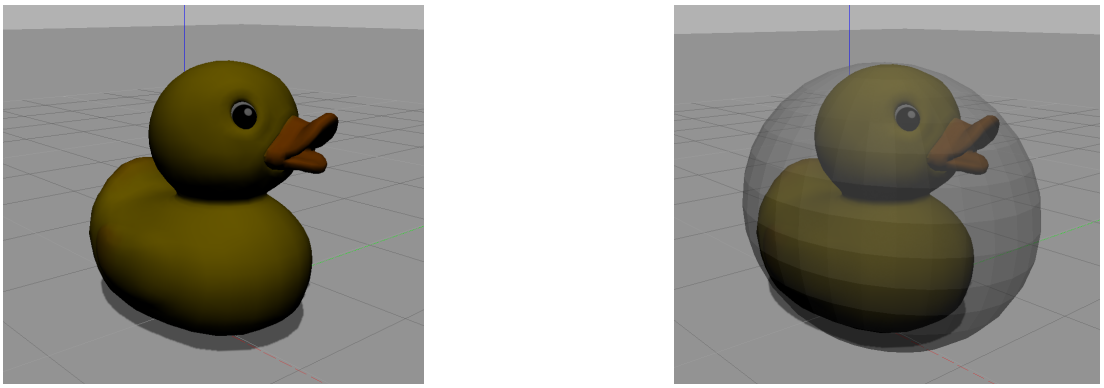

FIGURE 5.2 – A gauche : objet complexe sous Gazebo ; à droite : objet complexe entouré de son ellipse englobante optimale sous Gazebo

Chapitre 6

Résultats et demonstration

6.1 Travail effectué

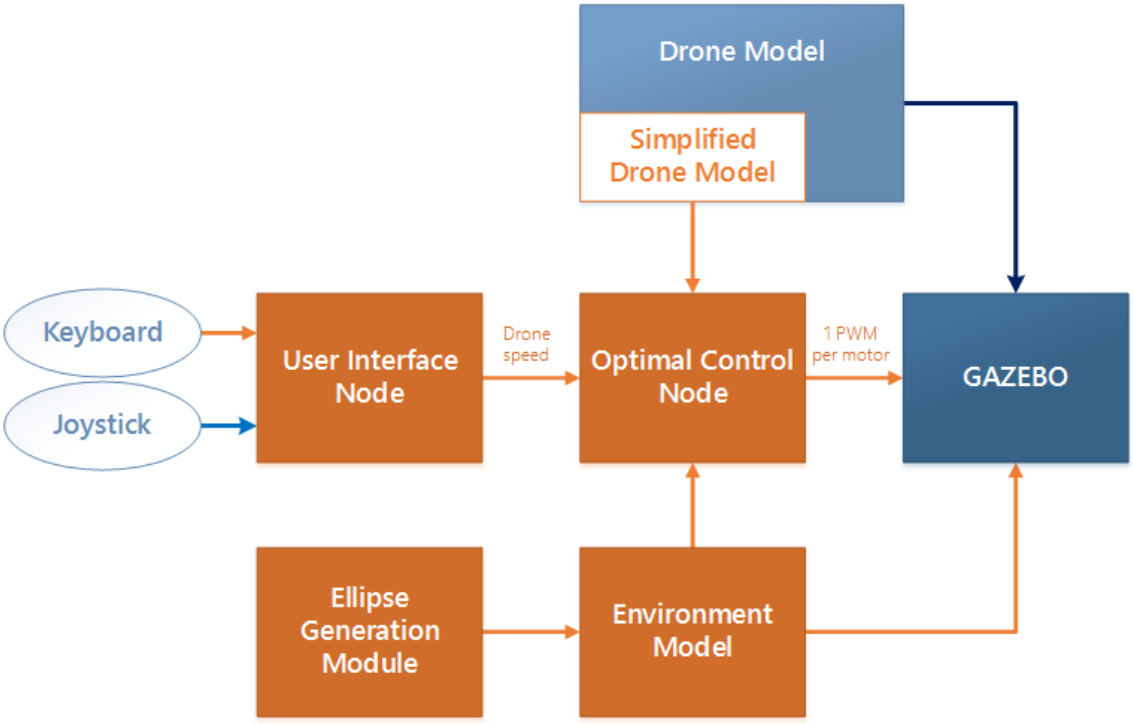

FIGURE 6.1 – Architecture logicielle du projet avec en rouge les éléments réalisés par l'équipe du projet 2017

Le schéma d'architecture présenté ci-dessus (fig. 6.1) fait apparaître en rouge les tâches qui ont effectivement été réalisées par l'équipe au cours du projet, en opposition aux éléments externes qui ont été intégrés au projet.

L'utilisation de ROS nous a permis de séparer les différents éléments du programme sous la forme de noeuds. Nous avons ainsi codé les noeuds *Interface utilisateur* et *Contrôle optimal*, ainsi que leurs interfaces.

La récupération des entrées clavier est effectuée depuis le noeud *Interface utilisateur* tantdis que la récupération des données joystick se fait par l'intermédiaire d'un noeud ROS externe (*joy*).

Le modèle de drone utlisé dans le modèle optimal est un modèle simplifié que nous avons élaboré à partir du modèle complet fourni par *Hector Quadrotor*.

6.2 Livrables

Les livrables (code, rapport, documentation technique) sont disponibles sur le dépôt GIT suivant : `https://github.com/DianeBury/IntelTeleop`

Le dossier *ProjetSupaero2016* contient les livrables du projet de l'équipe précédente, tandis que les livrables de cette année sont sous le dossier "ProjetSupaero2017".

6.2.1 Instructions d'installation

Afin de télécharger le code source, il faut effectuer (sur un système Linux, avec git installé) :

```
git clone https://github.com/DianeBury/IntelTeleop
```

Les instructions d'installation se trouvent dans le fichier *README.md*. La documentation technique *user_and_developper_guide.md* donne des détails plus techniques ainsi que des instructions d'utilisation plus poussées.

6.3 Lancer la démo

Afin de lancer la démo, il faut exécuter la commande :

```
roslaunch intel\_teleop\_demo demo.launch
```

L'environnement *Gazebo* va se lancer et charger la scène de démo ainsi que le drone. Les commandes seront (par défaut) avec le clavier. Pour lancer la démo avec une manette, il faut lancer :

```
roslaunch intel\_teleop\_demo demo.launch joystick:=true
```

Dans ce cas, sera peut-être nécessaire de modifier le fichier *demo.launch* afin de sélectionner le périphérique correspondant au joystick, dans la mesure où cette configuration peut varier d'un ordinateur à l'autre.

Il suffit alors de commander le drone en direction (gauche, droite, devant, derrière) et en hauteur (monter, descendre) en utilisant le clavier ou la manette. Le drone va alors obéir aux commandes, avec un certain temps de réaction. Il est possible de lancer la simulation à une certaine fraction du temps réel afin d'alléger la demande en ressources processeurs.

6.4 Résultats

Le contrôle optimal a nécessité de nombreux réglages et adaptations pour obtenir un résultat satisfaisant. Nous avons réussi à obtenir un réglage réaliste et cohérent qui fait suivre au drone des trajectoires acceptables avec une bonne réactivité. Utiliser une manette permet de commander le drone facilement et instinctivement, et ce même pour des personnes n'ayant aucune compétence préalable dans le maniement de drone.

Au niveau de la gestion de l'environnement, il est possible d'insérer des obstacles avant ou pendant l'exécution de la simulation. Afin d'insérer des objets plus complexes, on peut utiliser les fonctions de calcul d'ellipse englobante optimale : ces fonctions fonctionnent mais le passage du modèle d'objet au nuage de points à donner à la fonction n'est pas automatisé.

Seuls les cylindres verticaux infinis sont proprement gérés dans le contrôle optimal. Le drone réussit alors à éviter le cylindre : quand le drone arrive près de l'obstacle, il reçoit un vecteur qui annule la commande utilisateur et repousse le drone dans le sens inverse. Le drone recule donc, et il est impossible pour l'utilisateur de faire une collision avec l'obstacle.

On voit que le drone a un léger mouvement d'oscillation quand il se stabilise (il n'est jamais vraiment à l'arrêt total, ce qui est réaliste). Quand il évite un obstacle, le drone est repoussé dans l'autre sens, ce qui est réaliste et acceptable : on pourrait améliorer l'évitement en changeant ce comportement et en faisant contourner l'obstacle au drone.

On peut noter que même si les réglages du contrôle optimal et ceux des évitements sont satisfaisant, ils peuvent encore être améliorés.

Chapitre 7

Conclusion

7.1 Bilan des exigences

1. Système général

a. Le système doit être implémenté en utilisant un logiciel de gestion de versions collaboratif distribué
Respectée : le logiciel GIT a été utilisé

b. Les différentes parties du logiciel du système doivent être séparées du point de vue du code et placées chacune sur une branche différente au sein du gestionnaire de versions
Respectée : le contrôle optimal, l'interface utilisateur et l'environnement ont été développés dans des noeuds ROS séparés et sur des branches séparées

c. Le merging d'une branche de développement avec la branche principale devra être approuvée par un membre non impliqué dans le développement de cette branche
Respectée

d. Le système doit être fourni avec son code source et une documentation technique
Respectée : code et documentation présents sur le dépôt GIT (voir partie 6.2)

2. Documentation technique

a. La documentation doit détailler l'installation du logiciel
Respectée : les instructions d'installations se trouvent dans le fichier *README.md*

b. La documentation doit détailler les librairies externes utilisées avec le numéro de la version nécessitée
Respectée : Les instructions d'installation dans *README.md* contiennent les numéros de versions des dépendances

c. La documentation doit contenir les instructions d'utilisation du logiciel
Respectée : les instructions d'utilisation sont dans *user_ and_ developper_ guide.md*

3. Simulateur

a. Le simulateur doit opérer sous un environnement Linux
Respectée : notre application tourne sous ROS, qui est compatible avec les grandes distributions Linux

b. Le simulateur doit s'accompagner d'un visualisateur temps réel
Respectée : notre application utilise le visualisateur Gazebo en temps réel

c. Le simulateur doit simuler un drone à hélices commandable
Respectée : notre application simule le modèle Hector Quadrotor qui est un drone à 4 hélices commandable

d. Le simulateur doit simuler le drone dans le cas nominal (i.e. sans panne)
Respectée : notre application ne gère pas les pannes

e. Le simulateur doit utiliser un modèle réaliste du drone (propulsion...) et de l'environnement (aérodynamique...)
Respectée : les forces de propulsion et aérodynamiques sont prises en compte dans la simulation

f. Le simulateur doit simuler des obstacles
Respectée : il est possible d'intégrer deux types d'obstacles (cylindres et ellipses) en plus du sol dans le simulateur

g. Le simulateur doit comporter un modèle aérodynamique intégrant un modèle de vent
Respectée : la simulation du modèle de drone comporte une force de vent réglable

h. Le simulateur doit utiliser le contrôle optimal
Respectée : le simulateur est interfacé avec le noeud de contrôle optimal

4. Contrôle optimal

a. Le contrôle optimal doit assister en temps réel la téléopération du drone par l'utilisateur
Respectée : la contrainte temps réel de téléopération est respectée

b. Le contrôle optimal doit s'accompagner d'un module d'interface utilisateur
Respectée : le contrôle optimal est interfacé au noeud d'interface utilisateur

c. Le contrôle optimal doit transformer des commandes simples provenant de l'interface utilisateur en commandes d'entrée du drone
Respectée : l'utilisateur utilise soit les touches directionnelles du clavier, soit une manette, pour contrôler le drone

d. Le contrôle optimal doit empêcher le drone d'entrer en collision avec les objets de l'environnement simulé
Non respectée : le contrôle optimal permet l'évitement de collision avec des cylindres verticaux infinis mais l'évitement de cylindres quelconques et d'ellipses n'a pas pu être achevé dans les temps

e. Le contrôle optimal devra pouvoir s'exécuter sur un PC de bureau équipé d'un processeur i7 de de 3ème génération ou plus récent, à une fréquence de 10Hz
Respectée : la simulation a été testée sur un processeur i7 de 4ème génération à 80Hz

5. Interface utilisateur

a. L'interface utilisateur doit accepter des commandes utilisateurs provenant d'un clavier d'ordinateur ou d'une manette, ce en temps réel
Respectée : l'utilisateur peut choisir d'utiliser soit une manette soit les touches directionnelles du clavier

En conclusion, nous pouvons dire que toutes les exigences ont été respectées, à l'exception de celle portant sur l'évitement d'obstacles. Les obstacles eux-mêmes sont bien intégrés dans le simulateur, mais le contrôle optimal ne parvient pas à éviter les collisions. Le contrôle optimal n'a en effet pas pu être réglé de manière concluante.

Nous avons tout de même réussi à respecter la quasi-totalité des exigences, et ainsi à satisfaire les donneurs d'ordre. Le défi principal du projet 2017 étant le passage d'un démonstrateur à un simulateur : nous pouvons dire que c'est un succès.

7.2 Perspectives d'avenir

— Ajout d'un évitement d'obstacles pour les cylindres (actuellement, il n'existe que pour les cylindres infinis verticaux)

— Ajout d'un évitement d'obstacles pour les ellipses

— Test de différents réglages d'ACADO pour tenter d'améliorer la stabilité

— Automatisation de l'approximation par ellipses (notamment l'échantillonnage de points)

7.3 Bilan du projet

Ce PIE était l'occasion d'allier la découverte de la gestion de projet à une réalisation technique. Nous avons réussi à développer une application fonctionnelle en continuité du projet de l'année dernière. Un pas technique important a été franchi vers la portabilité du contrôle optimal sur un drone réel, ce qui constitue la réussite principale du projet. En plus de cela, plusieurs améliorations ont été apportées, notamment : la restructuration du code, la modélisation du vent, l'élargissement de l'environnement. Nous laissons à l'équipe qui prendra notre suite un code complet, commenté, facilement installable et maintenable, accompagné d'une documentation et de suggestions d'améliorations.

Bibliographie

[1] Documentation technique pour l'application teleopération intelligente pour microdrone complexe. 2017.

[2] Yoann Baillau, Thibault Barbié, Zhengxuan Jia, Francisco Pedrosareis, William Rakatomanga, and Baudouin Roullier. Rapport de projet ingénierie et entreprenariat - teleopération intelligente pour microdrone complexe. 2016.

[3] Hector quadrotor package summary. `http://wiki.ros.org/hector_quadrotor`. Accessed : 2017-03-22.

[4] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. Comprehensive simulation of quadrotor uavs using ros and gazebo. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 400–411. Springer, 2012.

[5] L.G. Khachiyan. Rounding of polytopes in the real number model of computation. *Math. Oper. Res. 21*, pages 307–320, 1996.

[6] Houska Boris et al. *ACADO toolkit user's manual.* 2009.

[7] Ariens David et al. *Acado for Matlab user's manual.* 2010.

[8] Diehl Moritz et al. Fast direct multiple shooting algorithms for optimal robot control. *Springer, Berlin*, pages 65–93, 2006.

[9] Mahony Robert Vijay Kumar and Peter Corke. Multirotor aerial vehicles : Modeling, estimation, and control of quadrotor. *IEEE robotics & automation magazine*, pages 20–32, 2012.

[10] Hamel Tarek and Philippe Soueres. Modélisation, estimation et contrôle des drones à voilures tournantes ; un aperçu des projets de recherche français. *IEEE robotics & automation magazine*, pages 20–32, 2005.

[11] Hehn Markus and Raffaello D'Andrea. Quadrocopter trajectory generation and control. *IEEE robotics & automation magazine*, pages 1485–1491, 2011.

[12] Geisert Mathieu and Nicolas Mansard. Trajectory generation for quadrotor based systems using numerical optimal control. *arXiv preprint arXiv :1602.01949*, pages 1485–1491, 2016.

[13] Castellanos Fermi Guerrero. *Estimation de l'attitude et commande bornée en attitude d'un corps rigide : application à un mini hélicoptère à quatre rotors*. Université Joseph-Fourier-Grenoble I, 2008.

[14] Joyo M. Kamran et al. Position controller design for quad-rotor under perturbed condition. *arXiv preprint arXiv :1602.01949*, pages 178–189, 2013.