POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Pilotage autonome agressif de drone dans un environnement de course

MATHIEU ULYSSE ASHBY
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie aérospatial

Décembre 2020

© Mathieu Ulysse Ashby, 2020.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Pilotage autonome agressif de drone dans un environnement de course

présenté par **Mathieu Ulysse ASHBY**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Richard GOURDEAU, président
David SAUSSIÉ, membre et directeur de recherche
Sofiane ACHICHE, membre et codirecteur de recherche
Jérôme LE NY, membre

DÉDICACE

À mes parents

REMERCIEMENTS

Tout d'abord, j'aimerais remercier mes directeurs de recherche, David Saussié et Sofiane Achiche, pour leur encadrement, leurs conseils judicieux et leur rigueur. J'aimerais apporter une mention particulière à M. Saussié pour son soutien académique et personnel hors-pair. Son écoute et ses conseils ont grandement été appréciés.

J'aimerais également remercier François Defaÿ et Joël Bordeneuve-Guibé du laboratoire DCAS de l'ISAE-SUPAERO qui m'ont accueilli et fait bénéficier de leur grande expertise au niveau des drones, ainsi que de leurs installations à la fine pointe de la technologie lors de mon séjour de recherche à Toulouse à l'automne 2019. Ce séjour fut inoubliable tant au niveau académique qu'au niveau personnel. J'aimerais apporter une mention spéciale à tous mes amis et collègues qui ont su rendre chacun à leur façon cette expérience mémorable.

Je remercie les membres du jury, M. Richard Gourdeau et M. Jérôme Le Ny d'avoir accepté de lire mon mémoire et de participer à la soutenance. Je tiens à remercier tous mes collègues du MRASL : Jérémie, Olivier, les Mathieu, Catherine, Tien, Florian et Justin. Vous avez tous contribué à rendre mes études supérieures un des plus beaux moments de ma vie jusqu'à présent. J'aimerais remercier Jérémie et Tien pour leur grande implication dans mes travaux expérimentaux. Vos conseils et connaissances techniques m'ont sauvé d'innombrables heures de travail. Jérémie et Olivier, non seulement avez-vous été de bons collègues, mais vous avez été de bons amis. Je me rappellerai toujours de nos moments passés ensemble.

J'adresse un merci spécial à mes amis proches Alexandre, Manuel, Antoine, Gabriel et Mikael d'avoir fait de ces deux années d'études supérieures un moment formidable. Vos conseils et nos scéances de spikeball thérapeutiques ont toujours réorienté mes esprits (sauf quand Antoine pogne une pocket). J'aimerais apporter une mention spéciale à mes colocataires Alexandre et Manuel qui ont toujours porté un intérêt envers mon projet et qui ont rendu le confinement pandémique beaucoup plus agréable.

Merci à mes parents, Richard et Suzanne, sans qui je n'aurais pu accomplir ce que j'ai fait dans ces travaux. Vous m'avez toujours encouragés et soutenus dans mes projets de vie et avez toujours su me conseiller judicieusement.

Pour terminer, j'aimerais remercier les organismes subventionnaires qui ont soutenu le travail effectué dans cette maîtrise, soit le Conseil de Recherche en Sciences Naturelles et en Génie du Canada (CRSNG), les Fonds de recherche du Québec en Nature et Technologie (FRQNT) et Mitacs.

RÉSUMÉ

Dans les dernières années, le marché des drones a connu un essor majeur dans plusieurs domaines, tant civil, commercial qu'académique. Parmi ceux-ci, un nouveau domaine d'intérêt a fait son apparition : les courses de drones. Ces courses mettent en évidence la supériorité des pilotes humains par rapport aux systèmes autonomes développés jusqu'à présent. Pour l'instant, ces systèmes d'autonomie sont très limités et plusieurs capteurs sont nécessaires pour un vol sécuritaire et performant. La piste d'une course de drone est un environnement particulièrement difficile comparativement aux conditions standards de vol du fait de hautes vitesses impliquées et d'un environnement visuel très complexe. Ce contexte pose donc d'importants défis dans le développement et l'implémentation en temps réel d'algorithmes d'estimation d'état et de perception, mais aussi de planification de trajectoires et de lois de commande.

Les lois de commande non-linéaires sont les plus appropriées pour contrôler un multicoptère dans des conditions non-standards de vol. En ce qui a trait à la génération de trajectoires, les trajectoires polynomiales optimales permettent, contrairement aux autres méthodes populaires, de générer des trajectoires satisfaisant les contraintes dynamiques du multicoptère. L'objectif principal de ce projet de recherche est donc d'améliorer l'autonomie des drones de course en se concentrant sur les problèmes de commande et de navigation à hautes vitesses.

Tout d'abord, une modélisation complète de la dynamique d'un multicoptère est effectuée afin de simuler son comportement, et de concevoir la loi de commande non-linéaire. La propriété de platitude différentielle du multicoptère est alors exploitée pour la génération de trajectoires optimales. La loi de commande non-linéaire est ensuite validée par voie de simulation haute fidélité dans l'environnement *ROS-Gazebo*. Des trajectoires complexes et agressives sont suivies avec une erreur relative faible.

Une version améliorée de la méthode de génération de trajectoires est ensuite présentée. Elle consiste essentiellement en des modifications au niveau de la robustesse numérique et l'ajout de l'optimisation du temps dans la formulation originale. Ces modifications sont alors utilisées pour générer des trajectoires optimales dans un parcours typique de course de drones.

Pour terminer, la loi de commande et la méthode génération de trajectoires développées sont combinées avec un module de détection et de cartographie pour former un système autonome de parcours de piste de courses. La performance de ces algorithmes est évaluée sur deux pistes de courses typiques pour drone. Les résultats des simulations montrent une performance et robustesse légèrement supérieure à la littérature.

ABSTRACT

In recent years, the UAV market has experienced major growth in several areas, including civil, commercial and academics. Among these, a new field of interest has emerged: UAV racing. These races highlight the superiority of human pilots over the autonomous systems developed to date. Currently, these autonomous systems are very limited and several sensors are required for safe and efficient flight. UAV racing tracks are a particularly difficult environment compared to standard flight conditions due to the high speeds involved and the very complex visual environment. This context therefore poses significant challenges in the development and implementation of real-time state estimation and perception algorithms, as well as trajectory planning and control laws.

Non-linear control laws are the most suitable to control a multicopter in non-standard flight conditions. As far as trajectory generation is concerned, optimal polynomial trajectories allow, contrary to other popular methods, to generate trajectories satisfying the dynamic constraints of the multicopter. The main objective of this research project is therefore to improve the autonomy of racing UAVs by focusing on control and navigation problems at high speeds.

First of all, a complete modeling of the dynamics of a multicopter is carried out in order to simulate its behavior, and to design the non-linear control law. The differential flatness property of the multicopter is then exploited to generate optimal trajectories. The non-linear control law is then validated by high-fidelity simulation in the *ROS-Gazebo* environment. Complex and aggressive trajectories are followed with a low relative error.

An improved version of the trajectory generation method is then presented. It essentially consists in modifications to the numerical robustness and the addition of time optimization in the original formulation. These modifications are then used to generate optimal trajectories in a typical UAV race course.

Finally, the control law and trajectory generation method developed are combined with a detection and mapping module to form a stand-alone autonomous drone racing system. The performance of these algorithms is evaluated on two typical UAV racetracks. Simulation results show a performance and robustness slightly superior to the literature.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vi

TABLE DES MATIÈRES vii

LISTE DES TABLEAUX x

LISTE DES FIGURES xi

LISTE DES SIGLES ET ABRÉVIATIONS xiii

CHAPITRE 1 INTRODUCTION 1
- 1.1 Contexte 1
- 1.2 Objectifs de recherche 3
- 1.3 Plan du mémoire 4

CHAPITRE 2 REVUE DE LITTÉRATURE 5
- 2.1 Modélisation d'un multicoptère 5
- 2.2 Méthodes de commande 6
  - 2.2.1 Commande classique linéaire 6
  - 2.2.2 Commande non-linéaire 8
- 2.3 Planification de trajectoires 10
- 2.4 Courses de drones autonomes 13
  - 2.4.1 Premiers travaux 15
  - 2.4.2 Premières approches 16
  - 2.4.3 État de l'art 17

CHAPITRE 3 MODÉLISATION DYNAMIQUE DU MULTICOPTÈRE ET MONTAGE EXPÉRIMENTAL 20
- 3.1 Modèle dynamique du multicoptère 20
  - 3.1.1 Repères et orientation 20

3.1.2 Représentations de rotations 22
3.1.3 Équation dynamique en translation 25
3.1.4 Équation dynamique en rotation 26
3.1.5 Matrice d'allocation 27
3.1.6 Modèle non-linéaire complet 28
3.1.7 Dynamique des moteurs 29
3.1.8 Paramètres du drone 29
3.2 Platitude différentielle (*differential flatness*) 30
3.3 Environnement de développement 35

CHAPITRE 4 LOI DE COMMANDE 36
4.1 Architecture du correcteur 36
4.1.1 Boucle Externe 37
4.1.2 Boucle Interne 40
4.2 Preuve de stabilité 40
4.2.1 Erreur de vitesse angulaire 41
4.2.2 Erreur d'attitude 42
4.2.3 Erreur en translation 45
4.2.4 Système Complet 48
4.3 Essais en simulation 52
4.3.1 Implémentation de la loi de commande 52
4.3.2 Simulations et Résultats 52
4.4 Conclusion 56

CHAPITRE 5 GÉNÉRATION DE TRAJECTOIRES 57
5.1 Formulation du problème d'optimisation 57
5.2 Reformulation du problème 62
5.2.1 Reformulation des contraintes 63
5.3 Optimisation du temps 66
5.4 Conclusion 68

CHAPITRE 6 PARCOURS D'UN ENVIRONNEMENT DE COURSE 69
6.1 Description de la problématique 69
6.2 Solution proposée 70
6.3 Détection et cartographie des balises 72
6.3.1 Formes des balises 72
6.3.2 Algorithme de détection 73

6.3.3 Cartographie des balises 76
6.4 Parcours autonome d'une piste et génération de trajectoires 77
6.4.1 Modifications pour évitement de parois 78
6.4.2 Allocation de temps 79
6.4.3 Détails d'implémentation 80
6.5 Simulations 80
6.5.1 Pistes simulées 81
6.5.2 Essais expérimentaux 83
6.5.3 Résultats 84
6.6 Conclusion 95

CHAPITRE 7 CONCLUSION 97
7.1 Synthèse des travaux 97
7.2 Limitations de la solution proposée 98
7.3 Développements futurs 98

RÉFÉRENCES 100

LISTE DES TABLEAUX

Tableau 3.1 Paramètres physiques du Asctec Firefly 29
Tableau 6.1 Taux de succès de la méthode d'autonomie pour la première piste . . 85
Tableau 6.2 Erreurs de suivi de trajectoire moyens pour les essais sur la première piste 87
Tableau 6.3 Taux de succès de la méthode d'autonomie pour la deuxième piste . . 90
Tableau 6.4 Erreurs de suivi de trajectoire moyens pour les essais sur la deuxième piste 92

LISTE DES FIGURES

Figure 1.1 Drone de course 1
Figure 1.2 Compétition de drone autonome iROS 2017 (tiré de [1]) 2
Figure 2.1 Contrôle de drone par pilotage 6
Figure 2.2 Architecture de commande typique pour multicoptères 7
Figure 2.3 Disposition de la piste dans la conférence iROS 2017 (tiré de [1]) . . . 14
Figure 3.1 Forces, moments et repères d'un quadricoptère 21
Figure 3.2 Visualisation des angles d'Euler 23
Figure 4.1 Architecture de la loi de commande 36
Figure 4.2 Réponse à l'échelon en $\mathbf{z}$ 53
Figure 4.3 Réponse à l'échelon en $\mathbf{x}$ 53
Figure 4.4 Réponse à l'échelon en $\mathbf{y}$ 53
Figure 4.5 Trajectoire du multicoptère dans le plan $\mathbf{x}_i - \mathbf{y}_i$ 54
Figure 4.6 Trajectoire du multicoptère selon l'axe $\mathbf{x}_i$ 54
Figure 4.7 Trajectoire du multicoptère selon l'axe $\mathbf{y}_i$ 54
Figure 4.8 Trajectoire du multicoptère selon l'axe $\mathbf{x}_i$ 55
Figure 4.9 Trajectoire du multicoptère selon l'axe $\mathbf{y}_i$ 55
Figure 4.10 Trajectoire du multicoptère selon l'axe $\mathbf{z}_i$ 55
Figure 4.11 Trajectoire du multicoptère en 3 dimensions 56
Figure 5.1 Trajectoire minimale en *snap* selon l'axe $\mathbf{x}_i$ pour la piste *AlphaPilot* [2] 63
Figure 5.2 Comparaison entre les trajectoires générées avec problème d'optimisation quadratique et avec une résolution directe 66
Figure 5.3 Trajectoire minimale en *snap* avec répartition des segments de temps optimisée 67
Figure 6.1 Architecture du système d'autonomie 71
Figure 6.2 Architecture du bloc de planification 71
Figure 6.3 Balise utilisée pour les pistes 73
Figure 6.4 Image de balise filtrée 75
Figure 6.5 Filtre de Canny appliquée sur l'image de la balise 75
Figure 6.6 Détection de la bande rectangulaire noire 75
Figure 6.7 Invariance de la trajectoire générée par l'algorithme original selon l'orientation des balises 78
Figure 6.8 Visualisation du parcours des pistes simulées 81
Figure 6.9 Visulation dans l'environnement Gazebo des pistes simulées 82

Figure 6.10 Image du drone simulé dans l'environnment Gazebo 83
Figure 6.11 Schématisation de l'implémentation ROS des simulations 83
Figure 6.12 Trajectoires optimales voisines 86
Figure 6.13 Vitesse totale pour le parcours de la première piste à 4 m/s 87
Figure 6.14 Résultats des travaux de [3] sur une piste similaire à la première piste 89
Figure 6.15 Visualisation de la trajectoire pour la phase de performance pour la première piste 90
Figure 6.16 Profil de vitesse pour le parcours de la deuxième piste 92
Figure 6.17 Profil de vitesse pour le parcours de la deuxième piste 93
Figure 6.18 Visualisation de la trajectoire pour la phase de performance pour la deuxième piste 94

LISTE DES SIGLES ET ABRÉVIATIONS

Acronymes

ENU	*East-North-Up*, est-nord-haut
FLU	*Front-Left-Up*, devant-gauche-haut
GPS	*Global Positioning System*, système de positionnement global
IMU	*Inertial Measurement Unit*, unité de mesure inertielle
LIDAR	*Light Detection And Ranging*, détection et localisation par la lumière
LQR	*Linear Quadratic Regulator*, régulateur linéaire quadratique
LTI	*Linear Time Invariant*, linéaire et invariant dans le temps
MIMO	*Multiple Input Multiple Output*, multi-entrées multi-sorties
MPC	*Model Predictive Control*, commande prédictive
MRASL	*Mobile Robotics and Autonomous Systems Laboratory*, laboratoire de robotique mobile et de systèmes autonomes
MSF	*Multi-Sensor Fusion Framework*, module de fusion multi-capteurs
PD	Proportionnel Dérivé
PID	Proportionnel Integral Dérivé
PNP	*Perspective-n-Points*, perspective depuis n points
ROS	*Robot Operating System*, système d'exploitation pour robots
RRT	*Rapidly exploring Random Trees*, exploration rapide par branches aléatoires
SISO	*Single Input Single Output*, mono-entrée mono-sortie
SLAM	*Simultaneous Localization And Mapping*, localisation et cartographie simultanées
VIO	*Visual Inertial Odometry*, odométrie visuelle-inertielle

Mécanique du vol

— Repères :

$\mathcal{F}_b =$	Repère objet $\{O_b, \mathbf{x}_b, \mathbf{y}_b, \mathbf{z}_b\}$
$\mathcal{F}_i =$	Repère inertiel $\{O_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i\}$
$\mathcal{F}_{cam} =$	Repère caméra $\{O_c, \mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c\}$

— Caractéristique du multicoptères :

m	Masse du multicoptère

$\mathbf{J}$	Matrice d'inertie du multicoptère
d	Longueur du bras du multicoptère
k_t	Coefficient de poussée des hélices
k_d	Coefficient de trainée des moteurs

— Paramétrage de la dynamique du multicoptères :

$\mathbf{p}$	Position dans le repère inertiel
$\mathbf{v}$	Vitesse dans le repère ineriel
$\mathbf{q}$	Orientation en quaternions entre $\mathcal{F}_b$ et $\mathcal{F}_i$
$\mathbf{\Omega}$	Vitesse de rotation angulaire dans $\mathcal{F}_b$

— Forces et moments appliqués au multicoptère :

$\mathbf{F}_t$	Force de poussée des hélices
$\mathbf{F}_g$	Force gravitationnelle
$\mathbf{M}_t$	Moment généré par la poussée des hélices
$\mathbf{M}_i$	Moment généré par la trainée des hélices

Notations mathématiques

$\cdot$	Produit scalaire
$\times$	Produit vectoriel
$\mathbb{R}^n$	Ensemble des vecteurs réels de dimension n
$\mathbb{H}$	Ensemble de tous les quaternions
S^3	3-sphère unitaire, représente aussi l'ensemble de tous les quaternions unitaires
$\hat{\mathbf{x}}$	Représentation matricielle du produit vectoriel $\mathbf{x}\times$
$\otimes$	Opérateur de rotation pour quaternions
$\lambda_m(\mathbf{A})$	Plus petite valeur propre de $\mathbf{A}$
x	Une variable scalaire
$\mathbf{x}$	Un vecteur
$\mathbf{X}$	Une matrice
$\mathbf{v}^1$	Expression d'un vecteur $\mathbf{v}$ dans le repère $\mathcal{F}_1$
$\mathbf{R}_{2/1}$	Rotation permettant le passage du repère $\mathcal{F}_1$ vers le repère $\mathcal{F}_2$

CHAPITRE 1 INTRODUCTION

1.1 Contexte

Dans la dernière décennie, le marché des drones a connu un essor majeur dans plusieurs domaines, tant civil, commercial qu'académique. Cette forte progression, participant à une démocratisation des multicoptères, s'explique notamment par leur dynamique relativement peu complexe par rapport aux autres types de véhicules aériens, leur simplicité mécanique, leur facilité de pilotage et leur faible coût de production. Parmi les nombreuses applications des multicoptères, on dénombre, entre autres, la livraison de colis, les opérations de recherche et de sauvetage, la cartographie et la surveillance de terrains, etc. Cette nouvelle technologie permet potentiellement de rendre ces services plus efficaces, mais aussi de les rendre disponibles aux régions lointaines ou inaccessibles (lieux de catastrophes naturelles, incendies, haute-mer, etc.).

Figure 1.1 Drone de course

Depuis 2015, un nouveau domaine d'intérêt a fait son apparition : les courses de drones. Suite à une hausse marquée de leur popularité, plusieurs ligues internationales ont été mises en place (*Drone Racing League*, *MultiGP*, *Drone Championship League*, etc.). Les drones sont alors pilotés à de très grandes vitesses dans un environnement complexe avec une vue à la première personne fournie par une caméra embarquée. Ces courses mettent en évidence la

supériorité des pilotes humains par rapport aux systèmes autonomes développés jusqu'à présent. Souhaitant diminuer l'écart de performance entre les vols pilotés et les vols autonomes, plusieurs compétitions ont été lancées afin de repousser les limites de l'autonomie des drones. Les premières compétitions ont eu lieu lors des conférences IROS 2016 et IROS 2017 : les participants étaient appelés à concevoir un système d'autonomie pouvant parcourir une piste de course intérieure composée de plusieurs cibles et de segments de difficulté variable. Les parcours étaient cependant petits et leurs caractéristiques connues à l'avance ; les vitesses en jeu n'étaient donc pas importantes et l'allure des trajectoires peu déterminante. C'est en 2019 que *Lockheed Martin* et *Drone Racing League* ont lancé le concours *AlphaPilot Challenge*[1]. Le but de ce défi est de concevoir un système de pilotage autonome pour parcourir une piste de course plus rapidement qu'un pilote humain, tout en ayant une puissance de calcul limitée et un nombre réduit de capteurs (une caméra et une centrale inertielle). Contrairement aux défis précédents, les pistes de courses sont beaucoup plus grandes et leur configuration exacte n'est pas connue par le pilote ou le système autonome ; les vitesses atteintes par le drone sont considérablement plus élevées et le problème de navigation devient d'autant plus complexe.

Figure 1.2 Compétition de drone autonome iROS 2017 (tiré de [1])

Jusqu'à présent, l'autonomie est très limitée et plusieurs capteurs sont nécessaires pour un vol sécuritaire. En effet, la grande majorité des conditions de vols autonomes sont à de basses vitesses avec des conditions minimales de vent et utilisent des contrôleurs linéaires avec une modélisation simplifiée du drone. Par ailleurs, ces contrôleurs sont souvent couplés avec des

1. `https://www.lockheedmartin.com/en-us/news/events/ai-innovation-challenge.html`

capteurs complexes et dispendieux (GPS, systèmes de capture de mouvement, LIDAR).

La piste d'une course de drone est particulièrement difficile à parcourir étant donné les hautes vitesses impliquées, l'environnement visuel très complexe (occlusions, flou de mouvement, basse luminosité, etc.), ainsi que les ressources informatiques limitées du drone. L'environnement visuel et la suite minimale de capteurs imposée posent d'importants défis pour la détection de cibles et pour l'estimation d'état. Au niveau du contrôle, les hautes vitesses et accélérations en jeu dépassent grandement les hypothèses simplificatrices de la commande classique (angles faibles de roulis et de tangage, condition de vol quasi-stationnaire, etc.) et rend leur performance imprévisible, instable et non-optimale. Les trajectoires générées en temps réel doivent être réalisables par la dynamique et les actionneurs du drone, et être les plus rapides possibles. Il faut aussi être capable de les recalculer de façon rapide et robuste étant donné l'environnement de course incertain et les ressources de calcul. Le contexte de course de drone autonome pose donc d'importants défis dans le développement et l'implémentation en temps réel d'algorithmes d'estimation d'état, de perception, de planification de trajectoires et de contrôle.

1.2 Objectifs de recherche

L'**objectif principal** de ce projet de recherche est **de développer un système autonome de contrôle et de navigation à hautes vitesses pour course de drones.**

Le **premier sous-objectif** de recherche est de synthétiser un contrôleur non-linéaire de suivi de trajectoire. Cette loi de commande sera conçue à partir de la modélisation complète de la dynamique d'un multicoptère. Elle aura pour but de garantir une stabilité asymptotique et une erreur de suivi nulle pour tout état et trajectoire dans le contexte hautement non-linéaire des courses de drone (hautes vitesses, angles d'inclinaison importants, grandes accélérations, etc.).

Le **deuxième sous-objectif** de recherche est de concevoir une méthode de génération de trajectoires optimales simples. Cette méthode sera conçue de sorte à être optimale en temps pour tous les segments de trajectoire, lisse et atteignable par le drone. De plus, cette méthode sera adaptée afin d'assurer la rapidité et la robustesse de calcul puisque la trajectoire est sujette à plusieurs changements considérant la nature incertaine d'une piste de course.

Le **troisième sous-objectif** de recherche est de concevoir un système de parcours de piste. À partir des détections des différentes balises dans le parcours, ce système devra cartographier leur position réelle et adapter la trajectoire que le drone devra suivre.

1.3 Plan du mémoire

Le chapitre 2 est consacré à une revue de littérature des divers sujets abordés dans ce mémoire. Une présentation des méthodes classiques de contrôle des multicoptères, ainsi que les méthodes plus récentes sera faite. Ensuite, une revue des différentes techniques de génération de trajectoires appliquées aux drones sera effectuée. Pour terminer, les plus récents et populaires systèmes de vol autonome seront introduits et décrits.

Dans le chapitre 3, la modélisation complète de la dynamique d'un multicoptère est faite. À partir de ce modèle non-linéaire, la propriété de platitude différentielle (*differential flatness*) du multicoptère, qui sera exploitée dans le Chapitre 5, est démontrée. Une description des environnements de tests sera présentée à la fin du chapitre.

À partir du modèle non-linéaire du multicopètre, on développe dans le chapitre 4 une loi de commande pour le suivi de trajectoire. La preuve de stabilité sur l'ensemble du domaine et l'ajustement des gains en vue de l'implémentation du contrôleur sur le système physique y sont présentés. La loi de commande est ensuite validée par le biais de simulations.

Le chapitre 5 présente la méthode de génération de trajectoires utilisée dans le système de parcours autonome. La formulation mathématique originale de la méthode, ainsi que les modifications robustes qui y sont apportées seront exposées.

Au chapitre 6, le système de parcours autonome d'une piste de course pour drone sera présenté. L'architecture générale du système et les différents algorithmes de détection, de cartographie et de gestion de trajectoire seront décrits. Ce système sera ensuite validé et analysé à différentes vitesses sur des pistes de courses de drones typiques.

Le chapitre 7 de conclusion permettra de faire un retour sur les objectifs de recherche et les différents résultats obtenus. Ce chapitre se termine par une ouverture sur les travaux futurs découlant de cette recherche et des limitations de la méthode proposée.

CHAPITRE 2 REVUE DE LITTÉRATURE

La revue de littérature est divisée selon la structure du mémoire. Tout d'abord, la section 2.1 est dédiée à la modélisation des multicoptères. Par la suite, une revue des différentes méthodes de commande utilisées pour ces systèmes est faite à la section 2.2. La section 2.3 présente les différentes méthodes de génération de trajectoire utilisées pour les drones. Pour terminer, une sélection d'algorithmes d'autonomie pour courses de drone seront énumérés et analysés brièvement à la section 2.4.

2.1 Modélisation d'un multicoptère

La modélisation de la dynamique des multicoptères est un problème qui a été largement étudié au cours des dernières années. Bien qu'il reste encore place à amélioration quant à la modélisation des effets aérodynamiques, le modèle mathématique de base est bien établi. Les travaux marquants sur le sujet ont été faits par [4], [5] et [6]. Ces travaux soulignent la dynamique hautement non-linéaire des multicoptères, ainsi que les divers couplages de la dynamique. Étant un système sous-actionné, chaque degré de liberté ne peut pas être commandé indépendamment. En effet, il ressort que la dynamique en translation horizontale est couplée à celle du roulis et du tangage, alors que l'altitude et le lacet du drone peuvent être commandés indépendamment. Une autre caractéristique importante soulevée par ces travaux est le fait qu'il est possible de contrôler le multicoptère par des entrées virtuelles plutôt que par les vitesses de rotation des moteurs en utilisant une matrice d'allocation ; ceci simplifie grandement les calculs pour les lois de commande.

Outre l'étude de la dynamique de ces robots aériens, plusieurs travaux se sont intéressés à la modélisation des effets aérodynamiques secondaires agissant sur les multicoptères. Dans [5], on décrit le phénomène de battement des hélices (*blade flapping*) qui affecte l'orientation de la poussée de chaque rotor. Dans [7], la traînée produite par les hélices est modélisée comme une force proportionnelle à la vitesse et l'orientation du drone. Certains articles comme [8] ont même modélisé l'effet du vent sur la dynamique du multicoptère. Ceci dit, ces effets ne sont généralement pas utilisés dans le modèle pour établir les lois de commande vu qu'ils ne sont généralement pas significatifs dans la majorité des conditions de vol. Dans ce cas, on suppose que les erreurs de modélisation peuvent être compensées par exemple par une loi de commande augmentée avec l'intégrale de l'erreur de suivi.

2.2 Méthodes de commande

Le premier but de la commande des multicoptères est de contrôler l'attitude du véhicule. Dans cette situation, les drones sont alors pilotés en utilisant une manette. En utilisant les joysticks de la manette, le pilote contrôle directement la poussée totale du drone et envoie des commandes d'attitude à la boucle de contrôle interne. La poussée commandée par le pilote et les consignes du correcteur d'attitude sont ensuite mixées dans la matrice d'allocation et une commande est envoyée à chaque moteur.

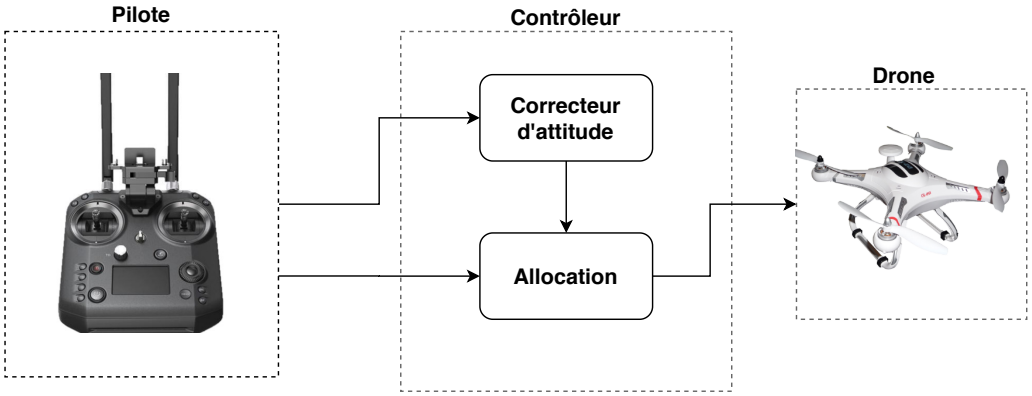


Figure 2.1 Contrôle de drone par pilotage

Pour un vol complètement autonome, l'architecture typique est d'ajouter une boucle externe pour la commande en position. Celle-ci fournit des consignes à la boucle interne qui elle s'assure de contrôler l'attitude. Ce choix est motivé par le couplage naturel du multicoptère et la dynamique plus rapide de la boucle interne par rapport à la boucle externe [9]. Les premières méthodes de contrôle en position à être appliquées sur les multicoptères ont été les méthodes classiques par contrôleur PID (*Proportional Integral Derivative*) et par contrôleur optimal LQR (*Linear Quadratic Regulator*). Une fois les méthodes linéaires explorées, plusieurs travaux se sont concentrés sur l'application de méthodes non-linéaires afin d'élargir l'enveloppe de vol et d'améliorer les performances de suivi des multicoptères.

2.2.1 Commande classique linéaire

Les méthodes de commande classiques émanant de la théorie de commande linéaire, ont été les premières méthodes à être utilisées sur les multicoptères. Parmi elles, le réglage de correcteurs PID demeure la première de par son réglage facile et intuitif, sa bonne performance et sa relative robustesse. La méthodologie de conception de loi de commande utilisant ce type de correcteur est la suivante :

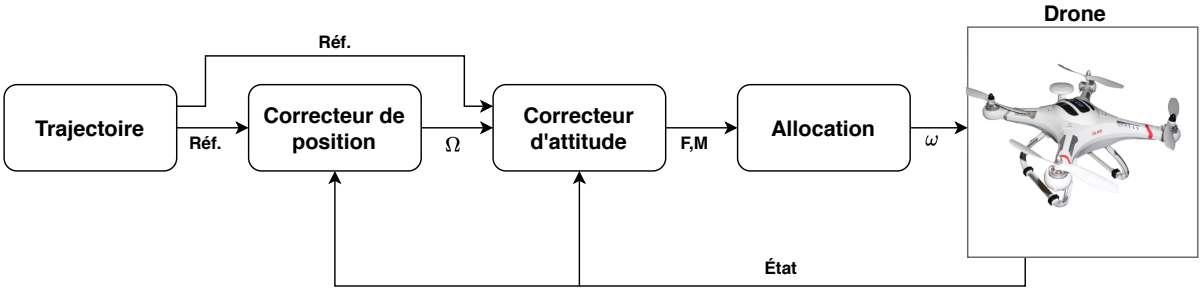


Figure 2.2 Architecture de commande typique pour multicoptères

— linéariser les équations dynamiques autour du vol stationnaire pour les réduire à un système LTI (*Linear Time Invariant*) ;
— découpler le système MIMO (*Multiple Input Multiple Output*) en plusieurs sous systèmes SISO (*Single Input Single Output*) ;
— appliquer les méthodes de synthèse de la commande classique (transformation dans le domaine de Laplace, lieu des racines, diagramme de Bode, etc.) pour concevoir un correcteur pour chaque sous-système.

Bien que cette technique soit suffisante pour le problème de stabilité en vol stationnaire, on constate que les performances se dégradent rapidement sur le système non-linéaire dès que l'on s'éloigne de cette condition. Ceci peut s'expliquer en partie par l'effet de cascade provenant de la conception indépendante des correcteurs pour chaque sous-système.

Une autre méthode linéaire couramment utilisée pour la commande de multicoptère est la commande LQR. Cette méthode de commande optimale a été développée dans les années 1960 par Rudolf Kalman [10]. Elle consiste à trouver le gain optimal qui minimise un problème d'optimisation quadratique. La fonction de coût associe un poids à chaque état et aux efforts de commande. Selon l'objectif de commande, le concepteur modifie les poids accordés aux états et aux entrées pour obtenir le comportement désiré. Les gains optimaux sont trouvés en résolvant une équation algébrique de Riccati. Les performances de suivi et de minimisation d'efforts de commande de la commande LQR sont en général meilleures que celles de la commande par PID.

À titre d'exemple, on peut citer [11], où un drone est stabilisé en vol stationnaire en implémentant chacune des deux méthodes de commande par voie de simulation et de vols expérimentaux. Dans [8], les deux approches sont aussi utilisées pour stabiliser un drone soumis à un vent de 5 m/s. Pour ce qui est de la performance en suivi, les travaux de [12] montrent que les deux méthodes offrent de très bonnes performances pour le suivi de tra-

jectoires lentes, avec cependant un avantage pour la commande LQR dès que la vitesse de parcours augmente.

Bien que les performances présentées dans les travaux cités ci-dessus soient bonnes, ces résultats ne sont pas assurés pour toute l'enveloppe de vol d'un multicoptère, i.e., pour de grands angles de roulis et de tangage. En effet, les caractéristiques utilisées pour la conception des lois de commandes proviennent de la linéarisation autour du vol stationnaire, soit un roulis et un tangage nuls. Ces hypothèses ne sont plus valides pour des angles de tangage ou de roulis supérieurs à 30 degrés, ce qui peut correspondre à des vitesses maximales d'environ 2 m/s selon le drone [9]. La performance et la stabilité de ces techniques de commande ne peuvent donc pas être garanties pour des scénarios nécessitant plus d'agilité lors du suivi de trajectoires.

2.2.2 Commande non-linéaire

Les limitations des techniques de commande linéaires étant bien connues, plusieurs travaux se sont intéressés à l'utilisation de lois de commande non-linéaires pour les tâches de stabilisation et de suivi de trajectoire. Les premières techniques à être appliquées ont été le *Backstepping* et la commande par mode de glissement (*Sliding Mode Control*).

Le *Backstepping* est une méthode de commande non-linéaire utilisable pour des systèmes présentant certaines propriétés. Cette technique récursive consiste à subdiviser le système principal en plusieurs sous-systèmes intermédiaires jusqu'à obtenir un sous-système irréductible. Chaque sous-système est ensuite conçu pour stabiliser le sous-système qui le précède jusqu'à obtenir un système complet stable. Quant à la commande par mode de glissement, elle consiste à appliquer un signal discontinu au système de sorte à le restreindre à « glisser» sur une surface de commande qui donne au système le comportement désiré. Des détails supplémentaires et des notions de stabilité avancées sont présentés dans [13]. Les travaux de [14] sont parmi les premiers à intégrer ces méthodes pour la commande d'un multicoptère. Dans [15], ces deux techniques de commande sont appliquées à un quadricoptère et comparées à une commande de type PID par voie de simulation. Malgré la bonne performance de suivi, ces méthodes de commande offrent de moins bonnes performances que le PID en termes d'erreur en régime permanent et de temps de réponse. Les auteurs ont aussi noté l'importance du phénomène de broutement (*chattering*) lors de l'utilisation de la méthode de commande par mode de glissement. Dans [16], une surface glissante de deuxième ordre basée sur la commande PID est utilisée pour contrevenir à ce problème de broutement.

Une autre méthode de commande non-linéaire populaire est la commande prédictive ou MPC (*Model Predictive Control*). Similairement à la commande optimale LQR, cette méthode

consiste à résoudre un problème de commande optimale pour suivre une trajectoire donnée. Cependant, à la différence de la méthode LQR, les commandes optimales sont calculées en ligne par la résolution d'un problème d'optimisation sur un horizon de temps T fini. Ces calculs sont effectués à chaque pas de temps et seulement la première commande trouvée est utilisée durant la période d'échantillonnage. Dans [17], un correcteur MPC linéaire et un correcteur MPC non-linéaire sont conçus et implémentés sur un hexacoptère. Le correcteur MPC non-linéaire offre de meilleures performances en suivi que le linéaire et est capable de suivre une trajectoire assez rapide (plus de 3 m/s) avec très peu d'erreur. Ceci dit, la commande MPC a plusieurs désavantages, le plus important étant qu'elle est très coûteuse au niveau des ressources informatiques. En effet, ce correcteur ne peut pas être utilisé de concert à un algorithme de SLAM (*Simultaneous Localization And Mapping*) ou de VIO (*Visual Inertial Odometry*) s'il est implémenté sur l'ordinateur embarqué d'un drone. Par ailleurs, la méthode de commande MPC n'est que localement optimale et n'a aucune garantie de stabilité puisque le problème d'optimisation non-linéaire est résolu de façon numérique.

Dans [18], les auteurs présentent la commande géométrique pour multicoptère qui rend le système presque globalement exponentiellement stable. La preuve de stabilité complète est fournie, ainsi que des simulations d'un drone qui se stabilise à partir d'une orientation presque renversée. Les performances de suivi de trajectoire sont remarquables en simulation, mais cette loi de contrôle ne peut pas être utilisée en pratique. En effet, l'erreur en *jerk* (troisième dérivée de la position) est nécessaire pour un terme de précommande et il n'existe présentement pas de méthode assez précise pour en faire l'estimation. Une version modifiée de cette loi de commande est présentée dans [19]. Bien que cette nouvelle formulation ne puisse pas garantir la stabilité du système, elle offre des bonnes performances de suivi. Dans ses travaux, [19] implémente cette loi de commande géométrique sur un quadricoptère qui passe au travers d'un cerceau lancé : il obtient une erreur de suivi maximale en dessous de 10 cm.

Dans [20], le contrôleur en position de [18] est repris et couplé avec le contrôleur d'attitude globalement asymptotiquement stable présenté dans [21] pour faire le contrôle d'un quadricoptère qui se réinitialise après une défaillance. Dans les travaux expérimentaux, le drone est lancé avec une grande force et avec une vitesse angulaire élevée. En utilisant son odométrie et la loi de commande, le drone arrive à se stabiliser. Cette loi de commande est reprise dans [22] pour faire le suivi d'un vol agressif à travers une fenêtre étroite. Les erreurs angulaires et en position moyennes sont d'environ 6 cm et 11 degrés pour des vols de 3 m/s et pouvant atteindre des vitesses angulaires de 400 °/s. Dans ces deux travaux, aucun système de capture de mouvement n'est utilisé pour faire l'estimation d'état du système.

2.3 Planification de trajectoires

Les méthodes traditionnelles de recherche de trajectoire telles que A*, D* ou RRT (*Rapidly exploring Random Trees*) ne sont pas adéquates pour les multicoptères [9]. En effet, l'optimalité de ces algorithmes est dépendante de la résolution de la grille de points et sa complexité augmente exponentiellement avec la taille du problème. Sachant cela, la méthode n'est pas appropriée dans le cas d'un système à 12 états tel que le multicoptère. En revanche, les méthodes optimales s'avèrent être les plus performantes.

Le problème général d'optimisation de trajectoire est le suivant :

$$\min_{\mathbf{u} \in \mathbf{U}_{feas}} \kappa(T, \mathbf{x}_f) + \int_0^T L(t, \mathbf{x}(t), \mathbf{u}(t)\mathrm{d}t \tag{2.1}$$

$$\text{s.c.} \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \tag{2.2}$$

$$\mathbf{x}(t) \in \mathbf{X}_{feas}, \forall t \in [0, T], \tag{2.3}$$

$$\mathbf{x}(0) = \mathbf{x}_i, \tag{2.4}$$

$$\mathbf{x}(T) = \mathbf{x}_f \tag{2.5}$$

avec $\mathbf{x}$ et $\mathbf{u}$ l'état et l'entrée du système décrit par la dynamique $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$, $\mathbf{X}_{feas}$ et $\mathbf{U}_{feas}$ les ensembles de fonctions contenant les états et les entrées possibles, T le temps total de la trajectoire, $L(t, \mathbf{x}(t), \mathbf{u}(t))$ la fonction de coût à optimiser, κ la fonction représentant le poids accordé à l'état final, ainsi que $\mathbf{x}_i$ et $\mathbf{x}_f$ les états initiaux et finaux. Ce problème n'est généralement pas résoluble analytiquement et les solutions numériques sont très longues à produire [9]. Dans la plupart des simplifications de ce problème, les équations dynamiques du multicoptère sont d'ailleurs rarement intégralement considérées, mais des limites sur les accélérations et les vitesses angulaires sont imposées pour représenter les contraintes.

Dans [23], les auteurs trouvent une trajectoire entre deux points en minimisant le temps total de parcours T via des *B-splines*. Pour trouver la trajectoire optimale, ils définissent une trajectoire $\{x, y, z, \psi\}$ par une fonction $\mathbf{P}(\lambda(t))$ qui est une forme paramétrisée de la trajectoire et où $\lambda(t) \in [0, 1],\ t \in [0, T]$ définit la dynamique au sein de la trajectoire. Cette forme paramétrique $\mathbf{P}$ est définie par un *B-spline* qui passe par des points de contrôle trouvés selon l'espace libre dans l'environnement. La dynamique λ est aussi trouvée en utilisant un *B-spline* avec $N \geq 5$ points de contrôle répartis également sur le domaine $[0, T]$. Les inconnues du problème d'optimisation à résoudre sont le temps total T et les coordonnées des points de contrôle pour $\mathbf{P}(\lambda)$ et $\lambda(t)$. La solution est trouvée numériquement avec un solveur non-linéaire en itérant le temps total et en vérifiant si les limites des actionneurs sont atteintes (vitesses angulaires des rotors). Les auteurs arrivent à générer des trajectoires pour

des environnements aussi bien simples que complexes et qui respectent les contraintes du multicoptère. Ceci dit, aucune information concernant le temps d'exécution est mentionné et l'algorithme ne fonctionne que pour des conditions initiales et finales au repos, ce qui n'est pas idéal lorsqu'une trajectoire doit être planifiée en temps réel pour des conditions initiales et finales variables.

Dans [24], une méthode de génération de trajectoire optimale en temps est faite à partir d'une dynamique découplée et simplifiée d'un multicoptère. Afin de pouvoir utiliser le *jerk* (troisième dérivée de la position) comme entrée au système, les auteurs simplifient la dynamique en translation selon chaque direction sous la forme :

$$\dot{\mathbf{s}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{s} + \begin{bmatrix} 0 \\ 0 \\ u \end{bmatrix}$$

avec $\mathbf{s} = \begin{bmatrix} x & \dot{x} & \ddot{x} \end{bmatrix}^\top$ et $x, \dot{x}, \ddot{x}$ représentant respectivement la position, la vitesse et l'accélération selon une direction donnée, et sous les contraintes :

$$\begin{aligned} \mathbf{s}(0) &= \mathbf{s}_0 \\ \mathbf{s}(t_f) &= \mathbf{0} \\ |u| &\leq u_{max} \\ |\ddot{x}| &\leq \ddot{x}_{max} \end{aligned}$$

La dynamique du multicoptère n'est donc pas directement exprimée dans le problème d'optimisation, mais plutôt par des contraintes maximales d'accélération et de vitesses angulaires. La solution au problème de temps minimal est :

$$u^* = \text{argmin } T_f$$

avec u^* la commande optimale et T_f le temps total de parcours. Par le biais de plusieurs

théorèmes, les auteurs montrent que la réponse optimale est composée de cinq segments :

$$u^* = \begin{cases} \pm u_{max} & \text{si } t = [0,\ T_1] \\ 0 & \text{si } t = [T_1,\ T_2] \\ \mp u_{max} & \text{si } t = [T_2,\ T_3] \\ 0 & \text{si } t = [T_3,\ T_4] \\ \pm u_{max} & \text{si } t = [T_4,\ T_f] \end{cases}$$

avec T_i représentant les temps de changement de segment.

En utilisant la dynamique simplifiée du multicoptère et des contraintes sur les différentes séquences de temps, la solution optimale est trouvée en résolvant un système linéaire de cinq équations. Selon les essais expérimentaux des auteurs, la solution à ce problème est trouvée en 0.2 ms sur un ordinateur portable typique. Après avoir trouvé la solution optimale, une vérification des valeurs limites des actionneurs est faite. Ces limites sont exprimées par une valeur maximale absolue à ne pas dépasser. Si les limites sont dépassées, la valeur de u_{max} est diminuée de façon itérative jusqu'à l'obtention d'une solution satisfaisante. Dans leurs travaux expérimentaux, les auteurs utilisent le drone *Asctec Hummingbird* pour suivre les trajectoires générées par leur algorithme. Le quadrirotor réussit à bien suivre des trajectoires agressives ayant une limite $u_{max} = 50$ m/s^3 et $\ddot{x}_{max} = \ddot{y}_{max} = \ddot{z}_{max} = 7$ m/s^2. Les auteurs notent cependant quelques limitations à leur algorithme. Parmi celles-ci, on mentionne l'absence de synchronisation des temps finaux pour chaque composante x, y et z de la trajectoire. Par ailleurs, tout comme dans les travaux de [23], l'état final doit être au repos pour que cette méthode fonctionne.

Dans [19] et [25], les auteurs montrent qu'un multicoptère a la propriété de platitude différentielle (*differential flatness*) pour une modélisation avec et sans trainée et que ses sorties plates sont x, y, z et ψ (l'angle de lacet du drone, définition plus précise à la section 3). Lorsqu'un système est différentiellement plat, on peut exprimer tous les états d'un système en fonction de ses sorties plates et les dérivées de celles-ci. Dans le cas d'un multicoptère, le robot est garanti de suivre toute trajectoire $\{x, y, z, \psi\}$ suffisamment dérivable. Plus spécifiquement, les trajectoires x, y et z doivent être dérivables au moins quatre fois et la trajectoire ψ doit l'être au moins deux fois. Cette propriété est importante puisque toute trajectoire de référence suffisamment dérivable respecte implicitement la dynamique du drone. Les contraintes dynamiques n'ont donc plus besoin d'être incluses dans le problème d'optimisation, ce qui diminue la complexité du problème.

La méthode la plus populaire de génération de trajectoire dérive de cette conclusion. Dans [19], on propose de trouver les coefficients de segments polynomiaux minimisant l'intégrale du *snap*[1] au carré tout le long de la trajectoire plutôt que de minimiser le temps total. Le *snap* est minimisé car il est liée aux entrées par la propriété de platitude différentielle et il permet d'obtenir une trajectoire lisse. La solution trouvée assure donc une minimisation indirecte des efforts de commande et une trajectoire lisse. À partir de points de passage par lesquels la trajectoire doit passer, l'auteur reformule la fonction de coût en un problème quadratique et le résout à l'aide d'une méthode numérique. Les travaux de [26] reprennent ces derniers travaux, mais reformulent le problème afin d'améliorer la robustesse numérique. Les conditions limites libres sont optimisées plutôt que les coefficients des segments polynomiaux. Cette nouvelle méthode est ensuite utilisée avec une méthode RRT pour trouver une trajectoire dans un environnement 3D complexe. Des points de passages libres d'obstacles sont trouvés avec l'algorithme RRT et ensuite une trajectoire optimale passant par ces points est trouvée avec des segments polynomiaux minimisant le *snap*. L'article [26] propose une méthode d'optimisation du temps total de parcours, ainsi que le temps alloué pour chaque segment tout en minimisant l'intégrale du carré du *snap* tout le long de la trajectoire. Le problème est formulé comme :

$$\begin{aligned} \min_{\mathbf{T}} \quad & \int_0^{T_f} [\zeta^{(4)}(t)]^2 \mathrm{d}t + k_t \sum_{i=0}^{n} T_i \\ \text{t.q.} \quad & T_1, T_2, ..., T_n \geq 0 \end{aligned}$$

avec $\mathbf{T}$ le vecteur des différents segments de temps T_i, T_f la somme des temps des segments T_i, $\zeta(t)$ la trajectoire de référence selon un axe x, y, z ou ψ, $[\zeta^{(4)}(t)]^2$ le carré du *snap* le long de la trajectoire unidimensionnelle x, y, z ou ψ, ainsi que k_t le poids accordé au temps total dans l'optimisation et n le nombre total de segments de trajectoire.

2.4 Courses de drones autonomes

Dans les dernières années, les courses de drones ont connu une importante montée en popularité. Ce sport a été créé par des hobbyistes qui s'amusaient avec leurs propres drones [27]. L'objectif d'une course typique est de franchir des balises selon une orientation et un ordre prédéterminés le plus rapidement possible. Le drone est piloté à distance en utilisant un système visuel de type vue à la première personne. La grande popularité du sport a mené à la création de plusieurs ligues professionnelles internationales telles que *MultiGP*, *DR1* et *DRL* avec des prix de plusieurs centaines de milliers de dollars [28]. La popularité de ce sport a mis

1. Quatrième dérivée de la position.

en lumière l'agilité et les performances phénoménales des drones. Ceci a piqué l'intérêt dans le domaine de la recherche. En effet, les systèmes d'autonomie étaient très loin d'atteindre les performances démontrées par les pilotes. Le problème de pilotage autonome agressif est d'importance pour le milieu de la recherche étant donné la charge utile limitée d'un drone, les très hautes vitesses linéaires et angulaires atteintes, les ressources de calcul limitées et l'absence de capteurs extéroceptifs. Les algorithmes utilisés doivent donc être précis, être exécutés en temps réel et ne pas être demandants en termes de ressources de calcul sinon une collision avec l'environnement ou une perte de contrôle sont inévitables. Plusieurs ensembles de données ont été conçus pour développer les algorithmes d'estimation d'état visuels ([29–32]) et plusieurs compétitions ont eu lieu afin de développer les algorithmes d'autonomie. La première de ces compétitions a eu lieu lors de la conférence IROS en 2016 [33]. Les compétiteurs étaient appelés à parcourir une piste connue à l'avance de façon autonome en utilisant seulement des algorithmes et capteurs sur un système embarqué. Depuis, les compétitions ont lieu chaque année dans les conférences IROS et ont mené à la création d'autres compétitions comme *DARPA Fast Lightweight Autonomy Program* [34] et *AlphaPilot* [2].

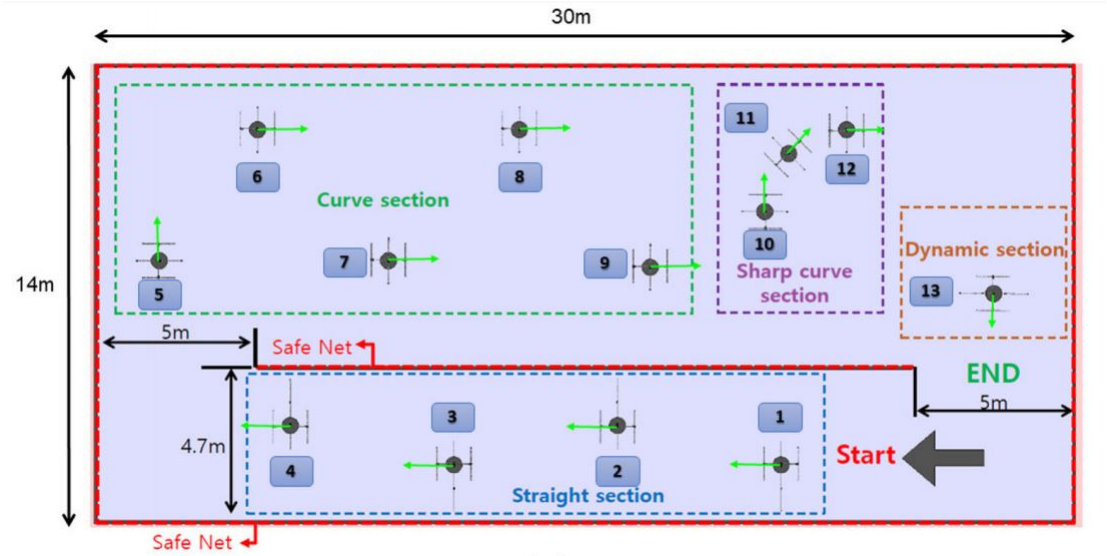


Figure 2.3 Disposition de la piste dans la conférence iROS 2017 (tiré de [1])

Les sous-sections suivantes présentent les travaux précurseurs de courses de drones, les premières méthodes utilisées et l'état de l'art concernant la course de drones autonomes. Il est à noter qu'on se concentre principalement sur les méthodes de navigation et de contrôle puisqu'il s'agit du sujet de ce mémoire, mais que d'autres aspects tels que l'estimation d'état et la détection doivent parfois être mentionnés puisqu'ils sont intrinsèques à certaines méthodes.

2.4.1 Premiers travaux

On s'intéresse d'abord aux travaux qui ont précédé la course de drones autonomes, soit la traversée d'une fenêtre étroite de façon agressive.

Dans [35], une loi de commande et une trajectoire adaptative sont utilisées pour permettre à un drone de passer au travers de fenêtres horizontales et verticales à des angles de 45°, 60°, 75° et 90°. Le processus s'effectue en plusieurs étapes séquencées. Tout d'abord, le drone est maintenu en vol stationnaire par un correcteur spécifique. Ensuite, une commande en position et en vitesse coïncidant avec le centre de la fenêtre est envoyée et suivie par un correcteur de suivi de trajectoire. Lorsque le drone se trouve proche de la fenêtre, une commande en attitude correspondant à l'orientation de la fenêtre est envoyée et suivie par un correcteur d'attitude. Une fois la balise traversée, l'attitude du drone est commandée à zéro. En reprenant le correcteur de la première étape, le drone est maintenu en vol stationnaire. L'auteur propose aussi une méthode itérative d'ajustement de la commande en position et en vitesse pour traverser la fenêtre afin de diminuer l'erreur de suivi. Bien que les performances de suivi soient bonnes (erreurs en vitesses linéaires et angulaires de moins de 0.1 m/s et de 2°), la méthode de commande est complexe (plusieurs lois de commande et séquencement selon le segment), elle ne considère qu'une seule fenêtre dont la position et l'orientation sont connues à l'avance, et la trajectoire à suivre est déterminée de façon empirique. Il n'est pas envisageable d'utiliser cette technique pour faire le parcours d'une piste de course.

Dans [22], des travaux similaires sont faits, mais en utilisant un seul correcteur et sans utiliser un système de capture de mouvement pour faire l'estimation d'état. Par ailleurs, la position et l'orientation de la fenêtre ne sont pas connues à l'avance et sont trouvés en ligne (pendant l'exécution de la manoeuvre). Pour ce faire, la fenêtre est conçue de sorte à avoir des dimensions spécifiques et des caractéristiques visuelles particulières. La fenêtre peut ensuite être trouvée pendant la manœuvre à l'aide d'algorithmes de reconnaissance d'images simples. Une fois la position et l'orientation de la fenêtre trouvées, une trajectoire passant par le centre de la fenêtre maximisant la distance entre les bordures de la fenêtre et le quadricoptère est calculée et suivie. Trente-cinq essais expérimentaux avec diverses orientations de fenêtre sont effectuées avec un pourcentage de réussite d'environ 80% et une erreur de suivi moyenne d'environ 6.4 cm. Ceci dit, il est montré dans [36] que cette méthode ne fonctionne pas pour plus d'une fenêtre.

2.4.2 Premières approches

Les premières approches au problème de courses de drones autonomes étaient relativement simples. Ces systèmes étaient robustes à différentes dispositions de la piste, mais étaient lents. Dans cette sous-section, une sélection de méthodes utilisées dans les premières compétitions de course autonomes de drones sont présentées.

Tout d'abord, nous présentons la méthode utilisée dans [37]. La stratégie de l'équipe était la suivante : détecter la prochaine balise et s'y rendre en utilisant un algorithme de navigation de type ligne de vue. La détection de balise se fait par un réseau de neurones convolutif développé par l'équipe (*ADRNet*) qui se base sur les réseaux SSD (*Single Shot Detection*) [38] et *AlexNet* [39]. Ce réseau de neurones permet de faire la détection de balises sur une plateforme embarquée à une vitesse d'environ 30 images par seconde. Le réseau est entraîné avec diverses images des balises utilisées dans la compétition. Cette nouvelle méthode a un taux de détection de 90%, ce qui est une nette amélioration au taux de détection de 50% de la méthode basée sur la couleur utilisée dans leurs efforts précédents [40]. Leurs algorithmes utilisent la carte de la piste pour orienter le drone entre chaque balise afin qu'il puisse détecter la prochaine balise à atteindre. L'objectif de l'algorithme de navigation est d'aligner le lacet du drone avec celui de la balise, ainsi que de faire passer le drone par le centre de la balise. Pour ce faire, l'algorithme de vision envoie les coordonnées du centre de la balise dans l'image à l'algorithme de navigation. L'erreur en pixels entre le centre de l'image et le centre de la balise est ensuite calculée ; de cette erreur en pixels, les erreurs en lacet et latérale sont calculées, puis fournies à une loi de commande de taux de lacet et de taux de roulis. Cette stratégie de parcours autonome est validée sur plusieurs pistes aléatoires de neuf balises avec des erreurs latérales de suivi maximales de 0.1 m. Ceci dit, bien que cette méthode soit robuste et réussit à parcourir une multitude de pistes, elle le fait à de très basses vitesses. Cette méthode est donc inefficace dans le contexte de courses de drones dans laquelle les vitesses en jeu sont élevées.

En second lieu, nous présentons la méthode utilisée dans [41]. Dans ces travaux, une méthode de navigation en deux étapes est utilisée. La première de ces étapes est la manœuvre de virage amorcée lorsque le drone traverse une balise. Pendant ce segment, le drone suit un arc calculé au préalable jusqu'à ce qu'il ait une vue sur la balise. Lorsque la balise est détectée, le drone se dirige vers la balise de sorte à rejoindre un segment droit aligné avec son centre. Le drone est alors en mesure de compléter l'ensemble du parcours en combinant les trajectoires entre chaque balise. L'estimation d'état se fait avec un filtre de Kalman étendu en utilisant les mesures de la centrale inertielle. Les auteurs utilisent l'accéléromètre pour mesurer la traînée sur le plan x-y du drone et établissent une relation linéaire entre la traînée mesurée et la

vitesse du drone ; l'estimation de la vitesse dans le filtre de Kalman est ainsi faite à partir de cette relation pour obtenir une mesure plus précise. L'algorithme d'estimation d'état utilise aussi les mesures d'un module de détection de balises avec une carte de la piste pour faire l'étape de mise à jour du filtre. La commande du drone est faite selon la structure classique présentée à la section 2.2 : la loi de commande de la boucle interne est tirée de [42] et la boucle de commande externe est, quant à elle, séparée en deux parties selon le segment de trajectoire à exécuter, soit lors du virage ou lors du segment droit. Puisqu'il n'y a pas d'estimation d'état précise lors de la manœuvre de virage (étant donné que la balise est hors de vue), la boucle externe ne comprend que des termes de précommande pour effectuer cet arc. Lorsque le drone est rendu dans le segment droit final, un correcteur de type PD (*Proportional Derivative*) est utilisé pour commander le lacet et l'erreur latérale. Il est à noter que dans les deux cas, une commande de tangage constante est envoyée à la boucle interne pour avoir une vitesse constante. Les auteurs ont validé leur système d'autonomie avec des essais expérimentaux sur une piste composée de cinq balises de petite taille et d'un environnement plus complexe que celui proposé dans la conférence IROS 2017. Les auteurs mentionnent que leur algorithme réussit à parcourir la piste à une vitesse de 1.8 m/s à presque tous les essais. Ils attribuent les non-réussites à la basse résolution de la caméra et l'environnement sombre de la piste lors des essais.

2.4.3 État de l'art

Nous présentons les trois méthodes les plus performantes trouvées dans la littérature. Contrairement aux méthodes présentées à la sous-section précédente, les méthodes de navigation et les lois de commande sont plus sophistiquées. Entre autres, des trajectoires optimales sont utilisées plutôt que des méthodes empiriques. La vitesse de parcours de piste est donc considérablement plus grande tout en ayant une certaine robustesse.

Les travaux de [3] présentent un système de parcours autonome de course ne nécessitant qu'une approximation grossière de la piste. La méthode proposée est sous-divisée en quatre modules : la détection de balises, la cartographie des balises, le calcul de trajectoire et la loi de commande. La détection de balises se fait à l'aide d'un réseau de neurones convolutif basé sur l'architecture de *DroNet* [43]. Ce réseau de neurones détermine la position et l'orientation en lacet de la balise par rapport au drone et fournit la covariance de ces mesures. La cartographie des balises est faite en faisant la fusion de l'odométrie et de la sortie du module de détection. Pour faire cette fusion, ils utilisent un filtre de Kalman étendu pour chacune des balises ; ce filtre est aussi utilisé pour compenser la dérive de l'estimation d'état. La génération de trajectoires et la commande du drone se font simultanément. La trajectoire est ainsi

calculée à partir d'une interpolation entre la position actuelle du drone et des points situés à une distance d de part et d'autre de la balise,alors que la méthode de commande utilisée est le MPC. Les auteurs valident ce système en simulation et de façon expérimentale. En simulation, la méthode est comparée à [37] et obtient de bien meilleurs résultats. Le système est validé sur plusieurs pistes avec un taux de réussite élevé jusqu'à une vitesse de 4 m/s. Expérimentalement, des résultats similaires sont obtenus, mais pour une vitesse maximale de 3 m/s.

Dans [44], une méthode par apprentissage est utilisée pour faire la navigation de drone dans un parcours de course. L'algorithme d'apprentissage est entraîné à imiter un drone suivant une trajectoire optimale pour parcourir une piste. Ce réseau de neurones convolutif est basé sur l'architecture de [43] et retourne un scalaire $v \in [0, 1]$ représentant la vitesse normalisée désirée et un vecteur $\mathbf{x} \in [-1, 1]^2$ qui représente la direction désirée en coordonnées d'image normalisées. La vitesse normalisée v est convertie en v_{des} en multipliant par un facteur v_{max}. La direction $\mathbf{x}$ est projetée le long du rayon de projection de la caméra sur une profondeur d pour trouver le point objectif p_g dans le repère inertiel. Le paramètre v_{max} est ajusté manuellement selon la disposition du parcours et le paramètre d est choisi proportionnellement à v. Ensuite, une trajectoire t_s minimisant le *jerk* entre la position du drone et le point p_g est calculée. Le correcteur envoie ensuite des commandes moteurs pour suivre t_s. La loi de commande utilisée est la même que [20]. En simulation, la méthode d'autonomie parcourt une piste de 8 balises et d'une longueur totale de 116 m de façon robuste pour des vitesses jusqu'à $v_{max} = 9$ m/s . Des essais expérimentaux ont validé la performance sur un système physique, mais à plus basse vitesse. Une extension de ces travaux est faite dans [44] en faisant l'entraînement dans différents environnements et avec des formes de balises variées pour rendre la méthode plus robuste.

Les travaux faits dans [45] sont la suite de ceux faits dans [3]. Plutôt que d'estimer la position des balises dans le repère inertiel, les coins et le contour de balises sont trouvés par un réseau de neurones convolutif basé sur [46] et [47]. Les auteurs choisissent de détecter des coins plutôt que d'estimer la distance entre la balise et le drone puisque plusieurs balises peuvent être détectées simultanément et les mesures sont plus faciles à intégrer dans le modèle d'estimation d'état. L'état du drone et de chaque balise sont estimés dans un filtre de Kalman étendu. Les états du drone sont propagés avec les données de la centrale inertielle et sont corrigés avec les mesures d'un système d'odométrie visuelle. L'état des balises est considéré comme une constante ayant un biais et est mise à jour avec la sortie du réseau de neurones. La loi de commande est similaire à celle dans [48], mais seulement la boucle externe est utilisée puisqu'un correcteur pour la boucle interne est déjà implémenté sur la plateforme expérimentale. Pour générer des trajectoires, les auteurs cherchent l'entrée

minimisant le temps total de la trajectoire tout en respectant une accélération et une vitesse maximales. Pour réduire la complexité du problème, les auteurs considèrent le drone comme une masse ponctuelle. Avec cette dynamique simplifiée, les auteurs prouvent, avec le principe du maximum de Pontryagin, que la solution optimale au problème est une commande de type *bang-cruise-bang*. Le temps le plus lent entre x, y et z est choisi comme le temps minimal et les trajectoires le long des autres dimensions sont recalculées avec ce temps. Lors du parcours d'une piste, la trajectoire de référence pour les N prochaines balises est recalculée lorsque l'état du drone ou des balises dans l'estimateur d'état varie plus qu'un certain seuil ou lorsque le drone traverse une balise. Lors du recalcul de trajectoire, M trajectoires aléatoires passant par le centre de chaque balise sont créées en contraignant l'angle entre la vitesse du drone et l'orientation de la balise à moins de 30 degrés pour les N balises. L'algorithme de Dijkstra [49] est ensuite utilisé pour trouver la solution au temps minimal des M trajectoires générées. Ce système a été utilisé pour la finale de la compétition *AlphaPilot* et a réussi le parcours de piste avec des vitesses moyenne et maximale de 6.5 m/s et 8 m/s respectivement. Au mieux de nos connaissances, cette méthode est la plus performante dans la littérature. Ceci peut s'expliquer par le fait que cette méthode utilise l'information sur la position des prochaines balises pour générer une trajectoire. En effet, les méthodes présentées dans [3] et [44] ne prévoient leur trajectoire qu'en fonction de la balise qu'ils voient et ne profitent pas de l'information qui leur est fournie. Le principal défaut de cette méthode est qu'elle ne considère ni la dynamique du drone, ni les limites de vitesses angulaires, ce qui peut expliquer la performance limitée.

CHAPITRE 3 MODÉLISATION DYNAMIQUE DU MULTICOPTÈRE ET MONTAGE EXPÉRIMENTAL

Ce chapitre a pour objectif principal de présenter la modélisation dynamique du multicoptère qui sera ensuite utilisée pour la synthèse des lois de commande. Cette modélisation est basée principalement sur les équations présentées dans [9] et [5]. Quatre hypothèses sont nécessaires pour simplifier la modélisation du drone :

1. La Terre est considérée localement plate et immobile.
2. La masse m du multicoptère est constante.
3. Le multicoptère possède une double symétrie massique, géométrique et propulsive.
4. Le multicoptère est un corps rigide.

L'hypothèse 1 est valide puisque la courbure de la Terre et sa vitesse sont négligeables par rapport aux distances parcourues dans les simulations. Cette hypothèse implique que le vecteur gravité pointe toujours vers le sol et que la Terre peut être considérée comme un référentiel galiléen. Les hypothèses 2 et 3 permettent de simplifier les équations dynamiques du multicoptère ($\dot{m} = 0$ et matrice d'inertie $\mathbf{J}$ diagonale). L'hypothèse 4 permet de négliger la déformation et les modes flexibles du drone.

La dérivation mathématique des équations cinématiques et dynamiques du multicoptère sont présentées dans la section 3.1 avec leurs hypothèses et leurs limitations respectives. La section 3.2 permet d'établir la platitude du système et la section 3.3 est dédiée à la présentation détaillée de l'environnement de simulation.

3.1 Modèle dynamique du multicoptère

3.1.1 Repères et orientation

Deux repères sont nécessaires afin de pouvoir modéliser la dynamique d'un drone : le repère inertiel $\mathcal{F}_i$ et le repère objet $\mathcal{F}_b$ lié au drone. Une illustration des deux repères est présentée à la Figure 3.1.

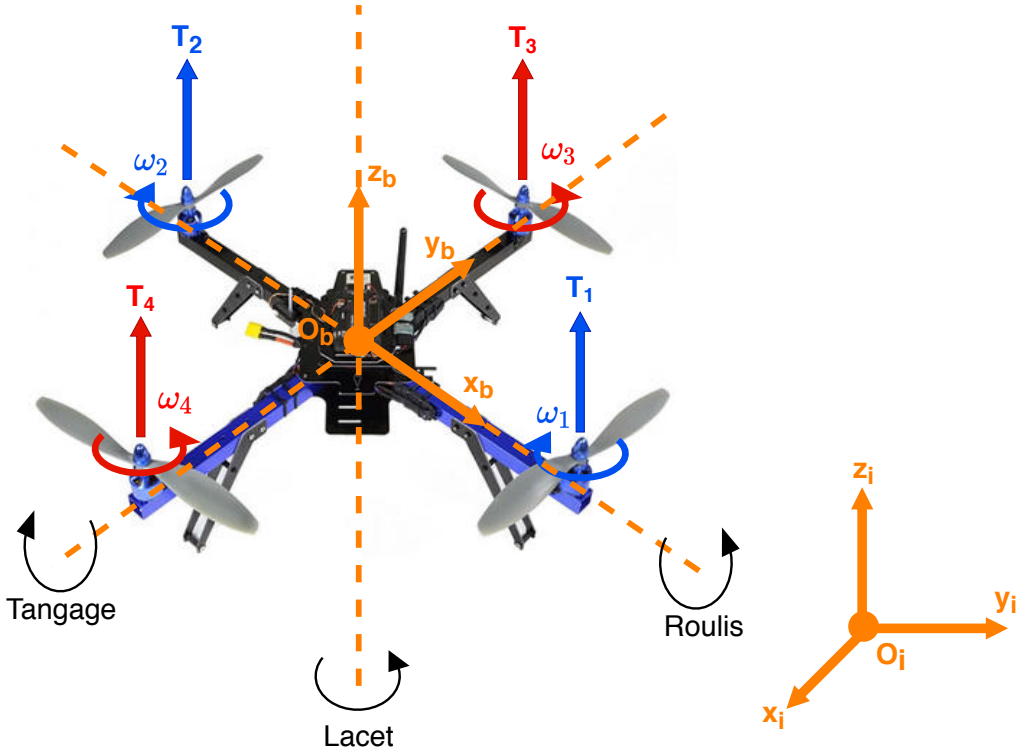


Figure 3.1 Forces, moments et repères d'un quadricoptère

- Repère inertiel $\mathcal{F}_i$
  Le repère inertiel est un repère fixe ayant pour origine un point arbitraire $\mathcal{O}_i$, généralement à la surface de la Terre. Ce repère étant inertiel, il permet d'y appliquer les lois de Newton et de déterminer la position, la vitesse linéaire, l'orientation et la vitesse angulaire de tout repère en mouvement par rapport à lui. Les vecteurs de base $\{\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i\}$, suivent la convention **ENU** (**E**ast-**N**orth-**U**p), c'est-à-dire que $\mathbf{x}_i$ pointe vers l'est, $\mathbf{y}_i$ pointe vers le nord et $\mathbf{z}_i$ pointe vers le haut.

- Repère objet $\mathcal{F}_b$
  Le repère objet est un repère lié au multicoptère. Son origine $\mathcal{O}_b$ se trouve généralement au centre de masse de celui-ci et les vecteurs de base $\{\mathbf{x}_b, \mathbf{y}_b, \mathbf{z}_b\}$ suivent la convention **FLU** (**F**ront-**L**eft-**U**p), c'est-à-dire que $\mathbf{x}_b$ pointe vers l'avant du drone, $\mathbf{y}_b$ pointe vers la gauche du drone et $\mathbf{z}_b$ pointe vers le haut.

On définit alors la matrice de rotation $\mathbf{R}_{b/i}$ qui permet de relier les projections d'un vecteur

$\mathbf{u}$ dans les repères $\mathcal{F}_i$ et $\mathcal{F}_b$, respectivement $\mathbf{u}^i$ et $\mathbf{u}^b$:

$$\mathbf{u}^b = \mathbf{R}_{b/i}\mathbf{u}^i \tag{3.1}$$

La matrice de rotation $\mathbf{R}_{b/i}$ est une matrice orthogonale appartenant au groupe spécial orthogonal SO(3) défini par

$$\mathrm{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3\times 3} \,|\, \mathbf{R}^\top\mathbf{R} = \mathbf{I}_3,\ \det(\mathbf{R}) = 1\} \tag{3.2}$$

On a ainsi $\mathbf{R}_{i/b} = \mathbf{R}_{b/i}^{-1} = \mathbf{R}_{b/i}^\top$.

3.1.2 Représentations de rotations

Dans ce mémoire, deux paramétrisations de la matrice de rotation $\mathbf{R}_{b/i}$ seront utilisées : les angles d'Euler et les quaternions. Chaque paramétrisation sera présentée brièvement ; le lecteur peut se référer à [50] pour plus de détails.

Angles d'Euler

Les angles d'Euler permettent de représenter l'orientation d'un repère par rapport à un autre selon 3 angles : ϕ (angle de roulis), θ (angle de tangage) et ψ (angle de lacet). Ces 3 angles représentent trois rotations successives du repère $\mathcal{F}_i$ pour atteindre le repère $\mathcal{F}_b$. Il est possible de faire ces 3 rotations dans n'importe quel ordre, mais la valeur des angles peut varier selon l'ordre choisi. Dans le reste de ce mémoire, nous allons suivre la convention "ZYX", c'est-à-dire :

- une rotation d'angle ψ autour de l'axe $\mathbf{z}_i$,
- ensuite une rotation d'angle θ autour de l'axe $\mathbf{y}_\psi$ du repère intermédiaire $\mathcal{F}_\psi$, obtenu après la rotation d'angle ψ du repère $\mathcal{F}_i$,
- et finalement, une rotation d'angle ϕ autour de l'axe $\mathbf{x}_\theta$ du repère intermédiaire $\mathcal{F}_\theta$, obtenu après la rotation d'angle θ du repère $\mathcal{F}_\psi$.

Ces rotations successives sont représentées à la figure 3.2. Les matrices de rotation d'angles ψ, θ et ϕ sont alors, respectivement :

$$\mathbf{R}_\psi = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix},\ \mathbf{R}_\theta = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix},\ \mathbf{R}_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{3.3}$$

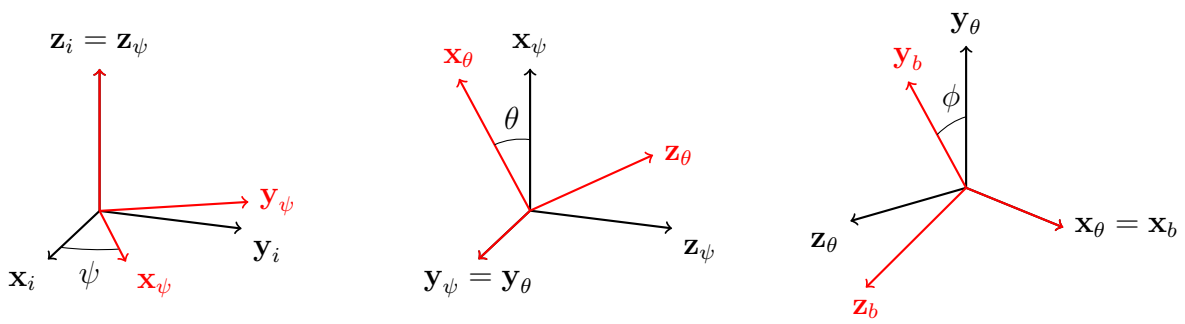


Figure 3.2 Visualisation des angles d'Euler

La matrice de rotation $\mathbf{R}_{b/i}$ pour passer du repère $\mathcal{F}_i$ au repère $\mathcal{F}_b$ est donc :

$$\mathbf{R}_{b/i} = \mathbf{R}_\phi \mathbf{R}_\theta \mathbf{R}_\psi \tag{3.4}$$

$$= \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{3.5}$$

Pour retrouver les angles ψ, θ et ϕ à partir d'une matrice $\mathbf{R}_{b/i}$ donnée, on utilise les relations suivantes :

$$\psi = \operatorname{arctan2}(r_{12}, r_{11})$$
$$\theta = -\arcsin(r_{13})$$
$$\phi = \operatorname{arctan2}(r_{23}, r_{33})$$

avec r_{ij} l'élément de la matrice $\mathbf{R}_{b/i}$ à la ligne i et à la colonne j.

Bien que les angles d'Euler soient une représentation minimale d'une rotation, un des principaux inconvénients est qu'ils ne sont valides que pour $\theta \in]-\pi/2, \pi/2[$. En effet, pour ces deux points de singularité, il n'est pas possible de déterminer ϕ et ψ de façon unique.

Quaternions

Un quaternion $\mathbf{q}$ permet une paramétrisation alternative d'une matrice de rotation. On note un quaternion unitaire $\mathbf{q} \in \mathbb{H}$ sous la forme :

$$\mathbf{q} = \begin{bmatrix} q_0 \\ \mathbf{q}_{1:3} \end{bmatrix} \tag{3.6}$$

où $\mathbb{H} = \{\mathbf{q} \in \mathbb{R}^4 \,|\, q_0 \in \mathbb{R}, \mathbf{q}_{1:3} = \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^\top \in \mathbb{R}^3, \mathbf{q}^\top \mathbf{q} = 1\}$.

Le théorème de rotation d'Euler stipule que toute rotation d'un corps rigide peut être représentée par une seule rotation d'angle ϕ_a autour d'un axe unitaire $\mathbf{n}$. L'expression du quaternion unitaire correspondant est alors donnée par :

$$\mathbf{q} = \begin{bmatrix} q_0 \\ \mathbf{q}_{1:3} \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\phi_a}{2}\right) \\ \mathbf{n} \sin\left(\frac{\phi_a}{2}\right) \end{bmatrix}$$

où la projection du vecteur $\mathbf{n}$ est la même dans $\mathcal{F}_i$ et dans $\mathcal{F}_b$. La matrice de rotation $\mathbf{R}_{b/i}$ s'écrit alors :

$$\mathbf{R}_{b/i} = (2q_0^2 - 1)\mathbf{I}_3 + 2\mathbf{q}_{1:3}\mathbf{q}_{1:3}^\top - 2q_0\hat{\mathbf{q}}_{1:3} \tag{3.7}$$

où l'opérateur $\hat{\cdot}$ associe au vecteur $\mathbf{q}_{1:3}$ la matrice antisymétrique $\hat{\mathbf{q}}_{1:3}$ donnée par :

$$\hat{\mathbf{q}}_{1:3} = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{3.8}$$

Cet opérateur revient à remplacer le produit vectoriel $\mathbf{x}\times$ par une matrice dans un repère donné, i.e., $\hat{\mathbf{x}}$.

Une rotation d'angle nul ($\phi_a = 0$) correspond à :

$$\mathbf{q_I} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^\top$$

Pour un quaternion unitaire, la rotation inverse d'angle $-\phi_a$ autour du même axe est :

$$\mathbf{q}^{-1} = \begin{bmatrix} q_0 \\ -\mathbf{q}_{1:3} \end{bmatrix}$$

La multiplication de deux quaternions $\mathbf{q}$ et $\mathbf{p}$, défini par l'opérateur $\otimes$, est :

$$\mathbf{q} \otimes \mathbf{p} = \begin{bmatrix} q_0 p_0 - \mathbf{q}_{1:3}^\top \mathbf{p}_{1:3} \\ q_0 \mathbf{p}_{1:3} + p_0 \mathbf{q}_{1:3} + \hat{\mathbf{q}}_{1:3}\mathbf{p}_{1:3} \end{bmatrix}$$

La rotation d'un vecteur $\mathbf{k}$ par un quaternion unitaire $\mathbf{q}$ est alors donnée par :

$$\mathbf{p}(\mathbf{k}') = \mathbf{q} \otimes \mathbf{k} = \mathbf{q} \otimes \mathbf{p}(\mathbf{k}) \otimes \mathbf{q}^{-1}$$

avec $\mathbf{k}'$ le vecteur après la rotation et $\mathbf{p}(\cdot)$ la représentation en quaternion d'un vecteur :

$$\mathbf{p}(\mathbf{k}) = \begin{bmatrix} 0 \\ \mathbf{k} \end{bmatrix}$$

L'avantage principal des quaternions est que l'on peut représenter une rotation sans singularité avec seulement quatre éléments, et il n'y a plus nécessité d'utiliser des fonctions trigonométriques lors des transformations. L'inconvénient principal est qu'il existe deux quaternions ($\pm\mathbf{q}$) pour une même attitude ; il faut donc porter une attention particulière lors de la conception d'une loi de commande. Un autre inconvénient est qu'un quaternion doit demeurer unitaire pour représenter une rotation. On doit donc toujours s'assurer de faire la normalisation de quaternions lorsque l'on effectue des opérations avec ceux-ci pour ne pas avoir d'erreurs numériques.

3.1.3 Équation dynamique en translation

En notant $\mathbf{p}$ le vecteur position du drone dans le repère $\mathcal{F}_i$ et $\mathbf{v}$ son vecteur vitesse, la dynamique en translation du multicoptère est donnée par la deuxième loi de Newton :

$$m\ddot{\mathbf{p}} = m\dot{\mathbf{v}} = \mathbf{F} \tag{3.9}$$

Pour avoir une modélisation haute fidelité du drone, la force totale $\mathbf{F}$ peut comprendre une panoplie de forces externes comme :

- le poids $\mathbf{F}_g$;
- les forces propulsives générées par les hélices $\mathbf{F}_t$;
- la force aérodynamique $\mathbf{F}_v$ causée par le vent ;
- les forces de portance et de traînée $\mathbf{F}_l$ et $\mathbf{F}_d$ causée par l'écoulement de l'air sur le châssis du multicoptère ;
- la force aérodynamique $\mathbf{F}_a$ provenant de l'interaction entre l'écoulement des hélices et le châssis.

On retient généralement les forces propulsives $\mathbf{F}_t$ et gravitationnelles $\mathbf{F}_g$ qui ont un impact majeur sur le comportement du drone [9]. L'équation dynamique en translation devient donc :

$$m\ddot{\mathbf{p}} = m\dot{\mathbf{v}} = \mathbf{F}_g + \mathbf{F}_t \tag{3.10}$$

Puisque nous avons supposé que la Terre est localement plate, $\mathbf{F}_g$ est constante et orientée

vers le bas selon l'axe $-\mathbf{z}_i$:

$$\mathbf{F}_g = -g\mathbf{z}_i \tag{3.11}$$

avec g accélération de la pesanteur à la surface de la Terre. Pour ce qui est des forces propulsives, nous négligeons l'effet de battement des pales (*blade flapping*) [5] et considérons qu'elles sont donc toujours orientées selon l'axe $\mathbf{z}_b$. La poussée générée par une hélice est :

$$T = \underbrace{C_T \rho A_r r^2}_{k_t} \omega^2$$

avec C_T, ρ, A_r, r et ω représentant respectivement le coefficient de poussée du profil d'hélice, la masse volumique de l'air, la surface du disque parcourue par l'hélice, le rayon et la vitesse angulaire de l'hélice. En pratique, on combine tous ces coefficients en un seul coefficient k_t déterminé expérimentalement. La force de poussée totale F est donc la somme de toutes les poussées générées par chaque hélice :

$$F = \sum_{i=1}^{n} T_i = \sum_{i=1}^{n} k_t \omega_i^2$$

avec n le nombre total de moteurs et k_t identique pour tous les rotors étant donné l'hypothèse de symétrie propulsive. L'équation dynamique en translation du multicoptère devient :

$$\ddot{\mathbf{p}} = \dot{\mathbf{v}} = \frac{F}{m}\mathbf{z}_b - g\mathbf{z}_i \tag{3.12}$$

Dans la suite, on note $\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^\top$ les coordonnées du centre de masse dans le repère inertiel $\mathcal{F}_i$.

3.1.4 Équation dynamique en rotation

Deux équations sont nécessaires pour caractériser la dynamique en rotation. La première est l'équation cinématique du quaternion, soit :

$$\dot{\mathbf{q}} = \frac{1}{2} \begin{bmatrix} -\mathbf{q}_{1:3}^\top \\ q_0 \mathbf{I}_3 + \hat{\mathbf{q}}_{1:3} \end{bmatrix} \mathbf{\Omega} \tag{3.13}$$

avec $\mathbf{\Omega} = \begin{bmatrix} p & q & r \end{bmatrix}^\top$ représentant la vitesse angulaire de $\mathcal{F}_b$ par rapport à $\mathcal{F}_i$ exprimé dans $\mathcal{F}_b$ et $\mathbf{q}$ le quaternion correspondant à la matrice de rotation $\mathbf{R}_{b/i}$.

La deuxième équation qui caractérise l'orientation du multicoptère provient du théorème du

moment cinétique et de l'équation de Coriolis. Elle prend la forme suivante :

$$\mathbf{J}\dot{\mathbf{\Omega}} = \mathbf{M} - \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega} \tag{3.14}$$

avec $\mathbf{J} \in \mathbb{R}^{3\times3}$ la matrice d'inertie du drone exprimé dans le repère $\mathcal{F}_b$ et $\mathbf{M} \in \mathbb{R}^3$ la somme des moments externes appliqués sur le multicoptère exprimée dans $\mathcal{F}_b$. Il est à noter, qu'étant donné l'hypothèse de double symétrie géométrique et massique, la matrice d'inertie $\mathbf{J}$ prend la forme diagonale :

$$\mathbf{J} = \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \tag{3.15}$$

Tout comme pour les forces externes, différents moments externes peuvent être considérés dans la modélisation dynamique d'un multirotor. Parmi ceux-ci, on retient principalement le moment dû aux forces propulsives, noté $\mathbf{M}_t$, et le moment induit par les moteurs, noté $\mathbf{M}_i$. On a alors $\mathbf{M} = \mathbf{M}_t + \mathbf{M}_i$.

Étant donné la géométrie du drone, les forces propulsives ne génèrent que des moments selon les axes $\mathbf{x}_b$ et $\mathbf{y}_b$. L'expression du moment $\mathbf{M}_t$ est alors :

$$\mathbf{M}_t = \sum_{j=1}^{n} \mathbf{OP}_j \times k_t \omega_j^2 \mathbf{z}_b \tag{3.16}$$

avec n le nombre total de moteurs et $\mathbf{OP}_j$ le vecteur partant du centre de masse du multicoptère jusqu'au centre du moteur j.

Le moment induit $\mathbf{M}_i$ provient essentiellement de la traînée causée par la rotation des hélices, qui induit un couple sur le châssis selon l'axe $\mathbf{z}_b$. L'expression du moment $\mathbf{M}_i$ est alors :

$$\mathbf{M}_i = \sum_{i=1}^{n} -s_i k_d \omega_i^2 \mathbf{z}_b \tag{3.17}$$

où $s_i = \pm 1$ représente le sens de rotation du moteur i et k_d le coefficient de traînée de l'hélice.

3.1.5 Matrice d'allocation

Afin de faciliter la conception de la loi de commande, il est commun de travailler directement avec F et $\mathbf{M}$ pour réduire la complexité mathématique du problème. À partir des expressions des forces et des moments des sections précédentes, on définit une matrice d'allocation qui permet de retrouver la vitesse angulaire de chaque rotor. Pour un quadricoptère avec la

configuration de la figure 3.1, on a la relation suivante :

$$\mathbf{u} = \begin{bmatrix} F \\ \mathbf{M} \end{bmatrix} = \begin{bmatrix} F \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} k_t & k_t & k_t & k_t \\ 0 & 0 & dk_t & -dk_t \\ -dk_t & dk_t & 0 & 0 \\ k_d & k_d & -k_d & -k_d \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{3.18}$$

avec d la distance d'un rotor au centre de masse du quadricoptère. Cette relation peut être inversée pour obtenir la matrice d'allocation :

$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4k_t} & 0 & -\frac{1}{2dk_t} & \frac{1}{4k_d} \\ \frac{1}{4k_t} & 0 & \frac{1}{2dk_t} & \frac{1}{4k_d} \\ \frac{1}{4k_t} & \frac{1}{2dk_t} & 0 & -\frac{1}{4k_d} \\ \frac{1}{4k_t} & -\frac{1}{2dk_t} & 0 & -\frac{1}{4k_d} \end{bmatrix} \begin{bmatrix} F \\ M_x \\ M_y \\ M_z \end{bmatrix}$$

Il est à noter que l'on impose la condition que les moteurs ne puissent pas changer de sens de rotation afin que cette allocation demeure valide.

3.1.6 Modèle non-linéaire complet

Les équations présentées dans les sections 3.1.3 et 3.1.4 permettent d'établir un modèle d'état non-linéaire complet d'un multirotor. Cette équation vectorielle est caractérisée par le vecteur d'état $\mathbf{x} \in \mathbb{R}^{13}$ et le vecteur d'entrée $\mathbf{u} \in \mathbb{R}^4$ donnés respectivement par :

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^\top & \mathbf{v}^\top & \mathbf{q}^\top & \mathbf{\Omega}^\top \end{bmatrix}^\top \tag{3.19}$$

$$\mathbf{u} = \begin{bmatrix} F & M_x & M_y & M_z \end{bmatrix}^\top \tag{3.20}$$

L'équation d'état est donc :

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ \frac{1}{m} u_1 \mathbf{z}_b - g\mathbf{z}_i \\ \frac{1}{2} \begin{bmatrix} -\mathbf{q}_{1:3}^\top \\ q_0 \mathbf{I}_3 + \hat{\mathbf{q}}_{1:3} \end{bmatrix} \mathbf{\Omega} \\ \mathbf{J}^{-1}(\mathbf{u}_{2:4} - \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega}) \end{bmatrix} \tag{3.21}$$

avec $u_1 = F$ et $\mathbf{u_{2:4}} = \begin{bmatrix} M_x & M_y & M_z \end{bmatrix}^\top$.

3.1.7 Dynamique des moteurs

Nous avons supposé jusqu'à présent dans notre modélisation que l'on pouvait obtenir les vitesses angulaires des rotors commandées de façon instantanée. En réalité, cette commande doit passer par des moteurs à courant continu, dont la dynamique est donnée par :

$$L\frac{\mathrm{d}i}{\mathrm{d}t} = u - Ri - k_e\omega \tag{3.22}$$

$$J\frac{\mathrm{d}\omega}{\mathrm{d}t} = k_m i - \tau_d \tag{3.23}$$

avec i le courant interne du moteur, L l'inductance interne du moteur, u le voltage appliqué au moteur, R la résistance interne du moteur, k_e la constante de force électromotrice du moteur, k_m la constante du couple moteur, ω la vitesse de rotation angulaire du moteur, J l'inertie totale du moteur et de l'hélice et τ_d le couple de charge du moteur.

Cependant, la dynamique des moteurs étant beaucoup plus rapide que la dynamique du drone, il est valide de la négliger. De plus, lors de l'implémentation expérimentale, le drone utilise déjà un correcteur pour réguler les vitesses de rotation des moteurs.

3.1.8 Paramètres du drone

Afin de pouvoir utiliser le modèle présenté dans (3.21) pour simuler et contrôler le drone, certains paramètres physiques du drone doivent être connus au préalable. Dans le cadre des essais expérimentaux, le drone utilisé est l'hexacoptère *Asctec Firefly*, utilisé au MRASL (*Mobile Robotics and Autonomous Systems Laboratory*) de Polytechnique Montréal. Les différentes valeurs de ses paramètres sont présentées au tableau 3.1.

Tableau 3.1 Paramètres physiques du Asctec Firefly

Paramètre	**Valeur**	**Unité**
m	1.507	kg
J_{xx}	0.0347563	kg.m^2
J_{yy}	0.0458929	kg.m^2
J_{zz}	0.0977	kg.m^2
d	0.215	m
k_t	$6.7 \cdot 10^{-6}$	N.s^2/rad^2
k_d	$3.1202317 \cdot 10^{-7}$	N.s^2/rad^2

La masse m du drone a pu être trouvée expérimentalement en pesant le drone avec la batterie. La matrice $\mathbf{J}$, la longueur du bras d et les coefficients k_t et k_d ont été tirés de [51].

3.2 Platitude différentielle (*differential flatness*)

Une conséquence importante de la modélisation précédente est que le modèle non-linéaire complet est différentiellement plat [19] [25]. Une des principales propriétés de ce type de systèmes est qu'il existe un ensemble de sorties plates (de même dimension que l'entrée du système) tel que l'on peut exprimer tous les états et toutes les entrées en fonction de ces sorties plates et de leurs dérivées [52]. Dans le cas d'un multicoptère, le vecteur de sorties plates $\boldsymbol{\sigma}$ est :

$$\boldsymbol{\sigma} = \begin{bmatrix} \mathbf{p} \\ \psi \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \psi \end{bmatrix} \tag{3.24}$$

Dans ce qui suit, nous détaillons la relation complète entre les sorties plates, les états et les entrées qui est brièvement présentée dans [25] et dans [19]. Les relations sont exprimées avec les angles d'Euler, mais la propriété de platitude différentielle est valide peu importe la représentation des rotations (angles d'Euler, quaternions, matrice de rotation). L'utilisation des angles d'Euler a été faite pour sa simplicité mathématique. Afin de clarifier la notation, nous posons nos sorties plates comme :

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}_{1:3} \\ \sigma_4 \end{bmatrix} \tag{3.25}$$

Il s'agit de démontrer que les vecteurs d'état $\mathbf{x}$ et d'entrée $\mathbf{u}$ peuvent s'exprimer en fonction de $\boldsymbol{\sigma}$ et de ses dérivées. Pour les états $\mathbf{p}$ et $\mathbf{v}$ du vecteur d'état $\mathbf{x}$, nous avons directement :

$$\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}_{1:3} \\ \dot{\boldsymbol{\sigma}}_{1:3} \end{bmatrix}$$

À partir de l'équation de Newton (3.12), on a :

$$F\mathbf{z}_b = m(\ddot{\mathbf{p}} + g\mathbf{z}_i) = m(\ddot{\boldsymbol{\sigma}}_{1:3} + g\mathbf{z}_i) \tag{3.26}$$

En prenant la norme de part et d'autre, on en déduit que la composante F du vecteur d'entrée $\mathbf{u}$ s'écrit :

$$F = u_1 = m\sqrt{\ddot{\sigma}_1^2 + \ddot{\sigma}_2^2 + (\ddot{\sigma}_3 + g)^2} \tag{3.27}$$

Pour le vecteur $\mathbf{q}$, on peut l'exprimer à partir de la matrice de rotation $\mathbf{R}_{b/i}$, ou de façon équivalente à partir de son inverse $\mathbf{R}_{i/b}$ définie par :

$$\mathbf{R}_{i/b} = \begin{bmatrix} \mathbf{x}_b^i & \mathbf{y}_b^i & \mathbf{z}_b^i \end{bmatrix}$$

À partir de (3.12), la projection du vecteur $\mathbf{z}_b$ dans $\mathcal{F}_i$ est donnée par :

$$\mathbf{z}_b^i = \frac{m}{F} \begin{bmatrix} \ddot{\sigma}_1 \\ \ddot{\sigma}_2 \\ \ddot{\sigma}_3 + g \end{bmatrix} = \frac{1}{\sqrt{\ddot{\sigma}_1^2 + \ddot{\sigma}_2^2 + (\ddot{\sigma}_3 + g)^2}} \begin{bmatrix} \ddot{\sigma}_1 \\ \ddot{\sigma}_2 \\ \ddot{\sigma}_3 + g \end{bmatrix} \tag{3.28}$$

Comme défini dans la figure 3.2, le repère intermédiaire $\mathcal{F}_\psi$ est obtenu après rotation du repère $\mathcal{F}_i$ d'un angle $\psi = \sigma_4$ autour de l'axe $\mathbf{z}_i$. Par inspection de la figure 3.2, et en considérant les deux rotations subséquentes en θ et ϕ, on peut définir $\mathbf{x}_b$ par :

$$\mathbf{x}_b = \frac{\mathbf{y}_\psi \times \mathbf{z}_b}{\|\mathbf{y}_\psi \times \mathbf{z}_b\|} \tag{3.29}$$

et par complétion du repère orthonormé direct, on en déduit $\mathbf{y}_b$:

$$\mathbf{y}_b = \mathbf{z}_b \times \mathbf{x}_b \tag{3.30}$$

Cela demeure valide tant que l'on évite la singularité $\mathbf{y}_\psi \times \mathbf{z}_b = \mathbf{0}$. Pour obtenir $\mathbf{x}_b^i$ et $\mathbf{y}_b^i$ en fonction de $\boldsymbol{\sigma}$ et de ses dérivées, il suffit de projeter les équations (3.29) et (3.30) dans le repère $\mathcal{F}_i$ où la projection de l'axe $\mathbf{y}_\psi$ dans le repère $\mathcal{F}_i$ est donnée par :

$$\mathbf{y}_\psi^i = \begin{bmatrix} -\sin\sigma_4 \\ \cos\sigma_4 \\ 0 \end{bmatrix}$$

On déduit alors $\mathbf{R}_{b/i}$, puis $\mathbf{q}$.

Pour montrer que le vecteur $\boldsymbol{\Omega}$ dépend de $\boldsymbol{\sigma}$ et de ses dérivées, on dérive (3.12) par rapport au repère inertiel $\mathcal{F}_i$:

$$m\dddot{\boldsymbol{\sigma}}_{1:3} = \dot{F}\mathbf{z}_b + F\dot{\mathbf{z}}_b = \dot{F}\mathbf{z}_b + F\boldsymbol{\Omega} \times \mathbf{z}_b \tag{3.31}$$

En projetant cette équation sur $\mathbf{z}_b$, on trouve :

$$\dot{F} = \mathbf{z}_b \cdot m\dddot{\boldsymbol{\sigma}}_{1:3} \tag{3.32}$$

L'insertion de (3.32) dans (3.31) nous permet de définir le vecteur $\mathbf{h}$ par :

$$\mathbf{h} = \frac{m}{F}\left(\dddot{\boldsymbol{\sigma}}_{1:3} - (\mathbf{z}_b \cdot \dddot{\boldsymbol{\sigma}}_{1:3})\mathbf{z}_b\right) = \boldsymbol{\Omega} \times \mathbf{z}_b \tag{3.33}$$

Comme $\boldsymbol{\Omega} \times \mathbf{z}_b = q\mathbf{x}_b - p\mathbf{y}_b$, on peut alors écrire :

$$\begin{aligned} p &= -\mathbf{h} \cdot \mathbf{y}_b \\ q &= \mathbf{h} \cdot \mathbf{x}_b \end{aligned}$$

et utiliser les expressions précédentes de $\mathbf{x}_b^i$, $\mathbf{y}_b^i$ et $\mathbf{z}_b^i$ en projetant dans le repère $\mathcal{F}_i$.

Pour trouver r, nous utilisons l'équation cinématique d'Euler :

$$\boldsymbol{\Omega} = p\mathbf{x}_b + q\mathbf{y}_b + r\mathbf{z}_b = \dot{\phi}\mathbf{x}_b + \dot{\theta}\mathbf{y}_\psi + \dot{\psi}\mathbf{z}_\psi \tag{3.34}$$

D'une part, en prenant le produit scalaire de (3.34) avec $\mathbf{y}_b$, on obtient :

$$q = \dot{\theta}\mathbf{y}_\psi \cdot \mathbf{y}_b + \dot{\psi}\mathbf{z}_\psi \cdot \mathbf{y}_b \Rightarrow \dot{\theta} = \frac{1}{\mathbf{y}_\psi \cdot \mathbf{y}_b}\left(q - \dot{\psi}\mathbf{z}_\psi \cdot \mathbf{y}_b\right) \tag{3.35}$$

D'autre part, en prenant le produit scalaire de (3.34) avec $\mathbf{z}_b$, on obtient :

$$r = \dot{\theta}\mathbf{y}_\psi \cdot \mathbf{z}_b + \dot{\psi}\mathbf{z}_\psi \cdot \mathbf{z}_b \tag{3.36}$$

Finalement, en injectant (3.35) dans (3.36), on a :

$$\begin{aligned} r &= \frac{\mathbf{y}_\psi \cdot \mathbf{z}_b}{\mathbf{y}_\psi \cdot \mathbf{y}_b}\left(q - \dot{\sigma}_4\mathbf{z}_\psi \cdot \mathbf{y}_b\right) + \dot{\sigma}_4\mathbf{z}_\psi \cdot \mathbf{z}_b \\ &= \frac{1}{\mathbf{y}_\psi \cdot \mathbf{y}_b}\left(\dot{\sigma}_4\mathbf{x}_\psi \cdot \mathbf{x}_b + q\mathbf{y}_\psi \cdot \mathbf{z}_b\right) \end{aligned} \tag{3.37}$$

où l'on a utilisé :

$$(\mathbf{z}_\psi \cdot \mathbf{z}_b)(\mathbf{y}_\psi \cdot \mathbf{y}_b) - (\mathbf{z}_\psi \cdot \mathbf{y}_b)(\mathbf{y}_\psi \cdot \mathbf{z}_b) = (\mathbf{z}_\psi \times \mathbf{y}_\psi) \cdot (\mathbf{z}_b \times \mathbf{y}_b) = \mathbf{x}_\psi \cdot \mathbf{x}_b \tag{3.38}$$

Cette expression de r demeure valide tant que l'on évite la singularité $\mathbf{y}_\psi \cdot \mathbf{y}_b = 0$ (équivalente à la singularité précédente $\mathbf{y}_\psi \times \mathbf{z}_b = \mathbf{0}$. Il reste seulement à préciser que la projection de

l'axe $\mathbf{x}_\psi$ dans le repère $\mathcal{F}_i$ est donnée par :

$$\mathbf{x}_\psi^i = \begin{bmatrix} \cos\sigma_4 \\ \sin\sigma_4 \\ 0 \end{bmatrix}$$

Pour conclure, il reste à exprimer le couple d'entrée $\mathbf{M}$ en fonction de $\boldsymbol{\sigma}$ et de ses dérivées. On dérive (3.31) par rapport au repère inertiel $\mathcal{F}_i$:

$$m\boldsymbol{\sigma}_{1:3}^{(4)} = \ddot{F}\mathbf{z}_b + 2\boldsymbol{\Omega} \times \dot{F}\mathbf{z}_b + \boldsymbol{\Omega} \times \boldsymbol{\Omega} \times F\mathbf{z}_b + \dot{\boldsymbol{\Omega}} \times F\mathbf{z}_b \tag{3.39}$$

où l'on précise que la dérivée de $\boldsymbol{\Omega}$ par rapport à $\mathcal{F}_i$ ou par rapport à $\mathcal{F}_b$ est la même du fait de la définition du vecteur de rotation $\boldsymbol{\Omega}$. En suivant la même procédure que pour $\boldsymbol{\Omega}$, on projette (3.39) sur $\mathbf{z}_b$ pour obtenir :

$$\ddot{F} = \mathbf{z}_b \cdot m\boldsymbol{\sigma}_{1:3}^{(4)} - \mathbf{z}_b \cdot (\boldsymbol{\Omega} \times \boldsymbol{\Omega} \times F\mathbf{z}_b) \tag{3.40}$$

L'insertion de (3.40) dans (3.39) nous permet de définir le vecteur $\mathbf{k}$ par :

$$\begin{aligned} \mathbf{k} &= \frac{1}{F}\left(m\left(\boldsymbol{\sigma}_{1:3}^{(4)} - (\mathbf{z}_b \cdot \boldsymbol{\sigma}_{1:3}^{(4)})\mathbf{z}_b\right) - (\boldsymbol{\Omega} \times \boldsymbol{\Omega} \times F\mathbf{z}_b - \mathbf{z}_b \cdot (\boldsymbol{\Omega} \times \boldsymbol{\Omega} \times F\mathbf{z}_b)\,\mathbf{z}_b) - 2\boldsymbol{\Omega} \times \dot{F}\mathbf{z}_b\right) \\ &= \dot{\boldsymbol{\Omega}} \times \mathbf{z}_b \end{aligned} \tag{3.41}$$

Comme $\dot{\boldsymbol{\Omega}} \times \mathbf{z}_b = \dot{q}\mathbf{x}_b - \dot{p}\mathbf{y}_b$, on peut alors écrire :

$$\begin{aligned} \dot{p} &= -\mathbf{k} \cdot \mathbf{y}_b \\ \dot{q} &= \mathbf{k} \cdot \mathbf{x}_b \end{aligned}$$

et utiliser les expressions précédentes de $\mathbf{x}_b^i$, $\mathbf{y}_b^i$ et $\mathbf{z}_b^i$ en projetant dans le repère $\mathcal{F}_i$.

Pour trouver $\dot{r}$, nous dérivons l'équation cinématique d'Euler par rapport à $\mathcal{F}_i$:

$$\begin{aligned} \dot{\boldsymbol{\Omega}} &= \dot{p}\mathbf{x}_b + \dot{q}\mathbf{y}_b + \dot{r}\mathbf{z}_b + p\dot{\mathbf{x}}_b + q\dot{\mathbf{y}}_b + r\dot{\mathbf{z}}_b \\ &= \dot{p}\mathbf{x}_b + \dot{q}\mathbf{y}_b + \dot{r}\mathbf{z}_b + \boldsymbol{\Omega} \times (p\mathbf{x}_b + q\mathbf{y}_b + r\mathbf{z}_b) \\ &= \dot{p}\mathbf{x}_b + \dot{q}\mathbf{y}_b + \dot{r}\mathbf{z}_b \\ \dot{\boldsymbol{\Omega}} &= \ddot{\phi}\mathbf{x}_b + \ddot{\theta}\mathbf{y}_\psi + \ddot{\psi}\mathbf{z}_\psi + \dot{\phi}\dot{\mathbf{x}}_b + \dot{\theta}\dot{\mathbf{y}}_\psi + \dot{\psi}\dot{\mathbf{z}}_\psi \\ &= \ddot{\phi}\mathbf{x}_b + \ddot{\theta}\mathbf{y}_\psi + \ddot{\psi}\mathbf{z}_\psi + \boldsymbol{\Omega} \times \dot{\phi}\mathbf{x}_b - \dot{\theta}\dot{\psi}\mathbf{x}_\psi \\ &= \ddot{\phi}\mathbf{x}_b + \ddot{\theta}\mathbf{y}_\psi + \ddot{\psi}\mathbf{z}_\psi + \dot{\phi}(r\mathbf{y}_b - q\mathbf{z}_b) - \dot{\theta}\dot{\psi}\mathbf{x}_\psi \end{aligned}$$

Soit finalement

$$\dot{p}\mathbf{x}_b + \dot{q}\mathbf{y}_b + \dot{r}\mathbf{z}_b = \ddot{\phi}\mathbf{x}_b + \ddot{\theta}\mathbf{y}_\psi + \ddot{\psi}\mathbf{z}_\psi + \dot{\phi}(r\mathbf{y}_b - q\mathbf{z}_b) - \dot{\theta}\dot{\psi}\mathbf{x}_\psi \tag{3.42}$$

Pour obtenir $\dot{\phi}$, on prend le produit scalaire de (3.34) avec $\mathbf{x}_b$:

$$p = \dot{\phi} + \dot{\psi}\mathbf{z}_\psi \cdot \mathbf{x}_b \Rightarrow \dot{\phi} = p - \dot{\sigma}_4\mathbf{z}_\psi \cdot \mathbf{x}_b \tag{3.43}$$

D'une part, en prenant le produit scalaire de (3.42) avec $\mathbf{y}_b$, on obtient :

$$\dot{q} = \ddot{\theta}\mathbf{y}_\psi \cdot \mathbf{y}_b + \ddot{\psi}\mathbf{z}_\psi \cdot \mathbf{y}_b + r\dot{\phi} - \dot{\theta}\dot{\psi}\mathbf{x}_\psi \cdot \mathbf{y}_b \Rightarrow \ddot{\theta} = \frac{1}{\mathbf{y}_\psi \cdot \mathbf{y}_b}\left(\dot{q} - \ddot{\psi}\mathbf{z}_\psi \cdot \mathbf{y}_b - r\dot{\phi} + \dot{\theta}\dot{\psi}\mathbf{x}_\psi \cdot \mathbf{y}_b\right) \tag{3.44}$$

D'autre part, en prenant le produit scalaire de (3.42) avec $\mathbf{z}_b$, on obtient :

$$\dot{r} = \ddot{\theta}\mathbf{y}_\psi \cdot \mathbf{z}_b + \ddot{\psi}\mathbf{z}_\psi \cdot \mathbf{z}_b - q\dot{\phi} - \dot{\theta}\dot{\psi}\mathbf{x}_\psi \cdot \mathbf{z}_b \tag{3.45}$$

Finalement, en injectant (3.44) dans (3.45), et avec des manipulations similaires à (3.38), on obtient :

$$\dot{r} = \frac{1}{\mathbf{y}_\psi \cdot \mathbf{y}_b}\left(\ddot{\sigma}_4\mathbf{x}_\psi \cdot \mathbf{x}_b + \dot{q}\mathbf{y}_\psi \cdot \mathbf{z}_b + \dot{\theta}\dot{\sigma}_4\mathbf{z}_\psi \cdot \mathbf{z}_b - \dot{\phi}(\mathbf{y}_\psi \cdot (q\mathbf{y}_b + r\mathbf{z}_b)\right)$$

dont toutes les composantes ont été précédemment exprimées en fonction de $\boldsymbol{\sigma}$ et de ses dérivées. Pour terminer, on trouve les relations pour $\mathbf{M}$ en utilisant (3.14), soit :

$$\mathbf{M} = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} J_{xx}\dot{p} + J_{zz}qr - J_{yy}qr \\ J_{yy}\dot{q} + J_{xx}pr - J_{zz}pr \\ J_{zz}\dot{r} + J_{yy}pq - J_{xx}qp \end{bmatrix} \tag{3.46}$$

Pour résumer, nous avons montré que tous les états et les entrées du modèle non-linéaire dynamique d'un multicoptère sont liées aux sorties plates $\begin{bmatrix}\mathbf{p}^\top & \psi\end{bmatrix}^\top$. Nous notons que la sortie platte $\mathbf{p}$ doit être dérivable au moins 4 fois et se retrouve dans les expressions de l'entrée $\mathbf{M}$ et de l'état $\dot{\boldsymbol{\Omega}}$. À partir de ces observations et de la propriété de platitude différentielle, il sera possible de générer des trajectoires optimales minimisant indirectement les efforts de commande sans avoir à inclure les équations dynamiques du système dans la formulation du problème optimal.

3.3 Environnement de développement

On utilise dans ce mémoire un environnement de développement en simulation pour valider l'implémentation discrète des lois de commande et pour effectuer les simulations pour diverses pistes de courses de drones. L'environnement virtuel *Gazebo* [53] est mis à profit, et plus spécifiquement, le simulateur *RotorS*[1] [54] pour *Gazebo* fait par l'ETH Zurich. Ce simulateur libre simule fidèlement la dynamique de multicoptères et plusieurs composantes essentielles aux blocs logiciels dont nous avons besoin comme un IMU (*Inertial Measurement Unit*), une ou plusieurs caméras, un capteur générique d'odométrie (similaire à un système de capture de mouvement), etc. Il est aussi possible de modifier l'environnement 3D (ajouts de murs, obstacles, textures de sol, etc.). Par ailleurs, plusieurs modèles de multicoptères sont présents par défaut dans le logiciel. Parmi ceux-ci se trouvent le *Firefly* et le *Pelican* produits par *Ascending Technologies*, deux modèles de drones qui sont utilisés au MRASL.

Le simulateur est aussi compatible avec ROS (*Robot Operating System*) [55], un intergiciel (librairies, conventions, outils logiciels, ...) permettant la programmation de robots. L'avantage de cette compatibilité est que le code développé pour la simulation peut directement être utilisé dans un environnement expérimental en modifiant très peu le code, ce qui peut être utile pour des travaux futurs.

1. Dépot Github : `https://github.com/ethz-asl/rotors_simulator.git`

CHAPITRE 4 LOI DE COMMANDE

Nous présentons dans ce chapitre une loi de commande non-linéaire de suivi de trajectoire pour un multicoptère. Tel que mentionné au chapitre 1, les courses pour drone ont lieu dans un environnement à très hautes vitesses et dans laquelle des manoeuvres agressives sont effectuées. Les hypothèses simplificatrices utilisées dans la commande linéaire, ainsi que pour la commande par backstepping et SMC sont invalidées dans ce contexte. Pour ce qui est du MPC, les contraintes quant aux ressources informatiques limitées empêchent son utilisation. Tel que mentionné au chapitre 2, les travaux de [20] et de [22] présentent une loi de commande qui permet le suivi d'un quadrirotor pour des manoeuvres à des hautes vitesses linéaire et des très hautes vitesses angulaires avec peu d'erreur. Nous reprenons donc cette loi de commande pour ces travaux de recherche. À la section 4.1, nous faisons la synthèse de ce correcteur. Par la suite, nous prouvons sa stabilité à la section 4.2 en utilisant des preuves similaires [18, 21]. Cette loi de commande est ensuite testée sur un modèle du *Asctec Firefly* et une série de simulations de la performance en suivi est présentée à la section 4.3.

4.1 Architecture du correcteur

La loi de commande est séparée selon la structure classique de correction pour un multirotor [9], i.e., une boucle interne et une boucle externe. Cette architecture est présentée à la Figure 4.1. Le signal de référence $\begin{bmatrix}\mathbf{p}_{ref}^{\top} & \psi_{ref}\end{bmatrix}^{\top}$ est fourni à la boucle externe qui calcule les vitesses angulaires désirées et l'amplitude de la force nécessaire pour faire le suivi de trajectoire. Ces deux données sont ensuite transmises à la boucle interne qui produit les vitesses angulaires des moteurs ω_i.

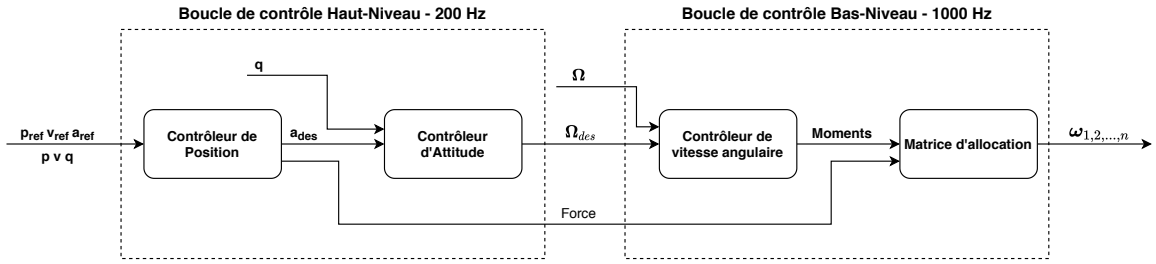


Figure 4.1 Architecture de la loi de commande

4.1.1 Boucle Externe

La boucle exterme est divisée en deux parties : le correcteur de position et le correcteur de position angulaire. Le correcteur de position calcule la force désirée pour suivre la trajectoire et fournit le vecteur d'accélération désirée au correcteur de position angulaire. À partir de l'accélération désirée et de l'angle de lacet de référence, le correcteur de position angulaire détermine les vitesses angulaires nécessaires pour avoir une erreur d'attitude nulle.

Correcteur de position

Pour faire le suivi d'une trajectoire $\mathbf{p}_{ref}$, nous utilisons un correcteur de type PD avec des termes de précommande (*feedforward*) pour l'accélération de référence $\mathbf{p}_{ref}$ et la gravité $-g\mathbf{z}_i$. Ce correcteur correspond à celui utilisé dans [18] et repris dans [20]. La formulation de cette loi de commande est la suivante :

$$\ddot{\mathbf{p}}_{des} = \mathbf{K}_p\mathbf{e}_p + \mathbf{K}_d\mathbf{e}_v + \ddot{\mathbf{p}}_{ref} + g\mathbf{z}_i \tag{4.1}$$

avec les matrices diagonales de gains $\mathbf{K}_p = \mathrm{diag}(k_{p,x}, k_{p,y}, k_{p,z})$ et $\mathbf{K}_d = \mathrm{diag}(k_{d,x}, k_{d,y}, k_{d,z})$, les erreurs $\mathbf{e}_p = \mathbf{p}_{ref} - \mathbf{p}$ et $\mathbf{e}_v = \dot{\mathbf{p}}_{ref} - \dot{\mathbf{p}}$, ainsi que $g = 9.81$ m/s^2.

Si nous considérons la dynamique en translation du système (3.12) en supposant une erreur en attitude nulle, c'est-à-dire $\mathbf{q} = \mathbf{q}_{des}$, le correcteur précédent garantit une erreur de suivi convergeant vers 0. Ceci dit, en pratique, il existe une erreur non-négligeable entre $\mathbf{q}$ et $\mathbf{q}_{des}$ qui ne peut être corrigée instantanément par le correcteur d'attitude. Plus cette erreur d'attitude augmente, plus la direction de $\mathbf{z}_b$ diverge de la direction de $\mathbf{z}_{b,ref}$, ce qui peut mener à des instabilités dynamiques. Pour pallier ceci, on projette le force désirée sur $\mathbf{z}_b$ comme dans [18] pour réduire la norme de la force si l'écart entre $\mathbf{z}_b$ et $\mathbf{z}_{b,ref}$ est trop grand :

$$F_{des} = m\ddot{\mathbf{p}}_{des} \cdot \mathbf{z}_b \tag{4.2}$$

Correcteur de position angulaire

Tel que mentionné dans la sous-section précédente, l'erreur en attitude doit être nulle pour assurer la convergence du correcteur en position. Pour ce faire, nous procédons à la synthèse d'un correcteur de position angulaire (tiré de [20]) qui détermine les vitesses angulaires à envoyer à la boucle interne. En analysant la dynamique du système (3.21) et la matrice d'allocation (3.18), nous notons deux comportements distincts au niveau de l'attitude. En effet, le roulis et le tangage sont liés à la poussée produite par les hélices, tandis que le

lacet est lié à la traînée causée par la rotation des hélices. La poussée étant d'une amplitude beaucoup plus importante que la traînée, la dynamique en roulis et en tangage est beaucoup plus rapide que celle du lacet. Suite à ces observations, la loi de commande est séparée en deux parties : une pour la dynamique plus rapide du roulis et du tangage, et une pour la dynamique plus lente du lacet.

La commande du roulis et du tangage a pour but de superposer l'axe $\mathbf{z}_b$ avec le vecteur d'accélération désirée $\mathbf{a}_{des}$. Pour y arriver, on commence par définir l'axe $\mathbf{z}_{b,des}$ à partir de l'accélération désirée trouvée à l'étape précédente :

$$\mathbf{z}_{b,des} = \frac{\ddot{\mathbf{p}}_{des}}{\|\ddot{\mathbf{p}}_{des}\|} \tag{4.3}$$

Il est ensuite possible de déterminer un quaternion d'erreur $\mathbf{q}_{e,xy}$ qui représente l'erreur rotationnelle entre les deux vecteurs. On trouve ainsi l'angle α entre les deux vecteurs et un vecteur $\mathbf{e}$ unitaire perpendiculaire aux deux vecteurs grâce aux formules suivantes :

$$\alpha = \arccos\left(\mathbf{z}_b \cdot \mathbf{z}_{b,des}\right) \tag{4.4}$$

$$\mathbf{e} = \frac{\mathbf{z}_b \times \mathbf{z}_{b,des}}{\|\mathbf{z}_b \times \mathbf{z}_{b,des}\|} \tag{4.5}$$

Puisque l'on s'intéresse aux rotations angulaires dans le repère $\mathcal{F}_b$, on transforme $\mathbf{e}$ dans ce repère :

$$\mathbf{e}_b = \mathbf{q}^{-1} \otimes \mathbf{e} \tag{4.6}$$

Le quaternion d'erreur $\mathbf{q}_{e,xy}$ est donc :

$$\mathbf{q}_{e,xy} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \mathbf{e}_b \sin\left(\frac{\theta}{2}\right) \end{bmatrix}$$

Il est à noter que cette mesure de l'erreur entre les deux vecteurs représente le chemin le plus court au niveau angulaire, mais pas nécessairement le plus court chemin au niveau temporel puisque cette mesure ne tient pas compte de la vitesse angulaire du corps.

À partir de $\mathbf{q}_{e,xy}$, on définit la loi de commande pour le tangage et le roulis :

$$\begin{bmatrix} p_{des} \\ q_{des} \end{bmatrix} = \begin{cases} 2k_{q,xy}\mathbf{q}_{e,xy,1:2} & \text{si } \mathbf{q}_{e,xy,0} \geq 0 \\ -2k_{q,xy}\mathbf{q}_{e,xy,1:2} & \text{si } \mathbf{q}_{e,xy,0} < 0 \end{cases} \tag{4.7}$$

Il est à noter que lorsque θ est nul, $\mathbf{e}$ est indéterminé. Lorsque c'est le cas, $\mathbf{q}_{e,xy}$ vaut $\mathbf{q}_I$

puisque, dans cette situation, les deux vecteurs sont superposés, et il n'existe donc pas d'erreur angulaire entre les deux vecteurs.

La loi de commande précédente assure qu'il n'y ait pas d'erreur entre $\mathbf{z}_b$ et $\mathbf{z}_{b,des}$. Il est à noter aussi que $\mathbf{q}_{e,xy,3}$ est nulle, ce qui assure qu'aucune rotation autour de l'axe $\mathbf{z}_b$ n'est commandée. Elle fixe donc 2 des 3 degrés de liberté du système. Il est possible de fixer le troisième degré de liberté avec r_{des}.

Afin de déterminer r_{des}, on doit déterminer la rotation $\mathbf{q}_{e,z}$ restante après la rotation $\mathbf{q}_{e,xy}$. Pour ce faire, on détermine la rotation complète désirée $\mathbf{q}_{des}$ à partir de $\mathbf{a}_{des}$ et ψ_{des}. On définit un repère intermédiaire entre $\mathcal{F}_i$ et $\mathcal{F}_{b,des}$, nommé $\mathcal{F}_\psi$. Ce repère représente le repère $\mathcal{F}_i$ ayant subi une rotation d'un angle ψ_{ref} autour de $\mathbf{z}_i$. Les axes $\mathbf{x}_\psi$ et $\mathbf{y}_\psi$ représentés dans $\mathcal{F}_i$ sont alors :

$$\mathbf{x}_\psi^i = \begin{bmatrix}\cos\psi_{ref} & \sin\psi_{ref} & 0\end{bmatrix}^\top$$
$$\mathbf{y}_\psi^i = \begin{bmatrix}-\sin\psi_{ref} & \cos\psi_{ref} & 0\end{bmatrix}^\top$$

À partir de ce repère intermédiaire, on détermine $\mathbf{x}_{b,des}$ par :

$$\mathbf{x}_{b,des} = \frac{\mathbf{y}_\psi \times \mathbf{z}_{b,des}}{\left\|\mathbf{y}_\psi \times \mathbf{z}_{b,des}\right\|} \tag{4.8}$$

Cette relation force $\mathbf{x}_{b,des}$ et $\mathbf{x}_\psi$ à être alignés sauf quand $\mathbf{z}_{b,des}$ a une composante négative selon $\mathbf{z}_i$. Lorsque c'est le cas, nous multiplions le résultat de (4.8) par -1, ce qui réaligne les vecteurs. Le troisième axe de notre repère est donné par :

$$\mathbf{y}_{b,des} = \frac{\mathbf{z}_{b,des} \times \mathbf{x}_{b,des}}{\left\|\mathbf{z}_{b,des} \times \mathbf{x}_{b,des}\right\|}$$

Avec ces trois vecteurs, on peut trouver l'attitude désirée complète $\mathbf{q}_{des}$, dont on déduit la rotation $\mathbf{q}_{e,z}$ par :

$$\mathbf{q}_{e,z} = \left(\mathbf{q} \otimes \mathbf{q}_{e,xy}\right)^{-1} \otimes \mathbf{q}_{des}$$

La deuxième partie de la loi de commande est finalement :

$$r_{des} = \begin{cases} 2k_{q,z}\mathbf{q}_{e,z,3} & \text{si } \mathbf{q}_{e,z,0} \geq 0 \\ -2k_{q,z}\mathbf{q}_{e,z,3} & \text{si } \mathbf{q}_{e,z,0} < 0 \end{cases} \tag{4.9}$$

4.1.2 Boucle Interne

Une fois les vitesses angulaires et la force désirée reçues, la boucle interne envoie les commandes virtuelles à la matrice d'allocation qui envoie elle-même les commandes moteurs au drone. Pour trouver les moments désirées, on utilise une loi de commande de type linéarisation par rétroaction (*feedback linearization*) :

$$\mathbf{M} = \mathbf{J}\dot{\mathbf{\Omega}}_{des} + \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega} + \mathbf{J}\mathbf{K}_{\Omega}(\mathbf{\Omega}_{des} - \mathbf{\Omega}) \tag{4.10}$$

avec $\mathbf{K}_{\Omega} = \mathrm{diag}(k_{\Omega,x}, k_{\Omega,y}, k_{\Omega,z})$.

4.2 Preuve de stabilité

Maintenant que nous avons présenté la loi de commande que nous avons utilisé, nous nous intéressons à sa stabilité. Tel que mentionné précédemment, la loi de commande présentée dans ce chapitre a été utilisée pour de nombreuses applications avec un niveau de précision élevé [20, 22, 36], mais sa preuve de stabilité n'a jamais été explicitée. Cette sous-section est dédiée à l'analyse de la stabilité de la loi de commande. En se basant sur les preuves présentées dans [18, 21], nous formulons la preuve complète de stabilité de notre loi de commande. Pour ce faire, nous analysons les dynamiques de l'erreur de la vitesse angulaire $\mathbf{e}_{\Omega}$, de l'erreur d'attitude $\mathbf{q}_e$, de l'erreur en translation $\{\mathbf{e}_p, \mathbf{e}_v\}$ et de l'erreur du système complet $\begin{bmatrix}\mathbf{e}_p & \mathbf{e}_v & \mathbf{q}_e\end{bmatrix}^{\top}$. Nous commencons par prouver la stabilité exponentielle de la loi de commande pour la vitesse angulaire. Ensuite, puisque la dynamique de l'erreur angulaire est significativement plus rapide que la dynamique de l'erreur en attitude et en position, nous posons l'hypothèse que l'erreur de vitesse angulaire est nulle pour les preuves de stabilité subséquentes. Ensuite, la stabilité asymptotique de l'erreur angulaire est démontrée. Pour terminer, nous étudions la stabilité du système complet et déterminons qu'il est stable sur un domaine restreint D.

Les notions de stabilité de Lyapunov et de systèmes autonomes hybrides seront utilisées pour faire cette analyse. Le lecteur est prié de se référer à [13] et [56] pour plus de détails. On rappelle cependant les théorèmes suivants :

Théorème 4.2.1 (Théorème de stabilité de Lyapunov) *Soit* $\mathbf{x} = \mathbf{0}$*, un point d'équilibre pour un système* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ *et* $D \subset \mathbb{R}^n$*, un domaine qui contient* $\mathbf{x} = \mathbf{0}$*. Soit* $V : D \to \mathbb{R}$*,*

une fonction C^1 tel que :

$$V(\mathbf{0}) = 0 \tag{4.11}$$

$$V(\mathbf{x}) > 0, \quad \forall \mathbf{x} \in D - \{\boldsymbol{0}\} \tag{4.12}$$

$$\dot{V}(\mathbf{x}) \leq 0, \quad \forall \mathbf{x} \in D \tag{4.13}$$

Alors, $\mathbf{x} = \mathbf{0}$ est stable. De plus, si :

$$\dot{V}(\mathbf{x}) < 0, \quad \forall \mathbf{x} \in D - \{0\} \tag{4.14}$$

Alors $\mathbf{x} = \mathbf{0}$ est asymptotiquement stable.

Théorème 4.2.2 (Théorème de stabilité exponentielle de Lyapunov) *Soit $\mathbf{x} = \mathbf{0}$, un point d'équilibre pour un système $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ et $D \subset \mathbb{R}^n$, un domaine qui contient $\mathbf{x} = \mathbf{0}$. Soit $V : D \to \mathbb{R}$, une fonction C^1 telle que :*

$$k_1 \|\mathbf{x}\|^a \leq V(t, \mathbf{x}) \leq k_2 \|\mathbf{x}\|^a \tag{4.15}$$

$$\dot{V}(\mathbf{x}) \leq -k_3 \|\mathbf{x}\|^a, \tag{4.16}$$

$\forall t \geq 0$ et $\forall \mathbf{x} \in D$, avec k_1,k_2,k_3 et a des constantes positives. Alors, $\mathbf{x} = \mathbf{0}$ est exponentiellement stable. De plus, si $D = \mathbb{R}^n$, alors $\mathbf{x} = \mathbf{0}$ est globalement exponentiellement stable.

4.2.1 Erreur de vitesse angulaire

On choisit une fonction candidate de Lyapunov :

$$V = \frac{1}{2}\mathbf{e}_\Omega^\top \mathbf{e}_\Omega \tag{4.17}$$

où l'erreur de vitesse angulaire $\mathbf{e}_\Omega$ est définie par :

$$\mathbf{e}_\Omega = \boldsymbol{\Omega}_{des} - \boldsymbol{\Omega} \tag{4.18}$$

La dérivée temporelle de la fonction candidate, $\dot{V}$, est alors :

$$\dot{V} = \dot{\mathbf{e}}_\Omega^\top \mathbf{e}_\Omega \tag{4.19}$$

En utilisant la dynamique angulaire du multicoptère donnée en (3.14), (4.19) devient :

$$\dot{V} = \left(\dot{\mathbf{\Omega}}_{des} - \dot{\mathbf{\Omega}}\right)^{\top} \mathbf{e}_{\Omega} \tag{4.20}$$

$$= \left(\dot{\mathbf{\Omega}}_{des} - \mathbf{J}^{-1}\left(\mathbf{M} - \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega}\right)\right)^{\top} \mathbf{e}_{\Omega} \tag{4.21}$$

En insérant la loi de commande définie à (4.10) dans (4.21), on trouve après simplification :

$$\dot{V} = -\mathbf{e}_{\Omega}^{\top}\mathbf{K}_{\Omega}\mathbf{e}_{\Omega} < 0 \tag{4.22}$$

pour $\mathbf{e}_{\Omega} \neq \mathbf{0}$ et en choisissant $\mathbf{K}_{\Omega} > 0$ et . On peut donc dire que le point d'équilibre $\mathbf{x} = \mathbf{0}$ est globalement exponentiellement stable car :

$$k_1 \left\|\mathbf{e}_{\Omega}\right\|^2 \leq V(\mathbf{e}_{\Omega}) \leq k_2 \left\|\mathbf{e}_{\Omega}\right\|^2 \tag{4.23}$$

$$\dot{V}(\mathbf{e}_{\Omega}) \leq -\bar{k}_{\Omega} \left\|\mathbf{e}_{\Omega}\right\|^2 \tag{4.24}$$

$$D = \mathbb{R}^n \tag{4.25}$$

avec $k_1 = k_2 = \frac{1}{2}$ et $\bar{k}_{\Omega} = \min\{k_{\Omega,x}, k_{\Omega,y}, k_{\Omega,z}\}$.

4.2.2 Erreur d'attitude

On commence par définir le terme d'erreur $\mathbf{q}_e$ qui correspond à :

$$\mathbf{q}_e = \mathbf{q}^{-1} \otimes \mathbf{q}_{des} \tag{4.26}$$

Pour cette preuve de stabilité, on suppose que le point d'équilibre $\mathbf{e}_{\Omega} = \mathbf{0}$ est atteint beaucoup plus rapidement que $\mathbf{q}_e = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^{\top}$, c'est-à-dire que l'on suppose que $\mathbf{\Omega} = \mathbf{\Omega}_{des}$. Sans perte de généralité, on pose le gain $\underline{k}_q$ positif défini par $\underline{k}_q = \min\{k_{q,xy}, k_{q,z}\}$ afin que les gains pour le contrôle d'attitude présentés à la section 4.1 soient identiques, donc qu'on puisse exprimer le gain par un scalaire plutôt que par une matrice. On pose aussi que $\mathbf{q}_{des} = \mathbf{q}_I$. Il est possible d'obtenir cette formulation peu importe la valeur de $\mathbf{q}_{des}$ par changement de repère.

Nous commençons notre preuve avec les rappels suivants :

Définition 4.2.1 (Système Autonome Hybride) *Un système autonome hybride H est un ensemble $H = (Q, X, \mathbf{f}, Init, Dom, E, G, R)$, avec :*

- $Q = \{q_1, q_2, ..\}$ *un ensemble d'états discrets ;*
- $X = \mathbb{R}^n$ *un ensemble d'états continus ;*

— $\mathbf{f}(\cdot,\cdot) : Q \times X \to \mathbb{R}^n$ *un champ vectoriel ;*
— $Init \subseteq Q \times X$ *un ensemble d'états initaux ;*
— $Dom\,(\cdot) : Q \to 2^X$ *un domaine ;*
— $E \subseteq Q \times Q$ *un ensemble de bords ;*
— $G\,(\cdot) : E \to 2^X$ *une condition de garde ;*
— $R\,(\cdot,\cdot) : E \times X \to 2^X$ *un "reset map".*

Théorème 4.2.3 (Stabilité de Lyapunov pour systèmes hybrides) *Considérons un système autonome hybride H avec $\mathbf{x} = \mathbf{0}$, un point d'équilibre et $R(q, \mathbf{x}) \in Q \times \{\mathbf{x}\}$. Supposons qu'il existe un ensemble ouvert $D \subset Q \times \mathbb{R}^n$ tel que $(q, \mathbf{0}) \in D$ quel que soit $q \in Q$. Soit V : $D \to \mathbb{R}$, une fonction C^1 pour son second argument tel que pour tout $q \in Q$:*

$$V(q, \mathbf{0}) = 0 \tag{4.27}$$

$$V(q, \mathbf{x}) > 0, \forall \mathbf{x} \in D - \{0\} \tag{4.28}$$

$$\frac{\partial V\,(q, \mathbf{x})}{\partial \mathbf{x}} \mathbf{f}\,(q, \mathbf{x}) \leq 0, \forall \mathbf{x} \in D \tag{4.29}$$

Alors, $\mathbf{x} = \mathbf{0}$ est stable.

En considérant la modélisation non-linéaire de notre système en attitude, on définit un système autonome hybride $H = (Z, Q, \mathbf{f}, Init, Dom, E, G, R)$, avec :

— $Q = S^3 \subset \mathbb{R}^4$, l'état continu $\mathbf{q}$ représentant l'attitude en quaternions ;
— $Z = \{z_1, z_2\}$ l'ensemble d'états discrets représentant la partie supérieure et inférieure de la 3-sphère unitaire dénotée S^3, qui représente aussi l'ensemble des quaternions unitaires. L'ensemble S^3 double couvre l'espace $SO(3)$, c'est-à-dire qu'il existe deux représentations dans S^3 pour une représentation dans $SO(3)$. Ceci peut être expliqué par le fait que toute rotation peut être représentée par une rotation par rapport à un axe ou par une rotation opposée par rapport à un axe pointant dans la direction opposée. La partie supérieure et inférieure de S^3 représentent ainsi ces rotations opposées. La valeur de Z correspond à la position de $\mathbf{q}$ dans S^3 ;
— le champ vectoriel :

$$\mathbf{f}(z, \mathbf{q}) = \begin{cases} \dfrac{1}{2} \begin{bmatrix} -\mathbf{q}_{1:3}^{\top} \\ q_0 \mathbf{I}_{3\times 3} + \hat{\mathbf{q}}_{1:3} \end{bmatrix} \mathbf{\Omega}_{des}, & si\ z = z_1 \\ -\dfrac{1}{2} \begin{bmatrix} -\mathbf{q}_{1:3}^{\top} \\ q_0 \mathbf{I}_{3\times 3} + \hat{\mathbf{q}}_{1:3} \end{bmatrix} \mathbf{\Omega}_{des}, & si\ z = z_2 \end{cases} \tag{4.30}$$

— $Init \subseteq Z \times Q$ un ensemble d'états initiaux ;
— $Dom(z_1) = Dom(z_2) = S^3$;
— $E = \{(z_1, z_2),\ (z_2, z_1)\}$ la possibilité d'aller de z_1 à z_2 et de z_2 à z_1 ;
— $G(z_1, z_2) = \{\mathbf{q} \in S^3 \mid q_0 < 0\}$ et $G(z_2, z_1) = \{\mathbf{q} \in S^3 \mid q_0 > 0\}$;
— $R(z_1, z_2, \mathbf{q}) = R(z_2, z_1, \mathbf{q}) = -\mathbf{q}$ puisqu'une rotation de 180° est égale à une rotation de −180° autour du même axe.

Conformément à (4.7) et (4.9), la loi de commande est :

$$\mathbf{\Omega}_{des}(\mathbf{q}) = 2\underline{k}_q \mathrm{sgn}(q_{e,0})\mathbf{q}_{e,1:3} = \begin{cases} -2\underline{k}_q \mathbf{q}_{1:3} & si\ z = z_1 \\ 2\underline{k}_q \mathbf{q}_{1:3} & si\ z = z_2 \end{cases} \tag{4.31}$$

avec sgn la fonction signe.

On choisit alors une fonction candidate de Lyapunov :

$$V_1(z, \mathbf{q}) = \mathbf{q}_{1:3}^\top \mathbf{q}_{1:3} + (q_0 - 1)^2 \quad \forall z \in Z \tag{4.32}$$

On trouve sa dérivée par rapport au temps :

$$\begin{aligned}
\dot{V}_1(z, \mathbf{q}) &= \frac{\partial V_1(z, \mathbf{q})}{\partial \mathbf{q}} \mathbf{f}(z, \mathbf{q}) && (4.33)\\
&= \begin{bmatrix} 2(q_0 - 1) & 2\mathbf{q}_{1:3} \end{bmatrix} \begin{bmatrix} \frac{1}{2}\mathbf{q}_{1:3}^\top 2\underline{k}_q \mathbf{q}_{1:3} \\ -\frac{1}{2}\left(\hat{\mathbf{q}}_{1:3} + q_0 \mathbf{I}\right) 2\underline{k}_q \mathbf{q}_{1:3} \end{bmatrix} && (4.34)\\
&= 2\underline{k}_q (q_0 - 1)\, \mathbf{q}_{1:3}^\top \mathbf{q}_{1:3} - 2\underline{k}_q \mathbf{q}_{1:3}^\top \left(\hat{\mathbf{q}}_{1:3} + q_0 \mathbf{I}\right) \mathbf{q}_{1:3} && (4.35)\\
&= -2\underline{k}_q \mathbf{q}_{1:3} \left(\hat{\mathbf{q}}_{1:3} + \mathbf{I}\right) \mathbf{q}_{1:3} && (4.36)\\
&= -2\underline{k}_q \mathbf{q}_{1:3}^\top \mathbf{q}_{1:3} \quad \forall z \in Z && (4.37)
\end{aligned}$$

Selon le théorème 4.2.2, $\mathbf{q}_e = \mathbf{q}_I$ est un point d'équilibre stable puisque :
— $V_1(z, \cdot) = 0$ ssi $\mathbf{q} = \mathbf{q}_I$;
— $V_1(z, \mathbf{q}) > 0 \ \ \forall \mathbf{q}, (z, \mathbf{q}) \in Dom(z) - \{\mathbf{q}_I\}$;
— $\dot{V}_1(z, \mathbf{q}) = \frac{\partial V(z, \mathbf{q})}{\partial \mathbf{q}} f(z, \mathbf{q}) \leq 0 \ \ \forall \mathbf{q}, (z, \mathbf{q}) \in Dom(z)$

Par ailleurs, on peut dire que $V_1(z, \mathbf{q})$ est strictement décroissant car :
— $\dot{V}_1(z, \mathbf{q}) < 0 \ \ \forall z \in Z \ \ \forall \mathbf{q} \in Dom(z) - \mathbf{q}_I$;
— $\dot{V}_1(z, \mathbf{q}_I) = 0 \ \ \forall z \in Z$;
— $V_1(z, \mathbf{q})$ ne change pas lorsque z change d'état.

On peut donc conclure que $\mathbf{q}_I$ est un point d'équilibre globalement asymptotiquement stable.

4.2.3 Erreur en translation

Avant de procéder à l'analyse de stabilité du système en translation, on pose les hypothèses suivantes :

— le domaine D d'analyse de la stabilité se limite à une erreur maximale $\bar{e}_p = \sup \|\mathbf{e}_p\|$ et à une erreur angulaire maximale de $\pi/2$ dans une représentation axe-angle (*Axis-Angle*) ;

— les matrices de gains $\mathbf{K}_p$ et $\mathbf{K}_d$ sont diagonales définies positives ;

— $\underline{k}_p = \min\{k_{p,x}, k_{p,y}, k_{p,z}\}$, $\bar{k}_p = \max\{k_{p,x}, k_{p,y}, k_{p,z}\}$;

— $\underline{k}_v = \min\{k_{v,x}, k_{v,y}, k_{v,z}\}$, $\bar{k}_v = \max\{k_{v,x}, k_{v,y}, k_{v,z}\}$.

Pour ce qui est de l'analyse de stabilité du système en translation, nous commençons par étudier la dynamique des erreurs $\mathbf{e}_p$ et $\mathbf{e}_v$:

$$\dot{\mathbf{e}}_p = \mathbf{e}_v \tag{4.38}$$

$$m\dot{\mathbf{e}}_v = m\ddot{\mathbf{p}}_{ref} - m\ddot{\mathbf{p}} \tag{4.39}$$

$$m\dot{\mathbf{e}}_v = m\ddot{\mathbf{p}}_{ref} - F\mathbf{z}_b + mg\mathbf{z}_i \tag{4.40}$$

Afin de pouvoir représenter la dynamique de l'erreur en fonction de l'erreur en attitude $\mathbf{q}_e$, nous faisons l'addition et la soustraction du terme $\dfrac{F}{\mathbf{z}_{b,des} \cdot \mathbf{z}_b}\mathbf{z}_{b,des}$ à l'équation (4.40)

$$m\dot{\mathbf{e}}_v = m\ddot{\mathbf{p}}_{ref} + mg\mathbf{z}_i + m\mathbf{b} - \frac{F}{\mathbf{z}_{b,des} \cdot \mathbf{z}_b}\mathbf{z}_{b,des} \tag{4.41}$$

avec

$$\mathbf{b} = -\frac{F}{m}\left(\mathbf{z}_b - \frac{\mathbf{z}_{b,des}}{\mathbf{z}_{b,des} \cdot \mathbf{z}_b}\right) = -\frac{F}{m(\mathbf{z}_{b,des} \cdot \mathbf{z}_b)}\left((\mathbf{z}_{b,des} \cdot \mathbf{z}_b)\,\mathbf{z}_b - \mathbf{z}_{b,des}\right) \tag{4.42}$$

Il est à noter que par hypothèse sur l'erreur angulaire maximale, le terme $\dfrac{F}{\mathbf{z}_{b,des} \cdot \mathbf{z}_b}\mathbf{z}_{b,des}$ est bien défini car :

$$0 < \mathbf{z}_{b,des} \cdot \mathbf{z}_b < 1 \tag{4.43}$$

À partir des équations (4.1), (4.2) et (4.3), on simplifie le terme $\dfrac{F}{\mathbf{z}_{b,des} \cdot \mathbf{z}_b}\mathbf{z}_{b,des}$:

$$\frac{F}{\mathbf{z}_{b,des} \cdot \mathbf{z}_b}\mathbf{z}_{b,des} = \frac{m\ddot{\mathbf{p}}_{des} \cdot \mathbf{z}_b}{\dfrac{\ddot{\mathbf{p}}_{des}}{\|\ddot{\mathbf{p}}_{des}\|} \cdot \mathbf{z}_b} \frac{\ddot{\mathbf{p}}_{des}}{\|\ddot{\mathbf{p}}_{des}\|} \tag{4.44}$$

$$= m\ddot{\mathbf{p}}_{des} \tag{4.45}$$

$$= m\mathbf{K}_p\mathbf{e}_p + m\mathbf{K}_v\mathbf{e}_v + m\ddot{\mathbf{p}}_{ref} + mg\mathbf{z}_i \tag{4.46}$$

En insérant (4.46) dans (4.41), la dynamique de l'erreur de $\mathbf{e}_v$ devient :

$$\dot{\mathbf{e}}_v = -\mathbf{K}_p\mathbf{e}_p - \mathbf{K}_v\mathbf{e}_v + \mathbf{b} \tag{4.47}$$

On pose comme candidate pour la fonction de Lyapunov :

$$V_2 = \frac{1}{2}\mathbf{e}_p^\top\mathbf{K}_p\mathbf{e}_p + \frac{1}{2}\mathbf{e}_v^\top\mathbf{e}_v + c_1\mathbf{e}_p^\top\mathbf{e}_v \tag{4.48}$$

avec c_1 une constante positive inférieure à $\underline{k}_v$. La dérivée de la fonction V_2 est alors :

$$\dot{V}_2 = \mathbf{e}_p^\top\mathbf{K}_p\dot{\mathbf{e}}_p + \mathbf{e}_v^\top\dot{\mathbf{e}}_v + c_1(\mathbf{e}_v^\top\dot{\mathbf{e}}_p + \mathbf{e}_p^\top\dot{\mathbf{e}}_v) \tag{4.49}$$

En utilisant (4.47), on simplifie $\dot{V}_2$ sous la forme :

$$\dot{V}_2 = \mathbf{e}_p^\top\mathbf{K}_p\mathbf{e}_v + \mathbf{e}_v^\top\dot{\mathbf{e}}_v + c_1(\mathbf{e}_v^\top\mathbf{e}_v + \mathbf{e}_p^\top\dot{\mathbf{e}}_v) \tag{4.50}$$

$$= \mathbf{e}_p^\top\mathbf{K}_p\mathbf{e}_v + \mathbf{e}_v^\top(-\mathbf{K}_p\mathbf{e}_p - \mathbf{K}_v\mathbf{e}_v + \mathbf{b}) + c_1(\mathbf{e}_v^\top\mathbf{e}_v + \mathbf{e}_p^\top(-\mathbf{K}_p\mathbf{e}_p - \mathbf{K}_v\mathbf{e}_v + \mathbf{b})) \tag{4.51}$$

$$= k_p\mathbf{e}_p \cdot \mathbf{e}_v + \mathbf{e}_v \cdot (-k_p\mathbf{e}_p - k_v\mathbf{e}_v + \mathbf{b}) + c_1\mathbf{e}_v \cdot \mathbf{e}_v + c_1\mathbf{e}_p \cdot (-k_p\mathbf{e}_p - k_v\mathbf{e}_v + \mathbf{b}) \tag{4.52}$$

$$= -c_1\mathbf{e}_p^\top\mathbf{K}_p\mathbf{e}_p - \mathbf{e}_v^\top(\mathbf{K}_v - c_1\mathbf{I})\mathbf{e}_v - c_1\mathbf{e}_p^\top\mathbf{K}_v\mathbf{e}_v + (c_1\mathbf{e}_p^\top + \mathbf{e}_v^\top)\mathbf{b} \tag{4.53}$$

On cherche à borner le terme $\|\mathbf{b}\|$. Pour cela, on débute en bornant $\|(\mathbf{z}_{b,des} \cdot \mathbf{z}_b)\,\mathbf{z}_b - \mathbf{z}_{b,des}\|$. Par manipulation des produits scalaires et vectoriels, il vient :

$$\|\,(\mathbf{z}_{b,des} \cdot \mathbf{z}_b)\,\mathbf{z}_b - \mathbf{z}_{b,des}\| = \|\mathbf{z}_b \times (\mathbf{z}_b \times \mathbf{z}_{b,des})\,\| \tag{4.54}$$

$$= \|\mathbf{z}_b\|\,\|\mathbf{z}_b \times \mathbf{z}_{b,des}\| \tag{4.55}$$

$$= \|\mathbf{z}_b\|\,\|\mathbf{z}_{b,des}\|\sin(\theta) \tag{4.56}$$

$$= \sin(\theta) \tag{4.57}$$

avec θ l'angle entre $\mathbf{z}_b$ et $\mathbf{z}_{b,des}$ défini par (4.4).

Pour un angle λ (plus petit que $\pi/2$ par hypothèse) entre $\mathbf{q}_{des}$ et $\mathbf{q}$ dans une représentation axe-angle, on sait que :

$$\sin(\theta) \leq \sin(\lambda) \tag{4.58}$$

En utilisant (4.57) et (4.58), on peut donc majorer la norme par :

$$\begin{aligned} \|(\mathbf{z}_{b,des} \cdot \mathbf{z}_b)\,\mathbf{z}_b - \mathbf{z}_{b,des}\| &\leq \sin(\lambda) && (4.59)\\ &\leq 2\sin\left(\frac{\lambda}{2}\right)\cos\left(\frac{\lambda}{2}\right) && (4.60) \end{aligned}$$

Finalement la conversion entre la représentation axe-angle et les quaternions permet d'établir la borne :

$$\|(\mathbf{z}_{b,des} \cdot \mathbf{z}_b)\,\mathbf{z}_b - \mathbf{z}_{b,des}\| \leq 2\left\|\mathbf{q}_{e,1:3}\right\| \|q_{e,0}\| \leq \mu \leq 2\left\|\mathbf{q}_{e,1:3}\right\| \tag{4.61}$$

avec μ une constante inférieure à 1.

En utilisant (4.46), on peut ainsi borner $\|\mathbf{b}\|$ par :

$$\begin{aligned} \|\mathbf{b}\| &\leq \|\mathbf{K}_p\mathbf{e}_p + \mathbf{K}_v\mathbf{e}_v + \ddot{\mathbf{p}}_{ref} + g\mathbf{z}_i\| \,\|(\mathbf{z}_{b,des} \cdot \mathbf{z}_b)\,\mathbf{z}_b - \mathbf{z}_{b,des}\| && (4.62)\\ &\leq \left(\bar{k}_p\,\|\mathbf{e}_p\| + \bar{k}_v\,\|\mathbf{e}_v\| + \|\mathbf{d}\|\right) \|(\mathbf{z}_{b,des} \cdot \mathbf{z}_b)\,\mathbf{z}_b - \mathbf{z}_{b,des}\| && (4.63) \end{aligned}$$

avec $\mathbf{d} = \ddot{\mathbf{p}}_{ref} + g\mathbf{z}_i$.

En appliquant l'hypothèse initiale $\mathbf{q}_{des} = \mathbf{q}_I$, nous savons que $\mathbf{q}_e = -\mathbf{q}$. Il est donc possible de borner $\mathbf{b}$ par :

$$\|\mathbf{b}\| \leq \mu\left(\bar{k}_p\|\mathbf{e}_p\| + \bar{k}_v\|\mathbf{e}_v\| + \|\mathbf{d}\|\right) \leq 2\|\mathbf{q}_{1:3}\|\left(\bar{k}_p\|\mathbf{e}_p\| + \bar{k}_v\|\mathbf{e}_v\| + \|\mathbf{d}\|\right) \tag{4.64}$$

En utilisant cette borne avec (4.53), on trouve :

$$\begin{aligned} \dot{V}_2 \leq\ & -c_1\underline{k}_p\|\mathbf{e}_p\|^2 - (\underline{k}_v - c_1)\|\mathbf{e}_v\|^2 - c_1\underline{k}_v\|\mathbf{e}_p\|\|\mathbf{e}_v\| \\ & + \mu\left(c_1\|\mathbf{e}_p\| + \|\mathbf{e}_v\|\right)\left(\bar{k}_p\|\mathbf{e}_p\| + \bar{k}_v\|\mathbf{e}_v\| + \|\mathbf{d}\|\right) \\ \leq\ & -c_1(\underline{k}_p - \mu\bar{k}_p)\|\mathbf{e}_p\|^2 - (\underline{k}_v - \mu\bar{k}_v - c_1)\|\mathbf{e}_v\|^2 - c_1(\underline{k}_v - \mu\bar{k}_v)\|\mathbf{e}_p\|\|\mathbf{e}_v\| \\ & + 2\|\mathbf{q}_{1:3}\|\left(c_1\|\mathbf{e}_p\|\|\mathbf{d}\| + \|\mathbf{e}_v\|\|\mathbf{d}\| + \bar{k}_p\|\mathbf{e}_p\|\|\mathbf{e}_v\|\right) \\ \leq\ & -c_1(\underline{k}_p - \mu\bar{k}_p)\|\mathbf{e}_p\|^2 - (\underline{k}_v - \mu\bar{k}_v - c_1)\|\mathbf{e}_v\|^2 - c_1(\underline{k}_v - \mu\bar{k}_v)\|\mathbf{e}_p\|\|\mathbf{e}_v\| \\ & + 2\|\mathbf{q}_{1:3}\|\left(c_1\|\mathbf{d}\|\|\mathbf{e}_p\| + (\bar{k}_p\bar{e}_p + \|\mathbf{d}\|)\|\mathbf{e}_v\|\right) \end{aligned}$$

À partir de $\dot{V}_2$, il n'est pas possible de tirer de conclusion quant à la stabilité du système en

translation puisque des termes concernant l'erreur en attitude sont inclus dans la fonction, ce qui est attendu étant donné la nature couplée de la dynamique d'un multicoptère. Nous procédons donc à l'étude du système complet avec les termes d'erreur en attitude afin de prouver la stabilité globale du système.

4.2.4 Système Complet

Avant de procéder à l'analyse de stabilité du système complet, il est à noter que l'on pose les mêmes hypothèses énoncées dans 4.2.2 et 4.2.3. À partir de V_1 et V_2 définies précédemment, on pose la fonction candidate de Lyapunov pour le système complet :

$$\begin{aligned} V &= V_1 + V_2 \\ V &= \mathbf{q}_{1:3}^\top \mathbf{q}_{1:3} + (q_0 - 1)^2 + \frac{1}{2} k_p \left\|\mathbf{e}_p\right\|^2 + \frac{1}{2} \left\|\mathbf{e}_v\right\|^2 + c_1 \mathbf{e}_p \cdot \mathbf{e}_v \end{aligned}$$

En utilisant les résultats de 4.2.2 et 4.2.3, on trouve alors :

$$\begin{aligned} \dot{V} \leq -c_1(\underline{k}_p - \mu \bar{k}_p)\|\mathbf{e}_p\|^2 - (\underline{k}_v - \mu \bar{k}_v - c_1)\|\mathbf{e}_v\|^2 - c_1(\underline{k}_v - \mu \bar{k}_v)\|\mathbf{e}_p\|\|\mathbf{e}_v\| \\ + 2\|\mathbf{q}_{1:3}\| \left(c_1 \|\mathbf{d}\| \|\mathbf{e}_p\| + (\bar{k}_p \bar{e}_p + \|\mathbf{d}\|)\|\mathbf{e}_v\|\right) - 2\underline{k}_q \|\mathbf{q}_{1:3}\|^2 \end{aligned}$$

En posant $\mathbf{z}_1 = \left[\|\mathbf{e}_p\| \ \|\mathbf{e}_v\|\right]^\top$, l'équation précédente devient :

$$\dot{V} \leq -\mathbf{z}_1^\top \mathbf{W}_1 \mathbf{z}_1 + 2\mathbf{z}_1^\top \mathbf{W}_{12} \|\mathbf{q}_{1:3}\| - 2\underline{k}_q \|\mathbf{q}_{1:3}\|^2 \tag{4.65}$$

avec

$$\mathbf{W}_1 = \begin{bmatrix} c_1(\underline{k}_p - \mu \bar{k}_p) & \frac{c_1}{2}(\underline{k}_v - \mu \bar{k}_v) \\ \frac{c_1}{2}(\underline{k}_v - \mu \bar{k}_v) & \underline{k}_v - \mu \bar{k}_v - c_1 \end{bmatrix} \tag{4.66}$$

$$\mathbf{W}_{12} = \begin{bmatrix} c_1 \|\mathbf{d}\| \\ \left(\bar{k}_p \bar{e}_p + \|\mathbf{d}\|\right) \end{bmatrix} \tag{4.67}$$

On peut reformuler (4.65) comme :

$$\dot{V} \leq -\underline{\lambda}\left(\mathbf{W}_1\right) \|\mathbf{z}_1\|^2 + 2 \|\mathbf{W}_{12}\| \|\mathbf{z}_1\| \|\mathbf{q}_{1:3}\| - 2\underline{k}_q \|\mathbf{q}_{1:3}\|^2 \tag{4.68}$$

$$\leq -\mathbf{z}^\top \mathbf{W} \mathbf{z} \tag{4.69}$$

avec $\underline{\lambda}(\mathbf{W}_1)$ la plus petite valeur propre de $\mathbf{W}_1$, $\mathbf{z} = \left[\mathbf{z}_1^\top \ \|\mathbf{q}_{1:3}\|\right]^\top$ et la matrice $\mathbf{W}$ donnée par :

$$\mathbf{W} = \begin{bmatrix} \underline{\lambda}(\mathbf{W}_1) & -\|\mathbf{W}_{12}\| \\ -\|\mathbf{W}_{12}\| & 2\underline{k}_q \end{bmatrix} \tag{4.70}$$

Il faut choisir des valeurs adéquates des gains $\underline{k}_p$, $\bar{k}_p$, $\underline{k}_v$, $\bar{k}_v$, $\underline{k}_q$ et c_1 tels que :

— $\mathbf{W}_1$ soit définie positive :

$$0 < \underline{k}_p - \mu\bar{k}_p \tag{4.71}$$

$$0 < c_1 < \frac{4(\underline{k}_p - \mu\bar{k}_p)(\underline{k}_v - \mu\bar{k}_v)}{(\underline{k}_v - \mu\bar{k}_v)^2 + 4(\underline{k}_p - \mu\bar{k}_p)} \tag{4.72}$$

— $\mathbf{W}$ soit définie positive :

$$\underline{k}_q > \frac{\|\mathbf{W}_{12}\|^2}{2\underline{\lambda}(\mathbf{W}_1)} \tag{4.73}$$

Avec ce choix de constantes, on peut affirmer que :

$$\begin{bmatrix} \mathbf{e}_p \\ \mathbf{e}_v \\ \mathbf{q}_e \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{q}_I \end{bmatrix}$$

est un point d'équilibre stable puisque :

— $V(z,\cdot) = 0$ ssi $\mathbf{s} = \left[\mathbf{0}^\top \quad \mathbf{0}^\top \quad \mathbf{q}_I^\top\right]^\top$;

— $V(z,\mathbf{s}) > 0 \ \ \forall\, \mathbf{s}, (z,\mathbf{s}) \in Dom - \{\left[\mathbf{0}^\top \quad \mathbf{0}^\top \quad \mathbf{q}_I^\top\right]^\top\}$;

— $\dot{V}(z,\mathbf{s}) \leq 0 \ \forall\, \mathbf{s}, (z,\mathbf{s}) \in Dom$.

De plus, on peut dire V est strictement décroissante car :

— $\dot{V}(z,\mathbf{s}) < 0 \ \forall z \in Z \ \forall\, \mathbf{s} \in Dom - \{\left[\mathbf{0}^\top \quad \mathbf{0}^\top \quad \mathbf{q}_I^\top\right]^\top\}$;

— $\dot{V}(z,\mathbf{s}) = 0 \ \forall z \in Z$;

— $V(z,\mathbf{s})$ ne change pas lorsque z change d'état.

On peut donc conclure que $\mathbf{e}_p = \mathbf{0}$, $\mathbf{e}_v = \mathbf{0}$ et $\mathbf{q}_e = \mathbf{q}_I$ est un point d'équilibre asymptotiquement stable sur le domaine défini D. À partir de ces résultats, nous pouvons énoncer le théorème suivant :

Théorème 4.2.4 (Stabilité asymptotique de loi de commande) *Considérons la dynamique d'un multirotor définie en* (3.21)*, ainsi que la force F et les moments $\boldsymbol{M}$ de la loi de commande définie par les équations* (4.1)*,* (4.2)*,*(4.7) *et* (4.9)*. Supposons que les conditions*

initiales satisfont :

$$\boldsymbol{q}_e \in D \tag{4.74}$$

$$\|\boldsymbol{e}_p\| \leq \bar{e}_p \tag{4.75}$$

Et que des gains positifs $\underline{k}_p$, $\bar{k}_p$, $\underline{k}_v$, $\bar{k}_v$ *et* $\underline{k}_q$, *ainsi qu'une constante positive* c_1 *sont choisis tel que :*

$$0 < \underline{k}_p - \mu\bar{k}_p \tag{4.76}$$

$$0 < c_1 < \frac{4(\underline{k}_p - \mu\bar{k}_p)(\underline{k}_v - \mu\bar{k}_v)}{(\underline{k}_v - \mu\bar{k}_v)^2 + 4(\underline{k}_p - \mu\bar{k}_p)} \tag{4.77}$$

$$\underline{k}_q > \frac{\|\boldsymbol{W}_{12}\|^2}{2\underline{\lambda}(\boldsymbol{W}_1)} \tag{4.78}$$

Où :

$$\boldsymbol{W}_1 = \begin{bmatrix} c_1(\underline{k}_p - \mu\bar{k}_p) & \frac{c_1}{2}(\underline{k}_v - \mu\bar{k}_v) \\ \frac{c_1}{2}(\underline{k}_v - \mu\bar{k}_v) & \underline{k}_v - \mu\bar{k}_v - c_1 \end{bmatrix} \tag{4.79}$$

$$\boldsymbol{W}_{12} = \begin{bmatrix} c_1\|\boldsymbol{d}\| \\ \left(\bar{k}_p\bar{e}_p + \|\boldsymbol{d}\|\right) \end{bmatrix} \tag{4.80}$$

$$\mu \geq 2\left\|\boldsymbol{q}_{e,1:3}\right\| \|q_{e,0}\| \tag{4.81}$$

Avec $\underline{\lambda}(\boldsymbol{W}_1)$ *la plus petite valeur propre de* $\boldsymbol{W}_1$ *et* $\boldsymbol{d} = \ddot{\boldsymbol{p}}_{ref} + g\boldsymbol{z}_i$.

Alors, le point d'équilibre $\{\boldsymbol{e}_p\ ,\ \boldsymbol{e}_v\ ,\boldsymbol{q}_e\} = \{\boldsymbol{0},\ \boldsymbol{0},\ \boldsymbol{q}_I\}$ *est asymptotiquement stable.*

Bien que ce théorème énonce la stabilité de notre système combiné à la loi de commande, il ne le défini que pour un domaine restreint D. Ceci dit, nous souhaitons que notre correcteur garantisse une erreur de suivi nulle pour toute erreur initiale. À partir des conclusions dans 4.2.2, nous savons que l'erreur angulaire décroît asymptotiquement jusqu'à une erreur nulle. Il est donc possible de se rendre dans D en un temps fini t^* peu importe l'erreur en attitude initiale. Si $\mathbf{e}_p$ et $\mathbf{e}_v$ sont bornés dans l'intervalle $[0, t^*]$, alors l'erreur totale est bornée pour tout $t > 0$ et décroît asymptotiquement pour $t > t^*$. Le correcteur garantit donc une erreur de suivi nulle pour tout état initial. Pour montrer que l'erreur est bornée, on pose la fonction Γ qui représente l'erreur de suivi en position et en vitesse :

$$\Gamma = \frac{1}{2}\|\mathbf{e}_p\|^2 + \frac{1}{2}\|\mathbf{e}_v\|^2 \tag{4.82}$$

pour laquelle on peut directement déduire :

$$\|\mathbf{e}_p\| < \sqrt{2\Gamma} \tag{4.83}$$

$$\|\mathbf{e}_v\| < \sqrt{2\Gamma} \tag{4.84}$$

On trouve la dynamique de la fonction Γ :

$$\begin{aligned}
\dot{\Gamma} &= \mathbf{e}_p^\top \dot{\mathbf{e}}_p + \mathbf{e}_v^\top \dot{\mathbf{e}}_v \\
&= \mathbf{e}_p^\top \mathbf{e}_v + \mathbf{e}_v^\top \left(-\mathbf{K}_p \mathbf{e}_p - \mathbf{K}_v \mathbf{e}_v + \mathbf{b}\right) \\
&\leq \|\mathbf{e}_p\| \|\mathbf{e}_v\| + \|\mathbf{e}_v\| \left(\bar{k}_p \|\mathbf{e}_p\| + \bar{k}_v \|\mathbf{e}_v\| + \|\mathbf{b}\|\right) \\
&\leq \left(1 + \bar{k}_p\right) \|\mathbf{e}_p\| \|\mathbf{e}_v\| + \bar{k}_v \|\mathbf{e}_v\|^2 + \|\mathbf{b}\| \|\mathbf{e}_v\| \\
&\leq d_1 \Gamma + d_2 \sqrt{\Gamma}
\end{aligned}$$

avec $d_1 = 2(1 + \bar{k}_p + \bar{k}_v)$ et $d_2 = \sqrt{2}\|\mathbf{b}\|$. On sait que lorsque $\Gamma \geq 1$, $\sqrt{\Gamma} \leq \Gamma$. Ainsi, pour $\Gamma \geq 1$, nous avons :

$$\dot{\Gamma} \leq (d_1 + d_2)\, \Gamma \tag{4.85}$$

En utilisant l'inégalité de Grönwall avec l'inégalité différentielle (4.85), on trouve :

$$\Gamma(t) \leq \Gamma(t_a) e^{(d_1 + d_2)(t - t_a)} \tag{4.86}$$

avec $t_a \in [0, t^*]$.

On peut donc dire que l'erreur de suivi $\{\mathbf{e}_p, \mathbf{e}_v\}$ est bornée puisque la fonction Γ est bornée pour tout intervalle de temps. Nous pouvons donc énoncer le théorème suivant :

Théorème 4.2.5 (Attraction de la loi de commande) *Considérons la loi de commande et le systèmes définis dans le* ***Théorème 4.2.4****, pour une erreur initiale :*

$$\boldsymbol{q}_e \notin D \tag{4.87}$$

La loi de commande garanti que $\boldsymbol{q}_e$ *se rend dans le domaine* D *de stabilité asymptotique en un temps fini* t^*.

Alors, le point d'équilibre $\{\boldsymbol{e}_p\ ,\ \boldsymbol{e}_v\ ,\boldsymbol{q}_e\} = \{\boldsymbol{0},\ \boldsymbol{0},\ \boldsymbol{q}_I\}$ *est atteint peu importe l'erreur* $\boldsymbol{q}_e$ *initiale.*

4.3 Essais en simulation

4.3.1 Implémentation de la loi de commande

Afin de pouvoir être réutilisable pour de futurs travaux expérimentaux, la loi de commande a été implémentée en C++ en utilisant ROS. Le processus de commande a été séparé en deux processus afin de respecter l'architecture de commande. Un nœud ROS est utilisé pour l'exécution de la boucle externe et un autre nœud ROS est utilisé pour l'exécution de la boucle interne et de l'allocation. Afin de respecter les différentes vitesses des systèmes, la boucle externe est executée à une fréquence de 200 Hz et la boucle interne est exécutée à une vitesse de 1000 Hz. Le nœud de la boucle externe publie un *Topic* avec la poussée F désirée et les vitesses angulaires $\mathbf{\Omega}_{des}$. Le nœud de la boucle interne reçoit ces messages et calcule la vitesse angulaire désirée de chaque moteur individuel ω_i et les publie dans un autre *Topic*.

L'ajustement des gains a été fait selon une procédure itérative. Tout d'abord, les gains pour la boucle de commande interne ont été trouvés indépendamment de ceux de la boucle externe. Ces gains ont été trouvés de sorte à avoir un temps de réponse assez rapide pour suivre des commandes de vitesses angulaires. Ensuite, les gains pour le correcteur de position angulaire ont été trouvés en se basant sur les essais expérimentaux de [21]. Les gains du correcteur de position sont ensuite ajustés en fonction des gains du correcteur de position angulaire et de sorte à respecter les contraintes suivantes :

- dépassement maximal de 5% pour un échelon unitaire pour toute sortie x, y, z ;
- temps de réponse d'au plus 3 s pour un échelon unitaire selon z ;
- temps de réponse d'au plus 2.5 s pour un échelon unitaire selon x et y.

Cette procédure particulière peut être expliquée par le fait qu'il n'existe pas vraiment de théorie sur l'ajustement de gains pour les lois de commande non-linéaires. En général, les gains sont ajustés de manière intuitive selon la dynamique du système.

4.3.2 Simulations et Résultats

Les simulations ont été effectuées dans l'environnement décrit à la section 3.3 et en utilisant le drone décrit à la section 3.1.8. Plusieurs simulations ont été effectuées pour valider la stabilité et la performance de la loi de commande. Tout d'abord, nous présentons les réponses à une entrée échelon selon chacun des axes $\mathbf{x}_i$,$\mathbf{y}_i$, $\mathbf{z}_i$ après l'ajustement des gains aux figures 4.2,4.3 et 4.4 .

Nous notons un temps de réponse de 1.29 s, 1.77 s et 1.66 s, ainsi qu'un dépassement maximal de 1.1%, 1.4% et 1.66%, ce qui est conforme au cahier des charges établi.

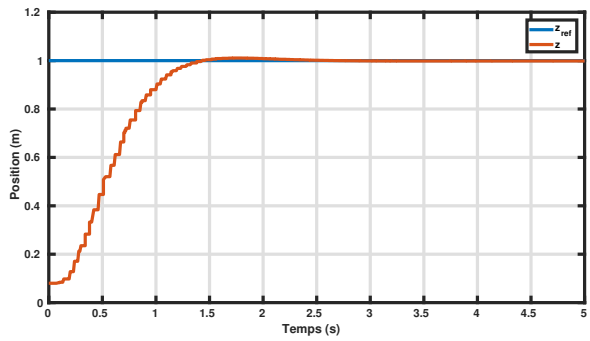


Figure 4.2 Réponse à l'échelon en **z**

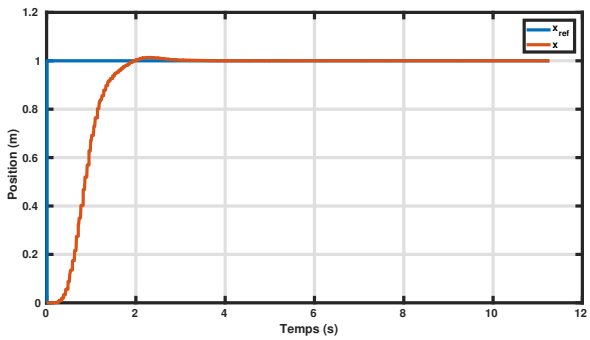


Figure 4.3 Réponse à l'échelon en **x**

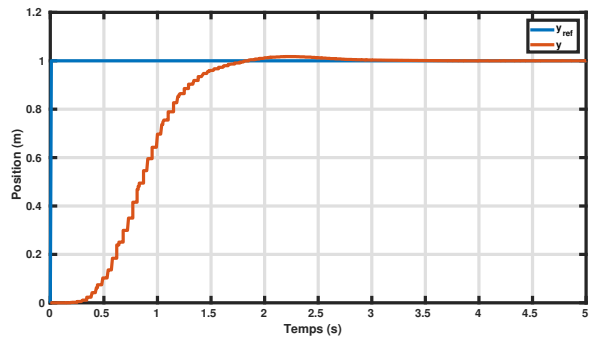


Figure 4.4 Réponse à l'échelon en **y**

Ensuite, pour illustrer les performances en suivi de trajectoire, deux trajectoires sont testées. Ces deux trajectoires sollicitent des efforts de commande selon les trois axes et représentent des trajectoires complexes. La première de ces trajectoires est une trajectoire circulaire de la forme :

$$x = \cos(0.75t)$$
$$y = \sin(0.75t)$$
$$z = 0.75$$

Pour effectuer la trajectoire, le drone est commandé à un vol stationnaire à une hauteur de 0.75 m et ensuite commandé à une position de 1 m selon $\mathbf{x}_i$. La trajectoire circulaire est ensuite amorcée. Une fois la trajectoire circulaire complétée, le multicoptère retourne en vol stationnaire à une hauteur de 0.75 m. Les figures 4.5, 4.6 et 4.7 présentent les résultats de suivi de trajectoire.

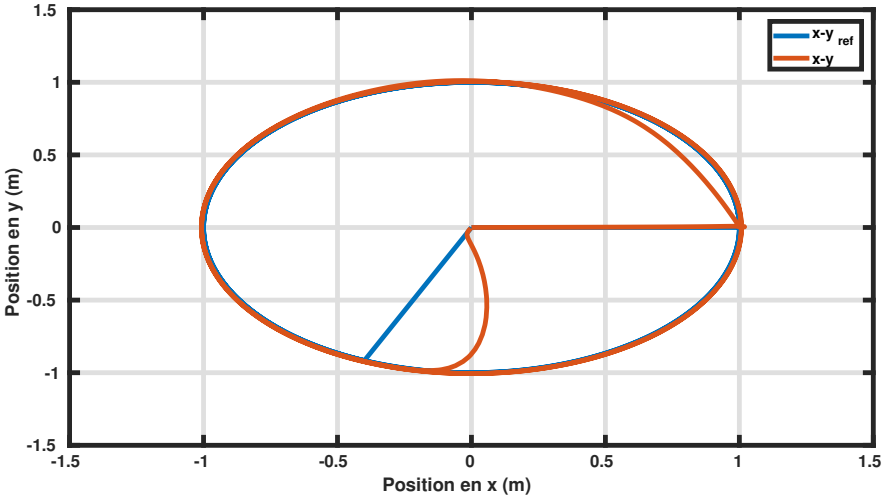


Figure 4.5 Trajectoire du multicoptère dans le plan $\mathbf{x}_i - \mathbf{y}_i$

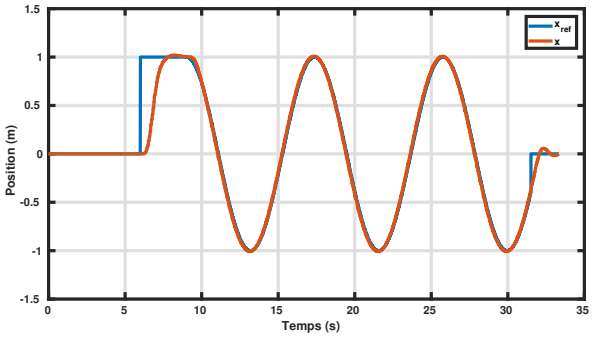


Figure 4.6 Trajectoire du multicoptère selon l'axe $\mathbf{x}_i$

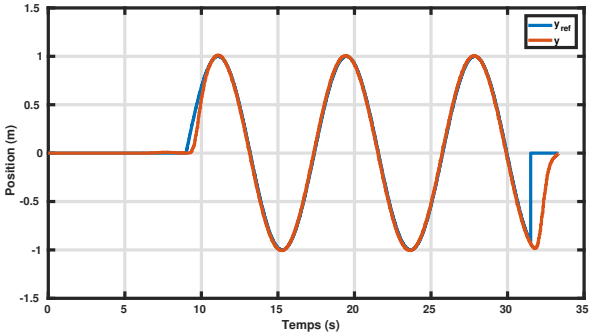


Figure 4.7 Trajectoire du multicoptère selon l'axe $\mathbf{y}_i$

Tel qu'illustré aux figures précédentes, nous notons une erreur de suivi quasi-nulle pour la trajectoire circulaire à l'exception des entrées échelon au début et à la fin de la séquence, ce qui valide la stabilité et la convergence de l'erreur de suivi vers une erreur nulle. Tout le long de la trajectoire, l'erreur moyenne est de 1.963 cm selon l'axe $\mathbf{x}_i$, 2.732 cm selon l'axe $\mathbf{y}_i$ et 0.165 cm selon l'axe $\mathbf{z}_i$, ce qui confirme les bonnes performances de suivi de la loi de commande conçue et est typique des erreurs de suivi pour multicoptères.

Pour le deuxième essai, une trajectoire plus agressive est suivie. En effet, cette trajectoire de

type hélicoïdale est représentée par :

$$x = \cos(t)$$
$$y = \sin(t)$$
$$z = 0.75 + 0.4t$$

Tout comme pour la première trajectoire, le drone est commandé à un vol stationnaire à une hauteur de 0.75 m et ensuite commandé à une position de 1 m selon $\mathbf{x}_i$. La trajectoire hélicoïdale est ensuite amorcée. Une fois la trajectoire complétée, le multicoptère retourne en vol stationnaire à une position nulle selon le plan $\mathbf{x}_i - \mathbf{y}_i$ tout en maintenant son altitude actuelle. Les figures 4.8, 4.9, 4.10 et 4.11 présentent les résultats de suivi de trajectoire.

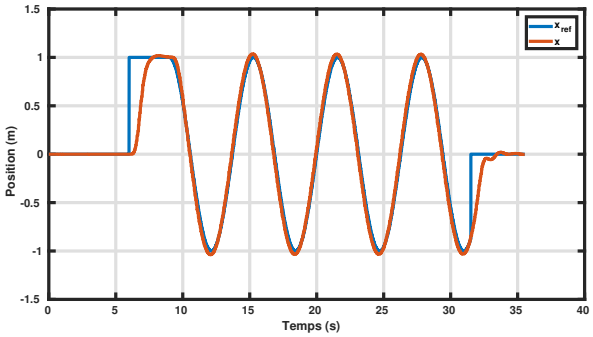


Figure 4.8 Trajectoire du multicoptère selon l'axe $\mathbf{x}_i$

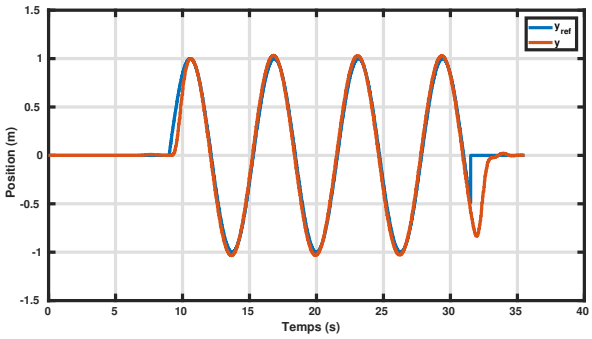


Figure 4.9 Trajectoire du multicoptère selon l'axe $\mathbf{y}_i$

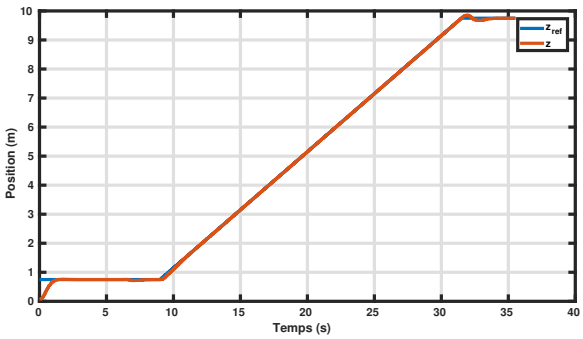


Figure 4.10 Trajectoire du multicoptère selon l'axe $\mathbf{z}_i$

À partir de ces figures, nous tirons les mêmes conclusions qu'à l'essai précédent, c'est-à-dire que l'erreur de suivi de trajectoire est quasi-nulle et que le correcteur conçu permet de faire le suivi adéquat de trajectoires de référence sollicitant des efforts selon chaque axe

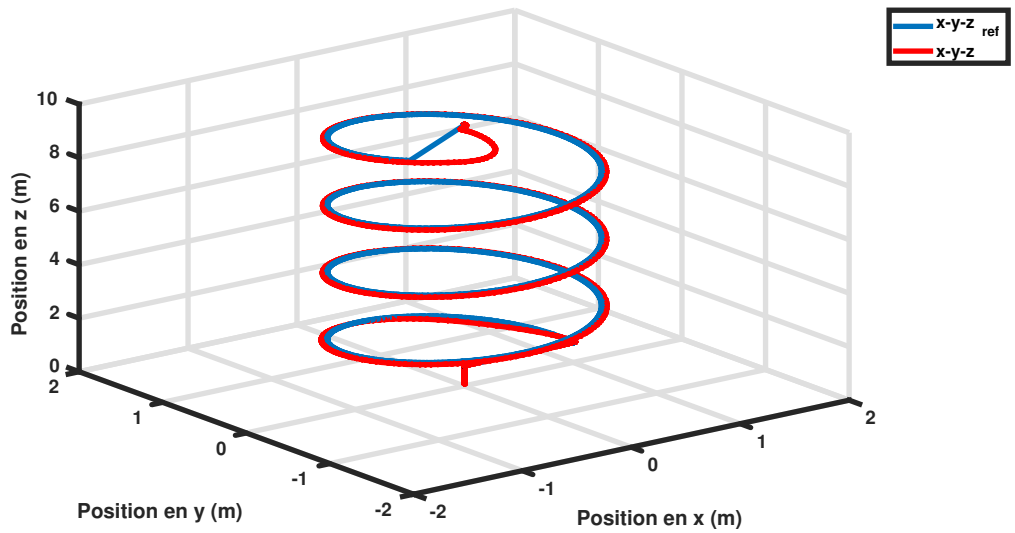


Figure 4.11 Trajectoire du multicoptère en 3 dimensions

simultanément. L'erreur moyenne selon chaque axe est, respectivement 4.782 cm, 5.492 cm et 0.845 cm.

Il est à noter qu'il faudrait effectuer plusieurs autres essais dans des scénarios variés afin de pouvoir pleinement valider les performances et la stabilité de ce correcteur. Pour ce faire, une multitude de primitives de mouvements présents dans les courses pour drone (manoeuvre de 180°, *zig-zag*, etc.) pourraient être suivies à différentes vitesses. À partir de ces résultats, les limites pratiques de la loi de commande pourraient être déterminées et la performance maximale de celle-ci sur une piste de course pour être quantifiable.

4.4 Conclusion

Dans cette section, nous avons présenté une loi de commande non-linéaire de suivi de trajectoire avec une boucle externe et une boucle interne. Nous avons démontré à l'aide de plusieurs théorèmes que celle-ci garantit une erreur de suivi nulle asymptotiquement stable, et ce, pour toute erreur initiale. Les gains de la loi de commande ont ensuite été ajustés selon une méthodologie itérative et la dynamique du système commandé a été simulée de façon *software-in-the-loop* dans l'environnement Gazebo modifié. Lors de ces simulations, nous avons trouvé un temps de réponse de 1.77 s, 1.66 s et 1.29 s selon les axes $\mathbf{x}_i$, $\mathbf{y}_i$ et $\mathbf{z}_i$ respectivement. Par la suite, les performances de suivi ont été évaluées sur une trajectoire circulaire et une trajectoire hélicoïdale. Dans les deux situations, le drone a suivi la trajectoire de référence avec très peu d'erreur, ce qui confirme les résultats théoriques obtenus.

CHAPITRE 5 GÉNÉRATION DE TRAJECTOIRES

Dans ce chapitre, nous nous intéressons à la méthode de génération de trajectoires utilisée pour parcourir une piste de course pour drone. Dans la section 5.1, nous commençons par présenter la formulation du problème de parcours optimal d'un multicoptère telle qu'elle est présentée dans [19,26]. Ensuite, à la section 5.2, nous reformulons le problème d'optimisation à partir des travaux de [26] afin d'améliorer la robustesse de la résolution du problème optimal. Pour terminer, nous présentons à la section 5.3 une méthode d'optimisation des segments de temps d'une trajectoire présentée dans [19] afin de minimiser la fonction de coût globale du parcours.

5.1 Formulation du problème d'optimisation

Étant dans un scénario de course, on souhaite parcourir une trajectoire passant le plus rapidement possible par plusieurs balises dans un ordre défini. Dans ce cas, la trajectoire optimale à parcourir serait la trajectoire qui minimise la fonctionnelle :

$$J = \int_0^T \mathrm{d}t$$

en incluant les équations dynamiques du multicoptère et la position des balises comme contraintes. Ceci dit, il n'y a pas de solution analytique à ce problème et la solution numérique est très longue à calculer et ne peut ainsi être utilisée en temps réel [9]. Une simplification de ce problème a été faite dans les travaux de [45] où la dynamique du multicoptère est réduite à celle d'une masse ponctuelle. En utilisant cette simplification avec des bornes sur l'accélération et sur la vitesse maximale, les auteurs trouvent une solution analytique au problème. Bien qu'elle garantisse un temps minimal, cette méthode néglige complètement la dynamique du drone et ne peut pas garantir qu'il est possible pour le drone d'atteindre l'état désiré.

À la section 3.2, nous avons montré qu'un multicoptère est différentiellement plat, c'est-à-dire qu'on peut exprimer tous les états du système à partir de sorties plates et de leurs dérivées. Dans le cas d'un drone, ces sorties plates sont $\left[\mathbf{p}^\top \psi\right]^\top$, soit quatre variables qui permettent d'exprimer entièrement une trajectoire. On note aussi que ces sorties doivent être dérivables au minimum 4 fois pour $\mathbf{p}$ et 2 fois pour ψ. Un choix simple pour une trajectoire ayant ces propriétés est une trajectoire polynomiale d'ordre suffisamment élevé qui sera facilement dérivable et simple à manipuler mathématiquement. Une trajectoire polynomiale qui ne dépasse

pas les limites des actionneurs du drone respecte donc indirectement sa dynamique. Il est donc possible de formuler un problème de trajectoire optimale avec des contraintes simples qui permet de trouver une trajectoire dynamiquement réalisable pour notre système sans inclure directement les équations dynamiques du multicoptère.

En analysant les relations entre les sorties plates, les entrées et les états, on note que les termes d'ordre 4 en $\mathbf{p}$ apparaissent dans M_x et M_y. Par ailleurs, on sait que les humains ont tendance à minimiser les variations d'accélération lorsqu'ils bougent. En effet, dans [57], les auteurs montrent que les humains minimisent la troisième dérivée de la position (*jerk*) lors de leur mouvement pour produire le mouvement le plus lisse possible. En nous basant sur ce constat, on en déduit qu'en minimisant la quatrième dérivée de la position (*snap*), on rend une trajectoire encore plus lisse. On propose donc de minimiser le *snap* de la trajectoire en fonction d'un temps de parcours prédéfini plutôt que de minimiser le temps de parcours avec des saturations en accélération et en vitesse. Ce choix devrait permettre de minimiser indirectement l'effort de commande tout en envoyant des commandes continues, ce qui rend la trajectoire plus facilement suivable. La fonctionnelle que l'on souhaite minimiser dans notre problème d'optimisation est :

$$J = \int_0^T [x^{(4)}(t)]^2 \mathrm{d}t \tag{5.1}$$

avec $x(t)$ la trajectoire désirée et T le temps de parcours total.

Pour une trajectoire polynomiale, l'équation (5.1) devient :

$$J = \int_0^T [\zeta^{(4)}(t)]^2 \mathrm{d}t \tag{5.2}$$

où le polynôme $\zeta(t)$ est donné par :

$$\zeta(t) = \zeta_N t^N + \zeta_{N-1} t^{N-1} + ... + \zeta_0 = \sum_{i=0}^{N} \zeta_i t^i \tag{5.3}$$

avec N le degré du polynôme et ζ_i ses coefficients. Dans la suite, on note $\boldsymbol{\zeta} \in \mathbb{R}^{N+1}$ le vecteur composé des coefficients polynomiaux de $\zeta(t)$, soit

$$\boldsymbol{\zeta} = \begin{bmatrix} \zeta_0 & \zeta_1 & \dots & \zeta_n \end{bmatrix}^\top \tag{5.4}$$

Nous allons montrer qu'il est possible d'exprimer (5.2) sous une forme quadratique :

$$J = \frac{1}{2} \boldsymbol{\zeta}^\top \mathbf{Q} \boldsymbol{\zeta} \tag{5.5}$$

avec $\mathbf{Q}$ une matrice symétrique semi-définie positive représentant la Hessienne de J par rapport aux coefficients polynomiaux. Cette forme est plus appropriée, car on peut résoudre ce problème numériquement avec des solveurs pour des problèmes quadratiques.

Matrice Hessienne

La dérivée d'ordre r $(1 \leq r \leq N)$ du polynôme $\zeta(t)$ est donnée par :

$$\zeta^{(r)}(t) = \sum_{i=r}^{N} \left(\prod_{j=0}^{r-1} (i-j) \right) \zeta_i t^{i-r} \tag{5.6}$$

Pour déterminer la matrice Hessienne $\mathbf{Q}$, on commence par la dérivée d'ordre 4 de $\zeta(t)$, soit :

$$\zeta^{(4)}(t) = \sum_{i=4}^{N} \left(\prod_{j=0}^{3} (i-j) \right) \zeta_i t^{i-4} \tag{5.7}$$

La mise au carré de $\zeta^{(4)}(t)$ donne alors :

$$[\zeta^{(4)}(t)]^2 = \sum_{i=8}^{2N} \sum_{j=4}^{N} \left(\prod_{m=0}^{3} (j-m)(i-j-m) \right) \zeta_j \zeta_{i-j} t^{i-8} \tag{5.8}$$

où les produits $\prod_{m=0}^{3}$ sont considérés nuls dès qu'un des facteurs est négatif. On peut donc réécrire (5.1) comme :

$$J = \int_0^T \sum_{i=8}^{2N} \sum_{j=4}^{N} \left(\prod_{m=0}^{3} (j-m)(i-j-m) \right) \zeta_j \zeta_{i-j} t^{i-8} \mathrm{d}t \tag{5.9}$$

$$= \sum_{i=8}^{2N} \sum_{j=4}^{N} \left(\prod_{m=0}^{3} (j-m)(i-j-m) \right) \zeta_j \zeta_{i-j} \frac{T^{i-7}}{i-7} \tag{5.10}$$

Avec la reformulation de la fonctionnelle, on peut trouver $\mathbf{Q}$ en dérivant (5.10) deux fois par rapport aux coefficients de $\zeta(t)$. La première dérivée de J par rapport au coefficient ζ_k est :

$$\frac{\partial J}{\partial \zeta_k} = \sum_{i=8}^{2N} \sum_{j=4}^{N} \left(\prod_{m=0}^{3} (j-m)(i-j-m) \right) \left(\frac{\partial \zeta_j}{\partial \zeta_k} \zeta_{i-j} + \frac{\partial \zeta_{i-j}}{\partial \zeta_k} \zeta_j \right) \frac{T^{i-7}}{i-7} \tag{5.11}$$

$$= 2 \sum_{i=8}^{2N} \left(\prod_{m=0}^{3} (k-m)(i-k-m) \right) \zeta_{i-k} \frac{T^{i-7}}{i-7} \tag{5.12}$$

En dérivant (5.12) par rapport à ζ_l, on trouve :

$$\frac{\partial^2 J}{\partial \zeta_k \partial \zeta_l} = 2 \sum_{i=8}^{2N} \left(\prod_{m=0}^{3} (k-m)(i-k-m) \right) \frac{\partial \zeta_{i-k}}{\partial \zeta_l} \frac{T^{i-7}}{n-7} \tag{5.13}$$

$$= 2 \left(\prod_{m=0}^{3} (k-m)(l-m) \right) \frac{T^{k+l-7}}{k+l-7} \tag{5.14}$$

À partir de (5.14), on peut construire la matrice Hessienne Q :

$$\mathbf{Q}_{kl} = \begin{cases} 2 \left(\prod_{m=0}^{3} (k-m)(l-m) \right) \frac{T^{k+l-7}}{k+l-7} & \text{si } k \geq 4 \wedge l \geq 4 \\ 0 & \text{si } k < 4 \vee l < 4 \end{cases} \tag{5.15}$$

avec k et l les indices respectifs des lignes et des colonnes de $\mathbf{Q}$.

Contraintes sur la trajectoire

Afin d'assurer la continuité entre chaque segment de la trajectoire complète et pour que la trajectoire passe par chacune des balises, des contraintes sur les positions et les dérivées doivent être ajoutées au problème d'optimisation. Pour chaque polynôme, on peut spécifier des conditions limites au début du segment et à la fin du segment, ou aussi les laisser libres. Pour un seul segment, les contraintes peuvent être exprimées par la fonction linéaire :

$$\mathbf{A}\boldsymbol{\zeta} = \mathbf{b} \tag{5.16}$$

$$\begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_T \end{bmatrix} \boldsymbol{\zeta} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_T \end{bmatrix} \tag{5.17}$$

L'équation $\mathbf{A}_0\boldsymbol{\zeta} = \mathbf{b}_0$ impose les conditions limites au début du segment et l'équation $\mathbf{A}_T\boldsymbol{\zeta} = \mathbf{b}_T$ impose celles en fin de segment au temps T. On suppose ci-après que l'on impose des conditions aux limites à la position et à toutes les dérivées successives jusqu'à l'ordre r. La ligne i des matrices $\mathbf{A}_0$ et $\mathbf{A}_T$ correspond au polynôme dérivé à l'ordre $i-1$, et la colonne j au coefficient ζ_{j-1}.

À $t = 0$, la matrice $\mathbf{A}_0$ s'écrit :

$$\mathbf{A}_{0,ij} = \begin{cases} (i-1)! & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases} \tag{5.18}$$

avec $0! = 1$ selon la convention du produit vide.

À $t = T$, la matrice $\mathbf{A}_T$ s'écrit :

$$\mathbf{A}_{T,ij} \begin{cases} \left(\prod_{m=0}^{i-2} (j-1-m)\right) T^{j-i} & \text{si } i \geq j \\ 0 & \text{si } i < j \end{cases} \tag{5.19}$$

avec $\prod_{m=0}^{-1} = 1$ selon la convention du produit vide.

Les vecteurs $\mathbf{b}_0$ et $\mathbf{b}_T$ correspondent aux valeurs des positions et des dérivées que l'on souhaite imposer :

$$\mathbf{b}_{0,i} = \zeta^{(i-1)}(0) \tag{5.20}$$

$$\mathbf{b}_{T,i} = \zeta^{(i-1)}(T) \tag{5.21}$$

Les matrices $\mathbf{A}_0$ et $\mathbf{A}_T$ ont été construites en supposant que l'on contraignait les positions et toutes les dérivées au début et à la fin du segment. Lorsque ce n'est pas le cas, les lignes de $\mathbf{A}_0$ ou de $\mathbf{A}_T$, ainsi que les éléments correspondants dans $\mathbf{b}$ pour les dérivées non-contraintes sont enlevées.

Pour une trajectoire composée de plusieurs segments, des contraintes supplémentaires sont nécessaires. En effet, on doit s'assurer d'une continuité entre les dérivées à la fin d'un segment et les dérivées au début du segment suivant. On exprime cette relation par :

$$\mathbf{A}_T^i \boldsymbol{\zeta}_i - \mathbf{A}_0^{i+1} \boldsymbol{\zeta}_{i+1} = 0 \tag{5.22}$$

Formulation complète du problème

Dans le cas de notre application, on souhaite imposer des contraintes jusqu'au *snap* (dérivée d'ordre 4). On choisit alors de prendre des polynômes d'ordre 9 pour chaque segment, afin d'imposer cinq conditions limites au début du segment et cinq conditions limites à la fin du segment. La trajectoire finale est alors la concaténation de plusieurs segments polynomiaux reliant chacune des balises dans un ordre prédéfini. Il est donc possible de définir une fonctionnelle J_t représentant le coût total de la trajectoire :

$$J_t = \frac{1}{2} \bar{\boldsymbol{\zeta}}^\top \mathbf{Q}_t \bar{\boldsymbol{\zeta}} \tag{5.23}$$

avec $\bar{\boldsymbol{\zeta}}$ la concaténation des vecteurs $\boldsymbol{\zeta}_i$ de chaque segment de trajectoire et $\mathbf{Q}_t$ la matrice bloc diagonale composée des matrices $\mathbf{Q}_i$ de chaque segment.

Cette fonction de coût est soumise aux contraintes suivantes :

— contraintes initiales complètes pour le premier segment ;
— contrainte de position à la fin de chaque segment (position des balises) ;
— contraintes de continuité entre chaque segment.

Ces contraintes se traduisent par les contraintes linéaires :

$$\bar{\mathbf{A}}\bar{\boldsymbol{\zeta}} = \bar{\mathbf{b}}$$

$$\begin{bmatrix} \mathbf{A}_0^i & \mathbf{0}_{5\times10} & \mathbf{0}_{5\times10} & \mathbf{0}_{5\times10} & \cdots & \mathbf{0}_{5\times10} \\ \mathbf{A}_{T,1}^i & \mathbf{0}_{1\times10} & \mathbf{0}_{1\times10} & \mathbf{0}_{1\times10} & \cdots & \mathbf{0}_{1\times10} \\ \mathbf{A}_T^i & -\mathbf{A}_0^{i+1} & \mathbf{0}_{5\times10} & \mathbf{0}_{5\times10} & \cdots & \mathbf{0}_{5\times10} \\ \mathbf{0}_{1\times10} & \mathbf{A}_{T,1}^{i+1} & \mathbf{0}_{1\times10} & \mathbf{0}_{1\times10} & \cdots & \mathbf{0}_{1\times10} \\ \mathbf{0}_{5\times10} & \mathbf{A}_T^{i+1} & -\mathbf{A}_0^{i+2} & \mathbf{0}_{5\times10} & \cdots & \mathbf{0}_{5\times10} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0}_{1\times10} & \mathbf{0}_{1\times10} & \mathbf{0}_{1\times10} & \cdots & \mathbf{A}_{T,1}^{n-1} & \mathbf{0}_{1\times10} \\ \mathbf{0}_{5\times10} & \mathbf{0}_{5\times10} & \mathbf{0}_{5\times10} & \cdots & \mathbf{A}_T^{n-1} & -\mathbf{A}_0^n \\ \mathbf{0}_{1\times10} & \mathbf{0}_{1\times10} & \mathbf{0}_{1\times10} & \cdots & \mathbf{0}_{1\times10} & \mathbf{A}_{T,1}^n \end{bmatrix} \bar{\boldsymbol{\zeta}} = \begin{bmatrix} \begin{bmatrix} \zeta_{init} & \dot{\zeta}_{init} & \ddot{\zeta}_{init} & \zeta_{init}^{(3)} & \zeta_{init}^{(4)} \end{bmatrix}^\top \\ \zeta_{balise1} \\ \mathbf{0}_{5\times1} \\ \zeta_{balise2} \\ \mathbf{0}_{5\times1} \\ \vdots \\ \zeta_{balise\,n-1} \\ \mathbf{0}_{5\times1} \\ \zeta_{balise\,n} \end{bmatrix}$$

avec n le nombre total de balises.

Par ailleurs, puisque chacune des dimensions x, y, z est découplée l'une de l'autre, nous pouvons générer une trajectoire le long de chaque dimension indépendamment. Il est donc possible de résoudre 4 problèmes d'optimisation séparés plutôt qu'un seul de grande dimension pour trouver une trajectoire complète. Cette propriété permet d'améliorer la robustesse numérique et le temps de calcul de la solution. Nous illustrons un exemple de trajectoire généré à l'aide de cette méthode à la figure 5.1. Nous pouvons y observer une trajectoire générée le long de l'axe $\mathbf{x}_i$ pour la piste nominale des qualifications de la compétition *AlphaPilot* [2]. Les différentes balises de la piste y sont indiqués afin de montrer que la trajectoire générée respecte les contraintes.

5.2 Reformulation du problème

Bien que l'on puisse résoudre numériquement le problème d'optimisation (5.23) avec un solveur pour formulation quadratique pour des trajectoires simples, le conditionnement du problème augmente considérablement lorsque le nombre de segments qui composent la trajectoire augmente. En utilisant la formulation précédente du problème, la solution numérique nécessite l'inversion de matrices presque singulières. Les matrices utilisées sont très sensibles et les coefficients sont de l'ordre de 10^{-20}, ce qui mène à des résultats peu précis [26].

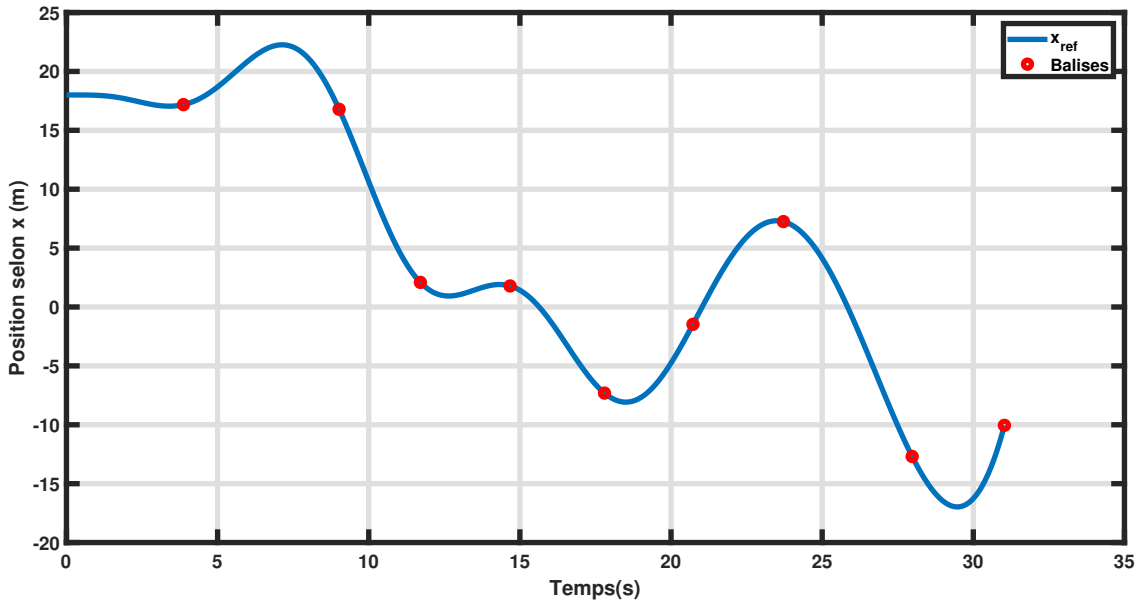


Figure 5.1 Trajectoire minimale en *snap* selon l'axe $\mathbf{x}_i$ pour la piste *AlphaPilot* [2]

De plus, la précision de la solution au problème varie grandement selon le solveur utilisé. Par expérience, nous avons trouvé que la qualité (préconditionnement des matrices, etc.) et la robustesse du solveur *quadprog* de MATLAB sont très bonnes. Ceci dit, ce solveur n'est pas utilisable pour des scénarios dans lesquels un ordinateur embarqué utilisant ROS doit être utilisé. De nombreux solveurs Python et C++ ont été testés et aucun d'entre eux n'est proche de la performance des solveurs dans MATLAB. À partir de ces constats et des problèmes identifiés quant à la sensibilité des matrices utilisées dans le problème optimal, le problème d'optimisation avec contraintes est reformulé en un problème sans contraintes en intégrant les contraintes directement dans la formulation de la fonction de coût selon la méthodologie proposée par [26]. Cette reformulation permet de transformer le problème d'optimisation avec contraintes en problèmes d'optimisation sans constraintes. Cette reformulation permet d'être beaucoup plus robuste numériquement [26].

5.2.1 Reformulation des contraintes

Dans la section précédente, le *snap* d'une trajectoire polynomiale était optimisée par rapport aux coefficients polynomiaux. Les contraintes en début et en fin de segment, ainsi que la continuité entre les segments étaient exprimées directement dans les contraintes linéaires du problème d'optimisation. Dans cette sous-section, nous reformulons le problème d'optimisation de sorte à ce qu'on optimise par rapport à des conditions limites de la trajectoire plutot que ces coefficients polynomiaux tel que proposé dans les travaux de [26]. Pour ce faire, nous

commencons par définir deux termes : les conditions limites fixes $\mathbf{D}_F$ et les conditions limites libres $\mathbf{D}_P$. Les conditions limites fixes correspondent à des contraintes sur la trajectoire. Dans notre cas, ces conditions correspondent à la position des différentes balises et de l'état initial et final de notre trajectoire. Ces valeurs sont connues avant la résolution du problème d'optimisation et sont considérées comme les entrées du problème. Les conditions libres correspondent à tous les autres états de la trajectoire qui ne sont pas fixés par les conditions fixes. Ces contraintes peuvent être concaténés en un vecteur $\mathbf{D}$:

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_F \\ \mathbf{D}_P \end{bmatrix} \tag{5.24}$$

Nous donnons aussi la relation qui existe entre les dérivées en début et en fin d'un segment de trajectoire en fonction des coefficients polynomiaux $\boldsymbol{\zeta}$:

$$\bar{\mathbf{d}} = \mathbf{C}\bar{\boldsymbol{\zeta}} = \begin{bmatrix} \mathbf{C}_0 \\ \mathbf{C}_T \end{bmatrix} \bar{\boldsymbol{\zeta}} \tag{5.25}$$

avec $\mathbf{C}_0\boldsymbol{\zeta}$ et $\mathbf{C}_T\boldsymbol{\zeta}$ représentant toutes les dérivées aux temps 0 et T. Les matrices $\mathbf{C}_0$ et $\mathbf{C}_T$ correspondent aux matrices $\mathbf{A}_0$ et $\mathbf{A}_T$ lorsque des contraintes sur toutes les dérivées sont appliquées.

Pour incorporer les conditions limites $\mathbf{D}$ dans la relation (5.25), on multiplie $\mathbf{D}$ par une matrice d'allocation $\mathbf{M}$ qui associe chacune des dérivées des segments $\bar{\mathbf{d}}$ à une condition limite fixe $\mathbf{D}_F$ ou à une condition limite libre $\mathbf{D}_P$. Par exemple, on pourrait contraindre seulement la position en début et en fin de chaque segment et laisser les autres dérivées libres. Pour prendre en compte les contraintes de continuité, on s'assure que la matrice d'allocation associe la même valeur de dérivée à la fin d'un segment et au début du prochain. L'équation (5.25) donne alors :

$$\mathbf{M} \begin{bmatrix} \mathbf{D}_F \\ \mathbf{D}_P \end{bmatrix} = \mathbf{C}\bar{\boldsymbol{\zeta}} \Rightarrow \bar{\boldsymbol{\zeta}} = \mathbf{C}^{-1}\mathbf{M} \begin{bmatrix} \mathbf{D}_F \\ \mathbf{D}_P \end{bmatrix} \tag{5.26}$$

À partir de cette relation, nous pouvons reformuler le problème de trajectoire optimal présenté

à (5.23). La fonction de coût devient :

$$J = \bar{\boldsymbol{\zeta}}^\top \bar{\mathbf{Q}} \bar{\boldsymbol{\zeta}} \tag{5.27}$$

$$J = \begin{bmatrix} \mathbf{D}_F \\ \mathbf{D}_P \end{bmatrix}^\top \mathbf{M}^\top \mathbf{C}^{-\top} \bar{\mathbf{Q}} \mathbf{C}^{-1} \mathbf{M} \begin{bmatrix} \mathbf{D}_F \\ \mathbf{D}_P \end{bmatrix} \tag{5.28}$$

Avec cette reformulation, les coefficients $\bar{\boldsymbol{\zeta}}$ ne sont plus directement les variables d'optimisation, mais plutôt les conditions limites libres $\mathbf{D}_P$. Par ailleurs, les contraintes sont directement comprises dans la formulation du problème donc le problème d'optimisation avec contraintes devient un problème d'optimisation sans contraintes. La solution au problème d'optimisation est donc :

$$\frac{\partial J}{\partial \mathbf{D}_P} = \mathbf{0} \tag{5.29}$$

Pour résoudre ce problème plus facilement, nous posons :

$$\mathbf{R} = \mathbf{M}^\top \mathbf{C}^{-\top} \mathbf{Q} \mathbf{C}^{-1} \mathbf{M} \tag{5.30}$$

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_{FF} & \mathbf{R}_{FP} \\ \mathbf{R}_{PF} & \mathbf{R}_{PP} \end{bmatrix} \tag{5.31}$$

La fonction de coût devient :

$$J = \mathbf{D}_F^\top \mathbf{R}_{FF} \mathbf{D}_F + \mathbf{D}_F^\top \mathbf{R}_{FP} \mathbf{D}_P + \mathbf{D}_P^\top \mathbf{R}_{PF} \mathbf{D}_F + \mathbf{D}_P^\top \mathbf{R}_{PP} \mathbf{D}_P \tag{5.32}$$

Et la solution de (5.29) est donnée par :

$$\mathbf{D}_p^{opt} = -\mathbf{R}_{PP}^{-1} \mathbf{R}_{FP}^\top \mathbf{D}_F \tag{5.33}$$

Ce qui veut dire qu'il n'existe qu'un seul extremum. En dérivant une autre fois par rapport à $\mathbf{D}_p$, on trouve :

$$\frac{\partial^2 J}{\partial \mathbf{D}_p^2} = \mathbf{R}_{PP} \tag{5.34}$$

Sachant que $\mathbf{R}_{pp}$ est définie positive par construction, la matrice (5.33) est le minimum global de la fonctionnelle. Puisque la valeur de $\mathbf{D}_F$ est définie en début de problème de par les contraintes sur les segemnts imposées, la valeur de $\mathbf{D}_p^{opt}$ est connue. On utilise ensuite (5.26) pour trouver les coefficients polynomiaux qui correspondent aux conditions libres et

la trajectoire optimale peut être calculée.

À la figure 5.2, nous comparons la trajectoire obtenue selon la méthode de résolution de problème quadratique avec celle présentée à cette section pour la même situation que celle présentée à la section 5.1. En observant le graphique, nous notons que les deux courbes sont superposées, ce qui confirme la validité de la nouvelle méthode proposée. Par ailleurs, le coût associé à chacune de ces trajectoires est identique à $28.5789\,(m/s^4)^2$.

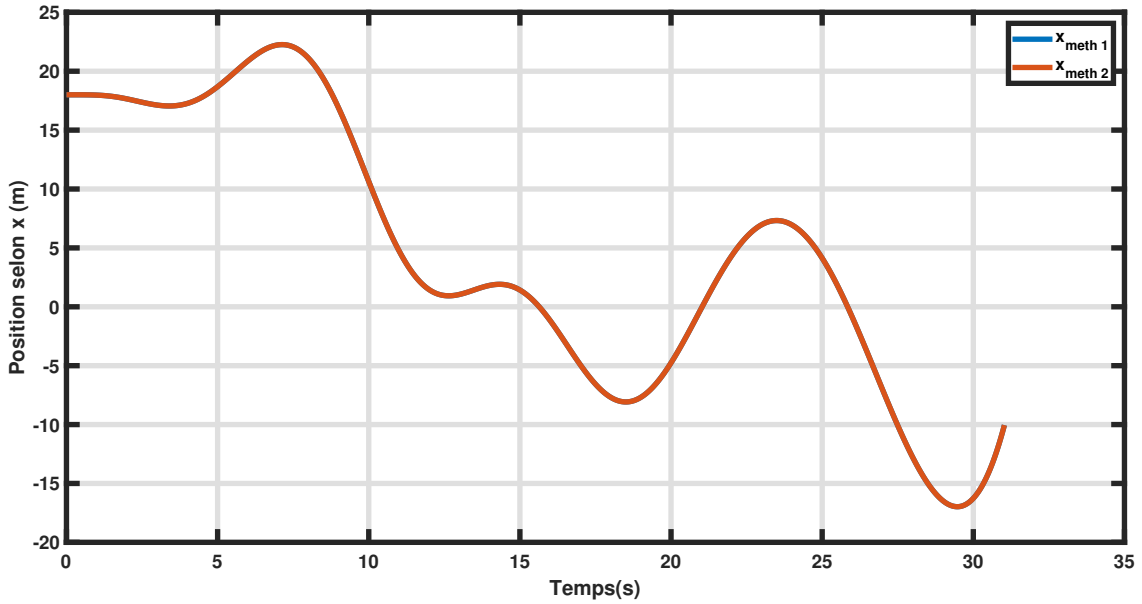


Figure 5.2 Comparaison entre les trajectoires générées avec problème d'optimisation quadratique et avec une résolution directe

5.3 Optimisation du temps

Dans les sections précédentes, on trouve une trajectoire minimisant le *snap* pour des temps de segments connus. Ceci dit, ces temps sont souvent choisis instinctivement et ne représentent pas nécessairement la répartition optimale du temps total entre segments. Par exemple, supposons que nous désirons parcourir une piste de course pour drone donnée (Fig. 5.1). Cette piste peut être constituée de segments droits, de longs virages, de virages abruptes, etc. Certains segments vont nécessiter plus de temps que d'autres selon le comportement qu'on désire obtenir. Afin de s'assurer d'avoir une répartition optimale du temps entre les différents segments au niveau du *snap* global, nous posons le problème d'optmisation tiré de [19], dans lequel le vecteur de temps $\mathbf{T}$ représentant le temps alloué à chaque segment de

trajectoire est optimisé :

$$\min \ f(\mathbf{T}) \tag{5.35}$$

$$\text{t.q.} \sum T_i = t_m \tag{5.36}$$

$$T_i \geq 0 \tag{5.37}$$

avec $f(\mathbf{T})$, la somme des solutions pour x,y et z au problème (5.23),T_i le temps alloué au $i^{ème}$ segment et t_m le temps de parcours total désiré (qui est fixe).

Ce problème d'optimisation peut être résolu avec un solveur numérique. Dans notre application, la fonction *fmincon* de MATLAB est utilisée pour trouver la solution au problème. Un exemple d'optimisation de trajectoire est présenté à la figure 5.3. Dans cet exemple, nous reprenons la trajectoire en 3 dimensions présentée à la figure 5.1 avec le temps déterminé de sorte à maintenir une vitesse moyenne constante de 6 m/s selon la distance en ligne droite entre chacune des balises. Le temps de chaque segment est ensuite optimisé selon la fonction de coût présentée plus haut. À partir de la figure 5.3, nous constatons clairement l'effet de l'optimisation du temps sur l'allure de la trajectoire. En effet, la trajectoire globale semble beaucoup plus lisse avec des virages avec un rayon de courbure plus grand. La comparaison du coût total de la trajectoire mène à la même conclusion : le coût pour la trajectoire non-optimisée est de $139.389\,(m/s^4)^2$ alors que celui de la trajectoire optimisée est de $27.3953\,(m/s^4)^2$.

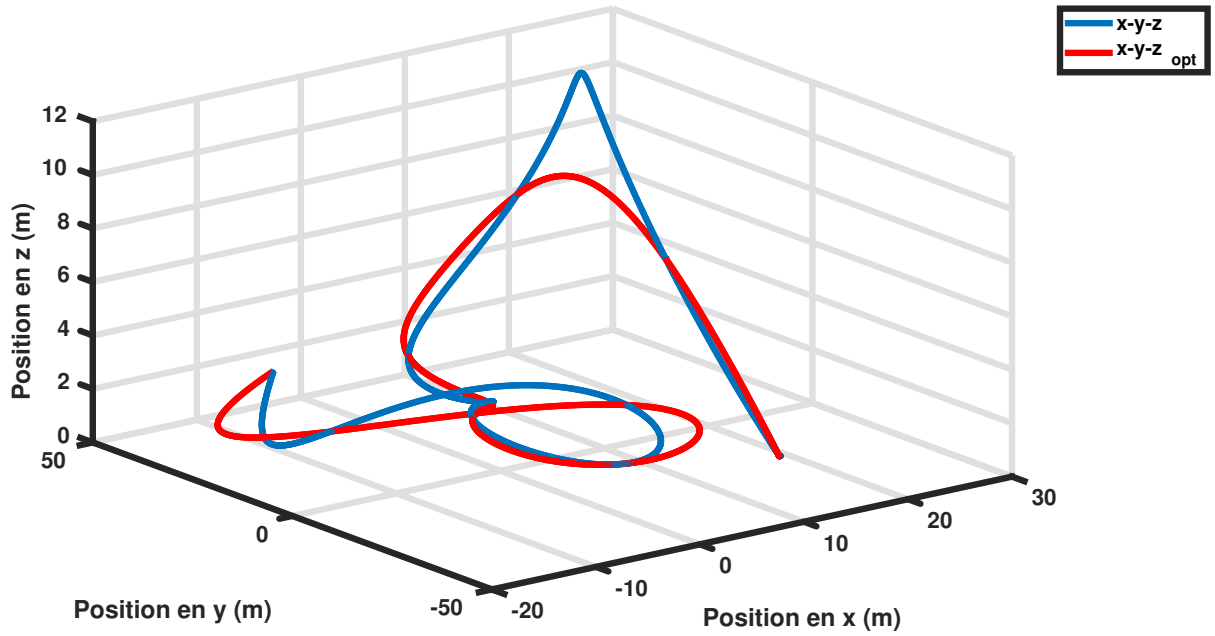


Figure 5.3 Trajectoire minimale en *snap* avec répartition des segments de temps optimisée

5.4 Conclusion

Dans ce chapitre, nous avons présenté les fondements théoriques d'une méthode de génération de trajectoires polynomiales minimisant le *snap* d'une trajectoire pour un temps de parcours fixe. La méthode est ensuite appliquée à un scénario de course de drone pour présenter sa validité. Par la suite, des modifications à la formulation du problème sont appliquées afin d'améliorer la robustesse numérique du problème pour des applications embarquées. Nous trouvons des résultats identiques à ceux trouvés par la méthode quadratique. Pour terminer, nous présentons une méthode d'optimisation de la répartition du temps entre les différents segments polynomiaux afin de réduire le *snap* d'une trajectoire tridimensionnelle. Un exemple de l'effet de cette optimisation est présentée pour une piste de course pour drone. Une réduction de 80.35% du *snap* global est observé.

CHAPITRE 6 PARCOURS D'UN ENVIRONNEMENT DE COURSE

Le but de ce chapitre est de présenter la méthodologie utilisée pour traduire les contraintes de course de drones piloté en contraintes pour un système autonome, ainsi que de simuler le comportement du système autonome développé. Tout d'abord, une présentation des contraintes pour le système autonome est faite à la section 6.1, suivie par la solution elle-même à la section 6.2. Ensuite, une présentation des algorithmes de détection et de cartographie des balises est faite à la section 6.3. La méthode de génération de trajectoires pour le parcours de la piste est formulée à la section 6.4. Pour terminer, les algorithmes développés seront simulés et analysés sur plusieurs pistes typiques de course de drone à la section 6.5.

6.1 Description de la problématique

Dans le contexte général d'une course de drone à la première personne piloté par un humain, l'objectif est de parcourir une piste le plus rapidement possible. Une piste est définie par différentes balises qui doivent être atteintes dans un ordre prédéfini selon une orientation particulière. Tout comme les travaux de [3, 36, 44, 45], nous modélisons une course comme une piste qui doit être parcourue pour plusieurs tours.

Pour la parcourir plus rapidement, les pilotes ont souvent une connaissance générale de l'allure de la piste pour laquelle ils prévoient une trajectoire grossière à suivre. Sachant cela, on peut définir la tâche du pilote comme :

- parcourir une piste grossièrement connue le plus rapidement possible ;
- ajuster sa trajectoire initiale en fonction de l'environnement qu'il percoit ;
- exécuter la trajectoire qu'il a prévu sans trop d'erreurs.

À partir de ces éléments, un système de pilotage autonome équivalent peut être conçu. En ce qui a trait à l'information connue à priori, tout comme le pilote, les algorithmes de pilotage autonome ont une idée générale de l'allure de la piste. Pour le pilote humain, l'idée générale de la piste provient souvent d'une vue de haut de la piste et une estimation de la position des points de passage. Pour ce qui est du système autonome, nous avons utilisé une approche différente. Nous avons modélisé la piste réelle comme étant une piste nominale à laquelle une erreur aléatoire aux bornes connues est ajoutée. La piste nominale peut donc être fournie aux algorithmes d'autonomie et agir comme l'équivalent de l'estimé initial de la piste pour le pilote humain. La borne des erreurs est aussi fournie à l'algorithme d'autonomie puisqu'elle lui permet d'identifier correctement les balises. En ce qui concerne les tâches que le système autonome doit accomplir, elles sont les mêmes que le pilote. Cependant, nous pouvons les

définir plus précisément comme suit :

— parcourir la piste le plus rapidement possible ;
— détecter les balises ;
— cartographier les balises détectées afin d'avoir une représentation plus précise de la piste ;
— adapter sa trajectoire en fonction des changements dans la carte des balises ;
— suivre la trajectoire le plus précisément possible.

6.2 Solution proposée

Pour résoudre la problématique de course de drone autonome, nous simplifions le processus de course en deux phases : la phase de découverte et la phase de performance. Nous définissons la phase de découverte comme étant la phase dans laquelle le drone parcourt la piste à plus basse vitesse afin de pouvoir identifier et cartographier les différentes balises de la piste. Lors de cette phase, la trajectoire n'est pas nécessairement optimale et le drone peut parcourir la piste plusieurs fois avant d'avoir correctement identifié toutes les balises. La phase de performance est la phase qui suit la phase d'exploration. Dans cette phase, la position des différentes balises est cartographiée grâce à la phase d'exploration et une trajectoire optimale au niveau de la performance est calculée et suivie. Les vitesses moyenne et maximale du drone lors de cette phase sont plus élevées que dans la phase de découverte, ce qui diminue le temps total de parcours. Nous posons l'hypothèse que cette simplification peut améliorer la performance des algorithmes d'autonomie par rapport à ceux présentés dans la littérature. En effet, les différents algorithmes d'autonomie présentés à la section 2 ne présentent pas de phase de performance et sont toujours en phase d'exploration, ce qui limite les performances.

Dans [37], la piste est parcourue de balise en balise par navigation de type ligne de visée visuelle sans construire de carte de la piste. Dans [41], une carte de la balise est disponible au début de la course, mais la trajectoire empruntée est construite en utilisant des segments simples et n'est pas optimale. Dans [36] et [44], bien que l'algorithme offre des bonnes performances de parcours autonome, aucune carte de la piste n'est construite, ni utilisée, et la trajectoire est déterminée de balise en balise, ce qui fait qu'il est impossible de noter une amélioration avec les tours complétés. Seuls les algorithmes présentés dans [3] et [45] effectuent une cartographie des différentes balises pour parcourir la piste. Ceci dit, ces algorithmes n'explorent pas l'utilisation d'une phase de performance pour le parcours d'une piste puisque l'utilisation de leur algorithmes de navigation et de commande est couplée à une odométrie de type visuelle-inertielle. Ce type d'odométrie n'est pas assez précis à haute vitesse et est sujet au phénomène de dérive, ce qui empêche d'avoir une carte précise et fixe des différentes

balises composant la piste. Il est donc impossible de pouvoir calculer une trajectoire optimale fixe et fiable pour parcourir la piste.

À partir des contraintes et des simplifications établies, nous proposons l'architecture d'autonomie suivante :

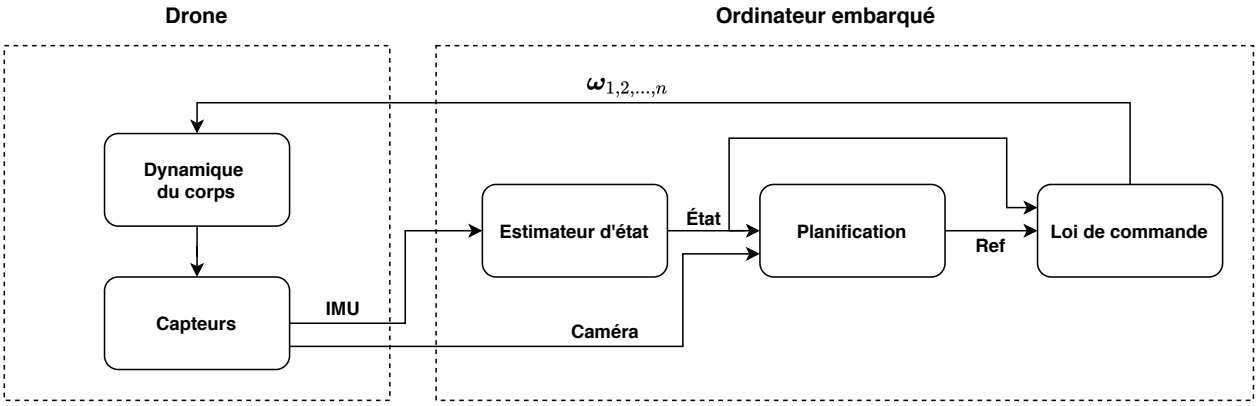


Figure 6.1 Architecture du système d'autonomie

Cette architecture s'inspire fortement des systèmes d'autonomie de l'état de l'art [3, 45]. Il y a 4 modules distincts : un estimateur d'état, un module de détection de balises, un module de cartographie et de planification, ainsi qu'une loi de commande pour suivre la trajectoire. Dans notre cas, l'estimateur d'état utilisé est un filtre de Kalman étendu de fusion de données multiples, MSF (*MultiSensor Fusion Framework*) [58]. Les données de la centrale inertielle sont utilisées pour faire la propagation de l'état et un capteur simulant un système de capture de mouvement est utilisé pour faire la mise à jour de l'état. La loi de commande développée au Chapitre 4 est reprise pour faire le suivi de trajectoire.

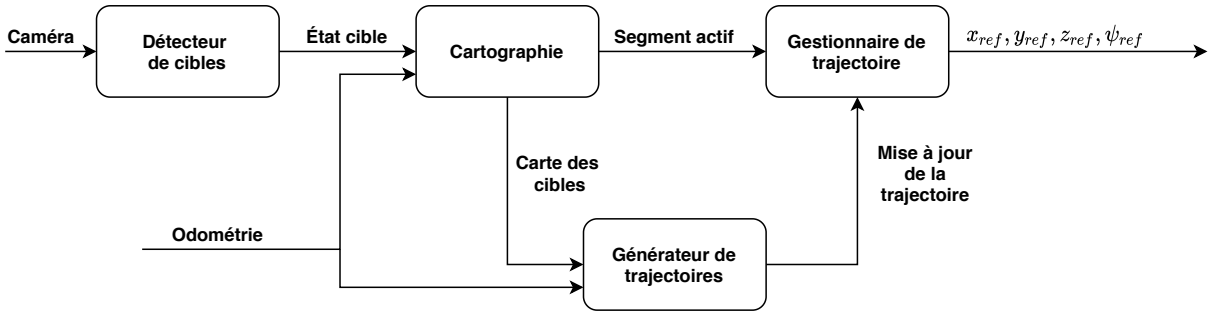


Figure 6.2 Architecture du bloc de planification

Dans le bloc de planification (Fig. 6.2), nous trouvons l'algorithme de détection de balise qui envoie les détections au module de cartographie. Les algorithmes de détection et de cartographie sont présentés à la section 6.3. Le module de cartographie suit la position du drone pour déterminer la prochaine balise à traverser et détermine lorsqu'on doit recalculer une nouvelle trajectoire. Finalement, le gestionnaire de trajectoire détermine les consignes à envoyer à la loi de commande selon la trajectoire mise à jour et selon la prochaine balise à traverser. Dans le bloc de génération de trajectoires, les algorithmes de génération de trajectoire présentés à la section 6.4 sont repris, mais avec quelques modifications pour améliorer la robustesse numérique, ainsi que pour prendre en compte différents délais dans le système.

Il est à noter que, puisque le sujet du mémoire est la navigation et le contrôle autonome d'un drone dans un scénario de course, nous supposons et utilisons une odométrie parfaite (sans erreur et sans dérive), ce qui n'est pas nécessairement valide pour les applications réelles. En effet, dans le cas de courses pour drones, la méthode principale d'odométrie est l'odométrie visuelle. À ce jour, les meilleures méthodes dans la littérature offrent une bonne précision à basse vitesse, mais les performances se dégradent grandement plus la vitesse augmente [32]. Pour l'instant, il s'agit donc d'une limitation de la solution proposée. Ceci dit, cette solution pourrait être appliquée dans un futur rapproché avec le développment et le raffinement des différents algorithmes d'autonomie visuelle. Par ailleurs, il est aussi à noter que les blocs de détection et de cartographie ne sont utilisés que pour permettre de simuler le comportement d'un système complet de course pour drones. Ceci permet aussi de rendre l'architecture modulaire et pourra être reprise dans le cadre de futurs travaux du laboratoire de recherche. L'objectif de l'utilisation de ces blocs est donc de détecter les changements dans la position des balises afin d'étudier la performance et le comportement de la loi de commande et de la méthode de navigation dans un contexte plus réaliste.

6.3 Détection et cartographie des balises

Dans cette section, nous présentons les détails de la détection de balises, soit la géométrie et les particularités visuelles des balises, les algorithmes utilisés pour les détecter et déterminer leur distance par rapport au drone, ainsi que l'algorithme utilisé pour cartographier les détections.

6.3.1 Formes des balises

La détection des balises a pour but de permettre au drone d'estimer la position réelle des balises et de réajuster sa trajectoire. Ceci dit, on souhaite valider les algorithmes de navigation et de contrôle et non développer un algorithme novateur de détection de balises de différentes

apparences dans des conditions visuelles difficiles. Sachant cela, chaque balise du parcours est identique, bien éclairée et libre d'occlusions afin de rendre la détection plus facile. De plus, la balise utilisée est conçue pour être facilement et robustement détectable, ainsi que d'être suffisamment large pour laisser une certaine marge de manœuvre au drone pour qu'il puisse passer au travers sans collision avec les parois. La forme et les caractéristiques visibles des balises utilisées dans la simulation de course de drone sont adaptées des travaux de [22]. L'allure de la balise est présentée à la figure 6.3.

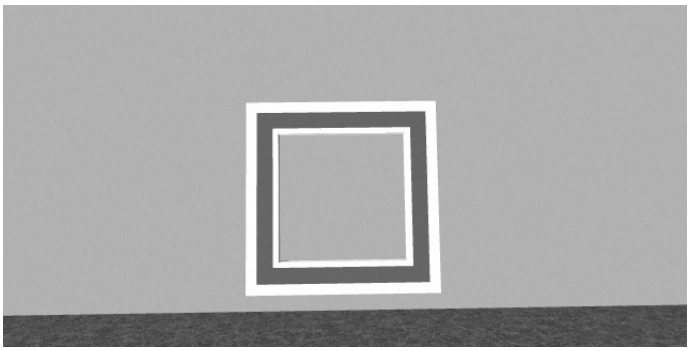


Figure 6.3 Balise utilisée pour les pistes

La forme rectangulaire des différentes composantes est justifiée par sa facilité de détection par des algorithmes de vision simples et pour imposer des contraintes géométriques qui permettent de rejeter de fausses détections. La bande rectangulaire noire sur un fond blanc offre un contraste très élevé qui permet aussi une robustesse accrue puisqu'elle minimise l'impact de l'arrière plan. Les dimensions de la balise ont été déterminées à partir du trou de passage et des ratios géométriques de la bande noire à respecter. La dimension du trou de passage a été choisie en fonction de la marge de manœuvre accordée au drone qui est de 60 cm dans notre cas. Les ratios géométriques sont définis comme étant les ratios d'aire des différents rectangles composant la balise. Ces ratios sont utilisés pour permettre une détection plus robuste de la balise dans l'algorithme de détection présenté à la section suivante.

6.3.2 Algorithme de détection

L'algorithme 1 présente la méthode utilisée pour détecter les balises présentées à la section précédente et pour calculer la distance relative et l'angle de lacet ψ relatif entre la balise et le drone.

Nous commençons par appliquer l'algorithme de Canny [59] à notre image (voir la figure 6.4 pour l'image filtrée et la figure 6.5 pour l'image de sortie de l'algorithme) pour obtenir une

Algorithme 1 : Détection de balises

Entrée : I, Image
Entrée : M, Dimensions de la balise
Entrée : K, Paramètres de la caméra
Sortie : $\mathbf{t}$, Translation entre la caméra et le centre la balise par rapport à la caméra
Sortie : $\mathbf{r}$, Vecteur d'orientation de la balise par rapport à la caméra
Sortie : *trouvé*, Détection de balises accomplie

```
trouvé ← 0 ;
condition ← 0 ;
t ← 0 ;
r ← 0 ;
B ← Canny(I);
C ← DétContoursFermés(B);
for i ← 0 to length(C) do
    P ← TrouverPolygone(C(i));
    condition ← TestGéométriques(P);
    if condition then
        trouvé ← 1 ;
        P ← ReordonneCoins(P);
        [t, r] ← RésoutPnP(P, M, K);
    end
end
```

image binaire B représentant les contours dans notre image. L'algorithme de [60] est ensuite appliqué à B pour produire une liste hiérarchisée des contours fermés C dans l'image binaire. Pour chaque élément de cette liste, nous approximons sa forme à un polygone avec l'algorithme Ramer–Douglas–Peucker [61]. Une série de tests géométriques sont ensuite appliqués au polygone. Parmi ces tests, on trouve :

— vérification que le polygone est un quadrilatère ;
— vérification que ce quadrilatère contient un autre quadrilatère ;
— vérification que le ratio d'aire entre ces deux rectangles est conforme à la géométrie de la balise.

Si les tests géométriques sont réussis, les quatre coins du polygone sont trouvés et sont réordonnés dans un ordre particulier. Finalement, avec la géométrie de la balise et les paramètres de la caméra, on associe les coins à leur dimension physique et on résout le problème *PnP* (Perspective-n-Points) avec l'implémentation dans *OpenCV* des travaux de [62] afin de trouver la translation et l'orientation de la balise par rapport à la caméra. Empiriquement, nous trouvons que l'erreur moyenne de détection est de 10 cm par axe, ce qui simule assez bien la précision offerte par les systèmes de détection récents.

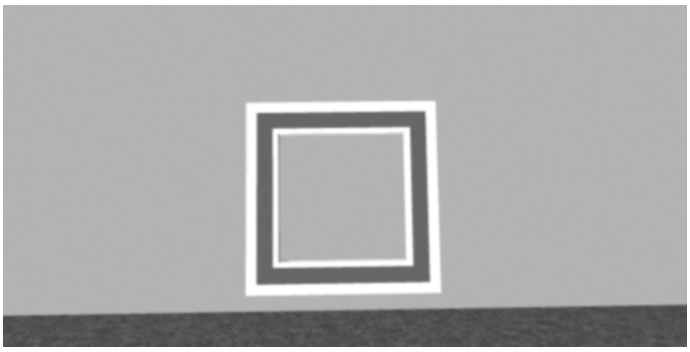


Figure 6.4 Image de balise filtrée

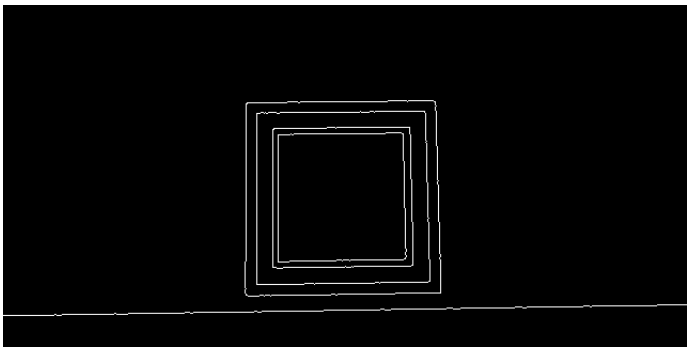


Figure 6.5 Filtre de Canny appliquée sur l'image de la balise

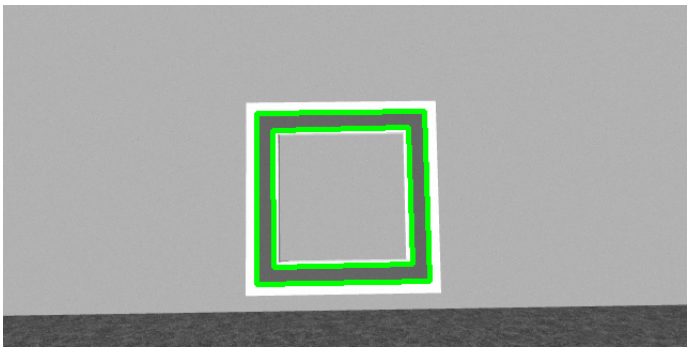


Figure 6.6 Détection de la bande rectangulaire noire

6.3.3 Cartographie des balises

Tel que mentionné précédemment, une carte des différentes balises doit être construite et mise à jour afin de pouvoir naviguer la piste. Initialement, cette carte est composée des positions nominales des balises dans la piste. Puisque la position des balises est sujette à des erreurs aléatoires, on doit les détecter au fur et à mesure du parcours et faire une mise à jour de la carte. Pour chaque balise dans la carte, deux éléments sont sauvegardés, soit la position $\mathbf{t}$ du centre du balise dans le repère inertiel $\mathcal{F}_i$ et le vecteur unitaire $\mathbf{r}$ de la base de $\mathcal{F}_{balise}$ exprimé dans le repère $\mathcal{F}_i$ représentant le lacet ψ de la balise dans le repère inertiel.

Mise à jour de la carte

L'algorithme 2 présente la méthode utilisée pour faire la mise à jour de la carte des balises à l'intérieur de la piste.

Algorithme 2 : Mise à jour de la carte

Entrée : I, Image
Entrée : M, Dimensions de la balise
Entrée : K, Paramètres de caméra
Entrée : C, Carte de la piste
Sortie : C^*, Carte de la piste mise à jour

```
[trouvé, t, r] ← Détecteur(I, M, K) ;
if trouvé then
    position ← TransformInertiel(t, r) ;
    [danscarte, numbalise] ← TrouverBalise(position) ;
    if danscarte then
        C* ← MoindresCarrésRécursifs(C, numbalise, position) ;
    end
end
```

À partir d'une image d'entrée I, nous utilisons l'algorithme 1 pour détecter s'il y a une balise et sa pose relative si elle est détectée. Si une balise est trouvée, on calcule la position globale de la balise dans le repère inertiel $\mathcal{F}_i$. Pour trouver la position du centre de la balise et son orientation dans le repère $\mathcal{F}_i$, on applique la transformation entre le repère $\mathcal{F}_{camera}$ et le repère $\mathcal{F}_i$ en utilisant la transformation entre $\mathcal{F}_{camera}$ et $\mathcal{F}_b$ (qui est connue à la base étant donné la géométrie du drone) et en utilisant la relation entre $\mathcal{F}_b$ et $\mathcal{F}_i$ fournie par l'odométrie. Une fois la position globale trouvée, on trouve le numéro de la balise à l'aide de la piste nominale et les bornes des erreurs. Si la position de la balise trouvée ne correspond à aucune balise à l'intérieure des bornes données, on néglige cette donnée. Ensuite, la position et l'orientation

de la balise détectée est mise à jour dans la carte en utilisant un moindre carré récursif.

Il est à noter que cette méthode d'estimation de position et d'orientation n'est précise que dans un contexte où l'estimation d'état est précise et n'est pas affectée par un biais. Une amélioration future possible serait donc d'utiliser un filtre de Kalman étendu pour compenser le manque de précision et le biais produits par le système d'estimation d'état comme dans [3]. Cette méthode de cartographie permettrait d'obtenir de meilleurs résultats lorsqu'incorporée à un système réel. Ceci dit, nous nous intéressons uniquement aux algorithmes de navigation et de contrôle et, ainsi, supposons une estimation d'état parfaite.

6.4 Parcours autonome d'une piste et génération de trajectoires

Tel que mentionné précédemment, nous utilisons des modifications de l'algorithme présenté au Chapitre 5 avec la carte des différentes balises de la piste pour générer des trajectoires pour faire le parcours de la piste. Plus spécifiquement, pour les trajectoires selon les axes $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i$ nous calculons une trajectoire débutant de la position actuelle du drone jusqu'à la balise $N+2$, avec N représentant la prochaine balise à traverser. Cette trajectoire est bien évidemment recalculée plusieurs fois lors du parcours de la piste selon les différentes balises traversées et selon les différentes observations faites par l'algorithme de détection de balises.

Une trajectoire est recalculée dans deux contextes particuliers. Le premier contexte est lorsque l'estimé de la position prochaine balise à traverser dans la carte change de plus de 10 cm par rapport au dernier estimé de référence. Ceci permet d'éviter d'entrer en contact avec les parois de la balise. Cette valeur a été déterminée de façon empirique de sorte à ne pas trop souvent initier de nouvelles trajectoires tout en assurant une certaine distance de sécurité avec la paroi. La deuxième situation dans laquelle une nouvelle trajectoire est calculée est lorsqu'une balise est traversée. Pour déterminer si le drone traverse une balise, nous calculons la distance entre un point situé à 1 m après le centre de la balise le long de son orientation et la position du drone. Si cette valeur est plus petite qu'un certain seuil, on considère que le drone a traversé la balise. Le seuil est déterminé empiriquement de sorte à compenser les erreurs d'estimation de la cartographie tout en restant assez précis.

En ce qui a trait au lacet, nous n'utilisons pas l'algorithme présenté au Chapitre 5. Dans le cadre d'une course, dans laquelle les vitesses en jeu sont élevées, il est nécessaire d'observer la prochaine balise le plus rapidement possible afin d'avoir le plus de temps possible pour réajuster la trajectoire selon les axes $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i$. Pour ce faire, la trajectoire pour le lacet correspond à un échelon d'amplitude égale à l'angle de lacet nécessaire pour que la caméra du drone soit alignée avec le centre de la prochaine balise.

6.4.1 Modifications pour évitement de parois

En ce qui concerne la méthode utilisée pour générer les trajectoires, des modifications à l'algorithme présenté au Chapitre 5 sont apportées. En effet, l'algorithme de base garantit que la trajectoire passe par le point situé au centre de la balise (Fig. 6.7) mais ne garantit pas que le drone n'entrera pas en contact avec le cadre puisque l'orientation n'est pas prise en compte dans l'algorithme. Il est donc nécessaire d'apporter des modifications à la formulation du problème afin de prendre en compte l'orientation des balises.

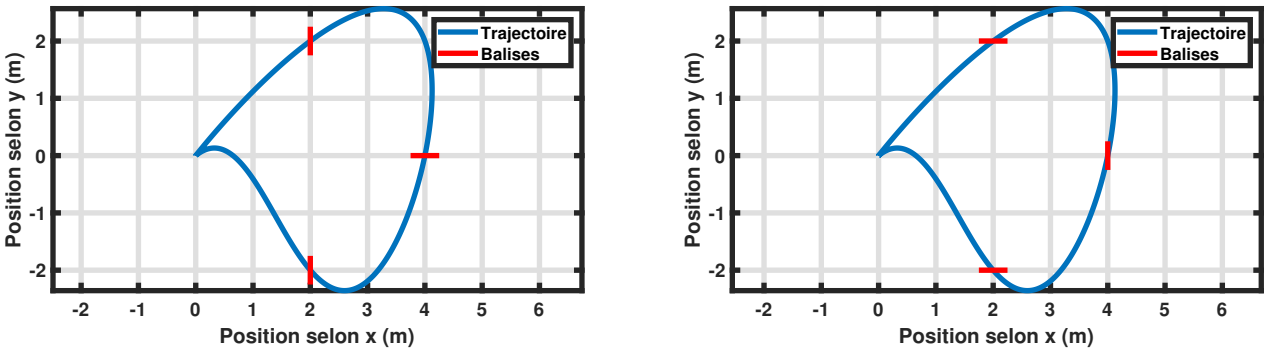


Figure 6.7 Invariance de la trajectoire générée par l'algorithme original selon l'orientation des balises

Méthode initiale d'évitement

La première méthode d'évitement de parois de balises développée dans nos travaux est inspirée de [3]. Dans leurs travaux, pour assurer que le drone n'entre pas en collision avec la cible, le point de référence au centre de la balise est remplacé par deux points de référence positionnés de part et d'autre de la balise selon son orientation. Dans notre implémentation, l'orientation utilisée provient de l'estimé provenant des algorithmes de détection et de cartographie. La distance par rapport au centre de la balise est déterminé empiriquement à 1 m. Bien que cette méthode ne puisse garantir analytiquement que la trajectoire générée n'entre en contact avec les parois, nous posons l'hypothèse que la faible distance entre les points intermédiaires et l'optimisation du *snap* force la trajectoire à être relativement droite. Cette méthode a été utilisée pour le développement initial du système, mais a vite été délaissée pour des problèmes de stabilité numérique. En effet, en observant l'équation (5.1), nous notons la présence de puissances du temps de parcours T du segment pour la construction de la matrice

Hessienne $\mathbf{Q}$. Selon l'allocation de temps entre les différents segments et la distance séparant les différentes balises, il peut arriver que le rapport des différents temps de segments T soit d'amplitude importante. Sachant que ces temps sont mis à des puissances élevées, il arrive parfois que le conditionnement de $\mathbf{Q}$ soit considérable, ce qui rend les opérations avec celle-ci très peu stables numériquement. Par ailleurs, le nombre de points utilisés double par rapport à la méthode originale, ce qui augmente la taille de $\mathbf{Q}$ et augmente le temps de calcul.

Méthode d'évitement implémentée

Suite aux conclusions tirées de la méthode initiale d'évitement de balises, une deuxième méthode d'évitement de balises plus raffinée et stable numériquement a été développée. Plutôt que de forcer la trajectoire indirectement à passer de façon perpendiculaire à la balise, nous imposons de façon directe l'orientation de la trajectoire en imposant une vitesse selon $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i$ à chacune des balises. Pour ce faire, nous modifions la formulation du problème d'optimisation. Les conditions limites fixes $\mathbf{D}_F$ et la matrice d'allocation $\mathbf{M}$ sont modifiées afin d'inclure les contraintes supplémentaires liées à la vitesse. Le principal inconvénient de cette méthode est que la vitesse de passage ne peut pas être trouvée en ligne. La solution utilisée pour contrevenir à ce problème est de déterminer les vitesses et les orientations aux balises au préalable selon la disposition de la piste nominale. Ces vitesses sont ensuite allouées selon chaque dimension en fonction de l'orientation des balises dans la carte de la piste. L'avantage principal de cette méthode par rapport à la précédente est sa plus grande robustesse numérique puisque moins de points sont nécessaires pour décrire la trajectoire ; nous diminuons ainsi la taille de la matrice Hessienne $\mathbf{Q}$. Par ailleurs, les rapports des segments de temps T sont plus petits, ce qui diminue le conditionnement de la matrice.

6.4.2 Allocation de temps

Deux méthodes sont utilisées pour l'allocation du temps entre les différents segments. La première méthode d'allocation est de séparer le temps également entre les segments de sorte à maintenir une vitesse constante le long de la piste. Pour déterminer le temps alloué à chaque segment, nous divisons la distance en ligne droite entre les deux balises par la vitesse moyenne désirée. La deuxième méthode d'allocation de segments de temps est celle présentée à la section 5.3. Afin de pouvoir comparer les deux méthodes, le temps total de parcours de la piste pour une vitesse moyenne désirée est utilisé comme temps à optimiser par l'algorithme d'optimisation. Les résultats obtenus à partir des deux méthodes peuvent donc être directement comparés.

Pour déterminer les segments de temps lorsqu'on recalcule une trajectoire, le temps est calculé

en fonction du rapport de la distance d (la distance entre le drone et la prochaine balise) et la distance de référence r (la distance en ligne droite entre la prochaine balise et la balise précédente). En supposant un suivi de trajectoire adéquat et en supposant que les balises sont détectées dans les segments droits de la trajectoire, cette méthode offre de bons résultats.

6.4.3 Détails d'implémentation

Certaines modifications ont dû être apportées aux algorithmes de génération de trajectoire afin qu'ils soient utilisables dans notre implémentation. Tout d'abord, par construction, la matrice Hessienne $\mathbf{Q}$ a un conditionnement infini. En effet, la matrice est bloc-diagonale avec certains blocs nuls. Pour réduire le conditionnement, nous faisons une opération de préconditionnement de la matrice $\mathbf{Q}$ en lui ajoutant une matrice identité $\mathbf{I}$ multipliée par un faible facteur ϵ. Le facteur ϵ est choisi de sorte à ne pas influencer le problème à résoudre tout en s'assurant de suffisamment diminuer le rapport entre la plus petite et la plus grande des valeurs propres de $\mathbf{Q}$. Cette méthode de *diagonal scaling* a permis de considérablement augmenter la stabilité numérique de la réponse trouvée. La résolution des équations (5.30) et (5.33) est faite avec l'implémentation de la routine *LAPACK* de la librairie Python *Numpy*. Afin d'assurer la continuité entre la trajectoire de référence précédente et celle nouvellement calculée, les délais de calcul doivent être pris en compte lors de l'imposition des conditions initiales. Pour ce faire, un temps fixe ΔT_{calc} est alloué pour le calcul des coefficients polynomiaux. Lorsque la durée du calcul est inférieure à ΔT_{calc}, le générateur de trajectoires est mis sur pause jusqu'à l'atteinte de ΔT_{calc}. Ce temps est choisi expérimentalement en se basant sur l'observation de différents temps de calcul et en déterminant le temps moyen de calcul avec une marge de manœuvre pour les cas exceptionnels. Lors du calcul des coefficients polynomiaux, les conditions initiales utilisées sont déterminées à partir de la trajectoire de référence précédente, mais en extrapolant de ΔT_{calc} secondes par rapport au temps actuel.

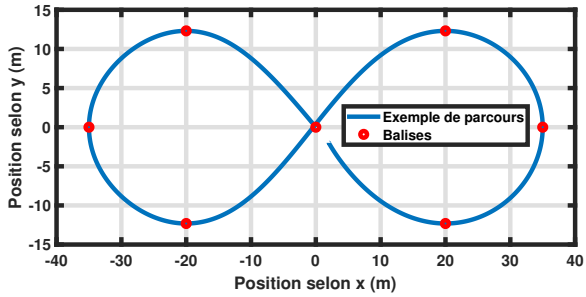
6.5 Simulations

Plusieurs simulations sont effectuées afin de valider les différents algorithmes d'autonomie développés (loi de commande, génération de trajectoire, cartographie de balises, etc.). Avant de présenter les résultats obtenus, les pistes simulées et la méthodologie d'évaluation sont présentées.

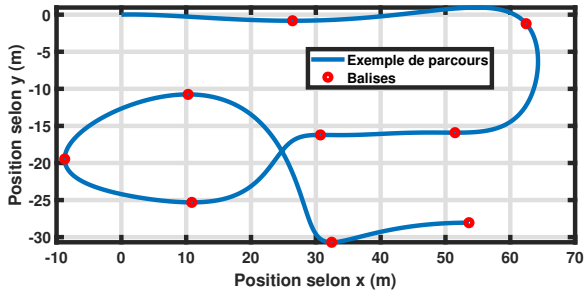
6.5.1 Pistes simulées

Dans le cadre de notre analyse, nous simulons deux pistes différentes. Chaque piste représente un niveau de difficulté différent et permet de valider la robustesse et la performance de la solution proposée. Dans la littérature, les méthodes sont souvent évaluées sur des pistes de petite taille, ce qui ne permet pas de montrer les limites des méthodes et des vitesses maximales pouvant être atteintes. Nous proposons donc des pistes de grande dimension permettant des manœuvres à haute vitesse.

La première piste utilisée est inspirée de celle trouvée dans les travaux de [3], mais à plus grande échelle afin de valider les algorithmes à plus grande vitesse. La piste est définie par la forme d'une courbe Lemniscate (Figs 6.8(a) et 6.9(a)). Cette piste est composée de segments droits et de segments courbés à différents degrés, ce qui est assez représentatif des conditions présentes dans une piste réelle de course pour drone , mais dans un scénario plus simple. Une fois les algorithmes validés sur cette piste, les limites de performance du système sont évaluées sur la deuxième piste, utilisée dans les qualifications virtuelles pour le concours *AlphaPilot* (Figs 6.8(b) et 6.9(b)). Cette piste est une combinaison de virages aggressifs et de segments droits à haute vitesse. La première piste, d'une longueur totale d'environ 180 m, est composée de 7 balises et la deuxième piste, d'une longueur d'environ 230 m, est composée de 9 balises.

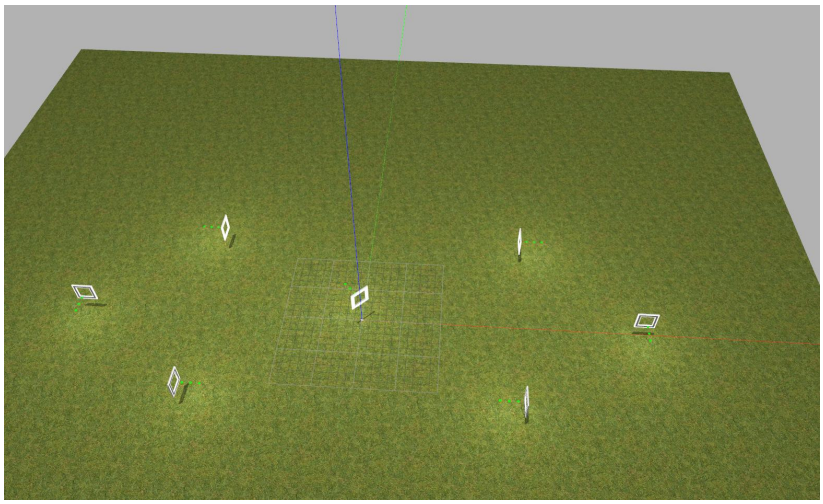


(a) Première piste

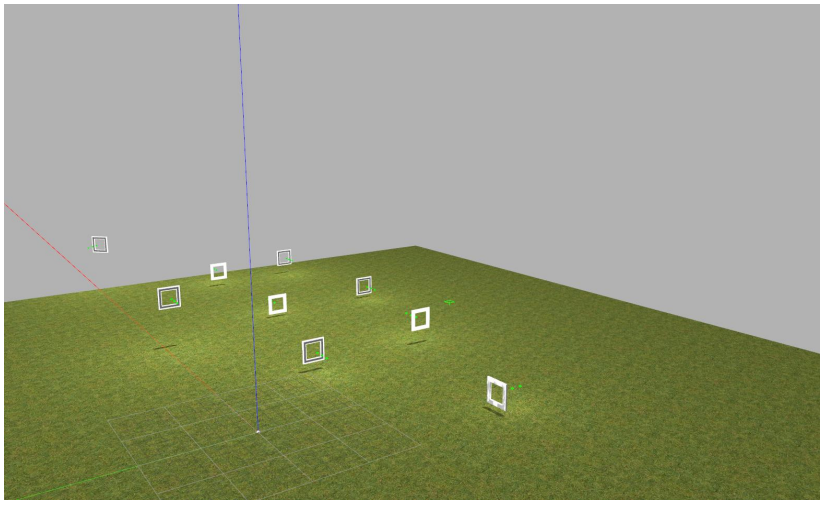


(b) Deuxième piste

Figure 6.8 Visualisation du parcours des pistes simulées



(a) Première piste



(b) Deuxième piste

Figure 6.9 Visulation dans l'environnement Gazebo des pistes simulées

Chacune des pistes est reproduite dans l'environnement de simulation Gazebo (présenté au Chapitre 3) avec les balises développées à la section 6.3.1. Le drone utilisé est le *Asctec Firefly* puisque les gains de la loi de commande ont été déterminés et validés pour ce système au Chapitre 4. Afin de faire la détection des balises, une caméra est ajoutée sur la face avant du multicoptère (Fig. 6.10).

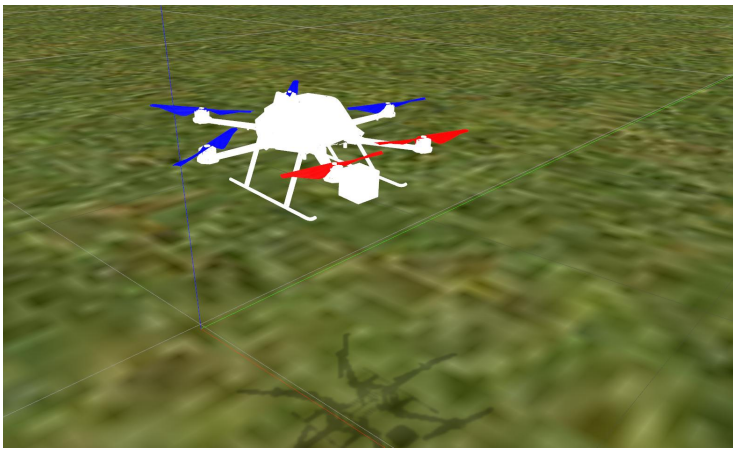


Figure 6.10 Image du drone simulé dans l'environnment Gazebo

6.5.2 Essais expérimentaux

L'architecture de l'autonomie présentée aux figures 6.1 et 6.2 a été implémentée dans l'environnement ROS. La dynamique du corps et les capteurs sont simulés par l'environnement *RotorS* de Gazebo. Pour ce qui est de l'estimateur d'état MSF (*MultiSensor Fusion Framework*) [58], nous utilisons l'implémentation de ETHZ-ASL [1]. Un nœud ROS pour le bloc de planification a été créé en *Python*. Dans ce nœud, le détecteur de cibles, le générateur de trajectoires, la cartographie et le gestionnaire de trajectoire sont tous exécutés en parallèle. Les codes pour le détecteur de cibles et le générateur de trajectoire ont été écrits dans le langage Python en utilisant pour le premier la librairie *OpenCV* et pour le second la puissante librairie de calcul et de manipulation algébrique *Numpy*. La loi de commande a été implémentée tel que décrit précédemment au Chapitre 4. La figure 6.11 montre un aperçu de la communication des différents modules avec la commande *rqt_graph* dans la librairie ROS.

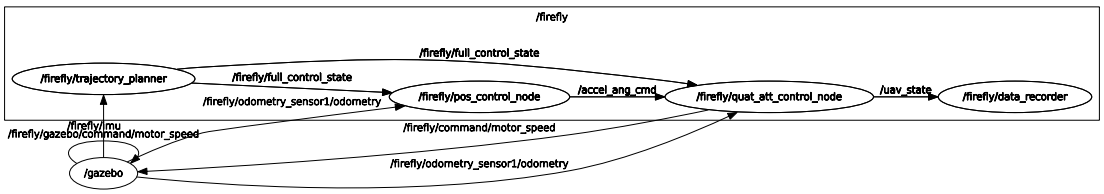


Figure 6.11 Schématisation de l'implémentation ROS des simulations

1. Dépôt Github : `https://github.com/ethz-asl/ethzasl_msf.git`

Notre méthode d'évaluation est différente de celle proposée dans la littérature. En effet, la performance du système d'autonomie est évaluée en deux temps conformément à la modélisation que nous en avons faite au début de ce chapitre.

Tout d'abord, la robustesse du système d'autonomie en phase d'exploration est évaluée selon différentes vitesses moyennes. Ici nous définissons la robustesse comme étant la capacité du système proposé de compléter le parcours d'une piste en passant par chacune des balises sans entrer en collision avec une balise ou le sol. Dans la littérature, lorsque les algorithmes d'autonomie sont évalués pour leur robustesse, ils ne sont évalués qu'un très petit nombre de fois, ce qui n'est pas statistiquement significatif. Par ailleurs, ces essais sont tous réalisés pour une même configuration de piste, ce qui ne montre pas nécessairement la capacité de l'algorithme à s'adapter à des variations d'une même piste. Nous proposons donc, pour valider la capacité d'adaptation et la robustesse de l'algorithme, de l'évaluer sur 30 variations de la piste nominale. Nous avons choisi d'utiliser 30 variations de piste afin de pouvoir tirer des conclusions statistiquement significatives tout en minimisant le temps de simulation nécessaire. Les variations appliquées à la piste correspondent à l'erreur aléatoire mentionnée à la section 6.1. Cette erreur aléatoire représente la différence qui existe entre l'allure de la piste présentée au pilote et la piste réelle. La borne maximale de l'erreur a été fixée à 1.5 m pour nos essais.

Ensuite, nous évaluons le système en phase de performance. Pour ce faire, nous effectuons des essais de suivi de trajectoire sur les pistes nominales pour chacune des pistes. Les trajectoires sont calculées au préalable puisque les positions des balises sont connues par hypothèse. Nous évaluons donc la vitesse maximale à laquelle la trajectoire peut être suivie par le drone sans entrer en contact avec les différentes balises. Ce critère de performance mesure la faisabilité des trajectoires optimales générées et la capacité de suivi de trajectoire de la loi de commande développée.

Pour terminer, la performance de l'algorithme d'autonomie sera évaluée selon les deux méthodes d'allocation de temps proposés.

6.5.3 Résultats

Évaluation de la première piste

Pour la première piste, nous avons évalué la robustesse de la phase d'exploration, pour des vitesses moyennes désirées de $\{4, 5, 6, 7\}$ m/s, ainsi que pour une allocation de temps manuelle ou optimale. La robustesse est mesurée comme étant le pourcentage de succès du système autonome pour tous les essais (qui est de 30 dans notre cas). Un succès est défini comme le

parcours complet de la piste en passant par toute les balises selon la bonne orientation sans entrer en collision avec les parois d'une balise ou le sol.

Les résultats sont présentés au tableau 6.1. Pour l'allocation de temps manuelle, le taux de succès s'améliore de 4 m/s à 5 m/s, mais diminue pour les vitesses suivantes. Pour l'allocation de temps optimale, nous constatons une augmentation significative du taux de succès jusqu'à une vitesse de 6 m/s mais une perte abrupte pour une vitesse de 7 m/s. Ceci semble contraire à notre intuition initiale. En effet, il devrait être plus facile de parcourir une piste à basse vitesse plutôt qu'à grande vitesse. Ceci dit, nous posons plusieurs hypothèses quant à l'origine des problèmes. Après avoir présenté ces hypothèses, nous proposons une méthodologie plus rigoureuse afin de mieux identifier les sources des erreurs.

Pour ce qui est de la première piste, nous posons deux principales sources aux problèmes rencontrés : les erreurs de génération de trajectoire et les erreurs de visibilité et de commande.

Tableau 6.1 Taux de succès de la méthode d'autonomie pour la première piste

	4 m/s	5 m/s	6 m/s	7 m/s
Allocation de temps manuelle	33.33%	36.67%	26.67%	10.00%
Allocation de temps optimale	33.33%	43.33%	46.67%	23.33%

Tout d'abord, après analyse des points de collision pour les essais à 4 m/s et à 5 m/s, nous notons que la majorité d'entre eux se situent au niveau du sol. La cause principale de ces échecs provient de la méthode de génération de trajectoire. Il a été trouvé que plus la vitesse moyenne est basse, plus il existe de trajectoires optimales ayant un niveau du *snap* similaire liant les points intermédiaires. Ceci dit, pour des basses vitesses, bien que ces trajectoires ait un *snap* similaire, leur trajectoire est complètement différente. Par ailleurs, malgré les modifications robustes apportées à l'algorithme de génération de trajectoire (reformulation et ajout de *diagonal scaling*), il existe toujours des erreurs numériques dans la solution trouvée, ce qui peut parfois entraîner que ce n'est pas la trajectoire optimale qui est trouvée ; la combinaison d'erreurs de suivi et d'erreurs numériques produit des trajectoires optimales différentes de celles prévues initialement. Dans le cas de la première piste, les erreurs au niveau de la trajectoire générée prennent la forme d'une large dérive de la trajectoire entre les balises 3 et 4 (voir figure 6.8 pour la position des cibles). La figure 6.12 illustre le principe pour une trajectoire réussie.

Nous pouvons voir que la trajectoire planifiée par l'algorithme d'autonomie est loin de celle prévue initialement. Dans le cas de la figure 6.12, la trajectoire calculée dévie dans le plan

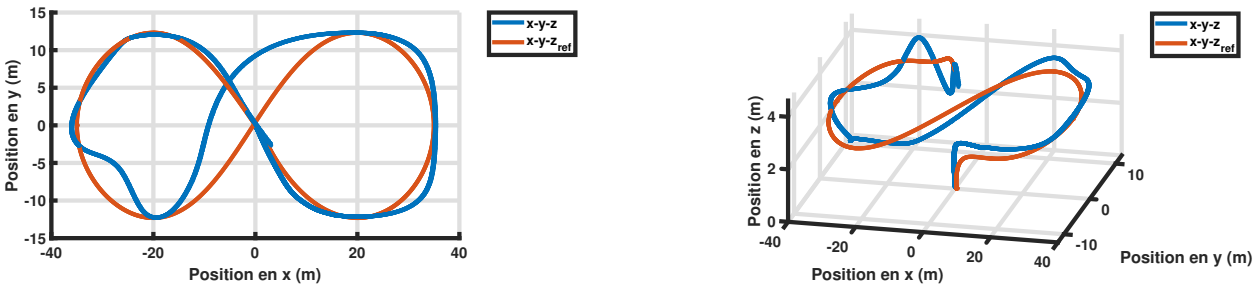


Figure 6.12 Trajectoires optimales voisines

$(\mathbf{x}_i, \mathbf{y}_i)$, et cette déviation est plus grande dans le cas des essais qui n'ont pas été réussis, ce qui cause une collision entre le multicoptère et une autre balise, ainsi qu'un manque de visibilité de la prochaine cible. Il arrive aussi que la dérive ait lieu dans la direction $\mathbf{z}_i$, ce qui occasionne un contact avec le sol. Dans ces situations, l'implémentation de l'algorithme de génération de trajectoire est à pointer du doigt. En effet, l'algorithme de génération de trajectoire, mis à part des erreurs numériques, produit la bonne sortie pour les conditions initiales et points intermédiaires fournis. Le problème vient du fait qu'aucune contrainte en espace est fournie à l'algorithme. Il est donc possible de générer des trajectoires entrant en contact avec le sol ou d'autres balises. Il pourrait ainsi être intéressant d'ajouter une étape supplémentaire à la génération de trajectoires afin de vérifier que les trajectoires générées soient comprises dans un certain espace (tel que proposé par [63]). Une vérification des contraintes pourrait être faite et si la trajectoire générée ne les respecte pas, des points intermédiaires pourraient être ajoutés jusqu'à ce que toutes les contraintes soient satisfaites. L'ajout de cet algorithme de vérification pourrait cependant nuire au fonctionnement en temps réel de l'algorithme et pourrait occasionner quelques problèmes numériques puisque la taille du problème à résoudre augmente avec l'ajout de points intermédiaires. Sachant cela, il est à noter que le phénomène de génération de trajectoires fautives s'atténue avec l'augmentation de la vitesse. En effet, plus la vitesse augmente, plus la différence de *snap* entre les solutions voisines augmente, ce qui force la trajectoire générée à être similaire à l'originale.

En ce qui concerne la méthode d'allocation de temps, elle a un impact sur cette source de collisions. En effet, l'avantage de l'allocation optimale du temps par rapport à l'allocation manuelle du temps est qu'elle optimise indirectement les vitesses de référence entre les différents segments. Pour le segment problématique de la première piste, qui se situe entre les points 3 et 4, l'optimisation produit une vitesse de référence plus grande que celle dans la méthode manuelle (Fig. 6.13). Sachant que le phénomène de génération de trajectoires problématiques s'atténue quand la vitesse augmente, il est possible de conclure que l'allocation optimale du temps permet de partiellement résoudre le phénomène de par l'augmentation

de vitesse, ce qui permet d'expliquer les performances supérieures du système d'autonomie lorsqu'une allocation optimale du temps est utilisée à plus basse vitesse.

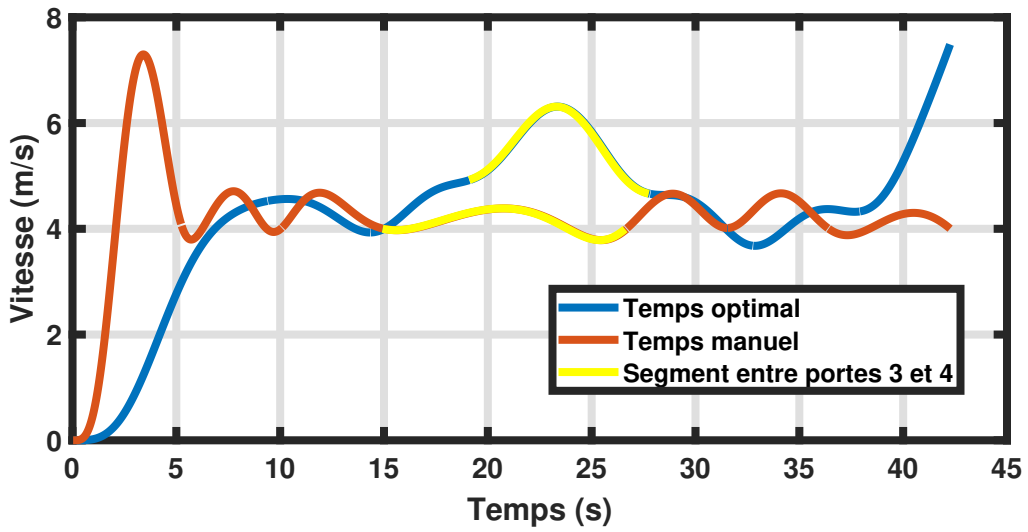


Figure 6.13 Vitesse totale pour le parcours de la première piste à 4 m/s

Pour ce qui est des essais à plus haute vitesse dans la phase d'exploration, la cause principale des échecs sont les erreurs de contrôle et du manque de visibilité des différentes balises. En effet, plus le parcours est exécuté à haute vitesse moyenne, plus les variations de vitesses sont grandes. Ces variations de vitesses sont difficiles à suivre puisqu'elles nécessitent un effort de commande particulièrement grand. Aussi, plus la vitesse moyenne désirée est grande, plus la vitesse de référence dans les virages est grande. Ces segments de trajectoire sont particulièrement difficiles à suivre puisqu'ils sollicitent un important effort de commande selon les 4 directions $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i, \psi)$. La combinaison de l'erreur de suivi et de l'imprécision de la cartographie des balises mène à des collisions. Nous pouvons voir dans le tableau 6.2 que l'erreur de suivi de trajectoire croît avec la vitesse des essais.

Tableau 6.2 Erreurs de suivi de trajectoire moyens pour les essais sur la première piste

	4 m/s	5 m/s	6 m/s	7 m/s
Allocation de temps manuelle	0.355 m	0.392 m	0.687 m	0.653 m
Allocation de temps optimale	0.171 m	0.186 m	0.278 m	0.265 m

Une autre problématique avec le parcours à plus haute vitesse dans la phase d'exploration de la piste est le manque de visibilité des balises. Plus la vitesse augmente, moins il y a de

temps pour faire la détection de la prochaine balise et ajuster la trajectoire. Lorsqu'un virage est parcouru, il arrive même que la cible ne puisse pas être vue à temps ou au complet. Une solution au niveau des algorithmes de génération de trajectoires serait d'inclure une pondération sur la visibilité de la balise dans le problème d'optimisation. Il s'agit donc d'une limitation de la méthode proposée. Une solution alternative qui pourrait être couplée à la solution actuelle serait d'utiliser un autre type de détecteur de balises. Dans [45], un réseau de neurones convolutif est utilisé pour faire la détection de balises avec une grande fiabilité. Le détecteur proposé permet de détecter plusieurs balises simultanément et permet de les trouver lorsqu'on ne les voit pas au complet. Il serait donc intéressant d'intégrer une méthode similaire pour une implémentation complète du système de navigation et de contrôle proposé.

En ce qui a trait aux résultats obtenus, bien que le taux de réussite ne soit pas particulièrement élevé, nous pouvons tout de même dire que les résultats sont concluants quant à l'efficacité de la méthode. En effet, après avoir souligné les différentes sources d'erreur, il est possible de dire que ces résultats seraient meilleurs après les modifications proposées, particulièrement dans le cas de la phase d'exploration de la piste. Il n'est aussi pas évident de faire un lien direct avec les résultats obtenus dans la littérature. En effet, dans [3], [36] et [44], les évaluations de robustesse sont faites à de plus basses vitesses et sur un total de 5 essais sur une même piste. Leurs résultats sont meilleurs, mais ne sont pas statistiquement significatifs. Dans [3], l'algorithme développé est évalué sur une piste similaire à la nôtre, mais à plus basse vitesse et à échelle réduite. Leur algorithme donne des résultats similaires aux nôtres pour des vitesses plus basses et une perturbation de balises équivalente (Fig. 6.14).

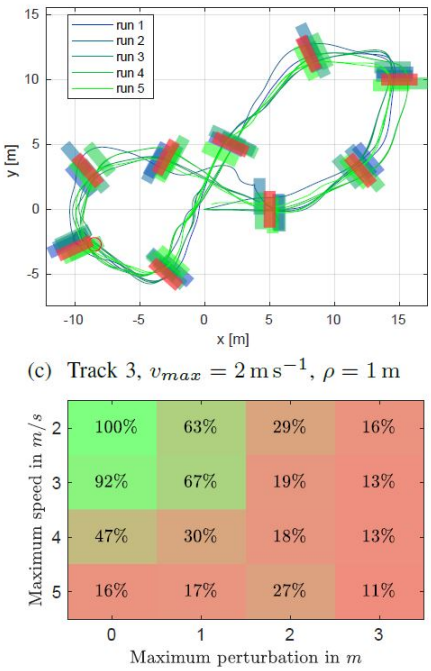
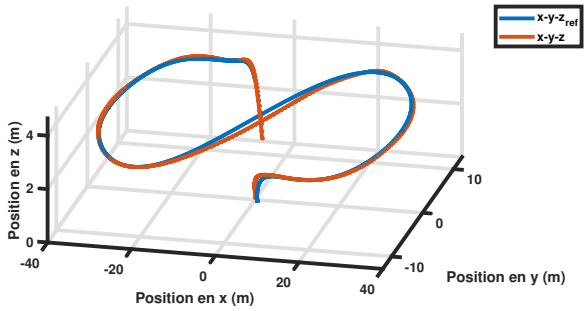


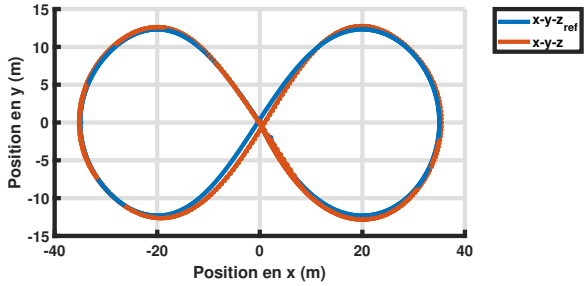
Figure 6.14 Résultats des travaux de [3] sur une piste similaire à la première piste

Comme deuxième mesure de la performance du système d'autonomie développé, nous avons évalué la vitesse moyenne maximale à laquelle le système est capable de parcourir la piste une fois que les positions de toutes les balises sont correctement identifiées. Tel que mentionné dans notre hypothèse de modélisation de piste en deux temps, cet essai permet d'évaluer le gain en vitesse de cette phase par rapport à la phase d'exploration. Pour procéder à cette évaluation, l'algorithme de détection de balise est désactivé et la trajectoire générée par le module de génération de trajectoire n'est faite qu'à partir des positions des balises qui sont connues à l'avance (Fig. 6.15). Les trajectoires sont donc générées seulement lorsque le drone traverse les balises. L'algorithme d'évaluation est utilisé sur la piste nominale pour des vitesses moyennes désirées avec les deux méthodes d'allocation de temps jusqu'à ce qu'il y ait collision ou échec du module de génération de trajectoire. Après nos essais, nous trouvons que la vitesse moyenne maximale à laquelle le système peut parcourir la piste sans échec est de 12 m/s avec une allocation de temps optimale et de 8 m/s avec une allocation de temps manuelle. Ces résultats sont ceux auxquels nous nous attendions étant donné que le *snap* est considérablement moins élevé pour la méthode optimale lorsque la vitesse augmente. Nous trouvons donc que la phase de performance permet de doubler la vitesse moyenne à laquelle le drone parcourt la piste en scénario de course. Ceci dit, ce résultat est à prendre avec un grain de sel. En effet, après analyse des données de l'essai, il est trouvé que la vitesse moyenne

réelle de parcours est d'environ 13 m/s et que la vitesse maximale est de 26 m/s, ce qui n'est pas très réaliste. En effet, ce résultat n'est possible qu' en simulation. Il n'existe pas d'erreur de modélisation, ni de perturbations extérieures ou de délais importants dans les processus embarqués. Ces résultats sont donc à valider expérimentalement dans de futurs travaux.



(a) Visualisation de la trajectoire pour la phase de performance en 3d



(b) Vue de haut de la trajectoire pour la phase de performance

Figure 6.15 Visualisation de la trajectoire pour la phase de performance pour la première piste

Évaluation de la deuxième piste

Après avoir évalué l'algorithme sur la première piste, nous l'avons évalué sur la deuxième piste inspirée de celle utilisée dans le défi *AlphaPilot*. Les résultats de l'évaluation de robustesse sont présentés au tableau 6.3.

Tableau 6.3 Taux de succès de la méthode d'autonomie pour la deuxième piste

	4 m/s	5 m/s	6 m/s	7 m/s
Allocation du temps manuelle	20.69 %	20.69 %	58.62 %	70.00 %
Allocation du temps optimale	26.67 %	23.33 %	36.67 %	33.33 %

Tout comme pour la première piste, nous notons que le taux de réussite augmente avec la vitesse. Après analyse des différentes essais échoués à basse vitesse, nous notons que la majorité des échecs sont causés par le même phénomène de génération de trajectoire identifié pour la première piste. Cependant, nous notons que le taux de réussite ne diminue pas pour les grandes vitesses lorsque l'allocation de temps manuelle est utilisée. Selon nous, ce résultat peut s'expliquer de par la disposition particulière de la piste. En effet, contrairement à la première piste, le niveau de difficulté de la deuxième piste est plus bas selon nous. Dans la première piste, la majorité du parcours s'effectue dans des virages. Ceci dit, tel qu'identifié plus haut, un des problèmes majeurs pour le parcours de piste à haute est vitesse est l'erreur de suivi et le manque de visibilité dans les segments constitués de virages. Nous croyons donc que ceci peut expliquer pourquoi nous obtenons de meilleurs résultats à haute vitesse dans la deuxième piste. De plus, il y a non seulement moins de virages dans la deuxième piste, mais aussi beaucoup plus de segments droits entre les virages, ce qui permet au drone de se stabiliser suite aux manœuvres en virage. Par ailleurs, la distance maximale entre deux balises dans la deuxième piste est de 36 m, tandis que cette distance est de 47 m dans la première piste. La plus petite distance maximale diminue les risques de reproduction du phénomène de génération de trajectoire déviant grandement de la trajectoire de référence initialement prévue.

Par ailleurs, contrairement au cas des essais sur la première piste, l'allocation de temps manuelle performe mieux au niveau de la robustesse que l'allocation de temps optimale. Après analyse des résultats, il a été trouvé que la principale raison qui explique cette différence est le profil de vitesse généré par l'allocation de temps optimal. En effet, bien qu'il est plus optimal au niveau du suivi de trajectoire d'augmenter la vitesse dans les segments droits et de la diminuer dans les virages (tel qu'il est possible de le voir avec l'algorithme d'optimisation à la figure 6.16), ce n'est pas le cas pour la recherche de balises à haute vitesse. En effet, lorsque les segments droits sont parcourus à trop haute vitesse, l'angle de tangage du drone diminue à un tel point qu'il n'est plus possible pour la caméra d'avoir la prochaine balise dans son champ de vue. Il est donc impossible de détecter avec précision la position et l'orientation de la prochaine balise, ce qui cause des collisions avec les balises à traverser. De plus, il a été noté que la disposition des différentes balises dans la deuxième piste fait en sorte que la piste est parcourue à plus haute vitesse. Ceci dit, pour de très grandes vitesses, il est difficile pour la loi de commande de suivre la trajectoire avec précision. On peut voir l'erreur de suivi moyen pour les différents essais sur la deuxième piste au tableau 6.4.

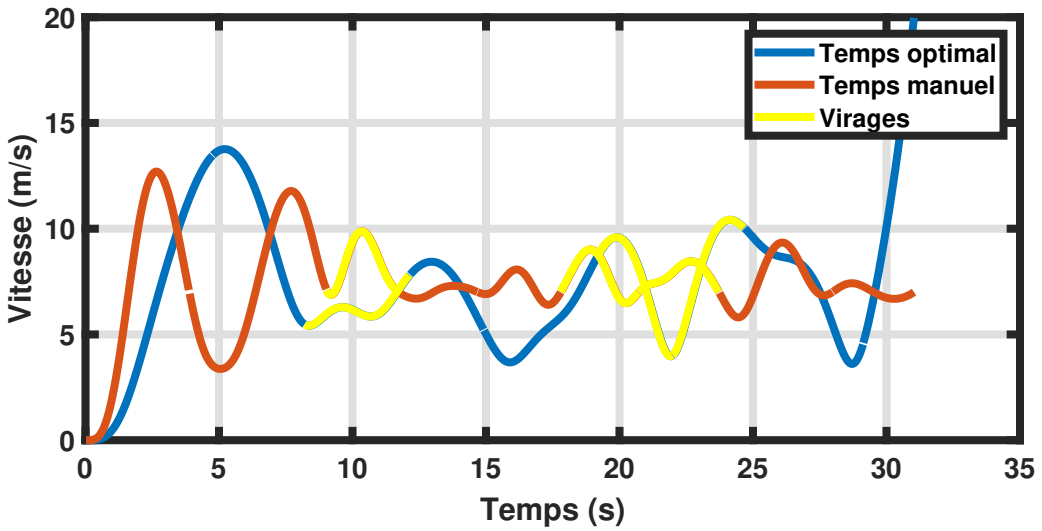


Figure 6.16 Profil de vitesse pour le parcours de la deuxième piste

Tableau 6.4 Erreurs de suivi de trajectoire moyens pour les essais sur la deuxième piste

	4 m/s	5 m/s	6 m/s	7 m/s
Allocation de temps manuelle	0.225 m	0.205 m	0.255 m	0.283 m
Allocation de temps optimale	0.177 m	0.185 m	0.350 m	0.447 m

Pour terminer, nous soulignons qu'à haute vitesse, le facteur principal d'échec est l'erreur de suivi dans les virages. En effet, pour l'allocation de temps manuelle, tous les échecs ont eu lieu au même point. Celui était à la sortie d'un virage et l'échec était causé par une collision avec la balise à la sortie du virage. La figure 6.17 présente tous les échecs de parcours du système d'autonomie pour une vitesse moyenne désirée de 7 m/s en utilisant une allocation de temps manuelle.

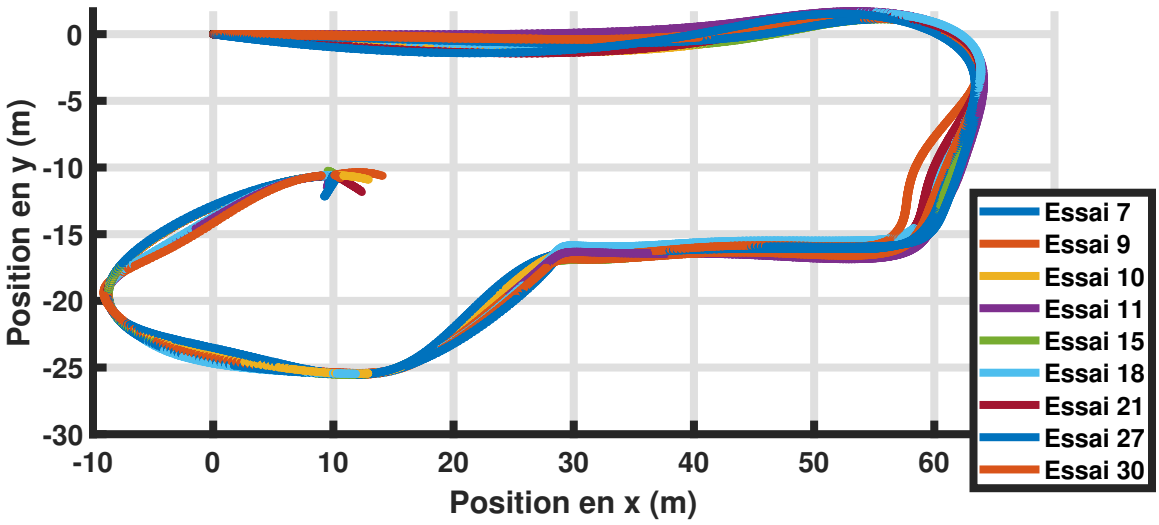
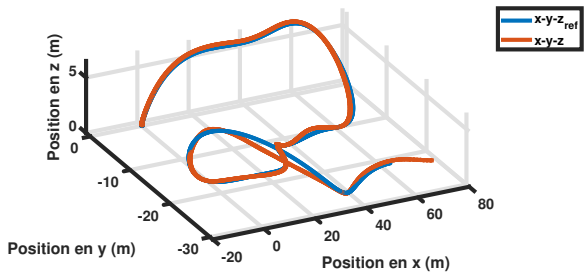
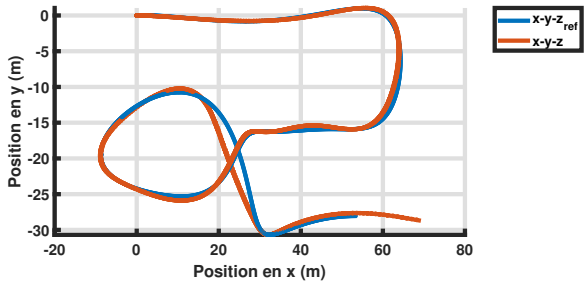


Figure 6.17 Profil de vitesse pour le parcours de la deuxième piste

Pour ce qui est de l'évaluation de la phase de performance maximale, nous obtenons des résultats similaires à ceux obtenus pour la première piste. Pour une allocation de temps manuelle, le système d'autonomie réussit à parcourir la piste sans échec jusqu'à une vitesse moyenne de 9 m/s pour une allocation de temps manuelle et jusqu'à 10 m/s pour une allocation de temps optimale. La figure 6.18 présente l'erreur de suivi de la trajectoire par rapport à la trajectoire de référence pour l'essai à 10 m/s.



(a) Visualisation de la trajectoire pour la phase de performance en 3d



(b) Vue de haut de la trajectoire pour la phase de performance

Figure 6.18 Visualisation de la trajectoire pour la phase de performance pour la deuxième piste

Méthodologie d'identification de sources d'erreurs proposée

Bien que les hypothèses énoncées précedemment semblent expliquer les résultats obtenus, une méthodologie plus rigoureuse est nécessaire afin de les confirmer. Au niveau de la génération de trajectoire, des essais supplémentaires de validation de l'algorithme proposé sont nécessaires. Nous proposons de tester l'algorithme de génération de façon isolée aux autres systèmes dans un environnement plus simple pour éviter des sources de bruit. Pour ce faire, il faudrait valider l'algorithme conçu avec des trajectoires dont la réponse optimale analytique au niveau du *snap* est connue. L'algorithme serait ensuite testé à différentes vitesses afin de reproduire le phénomène observé. Lorsqu'un échec est observé, l'environnement simple de simulation permettrait d'avoir toutes les données sur le problème tels le conditionnement des différentes matrices utilisées dans le problème optimal et la différence de snap entre la trajectoire de référence analytique et celle calculée. L'identification de la source de génération de trajectoires fautives pourrait donc être identifié avec précision.

Pour ce qui est des essais à plus haute vitesse dans la phase d'exploration, une analyse approfondie de la loi de commande présentée est nécessaire. Tel que proposé à la fin du chapitre

4, des essais sur différentes primitives de trajectoires présentes dans des trajectoires typiques de courses pour drone (virages à rayon de courbure et de longueur variable, manoeuvre de *zigzag*,etc.) sont nécessaires. Ces différents essais permettraient d'identifier les limitations et les erreurs de commande moyennes selon le type de segment parcouru. Ceci permettrait entre autre de déterminer si le drone est physiquement capable de suivre la trajectoire générée. Cette validation supplémentaire de la loi de commande serait aussi très utile dans l'évaluation des performances du système d'autonomie dans la phase de performance.

Pour terminer, une métrique intéressante à évaluer plus quantitativement serait la visibilité des balises à haute vitesse. Cette métrique serait constituée selon l'équation suivante :

$$visibilité = \frac{t_{seg.} - t_{rép.lacet}}{t_{seg.}} \tag{6.1}$$

Avec $t_{seg.}$ le temps alloué au segment particulier et $t_{rép.lacet}$ le temps nécessaire pour que le drone ait une vue sur la cible.

L'évaluation de cette métrique permettrait d'une part d'identifier les segments de piste de course problématiques dans la phase d'exploration et d'une autre part de servir de métrique de faisabilité au sein de l'algorithme d'autonomie et être utilisée comme mesure de sécurité afin de générer une nouvelle trajectoire qui offre une visibilité minimale de la balise.

6.6 Conclusion

Pour conclure ce chapitre, nous avons présenté un système de parcours autonome de piste de course pour drone. Nous avons utilisé une architecture similaire à celle présentée dans l'état de l'art, disposée selon quatre modules distincts. La loi de commande présentée au chapitre 4 et la méthode de génération de trajectoires présentée au chapitre 5 ont été reprises pour cette architecture. Des modules de détection et de cartographie de balises ont été créés afin de simuler les différents systèmes présents dans les systèmes d'autonomie pour drone. Ce système d'autonomie a ensuite été évalué pour sa robustesse et sa performance maximale sur deux pistes différentes typiques de course de drones. Il a été trouvé que la robustesse du système proposé s'améliore avec la vitesse, mais la vitesse de parcours maximale est limitée par les erreurs de commande et la visibilité des cibles lors du parcours de virages. Pour ce qui est des parcours à plus basse vitesse, une reformulation du problème d'optimisation pour inclure des contraintes au niveau de l'espace disponible pour exécuter la manoeuvre serait bénéfique. Par ailleurs, une méthodologie est proposée pour confirmer de façon quantitative les différentes hypothèses posées quant aux sources des échecs de notre système d'autonomie.

Pour ce qui est des résultats en général pour la phase d'exploration, pour la première piste,

nous obtenons des résultats similaires à ceux dans la littérature, mais pour des plus hautes vitesses. En ce qui a trait à la deuxième piste, nous obtenons un taux de réussite de 70% pour une vitesse de parcours moyenne désirée de 7 m/s (vitesse moyenne d'environ 7.5 m/s et vitesse maximale d'environ 12.7 m/s), ce qui est supérieur aux résultats trouvés dans la littérature. En ce qui a trait à l'évaluation de la phase de performance maximale de notre méthodologie proposée, nous trouvons que notre système est en mesure de parcourir la piste pour des vitesses moyennes désirées de 12 m/s et de 10 m/s, respectivement pour les première et deuxième pistes. Ces résultats permettent de valider notre hypothèse initiale que la modélisation de la piste en deux temps permettrait grandement d'améliorer les performances de drones dans le contexte de courses. Il est à noter que ces résultats sont à valider par voie expérimentale puisque les vitesses sont assez élevées et certains effets aérodynamiques non modélisés et non simulés font leur apparition dans ces conditions.

CHAPITRE 7 CONCLUSION

Ce chapitre résume les travaux faits dans ce mémoire et présente les limitations de la solution proposée par rapport au problème initial de course pour drone autonome.

7.1 Synthèse des travaux

L'objectif principal de ce projet de recherche était de développer un système autonome de contrôle et de navigation à hautes vitesses pour course de drones. Pour y parvenir, nous avons d'abord modélisé la dynamique complète d'un multicoptère à partir des équations de Newton. Par la suite, nous avons démontré la propriété de *platitude différentielle* du multicoptère qui permet de grandement simplifier le problème de génération de trajectoires dynamiquement faisables.

En deuxième partie, une loi de commande non-linéaire de suivi de trajectoire a été présentée. En utilisant diverses notions de stabilité (stabilité de Lyapunov, stabilité de systèmes autonomes-hybrides, etc.), nous avons démontré que la loi de commande était asymptotiquement stable pour un domaine restreint (erreur angulaire de type axe-angle inférieure à 90 degrés). Nous avons ensuite démontré que la loi de commande permet au drone de se rendre dans le domaine de stabilité asymptotique dans un temps fini, ce qui garantit une erreur de suivi nulle dans un temps fini et ce, peu importe les conditions initiales. L'implémentation discrète de cette loi de commande a ensuite été faite dans un simulateur haute fidélité dans l'environnement *ROS-Gazebo* et sa performance de suivi a été validée par des essais de suivi de trajectoires complexes et agressives.

En troisième partie, nous avons présenté une méthode de génération de trajectoires polynomiales optimales au niveau du *snap* de la trajectoire. Cette méthode de génération de trajectoire est ensuite utilisée pour générer des trajectoires optimales pour une piste de course complexe. Celle-ci est ensuite reformulée afin d'améliorer sa robustesse numérique et une méthode d'optimisation de l'allocation de temps entre les segments de trajectoire est présentée. Ces deux modifications sont ensuite appliquées à la même situation pour montrer leur bénéfice tant au niveau de la robustesse qu'au niveau de la réduction du coût de la solution optimale.

Pour finir, un système de parcours autonome de piste de course pour drone utilisant la loi de commande non-linéaire développée et la méthode de génération de trajectoire a été construit. Un module de détection de cibles simple a été développé afin de simuler les incertitudes

présentes dans un environnement de course. De concert, un système de cartographie des différentes balises en fonction des observations du système de détection a été construit. En agençant ces différents modules, un système d'autonomie a été formé. Ce système a ensuite été testé sur différentes pistes afin d'évaluer sa robustesse et sa performance maximale. Tout dépendant de la piste, nous obtenons des résultats similaires ou supérieurs à la littérature pour la phase d'exploration (taux de réussite maximal de 43.33% et 70% pour la première et deuxième piste) et nous pouvons presque doubler la vitesse moyenne de parcours lors de la phase de performance (12 m/s pour la première piste et 10 m/s pour la deuxième piste).

7.2 Limitations de la solution proposée

La solution proposée possède quelques limitations. Tout d'abord, la robustesse du système d'autonomie en phase d'exploration à basse vitesse est limitée par l'algorithme de génération de trajectoires. En effet, le calcul dynamique de trajectoires mène parfois à des trajectoires qui ne respectent pas l'espace disponible pour effectuer la manœuvre (collision avec d'autres balises ou le sol). Pour ce qui est de la performance maximale du système, elle est limitée par les erreurs de suivi de trajectoire et par la visibilité des cibles lors de virages. La performance du système se voit donc dégradée lorsqu'une piste présente beaucoup de virages abruptes. Par ailleurs, puisque la solution proposée avait comme but d'améliorer les algorithmes de navigation et de commande pour le parcours autonome de pistes de courses pour drone, la solution proposée n'est valide que pour des balises ayant une géométrie connue à l'avance et ayant une apparence particulière. Pour terminer, la solution proposée considère une odométrie parfaite, ce qui n'est pas représentatif des conditions réelles de courses pour drone. Le système développé n'est donc pas à l'abri des effets de dérive d'odométrie et la méthode de cartographie des cibles ne peut la compenser.

7.3 Développements futurs

Pour des travaux futurs, il serait intéressant d'ajouter un terme liée à l'observabilité des balises dans l'optimisation des trajectoires afin de limiter les collisions avec les balises par manque de visibilité. Pour ce qui est de la génération de trajectoires, il serait pertinent d'ajouter des limites exprimées en fonction de l'espace disponible pour éviter que le drone entre en collision avec le sol ou d'autres balises du parcours. Il serait aussi pertinent d'utiliser une méthode plus complexe telle que l'apprentissage profond pour faire une détection plus précise des différentes balises. Cette méthode pourrait possiblement résoudre le problème d'observabilité des balises sans avoir à reformuler le problème d'optimisation. Pour terminer,

il serait intéressant de déployer la solution proposée dans un environnement physique afin d'évaluer la performance réelle du système d'autonomie.

RÉFÉRENCES

[1] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, G. de Croon, S. Hwang, S. Jung, H. Shim, H. Kim, M. Park, T.-C. Au et S. J. Kim, "Challenges and implemented technologies used in autonomous drone racing," *Intelligent Service Robotics*, vol. 12, n°. 2, p. 137–148, avr. 2019. [En ligne]. Disponible : http://link.springer.com/10.1007/s11370-018-00271-6

[2] "Lockheed Martin and Drone Racing League Launch Groundbreaking AI Innovation Challenge," library Catalog : news.lockheedmartin.com. [En ligne]. Disponible : https://news.lockheedmartin.com/2018-09-05-Lockheed-Martin-and-Drone-Racing-League-Launch-Groundbreaking-AI-Innovation-Challenge

[3] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun et D. Scaramuzza, "Beauty and the Beast : Optimal Methods Meet Learning for Drone Racing," mars 2019, arXiv : 1810.06224.

[4] T. Hamel, R. Mahony, R. Lozano et J. Ostrowski, "Dynamic modelling and configuration stabilization for an X4-Flyer," *IFAC Proceedings Volumes*, vol. 35, n°. 1, p. 217–222, 2002. [En ligne]. Disponible : https://linkinghub.elsevier.com/retrieve/pii/S1474667015392697

[5] R. Mahony, V. Kumar et P. Corke, "Multirotor Aerial Vehicles : Modeling, Estimation, and Control of Quadrotor," *IEEE Robotics Automation Magazine*, vol. 19, n°. 3, p. 20–32, sept. 2012.

[6] P. Pounds, R. Mahony et P. Corke, "Modelling and control of a large quadrotor robot," *Control Engineering Practice*, vol. 18, n°. 7, p. 691 – 699, 2010. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0967066110000456

[7] J.-M. Kai, G. Allibert, M.-D. Hua et T. Hamel, "Nonlinear feedback control of Quadrotors exploiting First-Order Drag Effects," dans *IFAC World Congress*, Toulouse, France, juill. 2017. [En ligne]. Disponible : https://hal.archives-ouvertes.fr/hal-01544740

[8] N. K. Tran, E. Bulka et M. Nahon, "Quadrotor control in a wind field," dans *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, juin 2015, p. 320–328.

[9] S. Tang et V. Kumar, "Autonomous Flight," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, n°. 1, p. 29–52, mai 2018. [En ligne]. Disponible : https://www.annualreviews.org/doi/10.1146/annurev-control-060117-105149

[10] R. E. Kalman, “On the general theory of control systems,” *IFAC Proceedings Volumes*, vol. 1, n°. 1, p. 491–502, 1960. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S1474667017700948

[11] S. Bouabdallah, A. Noth et R. Siegwart, “PID vs LQ control techniques applied to an indoor micro quadrotor,” dans *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, sept. 2004, p. 2451–2456 vol.3.

[12] C. Liu, J. Pan et Y. Chang, “PID and LQR trajectory tracking control for an unmanned quadrotor helicopter : Experimental studies,” dans *2016 35th Chinese Control Conference (CCC)*, juill. 2016, p. 10 845–10 850, iSSN : 1934-1768.

[13] H. Khalil, *Nonlinear systems*, 3e éd. New Jersey : Prentice Hall, 2002.

[14] S. Bouabdallah et R. Siegwart, “Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor,” dans *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, avr. 2005, p. 2247–2252, iSSN : 1050-4729.

[15] V. K. Tripathi, L. Behera et N. Verma, “Design of sliding mode and backstepping controllers for a quadcopter,” dans *2015 39th National Systems Conference (NSC)*, déc. 2015, p. 1–6.

[16] H. L. N. N. Thanh et S. K. Hong, “Quadcopter Robust Adaptive Second Order Sliding Mode Control Based on PID Sliding Surface,” *IEEE Access*, vol. 6, p. 66 850–66 860, 2018, conference Name : IEEE Access.

[17] M. Kamel, T. Stastny, K. Alexis et R. Siegwart, “Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System,” dans *Robot Operating System (ROS)*, A. Koubaa, édit. Cham : Springer International Publishing, 2017, vol. 707, p. 3–39, series Title : Studies in Computational Intelligence. [En ligne]. Disponible : http://link.springer.com/10.1007/978-3-319-54927-9_1

[18] T. Lee, M. Leok et N. H. McClamroch, “Geometric tracking control of a quadrotor UAV on SE(3),” dans *49th IEEE Conference on Decision and Control (CDC)*, déc. 2010, p. 5420–5425.

[19] D. Mellinger et V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” dans *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China : IEEE, mai 2011, p. 2520–2525. [En ligne]. Disponible : http://ieeexplore.ieee.org/document/5980409/

[20] M. Faessler, F. Fontana, C. Forster et D. Scaramuzza, “Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor,”

dans *2015 IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA, USA : IEEE, mai 2015, p. 1722–1729. [En ligne]. Disponible : http://ieeexplore.ieee.org/document/7139420/

[21] D. Brescianini, M. Hehn et R. D'Andrea, "Nonlinear Quadrocopter Attitude Control : Technical Report," ETH Zurich, Rapport technique, 2013. [En ligne]. Disponible : http://hdl.handle.net/20.500.11850/154099

[22] D. Falanga, E. Mueggler, M. Faessler et D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," dans *2017 IEEE International Conference on Robotics and Automation (ICRA)*, mai 2017, p. 5774–5781.

[23] Y. Bouktir, M. Haddad et T. Chettibi, "Trajectory planning for a quadrotor helicopter," dans *2008 16th Mediterranean Conference on Control and Automation*, juin 2008, p. 1258–1263.

[24] M. Hehn et R. D'Andrea, "Quadrocopter Trajectory Generation and Control," *IFAC Proceedings Volumes*, vol. 44, n°. 1, p. 1485–1491, janv. 2011. [En ligne]. Disponible : https://linkinghub.elsevier.com/retrieve/pii/S147466701643819X

[25] M. Faessler, A. Franchi et D. Scaramuzza, "Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories," *IEEE Robotics and Automation Letters*, vol. 3, n°. 2, p. 620–626, avr. 2018.

[26] C. Richter, A. Bry et N. Roy, "Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments," dans *Robotics Research*, M. Inaba et P. Corke, édit. Cham : Springer International Publishing, 2016, vol. 114, p. 649–666. [En ligne]. Disponible : http://link.springer.com/10.1007/978-3-319-28872-7_37

[27] I. Frazier, "The Trippy, High-Speed World of Drone Racing," library Catalog : www.newyorker.com. [En ligne]. Disponible : https://www.newyorker.com/magazine/2018/02/05/the-trippy-high-speed-world-of-drone-racing

[28] D. Heitner, "Business Is Booming For The Leaders In Drone Racing," library Catalog : www.forbes.com Section : Business. [En ligne]. Disponible : https://www.forbes.com/sites/darrenheitner/2017/08/08/business-is-booming-for-the-leaders-in-drone-racing/

[29] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik et R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, n°. 10, p. 1157–1163, sept. 2016. [En ligne]. Disponible : http://journals.sagepub.com/doi/10.1177/0278364915620033

[30] A. Geiger, P. Lenz, C. Stiller et R. Urtasun, “Vision meets robotics : The KITTI dataset,” *The International Journal of Robotics Research*, vol. 32, n°. 11, p. 1231–1237, sept. 2013. [En ligne]. Disponible : http://journals.sagepub.com/doi/10.1177/0278364913491297

[31] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord et S. Karaman, “The Blackbird Dataset : A large-scale dataset for UAV perception in aggressive flight,” *arXiv :1810.01987 [cs]*, oct. 2018, arXiv : 1810.01987. [En ligne]. Disponible : http://arxiv.org/abs/1810.01987

[32] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler et D. Scaramuzza, “Are We Ready for Autonomous Drone Racing ? The UZH-FPV Drone Racing Dataset,” dans *2019 International Conference on Robotics and Automation (ICRA)*, mai 2019, p. 6713–6719, iSSN : 1050-4729.

[33] H. Moon, Y. Sun, J. Baltes et S. J. Kim, “The IROS 2016 Competitions [Competitions],” *IEEE Robotics Automation Magazine*, vol. 24, n°. 1, p. 20–29, mars 2017, conference Name : IEEE Robotics Automation Magazine.

[34] “Faster, Lighter, Smarter : DARPA Gives Small Autonomous Systems a Tech Boost.” [En ligne]. Disponible : https://www.darpa.mil/news-events/2018-07-18

[35] D. Mellinger, N. Michael et V. Kumar, “Trajectory generation and control for precise aggressive maneuvers with quadrotors,” *The International Journal of Robotics Research*, vol. 31, n°. 5, p. 664–674, avr. 2012, publisher : SAGE Publications Ltd STM. [En ligne]. Disponible : https://doi.org/10.1177/0278364911434236

[36] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun et D. Scaramuzza, “Deep Drone Racing : Learning Agile Flight in Dynamic Environments,” *arXiv :1806.08548 [cs]*, oct. 2018, arXiv : 1806.08548. [En ligne]. Disponible : http://arxiv.org/abs/1806.08548

[37] S. Jung, S. Hwang, H. Shin et D. H. Shim, “Perception, Guidance, and Navigation for Indoor Autonomous Drone Racing Using Deep Learning,” *IEEE Robotics and Automation Letters*, vol. 3, n°. 3, p. 2539–2544, juill. 2018.

[38] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu et A. C. Berg, “SSD : Single Shot MultiBox Detector,” dans *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe et M. Welling, édit. Cham : Springer International Publishing, 2016, p. 21–37.

[39] A. Krizhevsky, I. Sutskever et G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” dans *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou et K. Q. Weinberger, édit. Curran

Associates, Inc., 2012, p. 1097–1105. [En ligne]. Disponible : http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[40] S. Jung, S. Cho, D. Lee, H. Lee et D. H. Shim, “A direct visual servoing-based framework for the 2016 IROS Autonomous Drone Racing Challenge,” *Journal of Field Robotics*, vol. 35, n°. 1, p. 146–166, 2018. [En ligne]. Disponible : https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21743

[41] S. Li, M. M. O. I. Ozo, C. D. Wagter et G. C. H. E. d. Croon, “Autonomous drone race : A computationally efficient vision-based navigation and control strategy,” *Robotics and Autonomous Systems*, vol. 133, p. 103621, 2020. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0921889020304619

[42] E. J. Smeur, G. C. de Croon et Q. Chu, “Gust disturbance alleviation with Incremental Nonlinear Dynamic Inversion,” dans *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, oct. 2016, p. 5626–5631, iSSN : 2153-0866.

[43] A. Loquercio, A. I. Maqueda, C. R. del Blanco et D. Scaramuzza, “DroNet : Learning to Fly by Driving,” *IEEE Robotics and Automation Letters*, vol. 3, n°. 2, p. 1088–1095, avr. 2018, conference Name : IEEE Robotics and Automation Letters.

[44] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun et D. Scaramuzza, “Deep Drone Racing : From Simulation to Reality with Domain Randomization,” *IEEE Transactions on Robotics*, vol. 36, n°. 1, p. 1–14, févr. 2020.

[45] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar et D. Scaramuzza, “AlphaPilot : Autonomous Drone Racing,” dans *Robotics : Science and Systems XVI*. Robotics : Science and Systems Foundation, juill. 2020. [En ligne]. Disponible : http://www.roboticsproceedings.org/rss16/p081.pdf

[46] O. Ronneberger, P. Fischer et T. Brox, “U-Net : Convolutional Networks for Biomedical Image Segmentation,” dans *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells et A. F. Frangi, édit. Cham : Springer International Publishing, 2015, p. 234–241.

[47] Z. Cao, T. Simon, S. Wei et Y. Sheikh, “Realtime Multi-person 2D Pose Estimation Using Part Affinity Fields,” dans *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, juill. 2017, p. 1302–1310, iSSN : 1063-6919.

[48] D. Brescianini et R. D’Andrea, “Tilt-Prioritized Quadrocopter Attitude Control,” *IEEE Transactions on Control Systems Technology*, vol. 28, n°. 2, p. 376–387, mars 2020, conference Name : IEEE Transactions on Control Systems Technology.

[49] D. Bertsekas, *Dynamic Programming and optimal control*. Athena scientific Belmont, MA, 1995, vol. 1, n°. 2.

[50] J. Diebel, “Representing Attitude : Euler Angles, Unit Quaternions, and Rotation Vectors,” *Matrix*, vol. 58, n°. 15-16, p. 1–35, 2006.

[51] M. Kamel, K. Alexis, M. Achtelik et R. Siegwart, “Fast nonlinear model predictive control for multicopter attitude tracking on SO(3),” dans *2015 IEEE Conference on Control Applications (CCA)*, sept. 2015, p. 1160–1166, iSSN : 1085-1992.

[52] M. J. V. Nieuwstadt et R. M. Murray, “Real-time trajectory generation for differentially flat systems,” *International Journal of Robust and Nonlinear Control*, vol. 8, n°. 11, p. 995–1020, 1998.

[53] N. Koenig et A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator,” dans *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, sept. 2004, p. 2149–2154 vol.3.

[54] F. Furrer, M. Burri, M. Achtelik et R. Siegwart, “RotorS—A Modular Gazebo MAV Simulator Framework,” dans *Robot Operating System (ROS) : The Complete Reference (Volume 1)*, A. Koubaa, édit. Springer International Publishing, 2016, p. 595–625. [En ligne]. Disponible : http://dx.doi.org/10.1007/978-3-319-26054-9_23

[55] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler et A. Ng, “ROS : an open-source Robot Operating System,” dans *workshop on open source software*, vol. 3, Kobe, Japan, 2009, p. 5.

[56] J. Lygeros, S. Sastry et C. Tomlin, “Hybrid Systems : Foundations, advanced topics and applications,” *under copyright to be published by Springer Verlag*, 2012.

[57] T. Flash et N. Hogans, “The Coordination of Arm Movements : An Experimentally Confirmed Mathematical Model’,” *The Journal of Neuroscience*, vol. 5, n°. 7, p. 1688–1703, 1985.

[58] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli et R. Siegwart, “A robust and modular multi-sensor fusion approach applied to MAV navigation,” dans *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, nov. 2013, p. 3923–3929.

[59] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, n°. 6, p. 679–698, nov. 1986, conference Name : IEEE Transactions on Pattern Analysis and Machine Intelligence.

[60] S. Suzuki et K. be, “Topological structural analysis of digitized binary images by border following,” *Computer Vision, Graphics, and Image Processing*, vol. 30, n°. 1, p. 32–46, avr. 1985. [En ligne]. Disponible : https://linkinghub.elsevier.com/retrieve/pii/0734189X85900167

[61] D. H. Douglas et T. K. Peucker, “Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature,” *Cartographica : The International Journal for Geographic Information and Geovisualization*, vol. 10, p. 112–122, 1973.

[62] V. Leptit, F. Moreno-Noguer et P. Fua, “Epnp : An accurate o(n) solution to the pnp problem,” *Internation journal of computer vision*, vol. 81, n°. 2, p. 155, 2009.

[63] B. Morrell, R. Thakker, G. Merewether, R. Reid, M. Rigter, T. Tzanetos et G. Chamitoff, “Comparison of Trajectory Optimization Algorithms for High-Speed Quadrotor Flight Near Obstacles,” *IEEE Robotics and Automation Letters*, vol. 3, n°. 4, p. 4399–4406, oct. 2018.