HAL
open science

Évitement d'obstacles pour quadrirotors en utilisant un capteur de profondeur

Thibaut Tezenas Du Montcel

To cite this version:

Thibaut Tezenas Du Montcel. Évitement d'obstacles pour quadrirotors en utilisant un capteur de profondeur. Automatique. Université Grenoble Alpes, 2019. Français. NNT : 2019GREAT077 . tel-02534987

HAL Id: tel-02534987

https://theses.hal.science/tel-02534987

Submitted on 7 Apr 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITE GRENOBLE ALPES

Spécialité : **AUTOMATIQUE - PRODUCTIQUE**

Arrêté ministériel : 25 mai 2016

Présentée par

Thibaut TEZENAS DU MONTCEL

Thèse dirigée par **Nicolas MARCHAND,** directeur de recherche, CNRS,
et co-encadrée par **Amaury NEGRE, i**ngénieur de recherche, CNRS,
et par **José-Ernesto GOMEZ BALDERAS,** maître de conférences, Université Grenoble Alpes.

préparée au sein du **Laboratoire Grenoble Images Parole Signal Automatique (GIPSA-lab)**
dans **l'École Doctorale Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)**

Évitement d'obstacles pour quadrirotors en utilisant un capteur de profondeur

Thèse soutenue publiquement le **16 décembre 2019**,
devant le jury composé de :

Monsieur Cédric DEMONCEAUX
Professeur, Université de Bourgogne, Rapporteur – Président du jury
Madame Isabelle FANTONI
Directeur de Recherche, CNRS, Rapporteur
Monsieur Guillaume ALLIBERT
Maître de Conférences, Université Nice Sophia-Antipolis, Examinateur
Monsieur Nicolas MARCHAND
Directeur de Recherche, CNRS, Directeur de thèse
Monsieur Amaury NEGRE
Ingénieur de recherche, CNRS, Examinateur
Monsieur José-Ernesto GOMEZ-BALDERAS
Maître de Conférences, Université Grenoble Alpes, Examinateur

COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

ÉCOLE DOCTORALE EEATS

Électronique, Électrotechnique, Automatique
et Traitement du Signal

T H È S E

pour obtenir le titre de

docteur en sciences

de la Communauté Université Grenoble Alpes
Mention : Automatique et Productique

Présentée et soutenue par

Thibaut TEZENAS DU MONTCEL

Évitement d'obstacles pour quadrirotors en utilisant un capteur de profondeur

Thèse dirigée par Nicolas MARCHAND
et co-encadrée par Amaury NEGRE
et José-Ernesto GOMEZ-BALDERAS

préparée au laboratoire Grenoble Images Parole Signal Automatique

soutenue 16/12/2019

Jury :

Rapporteurs :	Isabelle FANTONI	-	CNRS
	Cédric DEMONCEAUX	-	Université Bourgogne
Examinateurs :	Guillaume ALLIBERT	-	Université de Sophia-Antipolis
	Nicolas MARCHAND	-	CNRS
	Amaury NEGRE	-	CNRS
	José-Ernesto GOMEZ-BALDERAS	-	Université Grenoble Alpes

Remerciements

J'ai beau être en passe d'obtenir un diplôme reconnaissant mon travail personnel au cours des trois dernières années, jamais un drone n'aurait évité les obstacles au Gipsa-Lab si j'avais été seul à travailler sur le sujet. Merci donc à Amaury et Matthieu avec qui j'ai travaillé en collaboration au jour le jour pendant toute la phase d'expérimentation. Tout ne s'est pas pas passé comme on l'aurait souhaité mais ces expérimentations resteront parmi les meilleurs moments de ma thèse. Merci à mes encadrants qui ont en particulier du subir la relecture de mes papiers pendant trois ans et merci à l'ensemble du jury qui s'est volontairement infligé la lecture de ma thèse ! Plus généralement, je tiens à remercier tous ceux qui ont travaillé sur des sujets proches du mien pendant et avant ma présence au laboratoire et qui ont laissé des notes de leurs travaux. Gloire à tous ces petits README jetés à l'eau qui émergent quand un nouvel arrivant tente de se repérer dans un environnement pas encore familier ! Et enfin, merci à tous ceux qui ont pris du temps spécifiquement pour moi, pour ma thèse. Merci à tous ceux qui laissent leur porte ouverte et chez qui on sent la possibilité de prendre quelques minutes pour profiter de conseils techniques. Merci donc à Etienne, Julien, Ludovic, Bruce, Ivan, Florent et j'en oublie !

Je n'apprendrai rien à ceux qui sont passé par là, une thèse ce n'est pas que du travail technique et/ou scientifique, c'est aussi une épreuve morale. Sans un environnement positif qui m'a poussé pendant ces trois ans, je n'aurais jamais terminé cette thèse. Je voulais donc aussi remercier tous ceux qui m'ont apporté un support moral pendant cette thèse. Merci à Matthieu qui m'a écouté me lamenter des dizaines de fois lors de tests ratés, quelle patience ! Merci à mes cobureaux pour avoir supporté tous mes exclamations ponctuant mes tests et pour m'avoir supportés à chaque fois que j'avais une heure de démotivation et une envie de discuter à les couper de leurs travaux ! Merci à Jonathan et Amaury pour cette discussion au retour de mes vacances il y a un peu plus d'un ans, quand toute l'envie de revenir était partie, que vous l'avez senti et m'avez conseillé de repartir sur quelque chose de différent. Merci à tous ceux qui font vivre la K'fet du DAuto, aux membres du Gipsadoc qui animent le labo et à tous ceux qui au cours d'une discussion m'ont glissé quelques mots d'encouragements !

Finalement merci à tous ceux qui hors du labo m'ont permis d'atteindre un équilibre. Merci à tous ceux avec qui j'ai pu sortir en montagne, dans cet espace pour moi synonyme de liberté ! Merci à tous ceux qui se reconnaîtront comme des licornes volantes et à mes différents partenaires de sports ! Merci à ma famille, proche comme lointaine, pour tout votre soutien ! Et merci à tous ces inconnus qui en s'extasiant devant mon sujet m'ont rappelés à de nombreuses reprises la chance que j'ai eu ces trois dernières années !

Table des matières

1 Introduction **1**

1.1 Introduction 1

1.2 Structure de la thèse 2

1.3 Les contributions de la thèse 2

1.4 Publications 3

1.5 Financement 3

2 Les quadrirotors **5**

2.1 Théorie simplifiée du Vol 5

2.2 Définition de l'espace 9

2.2.1 Le référentiel inertiel 9

2.2.2 Le référentiel drone 10

2.2.3 Définition des angles d'Euler utilisés 11

2.3 Théorie quantifiée du vol 12

2.3.1 Modélisation simple des contributions des hélices 13

2.3.2 Modélisation plus fine 14

2.3.3 Liens entre attitude, poussée et accélérations 16

2.4 Les quadrirotors en pratique 17

2.4.1 Législation et dangers liés aux quadrirotors 17

2.4.2 Les composants essentiels 19

2.4.3 Firmwares 21

2.4.4 Contrôle 22

2.4.5 Observateur 24

2.4.6 Besoins supplémentaires pour fonctionnalités avancées 25

3 Les algorithmes d'évitement d'obstacles 27

3.1 Revue d'algorithmes d'évitement d'obstacles pour drones 27

3.2 Notre définition de l'évitement d'obstacles 33

3.3 Les représentations de l'espace 35

3.3.1 Grilles d'occupations 3D 35

3.3.2 Points ou surfaces d'intérêts 36

3.3.3 Cartes de profondeur égocentriques 37

3.3.4 Notre représentation, l'egocubemap 38

3.4 Les différentes générations de commandes 44

3.4.1 Commandes autres que trajectoire 45

3.4.2 Librairies de trajectoires 46

3.4.3 Génération de trajectoires en ligne 48

3.4.4 Notre méthode génération de trajectoires, en ligne, basée sur une minimisation du jerk 49

3.5 Les différents modes de contrôle 54

3.5.1 Contrôleurs sans trajectoire 54

3.5.2 Contrôleurs pour suivi de trajectoires 55

3.5.3 Notre contrôle 56

3.6 Description de l'algorithme SOTA 58

3.7 Algorithme basé sur le filtrage spatial des trajectoires 66

3.8 Extension n°1 : Gestion des minimums locaux 71

3.9 Extension n°2 : Gestion des obstacles dynamiques 72

3.10 Quelques choix techniques 73

4 Les environnements de test et le benchmark 75

4.1 Les environnements simulés 75

4.1.1 Les simulateurs de quadrirotor utilisés 75

4.1.2 L'environnement simple 77
4.1.3 Les environnements dynamiques 79
4.1.4 La forêt simulée 81
4.2 L'environnement hardware-in-the-loop 84
4.2.1 L'arène de vol avec capture de mouvement 84
4.2.2 Les environnements simulés 87
4.3 L'environnement de vol extérieur 88
4.3.1 Le terrain d'expérimentation 88
4.3.2 Le quadrirotor expérimental 88
4.4 Le benchmark 93
4.4.1 Les motivations 94
4.4.2 Spécifications des environnements, capteurs et tests 95
4.4.3 Utilisation détaillée étape par étape 97
4.4.4 Analyse statistique des résultats 100

5 Résultats de contrôle et d'évitement 103

5.1 Étude de la qualité de suivi de trajectoires 103
5.2 Résultats d'évitement d'obstacles avec quadrirotors réels 108
5.3 Caractérisation des algorithmes en simulation 109
5.4 Résultats des tests avec obstacles dynamiques 113
5.5 Temps de calcul des algorithmes 115

6 Conclusion et Perspectives 117

6.1 Conclusions 117
6.2 Perspectives 118

Bibliographie 120

Table des figures

2.1 Forces et moments principaux qui s'appliquent à un quadrirotor. 6

2.2 Mouvements des quadrirotors (1). 7

2.3 Mouvements des quadrirotors (2). 8

2.4 Définition des repères cartésiens couramment utilisés dans le repère terrestre. . 10

2.5 Définition du repère $\mathbb{B}$ lié au drone, des centres géométriques et de gravité et numérotation des moteurs. 11

2.6 Définition des angles d'Euler. 12

2.7 Représentation schématique de l'effet de flapping. 15

2.8 Schéma des différents blocs logiciels de ROSflight. 21

2.9 Schéma d'un contrôleur en attitude. 23

2.10 Schéma de l'estimateur d'attitude de ROSflight. 25

3.1 Fonctionnalités principales et entrées et sorties des algorithmes d'évitement d'obstacles. 29

3.2 Vue schématique du travail effectué. 34

3.3 Egocylindre de Brockers et al. 37

3.4 Schémas d'une cubemap. 39

3.5 Comparaison des représentations cubemap (a) et sphérique (b). 40

3.6 C-Space : schéma de principe. 42

3.7 C-Space : reconstruction. 44

3.8 Scène simulée représentée par une egocubemap sans (a) et avec (b) C-space. . . 44

3.9 Projections équirectangulaires de deux égocubemap successives. 45

3.10 Schéma de principe du contrôle en cascade. 55

3.11 Schéma bloc du contrôle lors de l'utilisation des PIDs en cascade. 57

3.12 Schéma bloc du contrôle lors de l'utilisation du contrôleur à retour d'état. . . . 58

3.13 Architecture de l'algorithme SOTA. 59

3.14 Pré-sélection des directions pour $ND = 100$. 61

3.15 Distance tampon par rapport aux obstacles. 63

3.16 Schéma de génération des trajectoires selon une direction. 65

3.17 Répartition des trajectoires générées dans l'espace. 68

3.18 Matrice de coûts filtrée (haut gauche) et non filtrée (bas gauche) pour une situation décrite par une vue schématique de dessus (haut droite) et la représentation équirectangulaire de l'egocubemap correspondante (bas droite). 70

3.19 Masque pour prise en compte des minimums locaux. 71

4.1 Environnements légers générés pour les tests en phase de développement. 78

4.2 Visualisation du cylindre utilisé pour détecter les collisions du quadrirotor simulé. 79

4.3 Évolution sur 5 secondes d'un environnement dynamique aléatoire. 80

4.4 Évolution sur 5 secondes d'un environnement dynamique pour tester des cas d'évitement spécifiques. 81

4.5 Vue de dessus d'une forêt générée aléatoirement. 82

4.6 Comparaison d'une carte de profondeur bruitée simulée et d'une image de profondeur perçue par une Intel® RealSense™. 85

4.7 Photo de l'arène de vol du Gipsa-Lab. 86

4.8 Vue en parallèle d'un test HITL pour lequel on a placé des obstacles physiques et de l'environnement de simulation correspondant. 87

4.9 Vue satellite et photos du terrain d'expérimentation. 89

4.10 Photo du quadrirotor construit pour effectuer les tests en extérieur. 90

4.11 Schéma de communication du quadrirotor d'expérimentation. 92

4.12 Vue schématique de dessus des points de passage et du calcul de la *distance linéaire*. 99

5.1 Vues des deux drones impliqués dans les tests sur la qualité de contrôle. 104

5.2 Vue de dessus des trajectoires effectuées avec différentes options de contrôle. (a) munchkin42 et retour d'état. (b) munchkin42 et PIDs en cascade. (c) hummingbird simulé et lee contrôleur. (d) hummingbird simulé et retour d'état. (e) hummingbird simulé et PIDs en cascade. 106

5.3 Vue de dessus du résultat d'un test HITL. 110

5.4 Évolution sur 8 secondes d'un quadrirotor dans l'environnement dynamique aléatoire. 114

Liste des tableaux

4.1 Nombre de test pour différentes précisions et bornes d'incertitudes. 100

5.1 Résultats des tests de suivi de trajectoire. 107

5.2 Résultats du benchmark pour différents algorithmes d'évitement d'obstacles. . . 112

5.3 Temps moyen d'exécution des algorithmes proposés. 115

Table des sigles et acronymes

UAV Unmmaned Aerial Vehicle ou aéronef sans-pilote
PID Proportionnel, Intégral, Dérivatif
LQR Linear-Quadratic Regulator ou régulateur quadratique linéaire
MPC Model Predictive Control ou commande prédictive
HITL Hardware-In-The-Loop ou matériel dans la boucle
EKF Extended Kalman Filter ou filtre de Kalman étendu
SLAM Simultaneous Localization And Mapping ou localisation et cartographie simultanées
FPV First Person View ou Vue à la première personne
ESC Electronic Speed Controller ou variateurs de vitesse électronique
FPGA Field-Programmable Gate Array ou réseau de portes programmables in situ
ENU East, North, Up ou est, nord, haut
NED North, East, Down ou nord, est, bas
NWU North, West, Up ou nord, ouest, haut
IMU Inertial Measurement Unit ou centrale inertielle
Li-Po Lithium-ion Polymère
GPU Graphics Processing Unit ou processeur graphique
DGAC Direction Générale de l'Aviation Civile
DSAC Direction de la Sécurité de l'Aviation Civile
IID Indépendants et Identiquement distribués

Chapitre 1

Introduction

1.1 Introduction

Cette thèse s'inscrit dans le projet CAP2018 dont le but est d'améliorer la sûreté des drones quadrirotors. Le projet se concentre sur la création d'un auto-pilote dont le code sera certifiable au même titre que les auto-pilotes du transport aérien commercial. Pour atteindre la certification, les technologies choisies dans le projet devront veiller à minimiser le risque d'un crash matériel ou logiciel. ADA, un langage de programmation développé pour pouvoir répondre à tous les besoins du département de la Défense des États-Unis, du guidage des missiles à leurs applications de gestion, et réputé pour sa stabilité sera par exemple utilisé pour les fonctionnalités essentielles de l'auto-pilote.

Cette thèse est une extension de l'auto-pilote développé au cours du projet CAP2018. Elle se concentre sur l'évitement d'obstacles autonome. Son ambition est de proposer un logiciel générique qui puisse s'adapter facilement à différents drones du commerce. Il doit donc être utilisable par des drones de différentes tailles et poids et avec différents types de capteurs. L'objectif de cette thèse est d'exécuter l'algorithme d'évitement d'obstacles développé au cours de la thèse sur un drone construit par un autre acteur du projet. Le drone doit alors être capable de suivre des points de passage définis par l'utilisateur en évitant les obstacles présents sur le parcours. La conception de ce logiciel générique posera de nombreuses questions parmi lesquelles :

- Comment représenter l'espace et les obstacles dans l'unique but de faire de l'évitement autonome et quelle type de représentation peut facilement s'adapter à des changements de capteurs ?
- Comment contrôler des quadrirotors dont le poids peut varier d'un facteur de 1 à 8 (de 250 g à 2 kg) et le diamètre de 1 à 5 20 cm à 1 m ?
- Comment générer en temps réel des commandes réalisables par tous ces quadrirotors et quelles commandes privilégier pour maximiser les performances d'évitement ?

S'il est évident qu'un objectif majeur de cette thèse est de réaliser des vols d'évitement d'obstacles autonomes, il nous semble plus important encore d'être capable de s'assurer de la robustesse des solutions proposées. Cette réflexion, qui va dans le sens du projet CAP2018, vers une garantie de fiabilité, ouvre aussi un certain nombre de questions. Par exemple, question essentielle, comment s'assurer de la fiabilité des tests effectués ? Ou encore, est-il possible de définir des conditions sous lesquelles la sécurité d'un quadrirotor serait garantie et est-ce que

ces garanties restent valables en présence d'obstacles dynamiques ? Ces questionnements, sur une évaluation quantitative des solutions proposées, prendront finalement beaucoup de place dans cette thèse pour tenter de combler ce qui est perçu comme un manque dans la littérature.

1.2 Structure de la thèse

Après cette brève introduction, le chapitre 2 présentera le robot sur lequel porte cette thèse, les drones de type quadrirotor. Cette partie donnera lieu à de la vulgarisation sur du fonctionnement des quadrirotors, à l'étude, en utilisant le formalisme de la mécanique, des quadrirotors et à la mise en parallèle de la théorie issue de cette étude avec l'utilisation, en pratique, des quadrirotors.

Le chapitre 3 sera dédié à la présentation des algorithmes d'évitement d'obstacles développés par d'autres équipes au cours des dernières années et à la présentation des algorithmes développés au cours de la thèse. La structure de ce chapitre permettra de mettre en parallèle les principales possibilités lors du développement d'un algorithme d'évitement d'obstacles et les choix réalisés au laboratoire. Un algorithme permettant d'effectuer de l'évitement d'obstacles dynamique avec un unique capteur de profondeur possédant un champ de vision limité sera notamment présenté.

Le chapitre 4 se concentrera sur les différents environnements de tests créés autour de ces algorithmes. Comme annoncé en introduction, cette partie est le résultat d'un long travail de développement pour s'assurer de la fiabilité des tests proposés. Elle constitue plus de la moitié du travail effectué durant la thèse. Plusieurs environnements de simulation et de vol réels seront présentés.

La présentation des résultats constituera ensuite le chapitre 5. Les forces et faiblesses des différentes propositions seront mises en avant grâce à l'utilisation des tests présentés dans le chapitre 4.

Viendra finalement le moment de conclure sur le travail effectué dans le chapitre 6. Cette conclusion sera associée à des perspectives sur l'un des environnements de test proposés et sur les algorithmes d'évitement proposés.

1.3 Les contributions de la thèse

Cette thèse présente les contributions suivantes :

- développement d'un algorithme d'évitement d'obstacles agrégeant des briques préexistantes créées indépendamment les unes des autres dans les années précédant le début de la thèse
- création d'un benchmark statistique en simulation pour évaluer et comparer les performances d'algorithmes d'évitement d'obstacles pour quadrirotors

— création d'un algorithme d'évitement d'obstacles basé sur une nouvelle stratégie de génération des trajectoires dans l'espace.
— création d'un algorithme d'évitement dynamique grâce à une extension de l'algorithme précédent
— validation des algorithmes développés grâce aux résultats du benchmark proposé
— étude de modalités de suivi de trajectoires et de la transférabilité de la simulation à des vols réel

1.4 Publications

Cette thèse a donné lieu à la publication des travaux suivants :
— **Tezenas Du Montcel, T.**, Nègre, A., Muschinowski, M., Gomez-Balderas, J. E., & Marchand, N. : *An Egocubemap Based Algorithm for Quadrotors Obstacle Avoidance Using a Single Depth Camera*, 2018, 10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles at IEEE International Conference on Intelligent Robots and Systems (IROS)
— **Tezenas Du Montcel, T.**, Nègre, A., Gomez-Balderas, J. E., & Marchand, N. : *BOARR : A Benchmark for quadrotor Obstacle Avoidance based on ROS and RotorS*, 2019, dépôt HAL

De même, les présentations suivantes ont été données dans le cadre de la thèse :
— *An Egocubemap Based Algorithm for Quadrotors Obstacle Avoidance Using a Single Depth Camera*, 10$_{\text{th}}$ Workshop on Planning, Perception and Navigation for Intelligent Vehicles, IROS, Madrid, October 2018
— *BOARR : A Benchmark for quadrotor Obstacle Avoidance based on ROS and RotorS*, ROSCon Fr, Paris, June 2019
— *BOARR : A Benchmark for quadrotor Obstacle Avoidance based on ROS and RotorS*, Journée GT UAV, Marseille, June 2019
— *A spatial filter based algorithm and a benchmark for obstacle avoidance for quadrotors*, invited presentation, Naver Labs Europe, Meylan, June 2019

1.5 Financement

La Région Auvergne-Rhône-Alpes

Cette thèse a été financée par la région Auvergne Rhône Alpes dans le cadre du programme CAP2018 (FUI 20 AAP - pôle de compétitivité MINALOGIC) opération n°15 021140 01.

Chapitre 2

Les quadrirotors

La plate-forme utilisée durant cette thèse, un quadrirotor, sera décrite dans cette partie. On commencera par expliquer, en 2.1, en reprenant les mots d'élèves de troisième, les grands principes qui régissent les vols des quadrirotors. Dans un second temps , en 2.2, on définira un ensemble d'outils mathématiques, en utilisant le formalisme de la physique, qui permettent de décrire l'espace. Cela nous permettra dans une troisième partie (2.3), de modéliser mathématiquement au travers d'équations physiques le vol d'un quadrirotor. On s'attardera ensuite, dans la partie 2.4, sur des observations plus pratiques sur les quadrirotors. Quels sont les risques qu'ils causent ? Quelle est la législation qui les entoure ? Ou encore quels sont les éléments essentiels à un quadrirotor, physiquement et logiciellement ? On se demandera enfin quels ajouts sont nécessaires pour passer d'un drone minimal, qui ne comporte que ses éléments essentiels, à un drone capable d'effectuer de l'évitement d'obstacles en toute autonomie.

2.1 Théorie simplifiée du Vol

Remarque. *Cette section a été écrite pour expliquer le plus clairement possible le principe de fonctionnement des drones à des personnes n'ayant aucune connaissance préalable sur le sujet. Elle est inspirée d'une présentation créée par des élèves de troisième ayant effectué un stage sur les drones au sein du laboratoire. Ces élèves ayant travaillé une semaine entière sur le sujet, cette section s'adresse plutôt à des lycéens.*

Pour modéliser un quadrirotor très simplement, on peut le considérer comme un unique solide sur lequel s'appliquent la force de pesanteur ainsi que 4 forces et moments, créés par la rotation des hélices, au centre de chacune des hélices. Les forces *poussent* le drone dans une direction quand les moments *tournent* le drone selon un axe. En cas de doute sur ce qu'est un moment, vous pouvez consulter cette vidéo [pen16] d'**e-penser** qui les présente plus précisément. Les forces et moments qui s'appliquent à notre quadrirotor sont représentés sur la figure 2.1.

Comme on peut le voir sur la figure 2.1, les forces créées par la rotation des hélices sont portées par l'axe de rotation des hélices. De leur coté, les moments sont de sens opposés aux sens de rotation de l'hélice. Ces forces et moments créés par les hélices grandissent en fonction de la vitesse de rotation des hélices.

On veut maintenant montrer qu'en jouant sur les vitesses de rotation des hélices, et

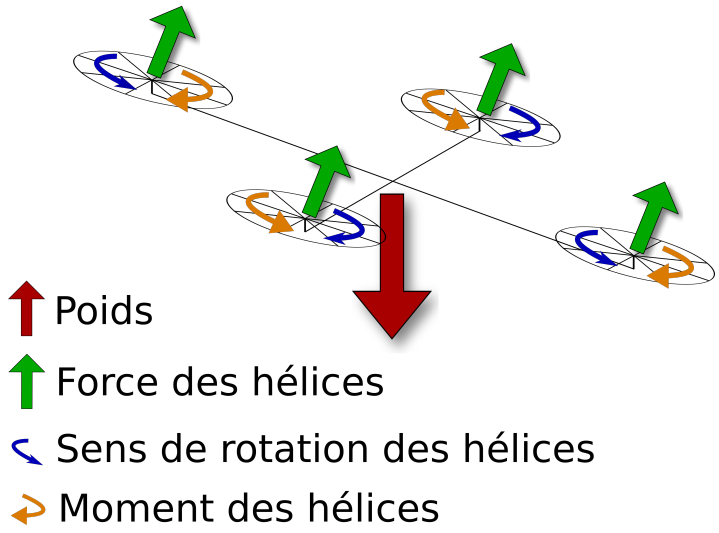


FIGURE 2.1 – Forces et moments principaux qui s'appliquent à un quadrirotor.

donc sur les forces et moments qu'elles créent, il est possible de contrôler les mouvements du quadrirotor. Mathématiquement, cela revient à calculer la somme des forces et des moments au centre de notre drone mais on va plutôt essayer de comprendre schématiquement quels sont les mouvements possibles.

On commence par les mouvements les plus simples qui sont présentés sur la figure 2.2. Tout premier mouvement, en posant le drone à plat au sol, on fait tourner toutes les hélices à la même vitesse. Quand les hélices dépassent une certaine vitesse de rotation, la somme des forces générées par les hélices est supérieure au poids et le drone décolle et monte verticalement. Dans le même temps, les moments générés par les hélices se compensent et le drone ne tourne donc pas. Ceci est schématisé sous le nom *déplacement vertical*.

Une fois en vol, en réduisant la vitesse de rotation des hélices, quand la somme des forces générées par les hélices est exactement égale au poids, il est possible d'obtenir une position d'équilibre comme schématisé sous l'intitulé *vol stationnaire*.

Dernier mouvement simple, la rotation autour de l'axe vertical. Pour créer un mouvement de rotation autour de l'axe principal, il faut augmenter la vitesse de rotation des deux moteurs qui tournent dans un sens et diminuer la vitesse de rotation de ceux qui tournent dans le sens opposé. On aura alors toujours un équilibre des forces mais la somme des moments elle ne sera pas nulle comme il est possible de le voir sur le schéma nommé *rotation autour de l'axe vertical*.

Ces mouvements sont les plus simples à comprendre mais ne représentent pas la totalité des mouvements possibles. Pour percevoir les autres mouvements possibles, il faut comprendre que si une force n'est pas appliquée au niveau du centre de gravité d'un solide, celle-ci va générer un nouveau moment qui va faire tourner le solide. L'exemple d'une planche plaquée contre un mur et fixée uniquement par son centre de gravité, comme présenté sur la figure 2.3 devrait être familier pour tout le monde. De la même façon, si on augmente la vitesse de

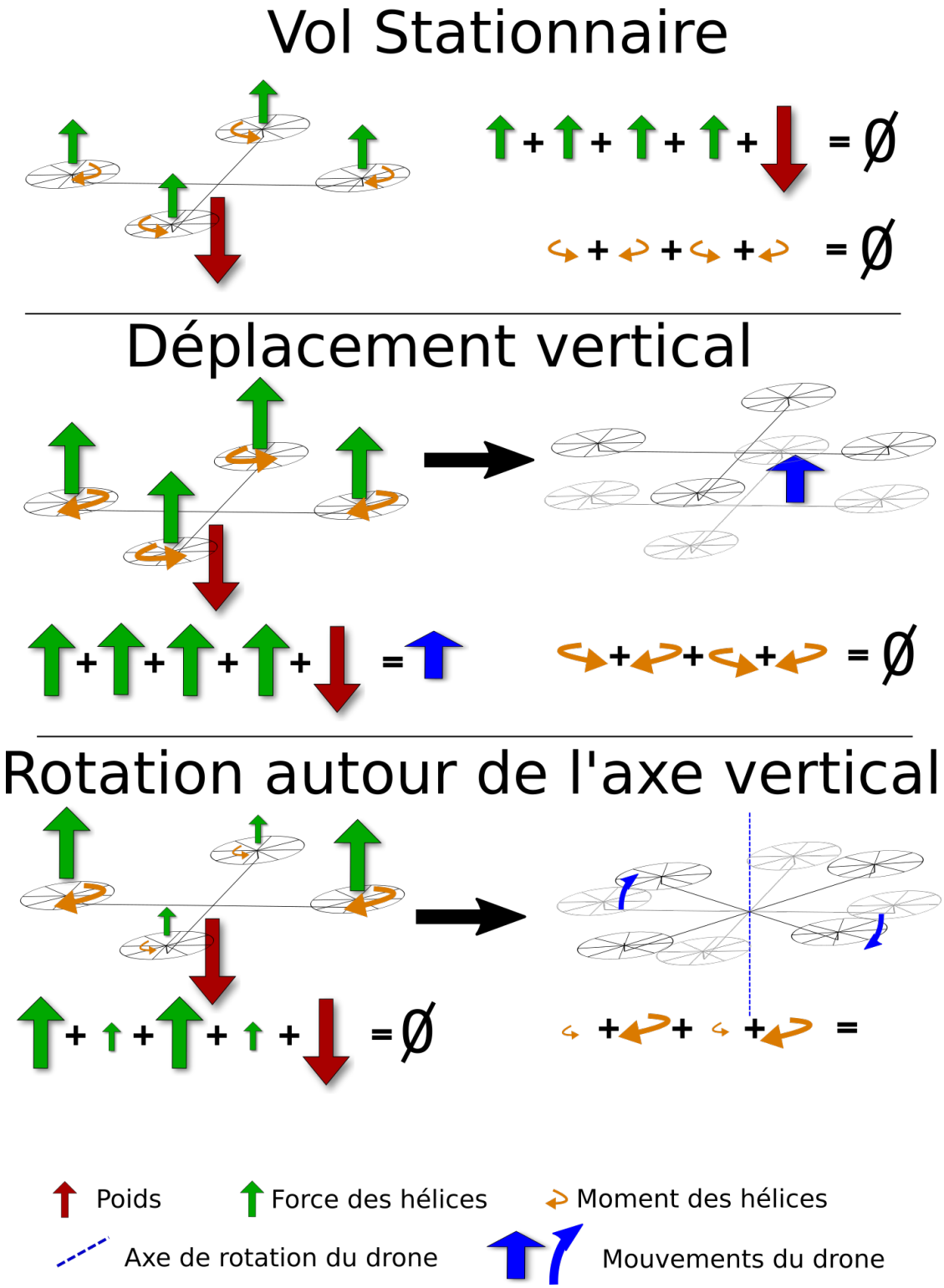


FIGURE 2.2 – Mouvements des quadrirotors (1).

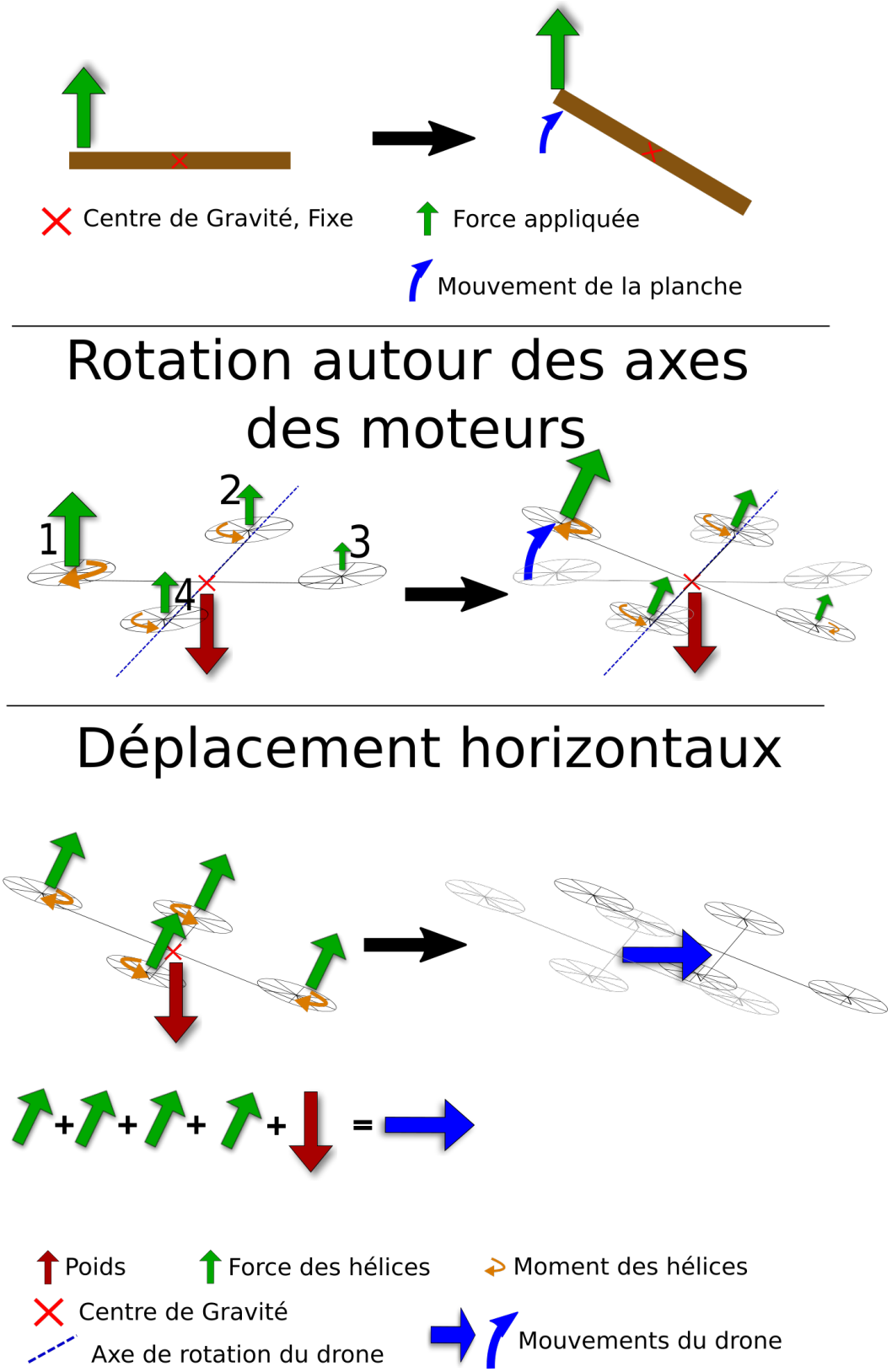


FIGURE 2.3 – Mouvements des quadrirotors (2).

rotation d'un moteur et qu'on réduit la vitesse de rotation du moteur opposé, on va faire tourner le drone comme présenté sur le schéma *Rotation autour des axes des moteurs*. Sur le schéma, on a augmenté la vitesse du moteur 1 et réduit la vitesse du moteur 3. Cela donne lieu à une rotation sur l'axe entre les moteurs 2 et 4. On peut imaginer augmenter la vitesse du moteur du 3 et diminuer celle du moteur 1, on tournera alors autour du même axe mais dans l'autre sens. Finalement, on peut aussi modifier les vitesses des moteurs 2 et 4 ce qui va nous permettre de tourner autour de l'axe des moteurs 1 et 3. En associant ces mouvements au mouvement autour de l'axe vertical présenté précédemment, il est donc possible d'orienter le drone selon notre bon vouloir.

Il reste une dernière question, comment le drone se déplace horizontalement ? La réponse est présentée sur le dernier schéma de la figure 2.3, il faut commencer par incliner le drone. La somme des forces créées par les hélices et le poids ne sera alors plus nulle mais orientée vers le moteur le plus bas. Le drone avancera donc dans cette direction.

Ces explications permettent de comprendre quels sont les mouvements possibles pour un quadrirotor. Il est par contre compliqué de savoir ce qu'il va se passer dans une situation précise. Par exemple, si on ralenti une hélice alors que le drone est déjà en train de tourner, que va-t-il se passer ? Plus généralement, lorsque le drone n'est plus immobile avant de modifier les vitesses de rotation des hélices, les mouvements générés ne sont plus intuitifs et il devient nécessaire de modéliser mathématiquement le vol du quadrirotor pour être capable de le décrire plus précisément et éventuellement être capable de le rendre autonome. C'est ce que nous ferons dans les deux parties suivantes.

2.2 Définition de l'espace

Pour pouvoir écrire les équations physiques qui décrivent les mouvements d'un mobile, il est nécessaire de définir au minimum un référentiel. Pour simplifier notre travail au cours de cette thèse, nous allons en définir deux ainsi que les passages de l'un à l'autre.

2.2.1 Le référentiel inertiel

Par définition, un référentiel inertiel, ou galiléen, est un référentiel dans lequel le principe d'inertie est vérifié. Cela signifie que toute masse ponctuelle, sur laquelle la résultante des forces est nulle, est en mouvement de translation rectiligne uniforme. On utilise généralement un tel référentiel comme référentiel de base pour simplifier la description des mouvements des mobiles étudiés. Aucun référentiel n'est réellement galiléen, cependant, vu les caractéristiques des expériences réalisées (durées de quelques minutes et vitesses $<100\,\mathrm{m\,s^{-1}}$), on considérera durant cette thèse le référentiel terrestre comme étant suffisamment galiléen pour être considéré comme tel.

S'il est assez classique d'utiliser le référentiel terrestre lors des études sur les quadrirotors, différents repères cartésiens sont couramment utilisés pour décrire ce référentiel. La Figure 2.4

présente ces différents repères.

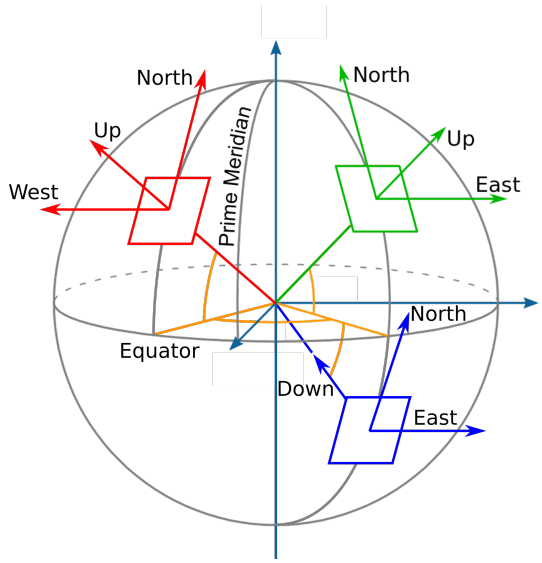


FIGURE 2.4 – Définition des repères cartésiens couramment utilisés dans le repère terrestre.

Le repère ENU [1] est le repère le plus fréquemment utilisé en aéronautique. Le repère NED [2] est celui qu'utilise la plupart des GPS. ROS [Qui+09], une suite de librairies informatiques dédiées à la robotique que l'on utilise et qui sera décrite dans la partie 3.10, utilise le repère NWU [3] qui sera aussi le repère utilisé, sauf indication contraire, au cours de cette thèse. On note $\mathbb{I}$ ce repère et i_1, i_2, i_3 les vecteurs directeurs unitaires associés respectivement au Nord, à l'Ouest et au Haut.

2.2.2 Le référentiel drone

Si le référentiel terrestre permet de décrire la trajectoire du quadrirotor d'une manière qui parait humainement naturelle, ça n'est pas le référentiel adapté pour décrire un certain nombre de forces et moments qui entrent en jeu dans le mouvement du drone. On pensera par exemple aux forces et moments issus des hélices qui sont à points d'applications et directions fixes dans n'importe quel repère lié au quadrirotor.

Pour exprimer ces forces plus simplement, on va donc définir le référentiel drone $\mathbb{B}$. Pour cela, on va noter W_1, W_2, W_3, W_4 les positions des quatre moteurs, C_g le centre de gravité du drone et C_{geo} le centre géométrique du drone. Les quatre moteurs étant, par construction des quadrirotors, dans le même plan, on définit C_{geo} comme l'intersection entre les droites (W_1W_3) et (W_2W_4). On note (b_1, b_2, b_3) les vecteurs directeurs unitaires de $\mathbb{B}$. L'origine de $\mathbb{B}$ est C_g, b_1 est portée par la bissectrice de $\angle W_1C_{geo}W_2$, b_2 est porté par la bissectrice de $\angle W_2C_{geo}W_3$ et b_3 est défini pour que le repère $\mathbb{B}$ soit orthogonal direct. La figure 2.5 illustre

1. East, North, Up ou est, nord, haut
2. North, East, Down ou nord, est, bas
3. North, West, Up ou nord, ouest, haut

toute ces définitions pour $C_g = C_{geo}$, configuration que l'on cherchera à atteindre lors de la construction du drone.

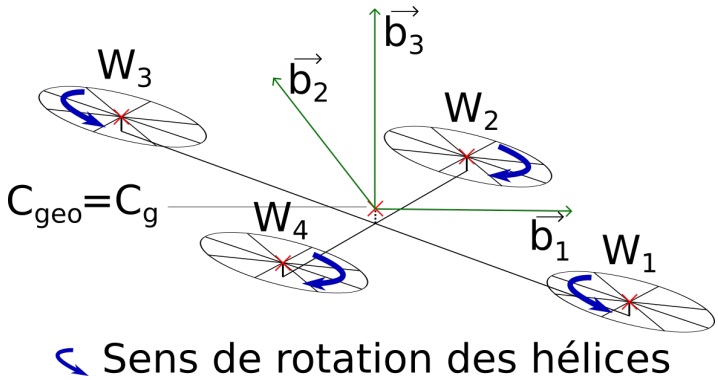


FIGURE 2.5 – Définition du repère $\mathbb{B}$ lié au drone, des centres géométriques et de gravité et numérotation des moteurs.

Note : $\mathbb{B}$ a été défini dans une orientation communément appelée **X**. Une deuxième orientation relativement classique est + où $b1$ est porté par la droite (W_1C_{geo}) et b_2 par la droite (W_2C_{geo}).

2.2.3 Définition des angles d'Euler utilisés

On définit maintenant la matrice de rotation $R \in SO(3)$ permettant de décrire l'orientation du repère lié au robot $\mathbb{B}$ dans le repère terrestre $\mathbb{I}$. On définit donc R tel que : $b_1 = Ri_1, b_2 = Ri_2, b_3 = Ri_3$.

Pour une plus grande lisibilité des rotations effectuées et parce que les contrôleurs classiques les utilisent, on choisit d'interpréter cette rotation sous forme d'angles d'Euler. Pour qu'une élévation nulle corresponde à une attitude horizontale, on choisit des angles d'Euler qui appartiennent à la catégorie des angles de Tait-Bryan. En particulier, on choisit le formalisme **Z-Y-X intrinsèque**. Du repère inertiel, on effectue d'abord une rotation ψ selon l'axe Z (correspondant à i_3). Cette rotation sera décrite par la matrice $R_Z(\psi)$ et le résultat sera porté par le repère intermédiaire $\mathbb{D}$ de vecteurs directeurs (d_1, d_2, d_3). Vient ensuite une rotation de θ selon l'axe Y (correspondant à d_2). Cette rotation sera décrite par la matrice $R_Y(\theta)$ et le résultat sera porté par le repère intermédiaire $\mathbb{C}$ de vecteurs directeurs (c_1, c_2, c_3). Finalement, on appliquera une rotation de ϕ selon l'axe X (correspondant à c_1). Cette rotation, décrite par la matrice $R_X(\phi)$ permettra d'obtenir le repère du quadrirotor $\mathbb{B}$ définit précédemment.

Ce procédé est illustré par la figure 2.6. Mathématiquement, cela donne :

$$R = Rz(\psi) * Ry(\theta) * Rx(\phi) \tag{2.1}$$

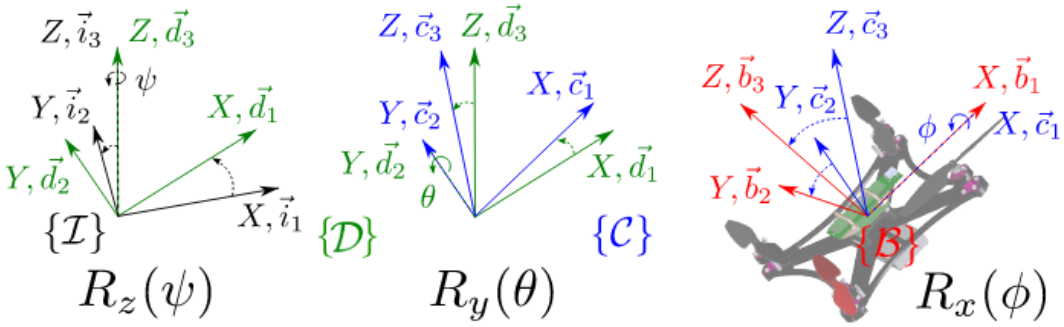


FIGURE 2.6 – Définition des angles d'Euler.

$$R = \begin{pmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 1 \\ -sin\theta & 0 & cos\theta \end{pmatrix} . \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{pmatrix} \tag{2.2}$$

$$R = \begin{pmatrix} cos\psi.cos\theta & cos\psi.sin\theta.sin\phi - sin\psi.cos\phi & cos\psi.sin\theta.cos\phi + sin\psi.sin\phi \\ sin\psi.cos\theta & sin\psi.sin\theta.sin\phi + cos\psi.cos\phi & sin\psi.sin\theta.cos\phi - cos\psi.sin\phi \\ -sin\theta & cos\theta.sin\phi & cos\theta.cos\phi \end{pmatrix} \tag{2.3}$$

2.3 Théorie quantifiée du vol

Ayant défini la totalité des outils nécessaires, il est maintenant possible de décrire grâce à la théorie physique le vol du quadrirotor.

Pour modéliser un quadrirotor, la méthode la plus courante est d'écrire les équations physiques d'un solide rigide à 6 degrés de liberté puis d'y ajouter différentes contributions qui modélisent les différences entre un quadrirotor et un solide rigide (effet des moteurs, de déformation des composants physiques, des frottements de l'air ...) . La modélisation de notre quadrotor sera donc :

$$\dot{\xi} = v \tag{2.4}$$

$$m\dot{v} = -mgi_3 + F_\Sigma \tag{2.5}$$

$$\dot{R} = R\Omega_\times \tag{2.6}$$

$$I\dot{\Omega} = -\Omega \times I\Omega + \tau_\Sigma \tag{2.7}$$

$\xi = (x, y, z)$ correspond à la position du centre de gravité du quadrirotor dans $\mathbb{I}$, $v = (v_x, v_y, v_z)$ à la vitesse du centre de gravité dans $\mathbb{I}$, R, présenté précédemment, est la matrice d'orientation du drone dans le repère inertiel, $\Omega = (p, q, r)$ le vecteur des vitesses de rotation de $\mathbb{B}$, I la matrice d'inertie du quadrirotor exprimée dans $\mathbb{B}$, m la masse du drone et g l'accélération de la pesanteur à la surface de la terre. Enfin, F_Σ et τ_Σ correspondent aux

contributions différenciant le solide rigide et un quadrirotor, l'étude de ces termes fera l'objet du reste de la partie, et $\Omega_\times$ est la matrice antisymétrique issue du vecteur Ω tel que, pour tout vecteur $A \in R^3, \quad A \times \Omega = A\Omega_\times$. On a donc :

$$\Omega_\times = \begin{pmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{pmatrix}$$

2.3.1 Modélisation simple des contributions des hélices

Les contributions les plus importantes à F_Σ et τ_Σ sont évidement celles issues de la rotation des hélices qui ont été présentées dans la section de vulgarisation (2.1). On note T la force de poussée et Q le couple créé par la traînée des hélices.

En ignorant les déformations des hélices, on a, pour chacune des hélices H_i :

$$T_i = C'_T.\rho.A_{r_i}.r_i^2.w_i^2 \cdot b_3 \tag{2.8}$$

$$Q_i = (-1)^{i+1}.C'_Q.\rho.A_{r_i}.r_i^3.w_i^2 \cdot b_3 \tag{2.9}$$

C'_T est un coefficient de poussée qui dépend de la géométrie de l'hélice, C'_Q est un coefficient de couple, ρ est la densité de l'air, A_{r_i} est l'aire balayée par l'hélice, r_i est le rayon de l'hélice et w_i est la vitesse de rotation de l'hélice.

En considérant ρ constant (faible variation d'altitude pendant les vols), tous les termes des équations 2.8 et 2.9, exceptés les vitesses de rotations et le signe, sont constants. Il est donc possible de définir la constante C_T telle que $C_T = C'_T.\rho.A_{r_i}.r_i^2$ de même pour $C_Q = C'_Q.\rho.A_{r_i}.r_i^3$

La contribution totale des hélices est évidemment la somme des contributions de toutes les hélices. Il faut simplement prendre en compte que les forces T_i ne sont pas appliquées au centre du quadrirotor mais à la jonction entre les moteurs et les hélices ce qui crée d'autres moments quand on considère le drone dans son ensemble. Pour exprimer ces moments, on rappelle que W_i est défini comme la position des hélices dans $\mathbb{B}$. On note Q_T le moment total créé par les hélices, on a alors :

$$T = \sum_{i=1}^{4} T_i = \sum_{i=1}^{4} (C_T.w_i^2) \cdot b_3 \tag{2.10}$$

$$Q_T = \sum_{i=1}^{4} (T_i \times W_i) + \sum_{i=1}^{4} Q_i \tag{2.11}$$

Avec une configuration en **X** définie précédemment et illustrée par la figure 2.5, h la hauteur du centre de gravité dans $\mathbb{B}$ et l la distance $C_{geo}W_1 = C_{geo}W_2 = C_{geo}W_3 = C_{geo}W_4$, on a :

$$\begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{pmatrix} = \begin{pmatrix} \frac{l\sqrt{2}}{2} & -\frac{l\sqrt{2}}{2} & h \\ \frac{l\sqrt{2}}{2} & \frac{l\sqrt{2}}{2} & h \\ \frac{-l\sqrt{2}}{2} & \frac{l\sqrt{2}}{2} & h \\ \frac{-l\sqrt{2}}{2} & -\frac{l\sqrt{2}}{2} & h \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{2.12}$$

En exprimant T_i, Q_i et W_i grâce aux équations 2.8, 2.9 et 2.12, on obtient :

$$Q_T = \begin{pmatrix} Q_x \\ Q_y \\ Q_z \end{pmatrix} = \begin{pmatrix} \frac{l\sqrt{2}}{2} * C_T(w_2^2 + w_3^2 - w_1^2 - w_4^2) \\ \frac{l\sqrt{2}}{2} * C_T(w_3^2 + w_4^2 - w_1^2 - w_2^2) \\ C_Q * (w_1^2 + w_3^2 - w_2^2 - w_4^2) \end{pmatrix} \tag{2.13}$$

En posant dans l'équation 2.5, $F_\Sigma = T$ et dans l'équation 2.7, $\tau_\Sigma = Q_T$, on obtient une première modélisation de notre drone. Cette modélisation ne prend en compte que les forces et moments principaux générés par les hélices. Il est intéressant de remarquer que dans cette modélisation, la hauteur du centre de gravité h n'influe pas sur la physique du quadrirotor.

Par ailleurs, on constate qu'il est possible de lier quadratiquement les forces et moments créés par les hélices aux vitesses rotations des hélices. On peut ainsi créer la matrice M qui lie les forces et moments directement aux vitesses de rotation des moteurs. Cette matrice, dite de mixage, peut être réécrite en fonction de la configuration géométrique du châssis, et permet d'appliquer facilement une commande composée d'une force et d'un moment à des drones configurés différemment. Pour un drone en **X** et en posant $l' = \frac{l\sqrt{2}}{2}$, la matrice M est la suivante :

$$\begin{pmatrix} T \\ Q_x \\ Q_y \\ Q_z \end{pmatrix} = M \cdot \begin{pmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{pmatrix} = \begin{pmatrix} C_T & C_t & C_t & C_t \\ -l'.C_T & l'.C_t & l'.C_t & -l'.C_t \\ -l'.C_T & -l'.C_t & l'.C_t & l'.C_t \\ C_Q & -C_Q & C_Q & -C_Q \end{pmatrix} \cdot \begin{pmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{pmatrix} \tag{2.14}$$

2.3.2 Modélisation plus fine

Dans la partie précédente, seule une modélisation simple des forces et moments créés par les hélices a été présentée. On va dans cette partie affiner ces modélisations, évoquer d'autres forces qui s'appliquent aux quadrirotors et présenter des références qui détaillent ces effets.

Commençons en reprenant la première équation de la partie précédente, l'équation 2.8 qui donne la poussée en fonction de la vitesse de rotation des hélices. Si la poussée est effectivement corrélée à la vitesse de rotation de l'hélice, elle dépend plus précisément de la vitesse de l'hélice par rapport à l'air qu'elle traverse. En conséquence, des variations de la vitesse du vent apparent, la différence entre la vitesse de l'air et la vitesse du quadrirotor, vont rendre inégales des poussées générées par des vitesses de rotation des hélices égales. Cet effet est présenté plus en détail par Huang et al. [Hua+09]. On notera qu'en présence de vent apparent, l'angle de lacet influe aussi sur la poussée générée.

Autre impact de ce vent apparent, l'effet de flapping. Lorsque l'hélice va à l'encontre du vent apparent cela augmente la différence de vitesse entre l'hélice et l'air et augmente donc la portance. Au contraire, lorsque l'hélice va dans le même sens que le vent apparent, sa portance en est réduite. Cette différence de portance va générer un effet balancier et les pales oscilleront

une fois par tour entre une position haute et une position basse. Cette oscillation des hélices modifie la direction de la poussée qui n'est alors plus portée par b_3.

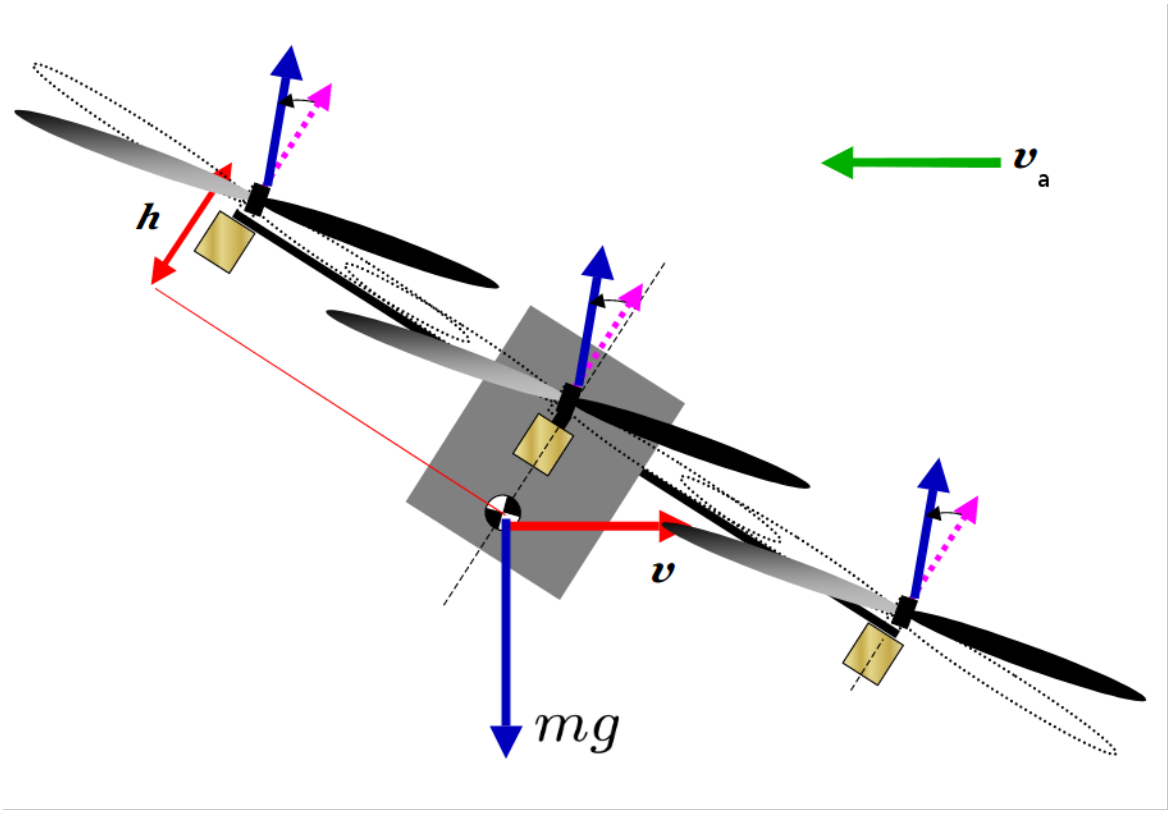


FIGURE 2.7 – Représentation schématique de l'effet de flapping.

La figure 2.7 illustre cet effet de flapping pour un vent apparent v_a. Les flèches violettes en pointillés indiquent la direction attendue de la poussée et les flèches en bleu la direction effective des différentes forces qui s'appliquent au drone.

Puisque cet effet de flapping modifie la direction de la poussée et non sa norme, il est possible de l'exprimer sous la forme d'une matrice de rotation R_{flap_i}. L'équation 2.8 devient alors :

$$T_i = \begin{pmatrix} T_{xi} \\ T_{yi} \\ T_{zi} \end{pmatrix} = R_{flap_i} \cdot C_T.w_i^2 \cdot b_3 \tag{2.15}$$

On remarquera que T_i n'est alors plus orienté selon b_3. Cet effet de flapping génère des mouvements de lacet parasites dont une partie dépend à présent de la hauteur du centre de gravité h (équation 2.11 avec T_i qui n'est plus orienté selon b_3).

Autre contribution influant sur le déplacement du quadrirotor, les frottements fluides. Ceux-ci peuvent être créés par le déplacement du quadrirotor, par la présence de vent ou par la somme de ces deux éléments (vent apparent). De nombreux travaux cherchent à estimer ces frottements fluides pour être, par la suite, capable de contrôler le drone en leurs

présences [SSP13] ; [MB04] ; [WW09]. Lorsque le vent est nul ou connu, il est possible de modéliser cette force plutôt que de l'estimer. Il faut alors prendre en compte la forme et la vitesse du quadrirotor [Man15].

Autres influences de l'environnement, les effets de sol et de plafond. Le premier de ces deux effets est relativement connu et de nombreux papiers l'évoquent et tentent de le modéliser [Man15] ; [BS07]. L'effet dû au plafond est lui beaucoup moins étudié. Il s'agit d'une augmentation de poussée à proximité des plafonds. Son principal inconvénient réside dans la nécessité d'être très proche des plafonds (<1 cm) ce qui oblige les quadrirotors l'utilisant à ajouter des éléments physiques pour se servir au maximum de cet effet et ce qui nullifie son impact sur les quadrirotors classiques. Cet effet peut par contre apporter une importante contribution à la poussée du drone ce qui permet d'envisager de doubler voir de tripler l'autonomie d'un drone qui utilise un tel effet [HC19].

Dernière modification que l'on évoquera par rapport au modèle simplifié présenté précédemment. On a précédemment évoqué un contrôle sur la poussée. Ce contrôle n'est cependant pas immédiat. Il passe par le contrôle des moteurs qui lui même est généralement assimilé à un système du premier ordre [BS07]. Cela engendre des retards de quelques milisecondes qu'il est intéressant de prendre en compte pour réaliser des commandes agressives réalisables par les quadrirotors.

Remarque. *Toutes les contributions présentées dans cette partie ne seront pas prises en compte lors de la construction des contrôleurs. Elles sont principalement présentées pour souligner que le modèle que l'on va utiliser lors du contrôle, le modèle simplifié, est plutôt imprécis ce qui engendrera nécessairement des erreurs de suivi de trajectoire qu'il faudra anticiper.*

2.3.3 Liens entre attitude, poussée et accélérations

En se replaçant dans le modèle simple et en utilisant l'équation 2.4, le principe fondamental de la dynamique (équation 2.5) et l'expression de la poussée (équation 2.10), il est possible d'écrire :

$$\ddot{\xi} = -gi_3 + \frac{\sum_{i=1}^{4}(C_T.w_i^2)}{m} \cdot b_3 \tag{2.16}$$

En se plaçant dans $\mathbb{I}$, en notant $\sum_{i=1}^{4}(C_T.w_i^2) = |T|$ et en utilisation l'expression de la matrice de rotation R (équation 2.3), on peut alors exprimer les accélérations en fonction de l'attitude du drone et de la poussée :

$$\begin{cases} \ddot{x} = (\sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi)\frac{|T|}{m} \\ \ddot{y} = (-\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi)\frac{|T|}{m} \\ \ddot{z} = -g + (\cos\theta\cos\phi)\frac{|T|}{m} \end{cases} \tag{2.17}$$

Cette information servira par la suite pour estimer les accélérations du drone et plus généralement sa position dans l'espace.

On aimerait aussi pouvoir inverser l'équation précédente. Cela nous permettrait de déduire la poussée et l'attitude nécessaire pour obtenir des accélérations spécifiques. S'il n'est pas possible d'inverser ces équations comme telles, les angles ϕ et θ ont été définis pour être nuls lorsque le drone est à l'horizontal, dans sa position d'équilibre. Lors des vols, ils seront donc la plupart du temps faibles. Pour contrôler le drone à faible incidence, on peut donc se permettre de linéariser les angles ϕ et θ. On le fait pour $\ddot{X}$ et $\ddot{Y}$ et on obtient :

$$\begin{cases} \ddot{x} \approx (\phi \sin\psi + \theta \cos\psi\ ) \frac{|T|}{m} \\ \ddot{y} \approx (-\phi \cos\psi + \theta \sin\psi\ ) \frac{|T|}{m} \\ \ddot{z} = -g + (\cos\theta \cos\phi) \frac{|T|}{m} \end{cases} \tag{2.18}$$

Ce qui nous permet finalement d'aboutir sur :

$$\begin{cases} |T| = \frac{m(\ddot{z}+g)}{\cos\theta \cos\phi} \\ \phi = \frac{m}{|t|} \cdot (\ddot{x} \sin\psi - \ddot{y} \cos\psi) \\ \theta = \frac{m}{|t|} \cdot (\ddot{x} \cos\psi - \ddot{y} \sin\psi) \end{cases} \tag{2.19}$$

En considérant le modèle simple et en linéarisant ϕ et θ, il est donc possible de convertir une accélération désirée en attitude désirée. Cela permettra de créer des contrôleurs en cascade comme présentés en 3.5.2. Il faudra simplement se rappeler que ces équations sont issues de linéarisations valables autour de l'attitude d'équilibre et que leur application lorsque l'attitude du drone s'en éloigne va générer de l'erreur dans cette conversion accélération/attitude.

2.4 Les quadrirotors en pratique

2.4.1 Législation et dangers liés aux quadrirotors

Cette thèse porte sur la sécurité des quadrirotors et autour des quadrirotors. Éloignons nous donc légèrement de la technique pour parler des dangers des quadrirotors et de la législation qui entoure ces engins volants dans le but de minimiser les dangers qu'ils causent.

Pour commencer, voici une liste non exhaustive des risques créés par l'utilisation de quadrirotors :

— risque de chute : Une panne en vol, qu'elle soit d'origine électrique, liée à un moteur, à un crash software, à un problème de communication ... va fréquemment conduire à une chute libre du quadrirotor. Si des travaux sont réalisés pour les pannes moteurs [Sun+18], il est difficile d'imaginer comment éviter une telle chute lors d'un problème d'alimentation électrique. Une telle chute peut bien évidemment provoquer des dégâts sur le drone lui même mais peut aussi blesser des personnes ou abîmer les structures sur lequel le drone va tomber.

— risque de collision : Il ne s'agit pas d'un risque négligeable. Premièrement, un quadrirotor représente un risque pour l'humain lors de vols à basse altitude. Même pour

de petits quadrirotors, les hélices, vu leurs vitesses de rotation, peuvent couper de la chair. Un ingénieur du laboratoire s'est, par exemple, entaillé 3 doigts en manipulant un drone de 500 g avec des hélices de 5 pouces. Plus important peut être, un quadrirotor représente un risque non négligeable s'il vient à heurter une partie sensible (moteurs ou hélices par exemple) d'avions transportant des personnes. A cause de ce risque, les deux aéroports de Londres ont, en décembre 2018 et janvier 2019, du annuler des vols ce qui a affecté plus de 140 000 passagers [dro19].

— risques liés à des utilisations frauduleuses : Grâce à leur grande mobilité, il est possible d'imaginer énormément d'actions frauduleuses avec des quadrirotor. Il est par exemple possible de transporter des matières interdites, faire de l'espionnage de sites interdits, brouiller des communications en se positionnant à des endroits clés, voir directement attaquer des cibles en larguant des charges explosives ou en s'écrasant volontairement.

Une législation a été mise en place à l'échelle nationale pour encadrer l'usage des drones. En considérant, toutes les utilisations frauduleuses listées précédemment après quelques minutes de réflexion, cette législation apparaît nécessaire. Comme souvent, cette législation apparaît contraignante pour les utilisateurs bien intentionnés mais elle permet de limiter les risques que peuvent, volontairement ou involontairement, représenter les quadrirotors. Les informations qui suivent proviennent de la DGAC [4] et plus précisément de la DSAC [5]. Le but ne sera pas de rentrer dans les détails de la législation mais de mettre en avant les grandes lignes de celle-ci. Pour plus d'information, il est recommandé de directement consulter les deux guides suivants : le *Guide DSAC, Aéromodélisme : modèles réduits et drones de loisirs* [civ18b] qui concerne, comme son nom l'indique, les drones de loisir ou le *Guide DSAC, Aéronefs circulant sans personne à bord : activités particulières* [civ18c] qui concerne les drones utilisés dans un cadre professionnel.

Comme l'organisation des guides permet de le supposer, la législation distingue différents régimes non pas en fonction des machines elles-mêmes mais des utilisations qui en sont faites. Lorsque cette utilisation est limitée au loisir (y compris la compétition), on parle d'*aéromodélisme*. Pour les utilisations autres que le loisir, que ces utilisations donnent lieu ou non à une transaction commerciale, on parle soit d' *activités particulières* soit d'*expérimentation*. La législation est nettement plus contraignante pour l'aéromodélisme que pour les autres utilisations mais les autorisations de vol sont beaucoup plus évidentes à obtenir voir automatiques dans la plupart des utilisations classiques.

Pour illustrer les limites fixées par la législation, en voici quelques unes pour l'aéromodélisme :

— Pas de vols dans de nombreuses zones (proximité immédiate des aéroports, zones de manœuvres d'aéronefs militaires, les agglomérations, les sites dit sensibles soit les hôpitaux, prisons, réserves naturelles, certains sites industriels...). Une carte, élaborée par la DGAC et disponible sur Géoportail [Fra19], référence les autorisations de vol en tout point du territoire français.

4. Direction Générale de l'Aviation Civile
5. Direction de la Sécurité de l'Aviation Civile

— Hauteurs limites de vol (souvent 150 m), heures limites de vol (pas de vol de nuit) et obligation de garder son aéronef en vue
— Pas de survol de personnes
— Enregistrement des tous les drones de plus de 800 g sur la plateforme en ligne AlphaTango
— Aucune utilisation commerciale directe ou de produits dérivés des vols (par exemple, pas de monétisation des prises de vues par drone de loisir)
— nécessité d'effectuer une formation de pilote pour tout aéronef de plus de 800 g (exigence minimum)

Pour ce qui est des *activités particulières* et *expérimentales*, la plupart des limites ci-dessus s'appliquent par défaut mais peuvent être nullifiées par l'acquisition d'autorisations spécifiques. L'obtention de ces autorisations demandera par contre de fournir un certain nombre de justificatifs supplémentaires et ajoute un certain nombre de contraintes administratives lors des vols et de leur archivage.

On notera que les expérimentations en cours de thèse sont considérées comme des *activités particulières* par la législation. On aurait pu s'attendre à ce qu'elles fassent parties de la catégorie législative des *expérimentations* mais ce n'est pas le cas car cette thèse se concentre sur l'ajout d'une fonctionnalité à un drone déjà fonctionnel et non à la construction d'un drone d'un nouveau genre. L'appartenance aux *activités particulières* implique notamment que le pilote soit titulaire d'un brevet de télépilote de drone ce qui a été la cause d'un retard important au cours de la thèse. Cela implique aussi que le drone doit être associé à un exploitant qui sera responsable des potentiels dommages qu'il causera. Le CNRS a choisi d'être l'exploitant de ses chercheurs mais en contrepartie demande des garanties qui vont au delà de celles définies dans la loi.

On s'arrêtera là sur la partie législative, le lecteur qui souhaite plus de détails sur la législation pour effectuer des expérimentations extérieures avec des drones en France est renvoyé vers le guide des activités particulières de la DSAC [civ18c] pour des informations exhaustives ou vers la synthèse qui en est extraite [civ18a].

2.4.2 Les composants essentiels

On va maintenant parler de la constitution **minimale** d'un drone. Cet état n'inclut que les composants et blocs logiciels nécessaires pour qu'un drone vole et soit manuellement pilotable par un amateur. On nommera ces composants et blocs logiciels les éléments **essentiels** du drone. On peut voir tout drone comme un drone **minimal** auquel certaines fonctionnalités ont été ajoutées. Ces fonctionnalités pourront donner lieu à des ajouts logiciels (contrôle en position, suivi de trajectoire, suivi de cible ...) ou à des ajouts matériels (prises de vues, mesure de grandeurs physiques ...). Le plus souvent, pour proposer une nouvelle fonctionnalité, il faudra à la fois un ajout matériel et un ajout logiciel.

Commençons la liste des éléments essentiels d'un drone par ses composants physiques. Les trois premiers composants sont des évidences. Pour construire, un quadrirotor, il y a besoin

d'un châssis, de quatre moteurs et de quatre hélices. On prendra simplement soin de choisir un châssis léger, rigide et qui permet de placer le centre de gravité du drone très proche du centre géométrique du drone. De leur coté, les moteurs sont généralement sans balais (brushless). Ces moteurs sont privilégiés car ils ont un meilleur rendement que les moteurs à courant continu classique et que l'apport énergétique est un des problèmes majeurs pour les drones. Enfin, les hélices doivent avoir un profil optimisant leur efficacité c'est à dire maximisant la poussée par rapport à la traînée. Il est intéressant de noter que l'efficacité des hélices de quadrirotor est plus faible d'environ 10% que l'efficacité des hélices de plus grandes tailles utilisées historiquement en aéronautique. Brandt et Selig propose un comparatif de 79 hélices mesurant entre 9 et 11 pouces [BS11] et leur étude fournit de multiples références sur la comparaison avec des hélices de plus grande taille. Pour compenser le poids et pouvoir décoller, il faut que la poussée maximale que puisse exercer chaque hélice soit au minimum du quart du poids en charge du drone. On préfère cependant généralement dimensionner les blocs moteurs et hélices de sorte à ce qu'il soit possible d'atteindre la moitié du poids du drone. En additionnant les forces de poussée de chacun des blocs moteurs et hélices, il est alors possible de générer une force de poussée équivalente à deux fois le poids. Une force d'une telle magnitude permet de voler à altitude constante avec des accélérations latérales de l'ordre de 1g ce qui confère au quadrirotor la maniabilité qui fait sa force.

À ces trois éléments, il faut ajouter cinq autres éléments pour obtenir un drone fonctionnel. On va commencer par évoquer le contrôleur de vol qui est un micro-contrôleur programmable sur lequel s'exécutera le logiciel embarqué ou *firmware* qui contient les blocs logiciels nécessaires pour voler. Ce micro-contrôleur contient généralement plusieurs capteurs. Parmi ceux-ci, il y a, au minimum, le deuxième élément nécessaire au vol manuel : une centrale inertielle ou *Inertial Measurement Unit* (IMU). Ce capteur composé de trois accéléromètres et gyroscopes permet d'estimer l'attitude du drone ainsi que ses vitesses angulaires, des états qu'il est nécessaire d'estimer pour voler manuellement. Troisième élément nécessaire mais sur lequel il y a peu à dire : un récepteur radio pour pouvoir communiquer avec le drone. Passons donc directement à l'élément suivant , quasi nécessaire, les variateurs de vitesse électronique ou *Electronic speed control* (ESC) qui servent à réguler les vitesses de rotation des moteurs brushless. Leur présence permet de s'assurer qu'une commande en vitesse moteur résultera bien en une sortie à peu près équivalente. L'absence d'ESC revient à commander les moteurs en boucle ouverte et résulte en de fortes variations des vitesses moteurs réelles comparées aux vitesses commandées. Ces fortes variations s'expliquent notamment par la chute en tension au fur et à mesure de la décharge de la batterie qu'il faut compenser par de plus fortes intensités pour transmettre la même énergie et conserver la même vitesse moteur en sortie. On arrive donc naturellement au dernier élément, celui qui alimente le drone en énergie, la batterie. Parce qu'elles permettent de fort courant de décharge tout en ayant une forte densité énergétique (rapport entre puissance stockée et masse), les batteries lithium-ion polymère (Li-Po) sont privilégiées sur les quadrirotors. Depuis 2018, il est aussi possible de trouver des batteries graphènes qui possèdent des performances comparables tout en promettant un cycle de vie plus long et un temps de recharge plus rapide [Son+17]. On notera qu'il y a, sur la plupart des drones, deux circuits électriques distincts. L'un, que l'on appellera le circuit de puissance, relie la batterie, les ESC et les moteurs. Sur des drones d'une cinquantaine de centimètres de diamètre, jusqu'à 100 A pour 14.8 V peuvent transiter dans ce circuit. Le deuxième circuit

est généralement à 5 V et ne transite dans ce circuit que quelques ampères au maximum. Ce circuit alimente généralement le contrôleur de vol ainsi que la totalité des capteurs embarqués.

Voici donc la totalité des éléments nécessaires pour créer un drone fonctionnel. On finira cette présentation des éléments physiques en signalant qu'il existe des outils en ligne pour aider à dimensionner correctement les éléments physiques en fonction des caractéristiques du quadrirotor que l'on souhaite construire. Parmi tous ces outils, ecalc.ch [eca19] semble faire référence.

2.4.3 Firmwares

On a maintenant un ensemble d'éléments physiques suffisant pour obtenir un quadrirotor fonctionnel. Il manque cependant quelques fonctionnalités clés à ces éléments physiques pour former un quadrirotor comme par exemple la capacité à communiquer entre eux. Toutes ces fonctionnalités sont réalisées par un logiciel embarqué ou **firmware**.

Pour présenter ces firmwares, le schéma des différents blocs logiciels de ROSflight [JK16] est mis en avant en figure 2.8. ROSflight est un des firmwares libres existant. L'étude de son architecture permet facilement d'identifier les blocs logiciels nécessaires dans un firmware de drone car ROSflight a pour but d'être minimal tout en assurant la connexion avec ROS [Qui+09], une suite de librairies et d'outils dédiés à la robotique. Sur le schéma du firmware, cela se traduit par le fait que tous les blocs sont nécessaires exceptés le bloc Mavlink ansi que les deux blocs qui n'ont qu'une connexion à ce bloc (State Manager et Parameter Server).

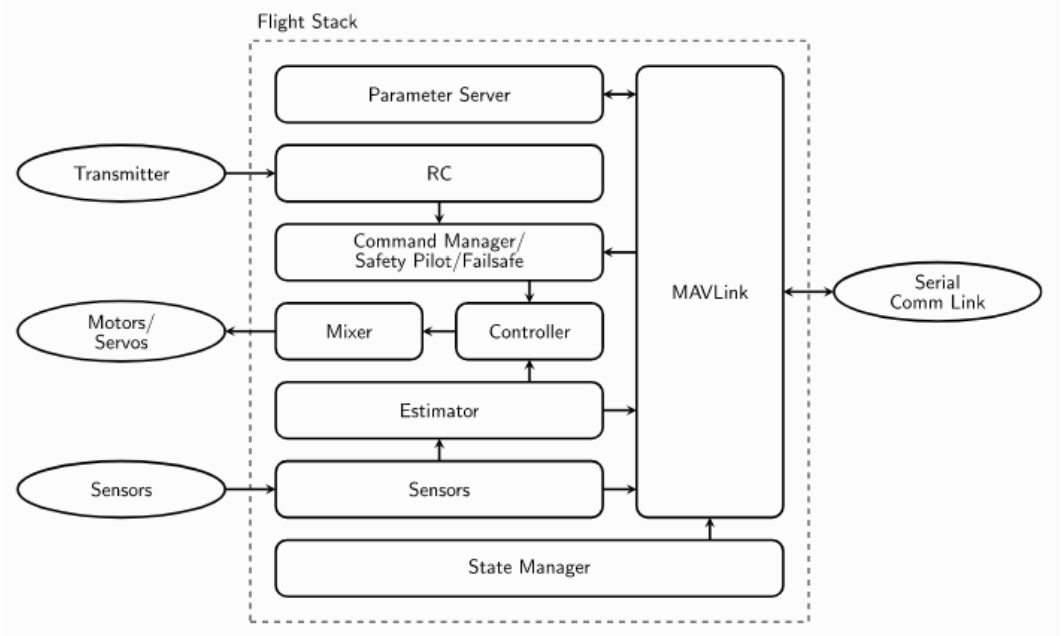


FIGURE 2.8 – Schéma des différents blocs logiciels de ROSflight.

Parmi les blocs nécessaires, on voit un contrôleur ainsi qu'un estimateur. On reviendra de ces deux blocs plus en détails dans quelques instants. Les autres blocs comprennent le mixer (l'implémentation de plusieurs matrices de mixage pour gérer plusieurs configurations de multirotors) ainsi que la gestion de l'armement pour sécuriser le drone en cas de problème. Les derniers blocs correspondent tous au même élément clé des firmwares, la communication entre les différents éléments. Il faudra au minimum assurer la communication RC pour que les pilotes puissent envoyer des commandes au quadrirotor, la communication avec les ESCs pour envoyer les commandes en sortie de firmware ainsi que les communications avec les capteurs. Au delà de ces communications qui sont strictement nécessaires, il peut être intéressant de communiquer avec plus d'éléments pour obtenir des informations supplémentaires. La lecture de la tension batterie est ainsi une information cruciale pour éviter les crashs en fin de vol à cause d'un manque de puissance disponible pour les moteurs.

Il existe de multiples firmwares. Certains, comme ROSflight, Cleanflight ou Betaflight visent à gérer uniquement les blocs logiciels essentiels à du vol manuel ou à ajouter une unique fonctionnalité sur ces blocs (dans le cas de ROSflight). Ces firmwares sont généralement utilisés par les passionnés du pilotage et du vol immersif (FPV).

D'autres auto-pilotes tel que Ardupilot, Inav ou le PX4 autopilot, proposent des fonctionnalités de vol plus avancées. Tous ces auto-pilotes gèrent par exemple la navigation entre points de passage par GPS. Ces fonctionnalités avancées génèrent de nouveaux problèmes et au cours des dernières années, ces auto-pilotes ont commencés à intégrer des algorithmes d'évitement d'obstacles. Le PX4 auto-pilote intègre ainsi un bloc d'évitement d'obstacles qui déroute le drone pour qu'il atteigne tout de même son objectif si des obstacles sont détectés sur la trajectoire initiale (hors ligne, s'exécutant sur un PC compagnon au travers de ROS, voir [Dro18]). Ardupilot, de son coté, propose un arrêt en cas de détection d'obstacles sur la trajectoire disponible directement en embarqué [Ard17]. On notera que les utilisateurs professionnels de drone utilisent le plus souvent ce type d'auto-pilotes. Ils se servent en effet de leurs fonctionnalités avancées pour simplifier, en interne, leur gestion des drones.

Finalement, les compagnies qui vendent des drones grand public tel que Parrot ou DJI possèdent aussi leurs propres auto-pilotes et proposent des fonctionnalités de haut niveau plus ludiques. Parrot offre par exemple la possible de contrôler son drone grâce à son smartphone.

2.4.4 Contrôle

La manière la plus directe de piloter un quadrirotor serait de piloter les vitesses moteurs. Cela n'est cependant pas possible pour deux raisons. D'une part, un quadrirotor est par nature instable et le stabiliser demande des actions à haute fréquence (100 Hz au minimum pour le contrôle des moteurs). D'autre part, il est humainement très compliqué de percevoir quel va être l'impact d'un changement de vitesse moteur sur le comportement du drone. Il est donc nécessaire d'ajouter une couche de contrôle automatique pour qu'il soit humainement possible de piloter un drone.

Pour avoir un contrôle relativement simple, le plus évident est de contrôler automatique-

ment le quadrirotor en attitude. Sans action humaine, le contrôle aura ainsi pour but d'obtenir des angles de tangage et de roulis nuls ce qui aura pour effet de stabiliser le drone. Le pilote pourra ensuite envoyer des commandes en attitude et en poussée et puisqu'on a précédemment vu qu'une attitude et une poussée étaient équivalentes à des accélérations dans l'espace (équation 2.19), le pilote pourra ainsi contrôler les déplacements du drone dans l'espace. Le contrôle en attitude est le mode de contrôle le plus proche possible des actionneurs qui permet de stabiliser automatiquement un quadrirotor dans l'espace. Il offre en prime la possibilité de contrôler assez intuitivement les déplacements dans l'espace. Pour toutes ces raisons, ce mode de contrôle s'est imposé comme le mode d'utilisation par défaut des quadrirotors.

Remarque. *On notera que d'autres modes de contrôle sont possibles. Pour du contrôle automatique, on parlera dans le chapitre suivant de suivi de trajectoire mais même pour du contrôle manuel d'autres modes de contrôle sont proposés aux pilotes. Les constructeurs qui proposent des jouets pour enfants offrent généralement des modes de contrôle plus assistés tel que du contrôle direct en position ou du contrôle à partir de mouvements du corps. Au contraire, les pilotes chevronnés cherchent à réduire l'impact du contrôle automatique et à se rapprocher au plus des actionneurs. Pour cela, ils renoncent à la stabilisation automatique et contrôlent leurs quadrirotors en vitesses angulaires.*

Voici, en figure 2.9, le schéma de contrôle en attitude le plus classique pour un quadrirotor.

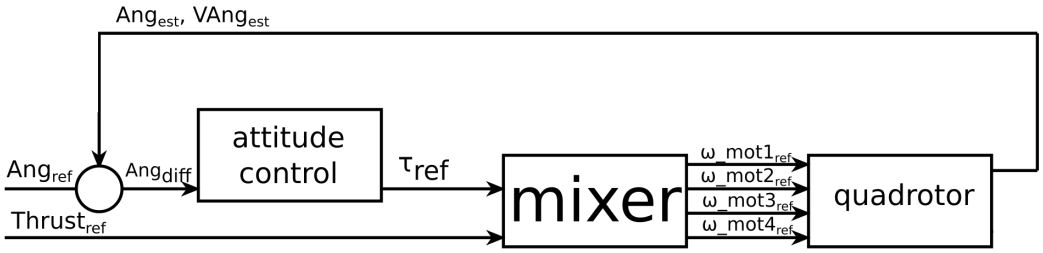


FIGURE 2.9 – Schéma d'un contrôleur en attitude.

On peut noter qu'il s'agit d'un contrôle en boucle fermée et que l'état nécessaire pour assurer ce contrôle en attitude est constitué de l'attitude du drone et de ses dérivées. On notera aussi la présence de la matrice de mixage en fin de chaîne. Celle-ci a été présentée avec l'équation 2.14 et permet de conserver le même schéma de contrôle (voir les mêmes paramètres de contrôle) pour différents multirotors.

Le contrôleur en lui même est dans la plupart des firmwares un régulateur PID ou plutôt un régulateur PID par angle qui compose l'attitude (ϕ, θ, ψ). C'est en particulier le cas dans tous les firmwares cités précédemment. Le régulateur utilise en entrée l'angle et la vitesse angulaire correspondante et propose en sortie une valeur qui sera équivalente à un moment. Les sorties des trois PIDs et la commande de poussée constitueront ensuite l'entrée de la matrice de mixage (d'où l'équivalence avec un moment). Les régulateurs PIDs sont notamment appréciés pour leur robustesse au bruit sur l'état dont ils disposent pour effectuer leur contrôle. Leurs

bonnes performances avec un même réglage en présence de changements physiques du drone permet aussi aux firmwares de pré-régler leurs PID et d'affirmer que ceux-ci fonctionne avec n'importe quel drone.

Il est intéressant de noter que l'utilisation d'un PID associé à chaque angle qui donne une force par axe dans le repère local (repère $\mathbb{B}$) suppose que chaque axe est indépendant. Or ceci ne correspond pas fidèlement à la réalité d'où une imprécision du contrôle angulaire. Pour pallier à ces imprécisions, de nombreuses solutions ont été proposées tel que des régulateur quadratique linéaire (LQR) [Arg+13], des contrôles adaptatifs [Bou+11] ou encore des contrôleurs utilisant les quaternions [LWZ15] ; [BHD13]. La plupart des solutions proposées permettent effectivement d'obtenir un meilleur contrôle qu'avec un PID mais elles sont moins pratiques à mettre en place et moins généralisables à un ensemble de conditions et un ensemble de quadrirotors d'où l'utilisation encore massive de PIDs basés sur les angles d'Euler dans la plupart des firmwares.

2.4.5 Observateur

Contrôler le drone requiert de connaître son état à chaque instant. Si cela semble évident, il s'agit pourtant d'une des contraintes les plus importantes en robotique. Lorsque cela est possible, des capteurs sont responsables de récupérer des informations pour obtenir l'état du drone. Ces capteurs ne sont cependant pas parfaits et il faudra alors estimer l'état réel du drone à partir des données bruitées des capteurs. Lorsqu'il est impossible de mesurer une grandeur, on l'estimera de manière indirecte en utilisant nos connaissances sur le robot et les mesures sur des grandeurs liées.

Dans le paragraphe sur le contrôle, on a vu que pour contrôler l'angle du drone, on se sert d'un état constitué des angles et des vitesses angulaires. On cherche à obtenir cet état à partir des accéléromètres et gyroscopes présents sur le drone. Ces capteurs ne mesurent pas directement l'attitude du drone et il faudra donc l'estimer en combinant nos connaissances sur ce dernier et les mesures des capteurs.

Il existe deux manières principales d'obtenir l'état désiré à partir des mesures. ROSflight utilise un filtre complémentaire comme proposé par Mahony, Hamel et Pflimlin[MHP08] en utilisant des suggestions de Casey et al.[Cas+13]. L'idée principale du filtre complémentaire est de privilégier les données d'un certain type de capteur en fonction de ses performances. On sait que les accéléromètres donnent des informations fiables dans des conditions statiques, on va donc privilégier les informations des accéléromètres pour les variations lentes. Au contraire, on sait que les gyroscopes ont un faible bruit lors de variations rapides, on va donc les privilégier dans les situations à dynamique forte. Pour cela, on va appliquer un filtre passe-bas aux accéléromètres, un filtre passe-haut aux gyroscopes puis combiner les deux informations. Dans l'implémentation de ROSflight, leur filtre complémentaire est précédé de filtres passe-bas (de fréquence de coupure différentes) sur chacune des mesures comme il est possible de le constater sur le schéma 2.10 de leur estimateur.

L'autre outil principalement utilisé pour estimer l'état du drone est le filtre de Kal-

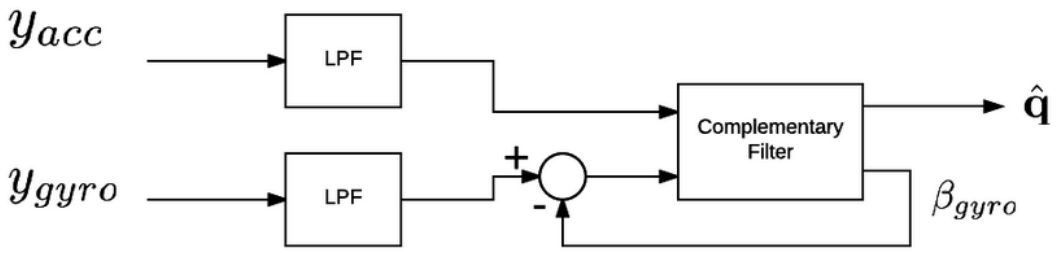


FIGURE 2.10 – Schéma de l'estimateur d'attitude de ROSflight.

man étendu (EKF). Il s'agit de l'outil le plus couramment utilisé pour fusionner des données provenant de multiples capteurs pour des systèmes non linéaires. C'est ce type d'estimateur qu'utilisent les auto-pilotes Ardupilot et PX4. En quelques mots, le filtre de Kalman étendu utilise à la fois des connaissances sur la dynamique du système et les observations issues des capteurs. La première étape du filtre, dite de prédiction, consiste à utiliser les connaissances sur la dynamique du système observé, l'état précédent du système et les commandes données au système pour estimer l'état courant du système. Au cours de cette première étape, une covariance sur la prédiction sera aussi calculée. Vient alors la seconde étape, dite de mise à jour, qui va modifier l'état prédit grâce aux observations. L'impact de chaque observation va dépendre de la matrice de covariance du capteur dont provient l'observation. Plus la source est fiable, plus l'information sera prise en compte. Finalement, un gain, dit de Kalman, sera mis à jour et servira à donner plus d'importance à la prédiction ou aux mesures en fonction des covariances de chacun. Conséquence de ses très nombreuses utilisations, de nombreux papiers détaillent l'utilisation des filtres de Kalman. Pour l'application à des quadrirotors, le mémoire de master de ASCORTI[Asc13] a l'avantage de décrire précisément à la fois le filtre et le système.

2.4.6 Besoins supplémentaires pour fonctionnalités avancées

Le setup décrit précédemment permet à un pilote de faire voler un quadrirotor. Il n'est cependant pas suffisant pour du pilotage automatique puisqu'il manque au drone des capteurs lui permettant de se repérer dans l'espace.

Pour des fonctionnalités basiques tel du suivi de points de passage sans évitement d'obstacles, la liste suivante présente les composants qui sont habituellement ajoutés :

- un magnétomètre pour obtenir un angle de lacet absolu par rapport au champs terrestre et non plus uniquement une estimation de la vitesse de lacet
- un gps pour obtenir une position absolue dans l'espace
- un baromètre pour affiner l'estimation de l'altitude ou un lidar pointant vers le sol qui permet d'obtenir la même mesure plus précisément
- une camera pointant vers le sol pour estimer les déplacements grâces à des algorithmes comme SVO [FPS14]

Lorsque l'on souhaite faire de la navigation autonome en toute sécurité, comprenant donc de

l'évitement d'obstacles, on ajoute :

- une carte de calcul supplémentaire (FPGA, GPU...)
- le lidar orienté vers le sol déjà listé précédemment devient presque obligatoire
- des capteurs de profondeurs (lidar, cameras RGBD, cameras de stero-vision ou simples cameras) au minimum pointant vers l'avant du drone

Au niveau logiciel, l'idée est généralement d'étendre l'état estimé et d'ajouter à l'attitude et aux vitesses angulaires, les positions, vitesses et accélérations linéraires dans un référentiel terrestre. Cette estimation d'état qui requiert une fusion des données de tous les capteurs cités précédemment est généralement réalisée grâce à un EKF.

Finalement, il est possible d'ajouter aux quadrirotors des éléments spécifiques pour réaliser des taches particulières. On peut ici penser à l'ajout de bras articulés qui occupe plusieurs équipes de recherches comme par exemple [KCK13], à l'ajout de capteurs pour étudier la qualité de l'air ou à l'ajout d'éléments protecteur pour se servir de l'effet de plafond [HC19].

Chapitre 3

Les algorithmes d'évitement d'obstacles

Dans ce chapitre, un certain nombre d'algorithmes d'évitement d'obstacles vont être présentés. La première partie 3.1 sera l'occasion de présenter rapidement un historique de l'évitement d'obstacles pour drones puis de se concentrer sur les méthodes proposées pour l'évitement d'obstacles dynamiques. On verra notamment dans cette partie qu'il y a plusieurs manières de poser le problème de l'évitement d'obstacles pour quadrirotors. Ce constat nous incitera à préciser notre conception de l'évitement d'obstacles dans la partie suivante. On prendra aussi soin dans cette partie 3.2 de détailler la répartition des tâches nécessaires pour aboutir à un quadrirotor autonome.

On se concentrera par la suite sur quelques fonctionnalités logicielles essentielles à l'évitement d'obstacles. Dans les parties 3.3, 3.4 et 3.5, on détaillera donc la représentation de l'environnement, la génération de trajectoires et les méthodes de contrôle utilisées. Pour chacune de ces fonctionnalités, on commencera par étudier l'état de l'art, notamment en essayant de classifier les alternatives principales pour clarifier les choix possibles. Une fois ces études réalisées, on présentera les choix effectués dans les algorithmes développés au cours de la thèse.

Une fois les études sur ces fonctionnalités terminées, on présentera deux algorithmes d'évitement d'obstacles développés durant la thèse ainsi que deux extensions à l'un de ces algorithmes. Le premier, décrit en 3.6, combine des éléments au niveau de l'état de l'art sans concept réellement novateur. En 3.7, le deuxième algorithme utilise une stratégie de génération et de sélection de trajectoires inédite. Finalement, deux extensions de ce deuxième algorithme, la gestion des minimums locaux et des obstacles dynamiques, seront présentées respectivement en 3.8 et en 3.9.

On conclura cette partie en discutant de quelques choix techniques qui ont été fait pendant le développement de ces algorithmes.

3.1 Revue d'algorithmes d'évitement d'obstacles pour drones

L'évitement d'obstacles lors de vols de drones est un sujet qui est traité depuis plus de quinze ans comme en témoigne la revue de l'état de l'art [GKM10] qui date de 2010. Le lecteur est invité à la consulter s'il souhaite plus d'informations sur la littérature originelle de l'évite-

ment d'obstacles en vol. Pour tout de même citer quelques résultats, en 2005 déjà, Zufferey et Floreano ont effectué 300 m de vols autonomes dans un environnement contrôlé avec un drone de 30 g [ZF05]. En 2007, Scherer et al. [Sch+08] effectuaient des vols en extérieur à $6\,\mathrm{m\,s^{-1}}$ *près* des obstacles avec un hélicoptère télécommandé originalement dédié à l'épandage. Dans ce papier, Scherer et al. font déjà la différence entre deux grandes catégories d'algorithmes. Ils différencient ce qu'ils appellent les méthodes de planifications aux méthodes réactives. Ils définissent les méthodes de planification comme des méthodes utilisant une carte de l'environnement, prévoyant une trajectoire réalisable pour le drone et gérant le suivi de ces trajectoires grâce à un contrôle dédié. Scherer et al. opposent à ces méthodes, les méthodes réactives qui, disent-ils, résolvent le problème temps réel qu'est l'évitement d'obstacles en utilisant une formule simple permettant de *réagir* aux obstacles quand ils apparaissent. Ils précisent que ces méthodes ne permettent pas de garantir une solution appropriée à toute situation.

Ces deux définitions replacent l'évitement d'obstacles dans un contexte plus général, la navigation autonome. Dans ce contexte, il est possible de considérer la génération de trajectoire comme la somme de trois fonctionnalités.

La première étape consiste à planifier un trajet très général. Son rôle est de définir une ébauche de la trajectoire qui sera ensuite suivie par le drone. Cette étape peut être très diverse. Elle peut à la fois définir une suite de couloirs de vols pour des vols de centaines de kilomètres comme être un algorithme de résolution de labyrinthe si la situation l'impose. Selon le problème, de nombreux algorithmes de planification de trajectoires relativement classiques peuvent être adaptés pour les quadrirotors. On peut par exemple citer les méthodes de recherches aléatoire dans des arbres (RRTs) [LaV98] ou les méthodes à base de recherche dans des graphes (a*) [HNR68]. Le point commun qui lie ces algorithmes aux sorties très diverses est qu'ils ne prennent généralement pas ou peu en compte la faisabilité des trajectoires réalisées. On notera de plus que toutes ces planifications, globales et souvent réalisées avant le vol depuis des cartes pré-enregistrées, ont un risque non négligeable de générer des trajectoires qui passent dans des obstacles non connus au moment de la planification.

Vient ensuite la seconde étape, son rôle principal est de corriger localement le trajet généré au cours de la première étape. On appellera cette étape replanification (re-planning). Cette étape est soit réalisée hors ligne, avant le vol, et ne se concentre que sur la faisabilité de la trajectoire [RBR16] soit partiellement ou totalement en vol ce qui permet en plus d'éviter d'éventuels obstacles non prévus lors de la planification initiale. Les algorithmes développés, comme par exemple [Zha+18], rentrent alors dans le cadre de l'évitement d'obstacles. Cette étape correspond à la *méthode de planification* que définissait Scherer et al.

Une dernière étape, facultative si l'étape précédente est effectuée en vol, consiste, forcément en cours de vol, à modifier, dans l'urgence, le vol du drone pour pallier à un évènement inattendu. Cette étape correspond à l'évitement réactif défini par Scherer et al. C'est par exemple ce type d'algorithme qui a été défini par Oleynikova, Honegger et Pollefeys dans un des papiers les plus populaires en évitement d'obstacles en 2015 [OHP15]. Les auteurs expliquent eux même en conclusion que leur algorithme peut être utilisé comme une couche robuste supplémentaire qui peut être ajouté à un système complexe.

Étudier l'état de l'art permet donc de constater que sous l'unique appellation *évitement d'obstacles* se cachent des réalités très différentes. Cette diversité n'est pas sans poser des problèmes, notamment sur les fonctionnalités qui font parties ou non de l'évitement d'obstacles. Par exemple, le contrôle fait-il partie de l'évitement d'obstacles ou faut-il le considérer comme la fonctionnalité logicielle venant après un algorithme d'évitement d'obstacles ? Si on considère les méthodes de planification, qui définissent toutes une trajectoire, il pourrait en effet être intéressant de chercher à comparer ces méthodes pour un même contrôle. Cela permettrait d'étudier les performances de chaque méthode indépendamment des performances atteintes en suivi de trajectoire. Pour les méthodes réactives, cela n'aurait par contre aucun sens puisque ces méthodes proposent justement des contrôles spécifiques des drones permettant de répondre le plus rapidement possible à la détection d'un obstacle. A défaut de savoir où mettre les limites, toutes les fonctionnalités qui jouent un rôle dans l'évitement d'obstacles peuvent se revendiquer de l'évitement d'obstacles et il est attendu des papiers présentant des **algorithmes d'évitement d'obstacles** de présenter la totalité de ces fonctionnalités. Ainsi, l'estimation d'état, la représentation de l'environnement, la génération de trajectoire s'il y en a une et le contrôle sont considérés dans la littérature comme partie intégrante des algorithmes d'évitement d'obstacles. Une architecture comprenant les entrées/sorties et les blocs logiciels généralement considérées comme faisant partie d'un algorithme d'évitement d'obstacles basé sur une méthode de planification est présentée dans la figure 3.1. Tout ce qui est en pointillé dans cette architecture est optionnel. Il est en effet possible de ne générer qu'une seule trajectoire ce qui supprime le besoin de sélection ou de ne pas en générer et de se servir d'une librairie pré-calculée.

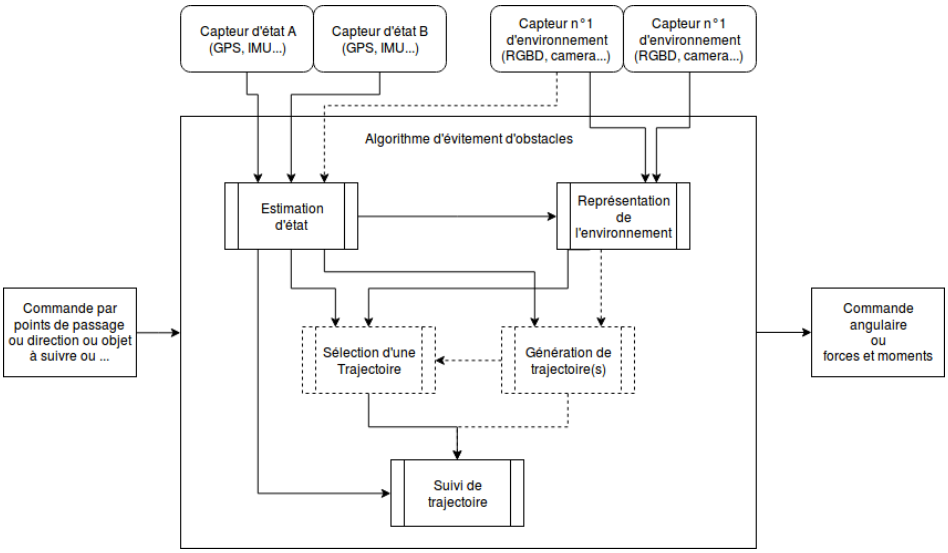


FIGURE 3.1 – Fonctionnalités principales et entrées et sorties des algorithmes d'évitement d'obstacles.

Vu la diversité du domaine, proposer un état de l'art exhaustif dans cette thèse n'est pas envisageable. Dans la suite et dans des parties dédiées, on dressera un état de l'art de certaines des fonctionnalités présentées plus tôt. Dans cette partie, on va se concentrer sur

l'étude d'une catégorie spécifique de papiers, ceux qui se préoccupe de l'évitement d'obstacles dynamiques pour quadrirotor.

Deux courtes parenthèses avant de se limiter à l'évitement dynamique. Premièrement, Tang et Kumar dans **Autonomous Flight** [TK18] proposent début 2018 une revue des vols autonomes pour quadrirotors. Eux aussi ont fait le choix de mettre en avant les différentes fonctionnalités mises en avant auparavant et sur lesquelles on proposera un état de l'art spécifique. Ils proposent par ailleurs une partie sur les différentes applications tel que la capacité à interagir avec l'environnement notamment avec des bras articulés et une partie sur la navigation multi-drone. Finalement, ils consacrent une partie à une méthode de plus en plus utilisée pour résoudre de nombreux problèmes : l'apprentissage automatique. Deuxièmement, l'algorithme le plus performant pour effectuer de l'évitement d'obstacles semble pour le moment être l'algorithme de la société Skydio sur lequel très peu d'informations sont disponibles. La société annonce dans son discours promotionnel des vols à 20km/h en environnement inconnu.

Concentrons nous donc à présent sur l'évitement dynamique. Alors que de plus en plus d'algorithmes permettent d'éviter des obstacles statiques, l'évitement d'obstacles dynamiques est une extension relativement évidente. Elle est en effet nécessaire pour garantir un vol sans collision dans la plupart des environnements réels (personnes en ville, branches en forêts, engins dans les champs, autres aéronefs et oiseaux...). Ajouter du mouvement à l'environnement pose de très nombreux problèmes. Certains de ces problèmes sont relativement évidents. Il est, par exemple, nécessaire d'anticiper le mouvement et d'estimer la position future des objets en mouvement pour pouvoir les éviter. D'autres problèmes sont plus inattendus. On peut par exemple signaler que les mouvements d'éléments de l'environnement vont ainsi dégrader les performances de la plupart des algorithmes de SLAM [1] que ce soit sur la capacité à cartographier l'environnement ou à s'y localiser.

Remarque. *L'évitement d'obstacles dynamiques n'est pas synonyme de navigation dans un environnement dynamique. Dans [Kau+18],* Kaufmann *et al. font varier la position d'éléments qui représente des portes à franchir. Si leurs résultats excèdent très largement les résultats sur de l'évitement d'obstacles dynamiques, c'est précisément parce que leur problème est très différent. Ils réalisent en fait du tracking de leurs portes et génèrent des trajectoires qui les traversent lorsque cela est possible. Il ne détaillent aucunement ce qu'il se passe si l'une de leurs portes, ou si un élément quelconque de l'environnement, se déplace en direction du drone et risque d'engendrer une collision avec celui-ci alors qu'il s'agit de la problématique principale à laquelle les algorithmes d'évitement d'obstacles dynamiques essayent de répondre. On notera aussi que l'évitement entre membres d'un essaim ne rentre pas dans l'étude puisque l'effort d'évitement devient alors coopératif ce qui modifie le problème.*

L'analyse des différents papiers traitant du sujet suggère une séparation des études dans deux directions qui, à terme, devraient se regrouper. D'une part, certaines études se concentrent sur l'évitement d'un unique obstacle qui peut avoir une grande vélocité dans un monde supposé épars. D'un point de vue pratique, cela revient à éviter une balle ou un autre drone. D'autre part, il faut être capable d'éviter de nombreux obstacles se déplaçant peu ou à

1. Simultaneous Localization And Mapping ou localisation et cartographie simultanées

des vitesses ne dépassant pas $5\,\mathrm{m\,s^{-1}}$. Pour les obstacles se déplaçant peu, on peut penser aux branches lors de vols avec du vent. Pour les obstacles se déplacent à vitesse réduite, on pourra notamment penser à du vol autonome proche d'humains ou de nombreux autres robots, deux capacités qui seront nécessaires pour des applications industrielles.

L'évitement d'un obstacle à forte vélocité pose principalement un problème de perception. En effet, en cas d'urgence et dans un espace considéré vide si l'on excepte l'obstacle dynamique qui s'approche, la réponse au niveau du contrôle peut tout simplement être une saturation de tous ou de certains des moteurs. Le challenge principal est de détecter l'obstacle suffisamment tôt pour avoir le temps de mettre en oeuvre cette commande d'urgence et donc de l'éviter. La plupart des papiers se concentrent sur la détection des obstacles à l'aide de caméras puisqu'il s'agit de capteurs ayant une grande portée ainsi qu'une très grande résolution. Ainsi, pour de l'évitement pour des avions de tourisme, Mejias et al. proposait dès 2010 un pipeline de vision se basant sur des opérations morphologiques et un filtre temporel permettent de détecter des aéronefs en approche à plus de 400m [Mej+10]. Le calcul du temps avant contact fait relativement souvent parti des solutions proposées, par exemple avec ce papier [PC16] qui travaille spécifiquement sur l'évitement de deux petits UAVs. En dehors de ces travaux avec des caméras, Falanga, Kim et Scaramuzza a récemment proposé l'utilisation d'une event-camera, une caméra qui détecte des changements d'intensité et communique ces détections de manières asynchrones ce qui permet de reconstruire des vidéos à 5000Hz, grâce à laquelle il a réussi à éviter des ballons envoyés sur un quadrirotor [FKS19]. Sanket et al. propose l'utilisation du même type de capteur et obtient 70% de réussite pour éviter des lancers d'objets de divers formes et tailles [San+19].

Pour l'évitement de multiples obstacles dynamiques à basses vélocités, il y a plusieurs challenges. Comme on le verra dans la partie 3.3, la plupart des systèmes de vision ne sont pas conçus pour représenter un environnement dynamique. Cela génère des erreurs de représentation de ces obstacles qui peuvent soit être ajoutés plusieurs fois à la représentation, soit être classifiés comme de fausses détection et ne pas y être ajouté. Comme signalé plus tôt, la présence d'obstacles dynamiques perturbe aussi les algorithmes visuo-inertiels : quel est le mouvement de la caméra par rapport à une scène non fixe ? Finalement, la navigation dans ces environnements dynamiques requiert aussi d'estimer les trajectoires des différents obstacles mouvants.

Andersson et al. [And+18] s'attaquent à la navigation dans un environnement statique et dynamique. Ils proposent de simplifier l'atteinte d'un point de passage en simplifiant le problème avec un mécanisme d'horizon variable. Les déplacements du drone, à faible vélocité, sont gérés grâce à une librairie de trajectoires (voir 3.4.2). S'ils obtiennent des performances intéressantes en simulation, on notera qu'ils ne prennent absolument pas en compte la perception des obstacles. Ils travaillent dans une carte d'occupation connue et estiment simplement la position future des obstacles dynamiques dans cette carte. Ils peuvent ensuite vérifier si les trajectoires de leur librairie vont entrer en collision avec les obstacles statiques ou avec la position estimée des obstacles dynamiques au temps correspondant. Si ces résultats sont intéressants, il ne sont applicables que dans un environnement totalement cartographié ce qui limite les possibilités d'utilisations de cet algorithme. Allen et Pavone proposent une autre

solution à un problème quasi équivalent [AP19]. Leur solution, qui a été testée au travers de vols réels, consiste en une génération d'une trajectoire dynamiquement non faisable jusqu'à l'objectif final et une transformation en une trajectoire faisable en s'inspirant des travaux de Richter, Bry et Roy [RBR16]. Ces trajectoires et optimisations sont fréquemment recalculés en cours de vol pour pouvoir contourner les obstacles dynamiques. Cette approche ne réalise pas d'estimation des positions futures et les trajectoires générées ne sont valables que lors de leur génération. Pour éviter des obstacles dynamiques qui se déplaceraient à proximité du drone, un contrôle réactif à l'environnement local est proposé. Cet algorithme souffre des mêmes défauts que l'algorithme précédent puisqu'il suppose une perception externe ainsi qu'une cartographie de l'environnement. De leur coté, Gao et Shen [GS17] proposent une génération de courtes trajectoires polynomiales qui utilisent la planéité différentielle (differential flatness en anglais, notion présentée dans la partie 3.4.3) des quadrirotors pour éviter de multiples obstacles dynamiques. Comme les travaux précédents, ils supposent cependant connaître les positions de tous les obstacles à chaque instant.

A ma connaissance, seuls quelques travaux préliminaires existent sur l'évitement d'obstacles dynamiques en prenant en compte la perception du drone tel que [Lem16]. Ces travaux, qui ont une approche plus pratique, proposent une démonstration avec un unique obstacle dans un environnement quasi-vide et ne donnent aucune garantie sur l'anti-collision. Néanmoins, il s'agit des seuls travaux présentant des expérimentations avec un obstacle dynamique et une perception intégrée au drone.

Pour terminer, signalons qu'un certain nombre d'algorithmes, bien que non construits pour réaliser de l'évitement dynamique, permettent tout de même d'éviter certains obstacles dynamiques. Dans ces algorithmes, on peut par exemple citer celui que propose Penin, Giordano et Chaumette [PGC18]. Il permet de suivre une cible mouvante tout en évitant des obstacles avec une attention particulière pour garder la cible dans le champs de vision des capteurs. Comme l'algorithme proposé est à 30HZ, il permet, dans une certaine mesure d'éviter certains obstacles dynamiques. En effet, si la vitesse du drone est plus élevée que celle de l'obstacle et si l'algorithme décide d'éviter l'obstacle dynamique en passant *derrière* celui-çi, alors cet algorithme, comme d'autres algorithmes qui proposent des approches de replannifcation en ligne, permet d'éviter l'obstacle. Le principal problème de ces approches qui ne prennent pas directement en compte les mouvements des obstacles est dans la sélection de trajectoires qui vont éviter un obstacle dynamique en passant par les positions futures de l'obstacle. Cela va très probablement générer des collisions. Il semblerait cependant qu'un certain nombre de ces approches, comme l'approche de Penin, Giordano et Chaumette citée précédemment, puissent bénéficier d'une extension prenant explicitement en compte les obstacles dynamiques qui permette d'étendre leur capacité d'évitement à un plus grand nombre d'obstacles dynamiques.

3.2 Notre définition de l'évitement d'obstacles

Dans la partie précédente, la distinction a été faite entre méthodes de planification et méthodes réactives. Le projet qui finance la thèse attendait de cette thèse la possibilité de transformer rapidement un drone capable de naviguer entre des points de passages dans un environnement sans obstacle en un drone capable de naviguer entre des points de passages dans n'importe quel environnement inconnu. Les méthodes de planification comme les méthodes réactives permettent d'atteindre cet objectif. On s'est orienté, par préférences personnelles, vers une méthode de planification parce qu'il nous semblait plus pratique de baser notre navigation sur une trajectoire que sur des instructions valables sur un intervalle temporel très court. On notera tout de même que l'utilisation de ce type de commande peut apporter une légère sécurité supplémentaire que ne peuvent, par définition, apporter les méthodes réactives. Il est en effet possible de ne générer que des trajectoires dont l'état final sera stable ce qui permet d'obtenir, à chaque exécution de l'algorithme, un état sécurisé qui restera valable sur le long terme. Si un crash logiciel ou matériel venait à perturber les nouvelles exécutions de l'algorithme d'évitement d'obstacles, cela permettrait de stabiliser le drone le temps qu'un pilote reprenne la main.

On a aussi expliqué dans la partie précédente, notamment au travers de la figure 3.1, que les algorithmes d'évitement d'obstacles étaient composés de multiples fonctionnalités et qu'un papier souhaitant présenter un algorithme devait nécessairement évoquer chacune de ces fonctionnalités. S'il est possible qu'un même travail intègre chacune de ces fonctionnalités, il est par contre impossible d'effectuer un travail de recherche sur chacune d'entre elles. Il faut aussi souligner qu'il ne faut pas négliger les possibilités d'améliorations dans l'utilisation de ces fonctionnalités. Prenons pour exemple la génération de trajectoire : une fois que l'on a choisi une méthode de génération pour générer **une** trajectoire, se contente-on d'une seule génération ? Si non, quelles trajectoires génère-t-on ? Combien en génère-t-on ? Quelle trajectoire va-t-on finalement sélectionner ?

Par intérêt personnel, pour mettre à profit mes connaissances, et pour essayer de mettre en avant un aspect que je trouve négligé dans la littérature, j'ai donc restreint le travail de recherche effectué à la représentation de l'espace et surtout à la stratégie de génération de trajectoires. La figure 3.2 reprend le schéma général proposé dans la partie précédente consacrée l'état de l'art et la précise dans le cadre de cette thèse. Les éléments en orange représentent les éléments sur lesquels un travail d'intégration a été effectué quand les éléments en rouge représentent les sujets sur lesquels un travail de recherche a été mené.

Cela permet de constater que tous les éléments logiciels considérés comme faisant partie de l'évitement d'obstacles ont été abordés durant la thèse. On notera que lorsqu'un travail d'intégration a été effectué, il a souvent été effectué en commun avec d'autres membres du laboratoire quand le travail de recherche a d'abord été personnel. Les seuls éléments sur lesquels aucun travail n'a été effectué sont les capteurs puisque je n'ai ni construit physiquement le drone ni réalisé les pilotes (drivers) nécessaires à la communication entre les capteurs et la carte de vol. Il est aussi intéressant de noter que l'estimation d'état est une fonctionnalité qui est nécessaire pour de simples vols avec pilotage automatique en extérieur (sans évitement

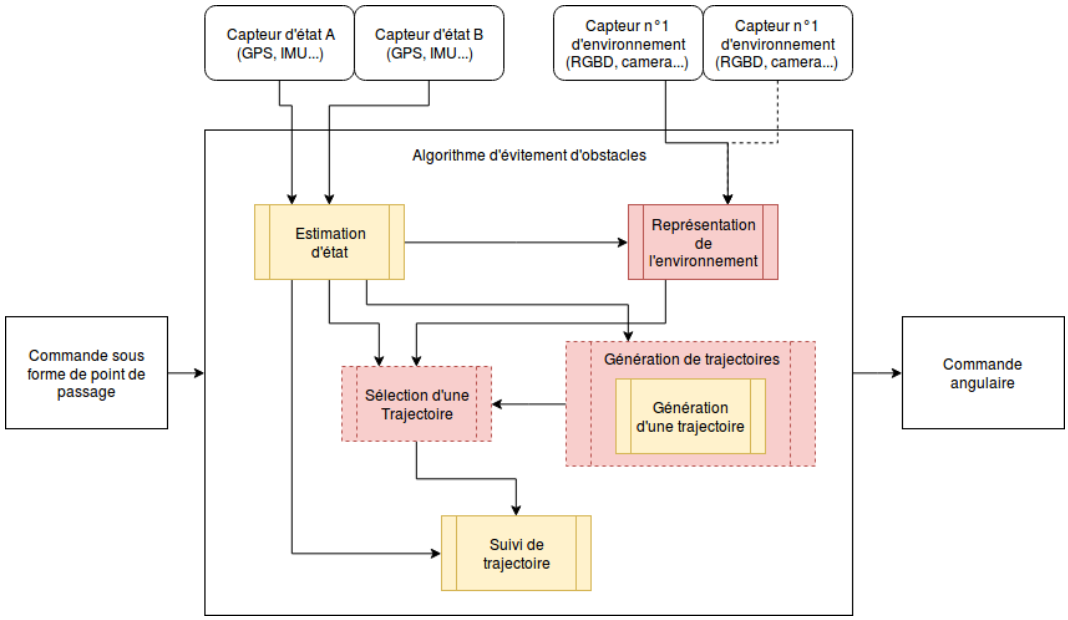


FIGURE 3.2 – Vue schématique du travail effectué.

d'obstacles) et que cette fonctionnalité sera donc développée par un autre acteur du projet avant d'être intégrée au sein du laboratoire au quadrirotor de test.

On peut aussi constater sur cette figure que les entrées et sorties du travail de recherche n'ont pas nécessairement besoin d'être les mêmes que pour la totalité de l'algorithme d'évitement d'obstacles. On remarque en particulier que les capteurs d'états ne sont en lien direct qu'avec le bloc d'estimation d'état qui agit comme une interface entre ces capteurs et le reste de l'algorithme. Puisque l'estimation d'état n'est pas considéré dans le travail de recherche, l'entrée de l'algorithme étudié n'a plus nécessairement à être liée aux sorties capteurs mais peut être constituée de l'état, bruité, qu'il est possible d'obtenir en sortie de l'estimateur.

Voici donc la liste des entrées telles que définies pour notre travail de recherche :
- un état du quadrirotor à 12 dimensions composé de la position dans un repère fixe lié à l'environnement, de la vitesse linéaire, de l'accélération linéaire et de l'attitude du quadrirotor.
- une ou des images de profondeur ou un ou des nuages de points issus d'un ou de plusieurs capteurs de profondeur.
- une commande sous la forme d'une suite de points de passages.

Si on reprend chaque entrée en détail. L'état de départ à 12 dimensions est assez courant, il n'est constitué que de grandeurs physiques qu'il est possible d'évaluer, directement ou indirectement, sur un quadrirotor. On remarque aussi que les vitesses angulaires ne figurent pas dans l'état d'entrée, elle ne seront donc pas utilisées par l'algorithme d'évitement. Ces vitesses pourraient servir entre autre pour affiner la génération de trajectoire mais leur apport est négligeable et ne saurait être significatif à cause du bruit que l'on aura sur l'observation des autres états. Elles pourraient aussi servir dans le bloc de contrôle d'attitude mais ce bloc est

en dehors des limites que nous avons fixé à notre algorithme d'évitement, nous reviendrons sur cela dans quelques instants.

Les sources données de profondeur ne sont pas précisées parce que le système de représentation devrait, si possible, être indépendant de ces données et pouvoir s'adapter à différents types d'entrées.

Finalement, la commande est définie sous la forme d'une suite de points de passage puisque cela est la manière la plus simple de se conformer aux attentes du projet dans lequel se déroule cette thèse. Ce choix a l'avantage d'autoriser une adaptation facile à n'importe quel type d'algorithme de planification général. Prenons par exemple un algorithme de planification qui fournit une trajectoire complète. On pourra aisément à chaque exécution de l'algorithme d'évitement d'obstacles, extraire un point de la trajectoire fournie, par exemple la position à t+10s, que l'on définira comme point de passage.

La sortie de l'algorithme est une commande angulaire, en Roulis-Tangage-Lacet-Poussée. Ce type de commande est privilégié parce qu'il s'agit d'un type de commande supporté par la totalité des firmwares pour quadrirotors. Utiliser ce type de commande en sortie d'algorithme revient à ne pas gérer tout le contrôle du drone mais à utiliser les contrôleurs en attitude (voir 2.4.4) déjà disponibles sur les drones sur lesquels nos algorithmes vont être ajoutés. Ce choix a été fait pour augmenter la généricité de l'algorithme proposé, pour que celui-ci ne soit pas dépendant d'un contrôle spécifique mais puisse au contraire s'interfacer sur de nombreux drones. Plus de détails sur le contrôle seront disponibles dans la partie 3.5 traitant exclusivement de cette problématique.

Pour finir, posons quelques notations sur le problème auquel nos algorithmes tenteront de répondre. L'objectif sera d'atteindre les points de passage P_i avec $P_i \in \mathbb{R}^3,\ i \in [1, n]$. Ces points forment une liste ordonnée par l'indice i de n points de passages. On considère un point de passage P_i atteint quand la distance entre ce point de passage et ξ la position du centre de gravité du quadrirotor définie en 2.3 est inférieure à $\mu_i \in \mathbb{R}^+$, un paramètre de distance qui dépend de la précision requise par l'entité qui génère les points de passages P_i.

3.3 Les représentations de l'espace

Maintenant que le problème est plus clairement posé, on va se concentrer sur l'étude des fonctionnalités principales et nécessaires à l'évitement d'obstacles qui ont été identifiées dans la partie 3.1. Commençons dès à présent par la représentation de l'espace qui peut prendre plusieurs formes.

3.3.1 Grilles d'occupations 3D

Il s'agit du type de représentation le plus souvent privilégiée par les algorithmes de planification. Cette représentation a l'avantage de permettre de construire de réelles cartes de

l'environnement ce qui peut sembler nécessaire dans les environnements encombrés et ce qui peut être l'objectif même de certaines missions.

Un exemple de ce type de représentation appliquée aux drones est offert par [Val+14]. Une octomap [Hor+13], un type de grille d'occupation 3D qui utilise une structure hiérarchique d'octree, est construite en embarqué sur un drone à partir d'images clés pré-sélectionnées. Il faut dans ce papier environ 1 s pour prendre en compte la nouvelle image traitée dans la carte de l'environnement.

D'autre types de représentations 3D se basant sur des grilles d'occupations sont aussi développées pour représenter l'environnement en temps réels avec des drones. Par exemple dans [Ole+17], Oleynikova et al. proposent une carte dense composée de voxels qui contiennent la distance à l'obstacle le plus proche. Sa méthode, qui fournie plus d'information qu'une simple grille d'occupation, est pourtant plus rapide que la plupart des implémentations d'octomap. Ce type de représentation a l'avantage d'autoriser l'exécution d'une plus grande diversité d'algorithmes de replanification tel que CHOMP [Rat+09].

3.3.2 Points ou surfaces d'intérêts

L'idée de ces représentations est de stocker directement un certain nombre de points ou de surfaces d'intérêts. La sortie des capteurs de profondeur étant généralement assimilable à un nuage de points, cela invite ce type de représentation.

Stocker directement des points est envisageable lorsque les capteurs utilisés ne retournent qu'un nombre limité de valeurs. C'est par exemple ce qui est fait dans [Bar16] où seuls les obstacles présents à une certaine distance des caméras de stéréo-vision sont détectés. Cette approche permet une exécution très rapide de l'algorithme de stéréo-vision ce qui en fait une solution adéquate pour de l'évitement d'obstacles réactif où la fréquence de mise à jour de la commande est directement liée à la durée d'éxécution de l'algorithme. Il est aussi possible de stocker la totalité des points en éliminant les mesures les plus anciennes ou en se limitant à une zone géographique.

L'une des solutions pour limiter la multiplication des points d'intérêts est de les regrouper en surfaces appelées surfels. Ce choix permet de réduire très fortement la complexité de la représentation et permet d'obtenir à des représentations adaptées à des applications embarquées [Can+19].

Le problème principal de ces méthodes quand appliquées à de l'évitement d'obstacles est qu'elles font l'hypothèse que l'environnement est vide puis ajoutent des obstacles lorsqu'ils sont détectés. Comment alors déterminer si les trajectoires générées passent dans des espaces observés et vides ou si elles passent dans des espaces non encore observés ?

3.3.3 Cartes de profondeur égocentriques

Par définition, une carte de profondeur, appelée aussi *z-map*, est une représentation de l'espace où chaque élément stocke une valeur qui représente une distance. Les coordonnées des différents éléments dans la carte de profondeur permettent quand à elles d'obtenir deux informations de position supplémentaires. La somme des trois informations contenues dans chaque élément (coordonnées et valeur stockée) permet alors de replacer cet élément dans l'espace.

En 2009, Otte et al. montrent que l'on peut utiliser ce type de représentation pour effectuer de l'évitement d'obstacles avec un robot terrestre en centrant la représentation autour du robot [Ott+09]. Leur représentation utilise des coordonnées sphériques. Toujours en centrant la représentation autour du robot, une carte de profondeur est utilisée pour effectuer de l'évitement d'obstacles sur les drones par Brockers et al. qui propose en 2016 dans [Bro+16] une représentation qu'il nomme egocylindrique visible sur la figure 3.3.

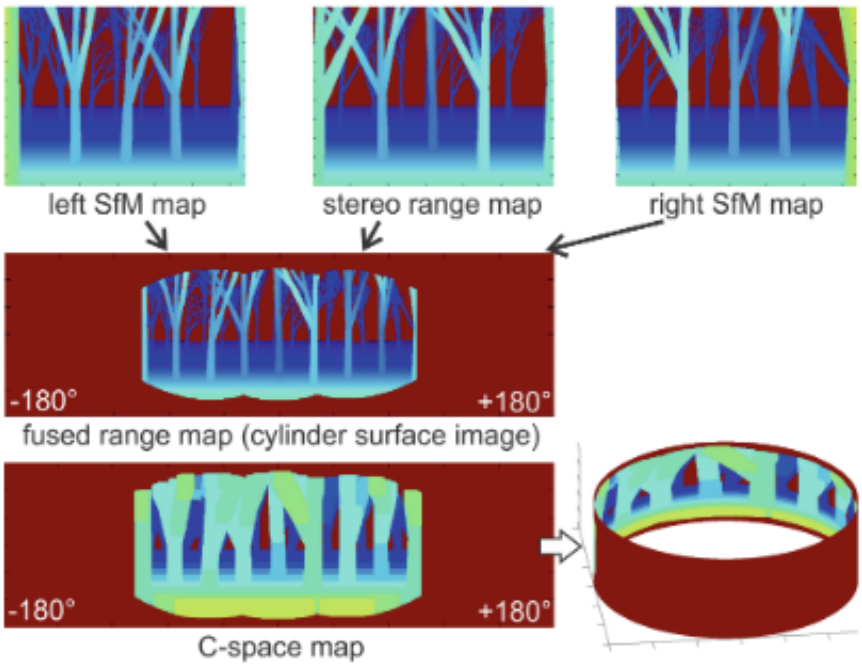


Figure 3.3 – Egocylindre de Brockers et al.

Il est intéressant de noter que pour les deux papiers précédemment cités, la représentation est utilisée pour fusionner de multiples capteurs (deux sur le robot terrestre et trois sur le quadrirotor). Il est aussi intéressant de noter que si l'egocylindre présenté par Brockers et al. utilise plusieurs images capturées simultanément pour affiner les mesures obtenues individuellement par chacun des capteurs, il n'y a pas de propagation de l'egocylindre au cours du temps.

Comparons à présent l'empreinte mémoire d'une grille d'occupation et d'une carte de

profondeur. Pour effectuer cette comparaison, on va chercher à représenter la sphère de 10m entourant notre quadrirotor avec une précision de 25 cm × 25 cm au maximum. La sphère que l'on souhaite cartographier va avoir un volume de $4/3 * Pi * 10^3 = 4186\,\mathrm{m}^3$. Un élément de la grille d'occupation de son côté occupera : $0.25^3\mathrm{m} = 0.015\,625\,\mathrm{m}^3$, il faudra donc légèrement plus de 267900 éléments pour couvrir cette zone. Pour une carte de profondeur sphérique une distance de 25 cm à 10 m correspond à un angle de $arctan(0.25/10.)$ soit un petit peu plus que 1.43°. Il font donc $360/1.43 * 180/1.43 = 31600$ pour obtenir une représentation sphérique comparable à la grille d'occupation définie précédemment. L'une et l'autre de ces représentations sont sous optimales (une représentation sphérique va être beaucoup plus dense aux pôles qu'à l'équateur ; une grille d'occupation à taille fixe pourrait être remplacée par d'autres représentations tel une octomap [Hor+13] dont on a déjà parlé) mais permettent de mettre en évidence un facteur 10 dans l'empreinte mémoire inhérente à chacune de ces méthodes. Ce facteur serait encore plus prononcé pour un rayon plus important puisque la taille de la représentation sphérique ne dépend pas du volume de l'espace représenté contrairement à une grille d'occupation. Cette comparaison met en avant l'avantage principal des cartes de profondeur, celles-ci sont beaucoup plus légères que les grilles d'occupation.

En contrepartie, ces cartes de profondeur ont un défaut que ne possèdent pas les grilles d'occupations : elles ne permettent pas de représenter les obstacles occludés puisque seule la distance à l'obstacle le plus proche est stockée dans chaque direction. Comme cela a été montré, notamment par les papiers cités précédemment, cela n'est pas nécessairement problématique pour effectuer de la navigation majoritairement rectiligne dans un repère local mais perdre toute information après le premier obstacle va nécessairement réduire artificiellement l'espace considéré comme libre autour du robot considéré. Cette perte d'information sera aussi problématique lors de la propagation de la carte de profondeur pour suivre le déplacement du robot et conduira à manquer d'information dans certaines directions.

3.3.4 Notre représentation, l'egocubemap

Notre objectif étant uniquement de réaliser un algorithme d'évitement d'obstacles et non une cartographie de l'environnement, reconstruire une carte de l'environnement en utilisant une grille d'occupation, même locale, semble être excessif. La quantité d'information stockée semble disproportionnée comparée à la quantité d'information nécessaire pour naviguer sans collision.

D'autre part, ne garder en mémoire que des points identifiés comme des obstacles semble être une grosse prise de risque. Comme signalé plus tôt, comment peut-on alors gérer l'espace non perçu qui est considéré comme vide ? Dans le but de sécuriser au maximum le quadrirotor, on souhaite, au contraire, utiliser une représentation de l'espace qui suppose que l'environnement est occupé sauf observation contraire.

Ces deux constats nous ont poussés à choisir une carte de profondeur égocentrique. On notera que l'on propose l'utilisation d'une carte de profondeur égocentrique en utilisant une représentation qui n'a jamais été utilisée jusqu'à présent en embarquée sur drone puisque

l'on propose d'utiliser une cubemap. Cette représentation, que l'on nomme egocubemap, est schématisée sur la figure 3.4.

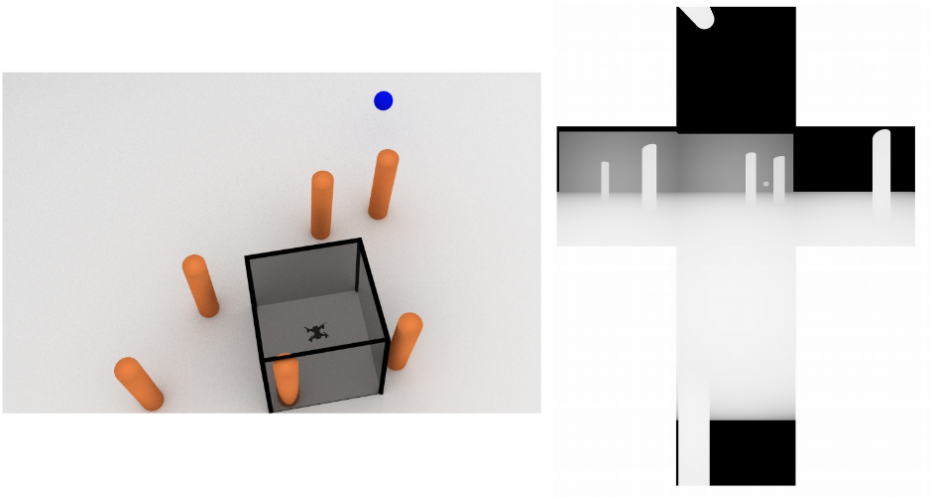
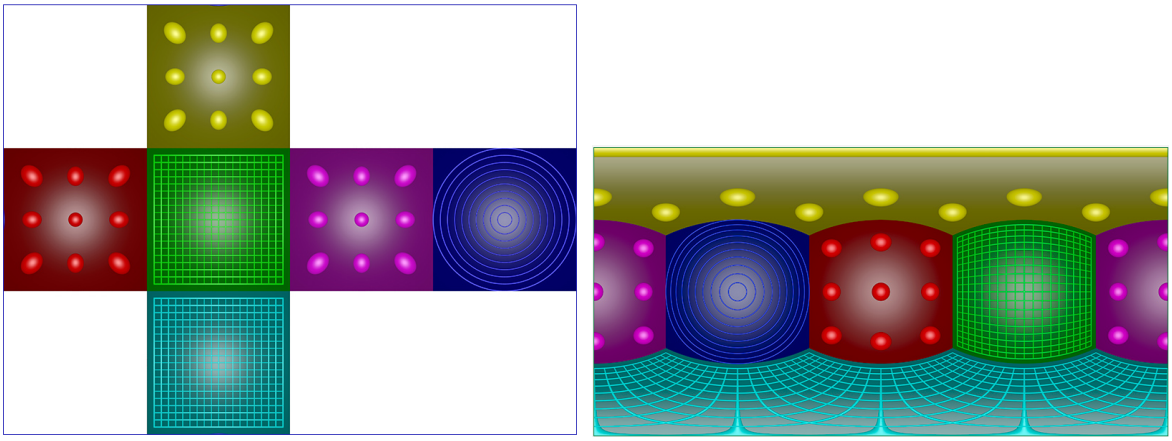


FIGURE 3.4 – Schémas d'une cubemap.

Une cubemap est simplement une représentation 360°utilisant un système de coordonnées différent des représentations sphériques ou cylindriques qui sont plus classiques. Cette représentation a deux avantages principaux. Pour commencer, la dispersion des points d'une cubemap dans l'espace est plus uniforme que la dispersion des points d'une représentation égocylindrique. Grâce à cela, on va pouvoir avoir une meilleure résolution angulaire qu'avec une représentation sphérique, aussi appelée équirectangulaire, tout en stockant moins de points en mémoire. La figure 3.5 montre la représentation sous forme de cubemap en (a) et en coordonnées sphériques en (b) d'un cube sur lequel est dessiné sur 3 faces des disques de même taille, sur deux faces des grilles régulières et sur la dernière face des cercles centrés de différentes tailles. Il est clair que la cubemap n'est pas une représentation parfaite puisque les disques n'apparaissent pas tous de même taille. La représentation sphérique semble même être plus performante pour les latitudes proches de zéro degré. Cependant, lorsque les latitudes s'éloignent de zéro, les déformations deviennent très importantes pour les coordonnées sphériques. On se retrouve alors, dans ce système de coordonnées, à représenter une petite partie de l'espace avec de très nombreux points ce qui est très peu efficace. En moyenne, la déformation créée par une représentation de type cubemap est donc plus faible que la déformation que crée une représentation sphérique.

L'autre avantage de la cubemap est son atout principal : les projections et reprojections entre espace réel et cubemap sont plus efficaces que leurs équivalents dans les autres représentations. Comparons par exemple la reprojection. Le code ci-dessous présente la reprojection du point x, y, z à partir d'un cubemap dont les 6 faces sont numérotés par *index* et dont les coordonnées dans chacune des faces sont comprises entre [0 et 1] et notées u et v :

```
void convert_cube_uv_to_xyz(int index, float u, float v,
                            float *x, float *y, float *z)
```



(a) (b)

FIGURE 3.5 – Comparaison des représentations cubemap (a) et sphérique (b).

```
{
  // Get the depth value stored inside the cubemap
  float depth = cubemap[index].at(u,v);

  // convert uv range from [0 1] to [-1 1]
  float uc = 2.0f * u - 1.0f;
  float vc = 2.0f * v - 1.0f;

  switch (index)
  {
  // FACE POSITIVE X
    case 0: *x =      depth; *y = vc * depth; *z =-uc * depth; break;
    // FACE NEGATIVE X
    case 1: *x =     -depth; *y = vc * depth; *z = uc * depth; break;
    // FACE POSITIVE Y
    case 2: *x = uc * depth; *y =      depth; *z =-vc * depth; break;
    // FACE NEGATIVE Y
    case 3: *x = uc * depth; *y =     -depth; *z = vc * depth; break;
    // FACE POSITIVE Z
    case 4: *x = uc * depth; *y = vc * depth; *z =      depth; break;
    // FACE NEGATIVE Z
    case 5: *x =-uc * depth; *y = vc * depth; *z =     -depth; break;
  }
}
```

En coordonnées sphériques, ce passage semble beaucoup plus simple puisque le code qui permet d'effectuer la reprojection présenté juste après ne fait que quelques lignes. Il est cependant plus long à exécuter à cause des multiples fonctions trigonométriques. On note $u, v \in [01]$ les coordonnées dans la représentation équirectangulaire et x, y, z les coordonnées

dans l'espace. on a :

```
void convert_equirectangular_uv_to_xyz(float u, float v,
                                        float *x, float *y, float *z)
{
  // Get the depth value stored inside the equirectangular map
  float rho = equirectangular.at(u,v);

  // convert u range from [0 1] to [-Pi Pi]
  float theta = 2*Pi * u - Pi;

  //convert v range from [0 1] to [-Pi/2 Pi/2]
  float phi = Pi * v - Pi/2.0f;

  // Get x, y z using the classical spheric to cartesian transformation
  *x = rho * sin(phi) * cos(theta)
  *y = rho * sin(phi) * sin(theta)
  *z = rho * cos(phi)
}
```

Contrairement aux papiers utilisant une carte de profondeur égocentrique présentés précédemment, on propose une solution n'utilisant qu'une seule caméra de profondeur avec un champ de vision limité ($< 70°$ x $50°$). Pour limiter l'impact de ce champ de vision réduit, on va vouloir conserver au cours du temps une information sur les obstacles qui quittent le champ de vision de notre camera. Cela implique de propager notre représentation de l'espace en fonction du déplacement du drone. L'efficacité de la reprojection devient donc cruciale pour effectuer cette opération rapidement d'où l'utilisation d'une cubemap.

On pourrait se contenter d'utiliser cette représentation de l'environnement au cours de nos vérifications de trajectoires mais on a préféré coupler cette représentation à une opération relativement classique en navigation : l'expansion de l'espace de configuration (C-Space expansion) [LP90]. Cette opération consiste à élargir tous les éléments représentés dans un environnement par un volume Vol. Cela permet ensuite de vérifier si le volume Vol, lorsqu'il se déplace en translation, va être en collision avec son environnement en connaissant uniquement la position de son centre géométrique. La figure 3.6 schématise en 2D le principe de fonctionnement du C-Space. Sur la figure de gauche, lors de la vérification des trajectoires, en chaque point de la trajectoire, on vérifie que la totalité du volume bleu n'entre pas en contact avec les obstacles. Sur la figure de droite, tous les obstacles sont élargis ce qui permet de créer la zone en rouge clair. Il suffit ensuite de vérifier si la trajectoire crée passe dans cette zone pour savoir s'il y aura collision ou non.

Le volume que l'on choisit pour être à la base de notre C-Space sera une sphère. Si un quadrirotor est presque planaire et est souvent représenté par un cylindre, son attitude va fortement évoluer lors de son évolution dans l'espace. Pour ne pas avoir à recalculer le

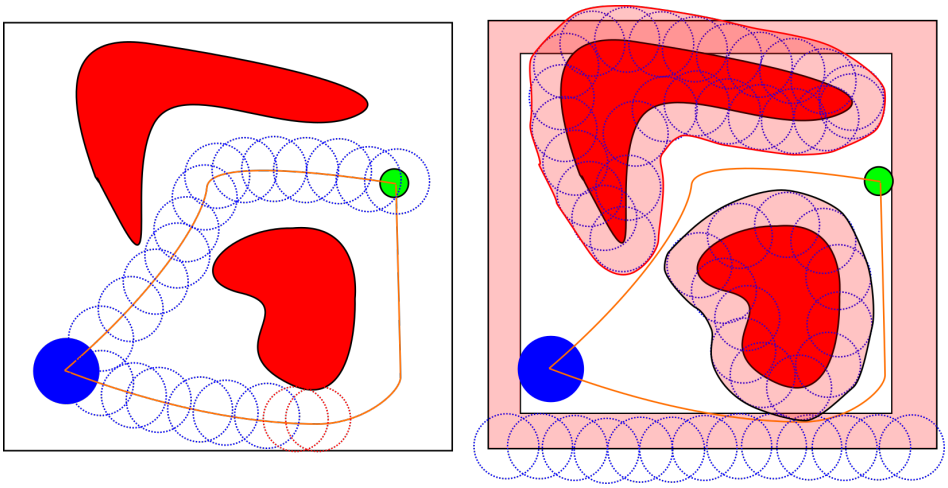


FIGURE 3.6 – C-Space : schéma de principe.

C-Space en fonction de l'attitude de notre quadrirotor, on choisit donc une sphère englobante comme volume de définition de notre C-Space. Le C-space est une transformation qui permet de gagner du temps de calcul si les algorithmes développés vont générer un besoin important de vérifications de collisions. Or l'un des buts de nos algorithmes va être de générer des trajectoires sans collisions. Il va donc falloir, sur chaque trajectoire testée, vérifier sur de multiples positions intermédiaires si il y a une collision avec l'environnement. Intuitivement, on a aussi l'idée qu'on va avoir besoin de générer de nombreuses trajectoires pour essayer de trouver une trajectoire satisfaisante, une idée qu'il sera possible de mettre en oeuvre grâce au C-space. Il faut considérer que le C-space est un investissement en calcul pour s'offrir plus de possibilités lors des phases de vérifications de trajectoires.

On va maintenant entrer dans les détails techniques de notre egocubemap qui comporte trois phases : une phase de propagation de l'ancienne egocubemap, une phase d'ajout de la nouvelle image de profondeur et une phase de calcul de l'espace de configuration. La première phase permet de conserver l'information passée et de la mettre à jour en fonction des mouvements du drone, la seconde phase permet l'ajout des dernières observations et la troisième et dernière phase est un investissement en temps de calcul qui permettra par la suite d'accélérer les vérifications de trajectoires. On a fait le choix de déporter la construction de l'egocubemap sur carte graphique pour pouvoir paralléliser le traitement des différents pixels et ainsi réduire le temps de calcul nécessaire à chacune des phases précédemment identifiées. On a, en pratique, fait le choix d'utiliser la librairie OpenGL pour effectuer la parallélisation et on va a présent voir comment on a du adapter l'algorithme décrit précédemment pour minimiser, à travers cette librairie, les temps d'exécutions.

1- propagation : Commençons par la phase de propagation qui, embarquée sur drone, est une nouveauté pour une carte de profondeur 360°. L'entrée de cette phase est constituée d'une ancienne egocubemap et de la transformation entre l'état du drone lors du calcul de cette ancienne egocubemap et l'image de profondeur courante. Le but de cette étape est de déplacer

l'ancienne egocubemap en accord avec le déplacement estimé. Pour obtenir une egocubemap dense, chaque pixel de l'ancienne egocubemap est considérée comme une surface rectangulaire et non comme un point. Pour cela, on considère les coins des pixels et non leur centre comme cela est fait habituellement. Notons u et v les coordonnées d'un pixel dans une cubemap et Δu et Δv le pas entre deux pixels, les quatre coins ont pour coordonnées $\left[u \pm \frac{\Delta u}{2}, v \pm \frac{\Delta v}{2}\right]$. En associant à ces coordonnées la profondeur z du pixel $[u, v]$, on obtient des coordonnées complètes et il sera ainsi possible de reprojeter ces points dans l'espace avec les formules présentées précédemment. OpenGl étant optimisé pour faire des calculs sur des triangles, chaque rectangle/pixel est scindé en deux triangles. Pour chaque triangle, les sommets sont projetés de l'espace de la cubemap à l'espace réel, déplacé de l'inverse du déplacement estimé puis projeté dans la nouvelle cubemap. Cela permet d'obtenir un nouveau triangle qui sera ajouté à cette nouvelle cubemap. Ce processus créé des chevauchements lors de la projection dans la nouvelle cubemap. Ces chevauchements sont résolus grâce à un mécanisme de test Z-buffer qui conserve uniquement la distance la plus proche sur chaque pixel de la nouvelle cubemap.

2- ajout de la nouvelle image : La nouvelle image de profondeur est ensuite ajoutée à la nouvelle cubemap en remplaçant la profondeur estimée par la profondeur de l'image de profondeur dans les zones observées. Tous les pixels de la nouvelle image de profondeur sont individuellement projetés sur la cubemap. Puisque la résolution angulaire de l'image de profondeur est plus élevée que la résolution angulaire de la cubemap, de multiples pixels de l'image de profondeur vont être projetés sur le même pixel de la cubemap. Pour régler les conflits, un mécanisme de Z-buffer permettant de ne conserver que la profondeur la plus faible sera encore une fois appliqué.

3- calcul de l'espace de configuration : Chaque pixel de la cubemap va être considéré comme un obstacle ponctuel dans l'espace réel. Chaque pixel sera donc individuellement projeté dans l'espace réel puis élargi par une sphère, le volume du C-space choisi précédemment. Cet élargissement n'est cependant qu'une étape transitoire virtuelle et à aucun moment ces pixels élargi par une sphère seront stockés en mémoire. Puisqu'OpenGL est optimisé pour travailler avec des triangles, on va extraire de cette sphère des triangles qui seront eux reprojetés dans une *cubemap élargie*. Pour cela, on calcule le le plus petit rectangle englobant la sphère sur le plus proche plan perpendiculaire au centre de la caméra dont l'intersection avec la sphère crée est nulle. Comme dans l'étape de propagation, ce rectangle est séparé en deux triangles qui seront les triangles reprojetés dans la cubemap. Le processus d'obtention du rectangle englobant est détaillée sur la figure 3.7. Finalement, et comme pour les étapes précédentes, un mécanisme de Z-buffer permet de ne conserver que la distance la plus courte pour chacun des points de la cubemap.

Pour conclure cette partie, la figure 3.8 montre l'impact d'une transformation C-space sur une cubemap et la figure 3.9 montre deux projections sphériques (ou équirectangulaires) de deux egocubemap successives obtenues lors d'un vol en simulation. La direction courante du drone correspond au pixel central de ces projections. Par construction, les premières et

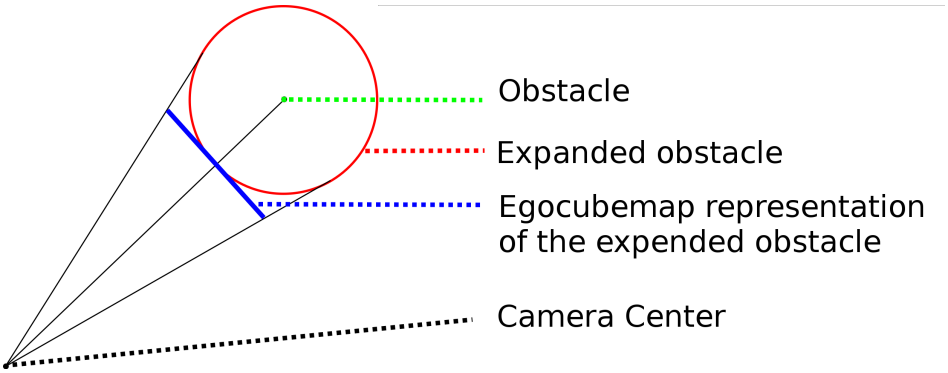
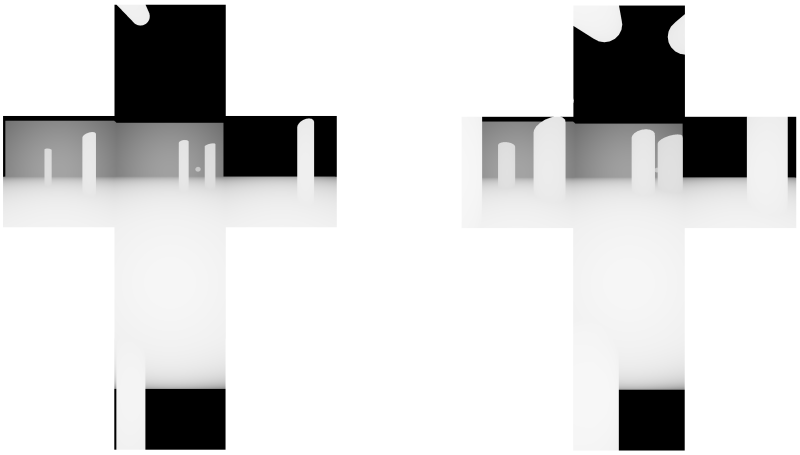


FIGURE 3.7 – C-Space : reconstruction.

dernières colonnes de ces projections correspondent donc à ce qui se trouve derrière le drone.

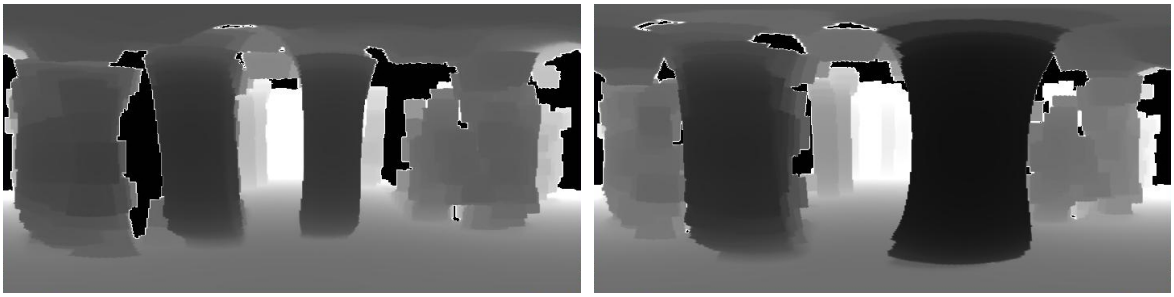


(a) (b)

FIGURE 3.8 – Scène simulée représentée par une egocubemap sans (a) et avec (b) C-space.

3.4 Les différentes générations de commandes

On a vu dans la partie 3.1 qu'il y a différentes catégories d'algorithmes d'évitement d'obstacles. On a notamment fait la distinction entre les algorithmes réactifs et les algorithmes plus proches de la planification et on a signalé que les types de commandes pouvaient différer en fonction des algorithmes. On structurera cette partie autour de la génération de trajectoire. On évoquera d'abord les commandes qui ne sont pas des trajectoires avant de se concentrer sur l'utilisation de trajectoires pré-calculées pour enfin terminer par la génération de trajectoires en cours de vol.



(a) (b)

FIGURE 3.9 – Projections équirectangulaires de deux égocubemap successives.

3.4.1 Commandes autres que trajectoire

Commençons par les commandes qui ne sont pas des trajectoires. Ces commandes sont généralement choisies par les algorithmes réactifs parce que leur calcul est moins coûteux que la génération et la vérification de trajectoires complètes. C'est par exemple ce type de commandes que génèrent les deux papiers récents permettant d'éviter des objets lancés en direction d'un quadrirotor [San+19] ; [FKS19]. La commande de SANKET et al. est une direction "de sécurité" qui maximise la distance du drone à la trajectoire prédite de l'obstacle. Un contrôleur PID à haut gain proportionnel vient ensuite traiter cette commande ce qui rend la commande quasiment identique à une commande saturée dans une direction. FALANGA, KIM et SCARAMUZZA proposent de leur coté une commande selon une direction de type bang-bang en accélération. Cela revient à saturer l'accélération dans la direction considérée avant de saturer l'accélération en sens inverse pour stabiliser le drone à une position considérée comme sécurisée.

On notera aussi qu'à ma connaissance, tous les algorithmes d'évitement d'obstacles utilisant des méthodes d'apprentissage génèrent des commandes qui ne sont pas des trajectoires. Par exemple, SMOLYANSKIY et al. [Smo+17] qui ont suivi des chemins en forêt pendant 1 km à $3\,\mathrm{m\,s^{-1}}$ classifient la position du drone par rapport au chemin (centré, sur la droite, sur la gauche) ainsi que l'orientation du drone par rapport au chemin (centrée, vers la droite, vers la gauche). Ils calculent ensuite un point de passage et un lacet en fonction de ces classifications qu'ils transmettent directement à l'autopilote PX4. De leur coté, SADEGHI et LEVINE [SL16] définissent des *actions* qui sont en fait des positions dans l'espace image. La sélection d'une action revient à un déplacement dans la direction de la position dans l'espace image correspondante.

Autre référence qui utilise un tel contrôle, OLEYNIKOVA, HONEGGER et POLLEFEYS [OHP15] dans une approche très légère calculent des points de passages situés à une profondeur fixe du drone qui ne sont autorisés qu'à se déplacer vers la gauche ou la droite pour effectuer l'évitement d'obstacles.

Les exemples précédents, et bien d'autres dans la littérature, montrent que d'autres

commandes que des trajectoires permettent d'effectuer efficacement de l'évitement d'obstacles. Certaines de ces commandes pourraient cependant bénéficier d'une génération de trajectoire (dans le cas de la définition de points de passages) ce qui permettrait d'améliorer le contrôle dont on dispose sur le drone entre les points de passage. Les autres commandes évoquées sont plus instantanées. Elles permettent de réduire la latence des réactions du quadrirotor et d'exploiter au maximum les capacités offertes par ce dernier, notamment grâce à l'utilisation de commandes saturées. Le défaut principal de ces commandes instantanées réside dans leur courte validité. Elles ne sont, en général, valables qu'au moment où elles ont été générées et leur état final n'est pas, en général, un état stable qui garantit la sécurité du drone en cas de crash logiciel ou matériel (carte de calcul et capteurs) .

3.4.2 Librairies de trajectoires

On en vient à présent aux commandes de type trajectoire où l'idée est de décrire une suite d'états $S_{des}(t)$ qui permette de passer d'un état initial S_0 à un état final S_f. On aimerait que le quadrirotor soit capable de suivre cette suite d'état et on aimerait générer cette suite d'état, cette trajectoire, dans un temps qui n'impacte pas les performances de l'algorithme. Ces souhaits nous incitent à définir deux notions : la faisabilité d'une trajectoire et la génération de trajectoires en temps réel.

Faisabilité d'une trajectoire : On peut dire d'une trajectoire qu'elle est faisable par un quadrirotor s'il est possible de définir une suite de commandes qui permette à un modèle de quadrirotor de décrire exactement cette trajectoire. Cette définition est imprécise mais elle illustre la diversité d'utilisation du terme dans la littérature et permet de bien se rendre compte que cette notion de faisabilité est une notion relative au modèle choisi. En considérant le drone comme un double intégrateur en position, il est ainsi possible de dire qu'une trajectoire est faisable en contraignant simplement les accélérations alors même que le modèle utilisé est un modèle très imprécis qui n'est valable que dans des conditions de vols spécifiques.

Génération en temps réel de trajectoires : La notion de temps réel n'a pas de définition stricte. On dit généralement d'un système qu'il est temps réel si ce système est capable de contrôler un procédé à une vitesse adaptée à l'évolution de ce procédé. Dans cette thèse, une génération de trajectoire sera temps réel si elle peut s'effectuer en ligne *sans trop impacter* les performances de l'algorithme d'évitement d'obstacle. Pour pouvoir prendre en compte l'évolution de l'environnement et donc s'attaquer au problème de l'évitement d'obstacles dynamiques et en partie arbitrairement, on souhaite passer un maximum de 100 ms dans la génération de trajectoires. Une méthode de génération de trajectoires prenant en compte l'environnement et ne nécessitant donc qu'une seule génération sera donc considérée comme temps réel si elle nécessite moins de 100 ms sur la carte de calcul embarquée sur le drone pour générer une trajectoire de quelques secondes. Si la méthode ne prend pas en compte l'environnement, il va falloir générer de multiples trajectoires avant d'en obtenir une qui ne rentre pas en collision avec l'environnement et une telle méthode sera considérée comme temps réel seulement si elle

dure moins d'une milliseconde.

Toute méthode de génération de trajectoires n'atteignant pas les objectifs fixés pour être considérée comme temps réel ne pourra donc pas être utilisée en ligne. La solution la plus courante pour contourner ce problème est de générer, hors ligne et avant le vol, des librairies de trajectoires et à simplement vérifier pendant le vol que les trajectoires ne rentrent pas en collision avec l'environnement. La génération hors ligne de trajectoires permet de créer et d'utiliser des méthodes de génération de trajectoires pour lesquelles le temps de calcul n'est plus un problème. Il est alors possible de générer des trajectoires faisables pour des modèles plus complexes et plus proches des comportements réels que ce que peut l'être le modèle simplifié défini précédemment ou d'explorer des concepts novateurs. L'un de ces concepts, particulièrement intéressant pour l'évitement d'obstacles mais qui demande pour le moment trop de calculs pour être utilisé en ligne, est la création de zones d'incertitudes autour des trajectoires générées. Le drone est alors garanti de rester dans ces zones lors de son suivi de trajectoire en présence de perturbations d'intensités bornées [MT17]. Singh et al.[Sin+17] construisent aussi une zone de sécurité hors ligne mais utilisent une approche différente. Ils ne calculent plus l'incertitude inhérente à chaque trajectoire mais proposent une méthode, à appliquer hors-ligne, permettant de calculer une marge d'erreur qui sera valide pour toute trajectoire respectant certaines contraintes.

L'utilisation d'une librairie de trajectoires a cependant des inconvénients relativement importants. Premièrement, elle nécessite de discrétiser l'espace d'état du drone et de générer des trajectoires pour chaque état ainsi défini. Or l'état nécessaire pour générer une trajectoire est au minimum constitué des vélocités et accélération du drones ce qui créé, au minimum, un état à 6 dimensions. Si on y ajoute la volonté de rejoindre un point de l'espace précis et donc une position finale, cela nous donne un espace à 9 paramètres. Si on ajoute la possibilité de faire varier la dynamique du vol (voir la remarque qui suit ce paragraphe), ce qui est une caractéristique très utile pour modifier la nature du vol en fonction de l'environnement, on obtient une génération dans un espace à 10 dimensions. Pour réduire la complexité de cet état, il n'y a généralement pas de but fixé aux trajectoires générées mais il s'agit plutôt de mouvements relativement divers et de trajectoires relativement courtes. Pour cette raison, on parle relativement souvent de librairie de primitives (de mouvements primitifs) [FCT16] plus que de librairie de trajectoires. De même, à ma connaissance, aucun papier utilisant des librairies de trajectoires pour de l'évitement d'obstacles ne fait varier la dynamique du vol. Des solutions ont aussi été proposées pour tenter de limiter cet effet, Green et Kelly ont par exemple proposé un algorithme basé sur le maximum de dispersion pour sélection des trajectoires dans une librairie [GK06]. Finalement, et malgré ces tentatives pour réduire la complexité, il faudrait générer de très nombreuses trajectoires pour avoir de multiples possibilités avec peu de bruit de quantification sur les états du drone. Cela n'est pas toujours fait et cela amène par exemple [Bar16] à pointer du doigt le manque de diversité de sa libraire de trajectoire et son impact négatif sur ses performances.

Remarque. *La dynamique d'une trajectoire est définie comme étant une grandeur liée à la difficulté à suivre cette trajectoire. Selon les publications, cette grandeur peut être calculée de différentes manières (accélérations maximales en un point de la trajectoire, vitesses de rotation angulaire maximales, moyennes de ces grandeurs sur la trajectoire...). On utilisera de manière*

*interchangeable avec cette notion de dynamique d'une trajectoire, la notion d'**agressivité** d'une trajectoire, une notion elle aussi vaguement définie mais utilisée à de nombreuses reprises comme par exemple par* RICHTER, BRY *et* ROY*[RBR16]. De notre point de vue, cette notion d'agressivité, mise en parallèle avec une conduite agressive, permet de faire ressortir plus clairement que la difficulté à suivre une trajectoire va être liée à de forts changements de directions, accélérations ou décélérations.*

L'autre inconvénient majeur dans l'utilisation des librairies de trajectoires est liée à son atout : si une librairie de trajectoire permet de générer des trajectoires sur des modèles beaucoup plus précis, la précision des modèles utilisés rend les trajectoires générées valides uniquement pour le drone pour lesquelles elles ont été générées. Les générations vont de plus avoir besoin d'informations très précises sur le drone en question pour pouvoir générer leurs trajectoires. Il va par exemple falloir connaître la matrice d'inertie exacte du drone. Ces méthodes engendrent donc une perte importante de généricité qui supprime la majeure partie de l'intérêt qu'elles auraient dans notre projet.

3.4.3 Génération de trajectoires en ligne

Abordons maintenant la dernière option restante, la génération de trajectoires en ligne qui concerne toutes les méthodes vérifiant les conditions détaillées dans la partie précédente pour être considérées temps réel.

La manière la plus simple de générer rapidement des trajectoires est de ne pas considérer leur faisabilité et de générer des trajectoires en définissant très simplement leur géométrie grâce à un rayon de courbure ou à une génération polynomiale. L'inconvénient principal est bien entendu que les trajectoires créées ne vont pas nécessairement être faisables par un quadrirotor. Elles vont alors nécessiter l'ajout d'une étape d'optimisation de trajectoire qui va prendre un certain nombre de points de passages sur la trajectoire définie et recalculer une trajectoire plus précise qui sera cette fois-ci dynamiquement faisable en modifiant légèrement les positions des points de passage et/ou en recalculant les vitesses et accélérations sur la trajectoire [Neu+16] ; [HWT08].

Une autre manière populaire de générer des trajectoires en ligne est d'utiliser les propriétés issues de la platitude (differential flatness) des quadrirotors pour calculer des contraintes sur les trajectoires puis de résoudre un problème d'optimisation en prenant en compte ces contraintes. Le snap minimum [MK11], le temps minimum [HD11], ou encore la distance la plus courte [VZT12] sont des exemples de critères qu'il est possible de minimiser. Si les travaux précédents utilisent le modèle simplifié présenté dans la partie 2.3.1, il a depuis été montré qu'en ajoutant dans le modèle la traînée crée par les rotors, le quadrirotor reste un système plat ce qui permet de générer rapidement des trajectoires prenant en compte la traînée [FFS17].

3.4.4 Notre méthode génération de trajectoires, en ligne, basée sur une minimisation du jerk

L'idée première en début thèse était d'obtenir rapidement un algorithme fonctionnel. Pour cela, au lieu de commencer en créant de nouveaux blocs logiciels pour la totalité des fonctionnalités nécessaires à un algorithme d'évitement d'obstacles, le parti pris était de construire sur les briques existantes, en particulier pour ce qui est relatif au contrôle qui n'est pas ma spécialité. Après l'étape classique de l'étude de l'état de l'art, on a donc travaillé par élimination pour sélectionner la méthode de génération de trajectoire qui nous semblait s'adapter au mieux à notre projet et on a arrêté notre choix sur la méthode de génération de trajectoires proposée par Mueller, Hehn et D'Andrea [MHD15].

La méthode choisie permet de générer un million de trajectoires faisables par seconde. Avant de la décrire plus en détails, attardons nous sur les raisons qui nous ont poussées à choisir cette méthode de génération de trajectoires en ligne. On a déjà expliqué dans la partie 3.2 pourquoi on a préféré une trajectoire aux commandes alternatives. Une fois ce choix réalisé, l'intérêt d'utiliser une librairie de trajectoires nous semblait réduite dans le projet considéré. S'il semble intéressant de pouvoir générer des trajectoires sur des modèles complexes, notamment parce que cela permet de s'approcher des limites dynamiques du quadrirotor modélisé, cela induit une perte de généricité que l'on ne souhaite pas dans le projet. En conséquence, puisqu'on ne va pas utiliser de modèle complexe et qu'il est possible de générer des trajectoires faisables en temps réel avec des modèles simples, on n'a pas d'intérêt à générer des librairies de trajectoires, ce qui force à discrétiser l'espace d'état et génère artificiellement de l'erreur, plutôt qu'à directement générer des trajectoires en ligne. Le choix entre une génération purement géométrique et une génération garantissant une faisabilité est encore plus simple. S'il est possible d'obtenir une garantie supplémentaire sur la génération de trajectoire pour un coût en calcul minime, pourquoi s'en passer. Finalement, c'est la vitesse d'exécution de la méthode choisie ainsi que la flexibilité dans la définition de l'état final qui nous a décidé à choisir cette méthode plutôt que les autres méthodes de génération permettant d'obtenir des trajectoires faisables en temps réel. On appréciera en particulier la possibilité de générer plusieurs milliers de trajectoires. Dans le même temps, on restera attentif au fait que la méthode choisie utilise un modèle simplifié du drone et il faudra nécessairement chercher les limites pratiques à partir desquelles ce modèle simplifié diverge trop du drone réel et rend la trajectoire impossible à suivre.

La méthode proposée par Mueller, Hehn et D'Andrea est constituée de deux phases. Une phase de génération de primitives puis une phase de vérification de faisabilité. On va commencer par introduire le modèle utilisé ainsi que les entrées et leurs bornes. On abordera ensuite chacune des deux phases. On finira par se concentrer sur le coût proposé dans ces travaux et on verra qu'il est possible de le lier à l'agressivité des trajectoires.

3.4.4.1 Modèle et entrées :

Dans ces travaux, en reprenant les notations présentées dans la partie 2.3, le modèle de quadrirotor utilisé est le suivant :

$$\ddot{\xi} = -g\, i_3 + \frac{T}{m} \tag{3.1}$$

$$\dot{R} = R\Omega_\times \tag{3.2}$$

On pose finalement $\frac{T}{m} = R\, b_3\, f$ et on définit f et $\Omega = (p, q, r)$ comme étant les entrées de ce modèle. Par rapport au modèle présenté dans la partie 2.3, les auteurs ont choisi de considérer que les vitesses angulaires peuvent être parfaitement suivies ce qu'ils justifient par une mise en parallèle de la faible inertie de rotation des quadrirotors et de leur capacité à produire d'importants moments. On notera que f est la norme de la poussée normalisée par la masse, il s'agit donc d'une accélération. Les auteurs ont choisi de définir les bornes suivantes pour leurs entrées :

$$0 < f_{min} < f < f_{max} \tag{3.3}$$

$$||\Omega|| < \omega_{max} \tag{3.4}$$

On remarquera que f_{min} est positif ou nul ce qui est induit par le sens de rotation fixe des hélices. De leur coté, les vitesses angulaires sont bornées par leur magnitude totale et non par axe ce qui diminue la précision de la borne mais la rend plus simple à vérifier comme on pourra le constater par la suite.

En suivant ce modèle, il faudra donc 9 variables pour décrire précisément l'état du quadrotor : la position, la vitesse et l'orientation.

Notons $\sigma(t) = (\xi, \dot{\xi}, \ddot{\xi}) \quad \in \mathbb{R}^9$ et $\sigma_i(t), \quad i \in [1, 9]$ les composantes individuelles de $\sigma(t)$. Notons t_f la durée souhaitée de la primitive à générer et $\hat{\sigma}_i$ l'état final souhaité avec $i \in \mathbb{I} \subseteq \{1, 2, ..., 9\}$. La trajectoire atteint son but si :

$$\sigma_i(t_f) = \hat{\sigma}_i \quad \forall i \in \mathbb{I} \tag{3.5}$$

Le but du papier est de trouver $f(t), \Omega(t) \quad \forall t \in \mathbb{T} = [0, t_f]$ pour un quadrirotor commençant dans un état initial définit par sa position, sa vitesse et son orientation tout en satisfaisant la dynamique du quadrirotor et les contraintes sur les entrées.

3.4.4.2 Génération de primitives :

On considérera dans cette partie de l'algorithme qu'il est possible de découpler les axes et de planifier les mouvements du quadrirotor sur chaque axe. La partie de vérification de faisabilité permettra de montrer que les trajectoires résultantes de cette simplification sont bien réalisables par le modèle présenté il y a quelques lignes.

Pour chaque axe k, $k \in [1, 3]$, le but dans cette partie est simplement de générer une trajectoire trois fois dérivable qui guide le système d'un état initial à un état final (potentiellement partiellement indéfini) tout en minimisant une fonction de coût C_k. La fonction de coût

choisie par l'auteur fait intervenir le jerk J_k et est la suivante :

$$C_k = \frac{1}{t_f} \int_0^{t_f} J_k(t)^2 dt \tag{3.6}$$

Ce coût sera discutée plus en détail par la suite.

Grâce au principe du minimum de Pontryargin, il est alors possible de trouver l'expression de la trajectoire optimale. Ainsi, la position optimale obtenue au cours du temps $p^*(t)$ est donnée par l'équation suivante :

$$p^*(t) = \frac{\alpha}{120}t^5 + \frac{\beta}{24}t^4 + \frac{\gamma}{6}t^3 + \frac{a_0}{2}t^2 + v_0 t + p_0 \tag{3.7}$$

avec a_0, v_0 et p_0 respectivement les accélérations, vitesses et positions initiales dans l'axe considéré. Pour un état final totalement défini, comme ce sera le cas dans cette thèse, on a :

$$\begin{bmatrix} \frac{1}{120}{t_f}^5 & \frac{1}{24}{t_f}^4 & \frac{1}{6}{t_f}^3 \\ \frac{1}{24}{t_f}^4 & \frac{1}{6}{t_f}^3 & \frac{1}{2}{t_f}^2 \\ \frac{1}{6}{t_f}^3 & \frac{1}{2}{t_f}^2 & t_f \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \Delta p \\ \Delta v \\ \Delta a \end{bmatrix} \tag{3.8}$$

avec

$$\begin{bmatrix} \Delta p \\ \Delta v \\ \Delta a \end{bmatrix} = \begin{bmatrix} p_f - p_0 - v_0\, t_f - \frac{1}{2}a_0{t_f}^2 \\ v_f - v_0 - a_0\, t_f \\ a_f - a_0 \end{bmatrix} \tag{3.9}$$

3.4.4.3 Vérification de faisabilité :

Comme annoncé, une vérification à posteriori de la faisabilité est effectuée. Cette vérification revient à contrôler que la trajectoire qui vient d'être générée est réalisable par le modèle défini en tenant compte des bornes sur les entrées.

Commençons par vérifier la poussée. La trajectoire est réalisable pour cette entrée si et seulement si :

$$\max_{t \in \mathbb{T}} f(t)^2 \leq f_{max}^2 \quad et \tag{3.10}$$

$$\min_{t \in \mathbb{T}} f(t)^2 \geq f_{min}^2 \tag{3.11}$$

Par ailleurs, en appliquant la norme euclidienne à l'équation 3.1, on obtient :

$$f = \left\| \ddot{\xi} - g \right\| \tag{3.12}$$

ce qui, élevé au carré donne :

$$f^2 = \left\| \ddot{\xi} - g \right\|^2 = \sum_{k=1}^{3} (\ddot{\xi}_k - g_k)^2 \tag{3.13}$$

avec $\ddot{\xi}_k$ la composante de l'accélération de la trajectoire générée sur l'axe k et g_k la composante de la gravité sur l'axe k. En combinant, 3.10-3.13, les contraintes de poussée peuvent être vues comme des contraintes sphériques sur l'accélération. En prenant les extremums par axes de 3.13, on obtient les bornes suivantes :

$$\max_{t\in\mathbb{T}}(\ddot{\xi}_k(t) - g_k)^2 \leq \max_{t\in\mathbb{T}} f(t)^2, \quad k \in 1,2,3 \tag{3.14}$$

$$\max_{t\in\mathbb{T}} f(t)^2 \leq \sum_{k=1}^{3} \max_{t\in\mathbb{T}}(\ddot{\xi}_k(t) - g_k)^2 \tag{3.15}$$

$$\min_{t\in\mathbb{T}} f(t)^2 \geq \sum_{k=1}^{3} \min_{t\in\mathbb{T}}(\ddot{\xi}_k(t) - g_k)^2 \tag{3.16}$$

On dérive de ces bornes des contraintes suffisantes pour montrer la faisabilité ou l'infaisabilité de la majorité des trajectoires : si la partie gauche de l'inégalité 3.14 est supérieure à f_{max}, la trajectoire est définitivement infaisable. Au contraire, si la partie droite de 3.15 est inférieure à f_{max} et la partie de 3.16 est supérieure à f_{min}, la trajectoire est définitivement faisable. Finalement, en utilisant le fait que $\ddot{\xi}_k - g_k$ est un polynôme de degré trois pour la trajectoire définie en 3.7, dont on connaît donc les extrema, il est possible de vérifier ces contraintes quasi-instantanément.

Passons à présent à la vérification de la faisabilité par rapport aux vitesses angulaires. En dérivant les équations 3.1 et 3.12, on obtient :

$$J = R\Omega_\times b_3 f + R b_3 \dot{f} \tag{3.17}$$

$$\dot{f} = b_3^\intercal R^{-1} J \tag{3.18}$$

En substituant et en évaluant le produit $\Omega_\times b_3$, on peut voir que le jerk et la poussée fixent deux composantes des vitesses de rotations :

$$\begin{bmatrix} q \\ -r \\ 0 \end{bmatrix} = \frac{1}{f} R^{-1} J \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.19}$$

On remarque que la troisième composante des vitesses de rotation, r, n'affecte pas les déplacements linéaires. En conséquence, elle sera fixée à zéro durant le reste de ces vérifications : $r = 0$.

En élevant l'équation précédente au carrée et en utilisant la norme induite suivante :

$$\left\| \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right\| \leq 1 \tag{3.20}$$

il est alors possible d'obtenir :

$$w_1^2 + w_2^2 \leq \frac{1}{f^2} \|J\|^2 \tag{3.21}$$

On borne le terme de droite de cette équation par un terme que l'on note $\bar{w}^2$:

$$w_1^2 + w_2^2 \leq \frac{1}{f^2} \|J\|^2 \leq \bar{\omega} = \frac{\sum_{k=1}^{3} \max_{t \in \mathbb{T}} J_k(t)^2}{\sum_{k=1}^{3} \max_{t \in \mathbb{T}} (\ddot{\xi}_k(t) - g_k(t))^2} \tag{3.22}$$

En utilisant l'équation ci dessus, la trajectoire est considérée comme faisable pour les vitesses angulaires si $\bar{w}^2 \leq w_{max}^2$. Si ce n'est pas le cas, la trajectoire est considérée comme indéfinie. $\bar{w}^2$ est alors évalué sur des intervalles de plus en plus petits jusqu'à ce que la condition $\bar{w}^2 \leq w_{max}^2$ soit satisfaite sur tous les intervalles où jusqu'à ce que les intervalles soient plus petits qu'une taille d'intervalle minimum défini par l'utilisateur. Dans le premier cas la trajectoire est classifiée comme faisable. Dans le second cas, elle est définie comme infaisable.

3.4.4.4 Agressivité des trajectoires :

A partir des fonctions de coûts par axe 3.6, il est possible de définir une fonction de coût globale :

$$C = \sum_{k=1}^{3} C_k = \sum_{k=1}^{3} \frac{1}{t_f} \int_0^{t_f} J_k(t)^2 dt = \frac{1}{t_f} \int_0^{t_f} J(t)^2 dt \tag{3.23}$$

Or, en récrivant l'équation 3.19, on obtient :

$$f^2 \|\Omega\|^2 = \left\| f \begin{bmatrix} \Delta w_2 \\ \Delta - w_1 \\ \Delta 0 \end{bmatrix} \right\|^2 = \left\| \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} R^{-1} j \right\|^2 \leq \|j\|^2 \tag{3.24}$$

d'où

$$\frac{1}{t_f} \int_0^{t_f} f(t)^2 \|\Omega(t)\|^2 dt < C \tag{3.25}$$

La fonction de coût C borne donc une moyenne du produit des entrées du modèle du quadrirotor, une quantité que l'on peut légitimement choisir comme une approximation de l'agressivité de la trajectoire générée.

Les auteurs signalent qu'il peut être intéressant de classer les trajectoires en fonction de leur coût pour discriminer plusieurs trajectoires qui atteindraient un même objectif. On utilise une approche légèrement différente, on va chercher par dichotomie sur la durée t_f à générer des trajectoires qui ont des coûts équivalents. On espère que cela conférera au vol final une impression de fluidité et que cela nous permette d'obtenir des erreurs de suivi de trajectoire à peu près constantes.

On conclura sur la méthode de génération choisie en la replaçant dans la littérature. Elle est à mi-chemin entre les méthodes ne prenant pas en compte la dynamique et les méthodes

utilisant la platitude des quadrirotors. Ses deux phases, génération et vérification de faisabilité, ressemblent aux deux phases des méthodes ne prenant pas en compte la dynamique lors de la génération (génération et optimisation). La différence principale étant que la deuxième phase est ici une simple vérification et qu'on préférera régénérer une primitive plutôt que de chercher à la rendre faisable. En même temps, la phase de génération est proche des générations utilisant la platitude. Ces dernières dérivent des modèles physiques des contraintes, entre autre sur le jerk, puis minimisent un critère en prenant en compte ces contraintes. De son coté, la méthode proposée minimise directement le jerk pour vérifier à posteriori des contraintes sur l'accélération et la vitesse angulaire.

3.5 Les différents modes de contrôle

On a vu dans la partie précédente que selon les objectifs des différents algorithmes d'évitement d'obstacles et selon les choix faits par leurs concepteurs, le type de commande associée aux algorithmes d'évitement d'obstacles diffère. Ceux qui conçoivent l'évitement comme une fonctionnalité d'urgence qui doit uniquement prendre le pas en cas de défaut du système de navigation vont limiter leurs commandes aux situations d'urgences et ont tendance à utiliser des contrôles saturés. Au contraire, lorsque les algorithmes créés se rapprochent d'algorithmes de navigation, des trajectoires sont générées et le contrôle doit alors consister en du suivi de trajectoires. Ces différences importantes vont nécessairement donner lieu à l'utilisation de contrôleurs de différentes natures. Certains de ces contrôleurs vont être présentés dans les lignes qui suivent en séparant, comme lors de la génération des commandes, les contrôleurs permettant de suivre une trajectoire des autres contrôleurs qui peuvent être très divers.

3.5.1 Contrôleurs sans trajectoire

L'une des manières les plus simples de contrôler automatiquement un drone sans suivi de trajectoires est tout simplement de le contrôler en angle et en poussée, comme le font les pilotes humains et comme cela a été présenté dans la partie 2.4.4. Cette solution a l'avantage de ne pas nécessiter d'estimation de la position comme le demande un suivi de trajectoire en boucle fermée.

En utilisant les transformations présentées dans la partie 2.3.3 ou des transformations équivalentes, il est aussi possible de convertir des commandes en accélérations en commandes composées de l'attitude et de la poussée. Il est probable que la majorité des papiers utilisant une commande en accélération utilisent ce type de transformation avant d'effectuer leur contrôle avec des contrôleurs en attitude et poussée.

Remarque. *On notera dans la phrase précédente le terme probable qui reflète le fait que, pour la majorité des papiers d'évitement d'obstacles, les détails des contrôleurs utilisés ne sont pas donnés et seule une courte et parfois vague explication sur la méthode de contrôle utilisée est disponible. Ceci s'explique par la diversité des tâches à accomplir pour effectuer de l'évitement d'obstacles et par le fait que le contrôle n'est, dans cette optique, qu'un outil et non la finalité.*

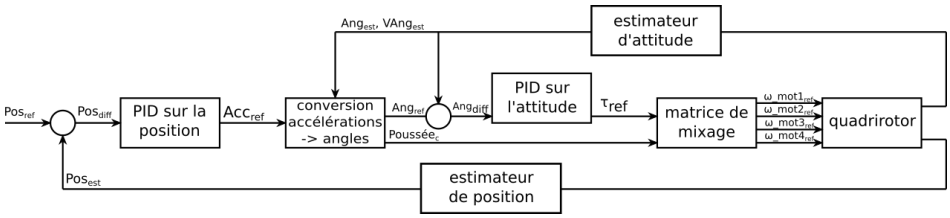


FIGURE 3.10 – Schéma de principe du contrôle en cascade.

On ne peut cependant que regretter cette situation quand on verra dans la partie de résultat sur le suivi de trajectoire (5.1) à quel point les performances des contrôleurs impactent les performances globales des algorithmes d'évitement d'obstacles.

Comme on a pu le voir dans la partie sur les commandes, l'autre manière relativement classique d'évoluer dans l'espace sans définir de trajectoires complètes est simplement de suivre des points de passages qui peuvent être définis directement dans une représentation de l'espace [OHP15] ou être définis comme l'interprétation d'une action sans passer par une représentation de l'espace [Smo+17] ; [SL16]. L'utilisation d'une telle commande peut être interprétée comme l'utilisation d'une trajectoire élémentaire définie par un unique état, l'état final, constitué du point de passage, d'une vitesse nulle et d'une accélération nulle. Une partie des contrôleurs présentés dans la partie suivante et utilisés pour suivre des trajectoires sont donc aussi utilisés pour effectuer le suivi par points de passages.

3.5.2 Contrôleurs pour suivi de trajectoires

Il en existe deux types. Les contrôleurs appartenant au premier type, dont *Geometric tracking control of a quadrirotor UAV on SE (3)* [LLM10] est très certainement l'élément le plus en vue, cherchent directement à convertir la trajectoire en une commande composée des moments et de la poussée qu'il est ensuite possible de transformer en vitesse moteur grâce à la matrice de mixage (voir 2.14).

L'autre type de contrôleur, qui est le plus utilisé, fait intervenir une architecture de contrôle en cascade. Un premier contrôleur, que l'on nommera abusivement contrôleur de haut niveau, génère une accélération linéaire à partir de la trajectoire commandée et de l'état du drone. En utilisant la transformation cinématique (équation 2.18), cette accélération linéaire est convertie en une commande en attitude qui sera l'entrée d'un deuxième contrôleur, de bas niveau. Ce contrôleur de bas niveau correspond au contrôleur en attitude présenté en 2.4.4. Sa sortie est composée de moments dans le repère du drone et d'une poussée qui sont ensuite converties en vitesses moteur grâce à la matrice de mixage. Le schéma 3.10 présente une vue en schéma bloc de la totalité de cette architecture de contrôle.

Comme expliqué dans la partie 2.4.4, le contrôleur en attitude est généralement un PID. D'autres contrôleurs ont été proposés mais ne se sont pas encore démocratisés et ce en particulier pour les vols en extérieur. Plus de détails sont disponibles en 2.4.4.

De l'autre côté, il est possible d'effectuer le contrôle de haut niveau grâce à un PID par axe du repère inertiel mais c'est relativement rarement le cas car de nombreuses autres solutions permettent d'améliorer très fortement la qualité du suivi de trajectoire. La grande tendance pour ce contrôleur de haut niveau est à l'utilisation de contrôleurs basés sur de la commande prédictive (MPC) tel que [Kam+17] ; [KBS16]. Si les MPCs permettent d'obtenir des performances qui vont bien au delà de ce qu'il est possible d'obtenir avec deux simples PID en cascade, ce qui les a fortement démocratisé et ce qui en fait l'une des alternatives couramment utilisée, d'autres contrôleurs permettent d'obtenir des résultats à peu près équivalents comme le contrôleur à retour d'état qui sera présenté dans la partie suivante.

3.5.3 Notre contrôle

3.5.3.1 Notre contrôle par PID

Comme présenté en 3.2, on a fait le choix de générer des trajectoires et donc d'avoir un contrôle par suivi de trajectoire. On vient d'expliquer que le schéma de contrôle le plus classique pour faire du suivi de trajectoire est d'utiliser un contrôleur en cascade et cela nous convient parfaitement pour rester dans l'esprit plug-and-play. On va, en effet, pouvoir utiliser une architecture de contrôle qui comprend un contrôleur de haut niveau fait sur mesure et utiliser les contrôleurs en attitude qui sont disponibles nativement sur la quasi-totalité des drones pour effectuer le contrôle final. Le contrôleur en attitude n'étant donc plus de notre fait, il ne sera pas plus discuté dans cette partie qui se concentrera sur le contrôleur de haut niveau. On signalera simplement que ce choix a une conséquence importance : l'estimation des performances des quadrirotors devra être effectuée avec leur contrôleur en attitude. Quand on sait que la majorité des contrôleurs en attitude contraignent les quadrirotors à des angles de roulis et de tangage inférieur à 30°, cela fait une différence très importante avec leurs capacités sans ces contraintes !

Il reste donc à créer ou à implémenter un contrôleur de haut niveau dont l'entrée sera une trajectoire et dont la sortie sera une accélération. Comme pour la génération de trajectoire, on a fait le choix de commencer par des solutions simples qu'il est possible de mettre en place rapidement. Pour cette raison, on s'est dirigé vers l'implémentation d'un simple PID. Ce contrôleur, s'il est effectivement bien moins performants que d'autres contrôleurs développés ces dernières années comme l'attesteront les résultats des tests de contrôle présentés en 5.1, permet néanmoins d'avoir un suivi de trajectoire suffisamment fiable à basse vitesse pour pouvoir évaluer les performances des algorithmes d'évitement d'obstacles créés. Cela permettra en outre de montrer que, même avec un contrôleur très simple, notre algorithme permet d'éviter efficacement les obstacles à basse vitesse.

On a tout de même légèrement raffiné le PID en ajoutant quelques fonctionnalités. La première consiste à détecter des messages spéciaux d'arrêt d'urgence. Une fois un arrêt d'urgence détecté, le terme de position du PID est mis à zéro et une commande en vitesse, opposée à la vitesse courante du quadrirotor est donnée en entrée du PID classique. Ces modifications ont pour but d'arrêter le drone très rapidement si l'algorithme d'évitement envoie

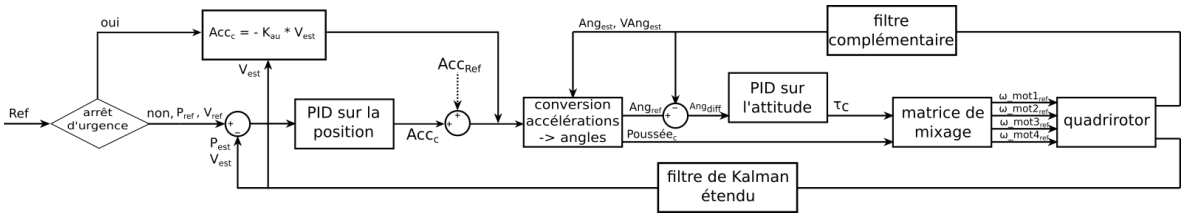


FIGURE 3.11 – Schéma bloc du contrôle lors de l'utilisation des PIDs en cascade.

une commande d'erreur. Dans un deuxième temps, il a fallu ajouter une fonctionnalité qui, en comparant le temps courant et le temps prévisionnel des différents points de la trajectoire, sélectionne l'état que l'on souhaite atteindre dans la trajectoire de commande. Dans un troisième temps, on a ajouté la prise en compte de la vitesse de référence (et non plus simplement de la position de référence). Enfin, on a ajouté une option permettant d'activer ou non la transmission en boucle ouverte de l'accélération sur la trajectoire. Cet ajout est nécessaire pour que mathématiquement, le contrôle ait du sens. Ne pas transmettre cette accélération revient à demander au drone une accélération nulle s'il est parfaitement dans l'état désiré alors même que la trajectoire à ce moment peut comprendre une accélération non-nulle. Bien que mathématiquement nécessaire, cet ajout de l'accélération en boucle ouverte a eu tendance à dégrader les performances du PID. Pour avoir un contrôle correct, et ce malgré l'incohérence mathématique, on a donc finalement remplacé cette accélération par un offset temporel dans la sélection de l'état commandé. Une représentation en schéma bloc de l'architecture de contrôle avec ce contrôleur est proposée sur la figure 3.11.

3.5.3.2 Notre contrôle par retour d'état

On s'est contenté du PID en position présenté précédemment pendant une longue période de la thèse où l'on s'est concentré sur le développement d'autres parties de l'algorithme. Lors des tests effectués pour la présentation du premier algorithme fonctionnel obtenu, il est cependant clairement apparu que l'utilisation de ce contrôleur était l'un des facteurs impactant le plus les performances de l'algorithme. On a alors eu la chance de bénéficier, dans la dernière partie de la thèse, du travail de deux autres personnes du laboratoire, John-Jairo Martinez-Molina et Matthieu Muschinowski, qui ont, pour un autre projet, créé un contrôleur à retour d'état pour faire du suivi de trajectoire avec un quadrirotor. Ce qui va suivre n'a donc pas été développé par mes soins mais nécessite d'être présenté pour que les résultats présentés par la suite aient du sens.

L'état considéré dans ce contrôleur est constitué de la position, de la vitesse et des incertitudes. De plus, et comme pour le PID, ce contrôleur est constitué de trois contrôleurs indépendants pour chaque axe du référentiel inertiel $\mathbb{I}$. Considérons l'axe X, on a :

$$\ddot{x}_c = \ddot{x}_{ref} + K_v * (\dot{x}_{ref} - \hat{\dot{x}}) + K_p * (x_{ref} - \hat{x}) + K_u * \hat{U}_x \tag{3.26}$$

$\ddot{x}_c$ correspond à la commande en accélération en sortie du contrôleur, X_{ref} à la commande de référence de la grandeur X, $\hat{X}$ à l'estimation de la grandeur X, K_v, K_p et K_u aux

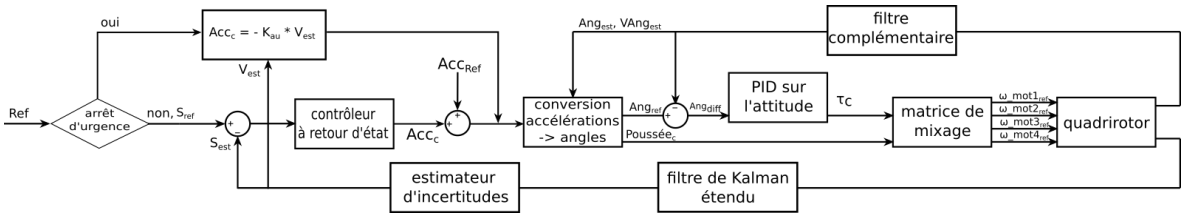


FIGURE 3.12 – Schéma bloc du contrôle lors de l'utilisation du contrôleur à retour d'état.

gains du contrôleur et U_x aux incertitudes selon l'axe X. En notant $X(-1)$ la grandeur X à l'exécution précédente du contrôleur, dt le temps entre l'exécution précédente et l'exécution courante et L_p, L_v et L_u les différents gains de l'observateur, on a :

$$\hat{x} = \hat{x}(-1) + dt * \hat{\dot{x}}(-1) + L_p * (x_{mes} - \hat{x}(-1)) \quad (3.27)$$

$$\hat{\dot{x}} = \hat{\dot{x}}(-1) + dt * \hat{U}_x(-1) + dt * x_c(-1) + L_v * (x_{mes} - \hat{x}(-1)) \quad (3.28)$$

$$\hat{U}_x = \hat{U}(-1) + L_u * (x_{mes} - \hat{x}(-1)) \quad (3.29)$$

Les gains du contrôleur et de l'observateur sont calculés grâce à la commande linéaire quadratique. Une représentation en schéma bloc de l'architecture de contrôle avec ce contrôleur est proposée sur la figure 3.12.

3.6 Description de l'algorithme SOTA

On a présenté dans les sous-parties précédentes les blocs logiciels qui font l'objet d'études particulières lors de la conception d'algorithmes d'évitement d'obstacles. Dans les parties suivantes, on décrira comment on les a articulés pour obtenir des algorithmes fonctionnels.

Les limites des algorithmes ont été présentés précisément dans la partie 3.2. Dans un souci de simplicité, on rappelle tout de même les quelques éléments les plus importants :

— l'objectif est d'atteindre une série de points de passages ordonnés P_i
— l'état en entrée, à 100Hz, possiblement bruité, est composé des positions, vitesses et accélérations linéaires ainsi que de l'attitude
— l'information sur les obstacles sera sous la forme de données de profondeur
— la sortie est une commande en attitude à 100Hz

La figure qui suit, numéro 3.13, présente l'architecture que l'on a choisie pour lier simplement notre représentation 3D, notre génération de trajectoire et notre contrôle dans notre premier algorithme. On nomme cet algorithme SOTA (pour *state of the art*) pour mettre en évidence qu'il s'agit d'un algorithme qui lie ces blocs logiciels de manière relativement simple, sans proposer de concept novateur.

Comme il est possible de le constater sur la figure 3.13, l'algorithme proposé est séparé en trois blocs. Ces trois blocs s'exécutent en parallèle. Si une nouvelle image de profondeur est disponible, dès que sa précédente exécution est terminée, la partie vision se ré-exécute sur la

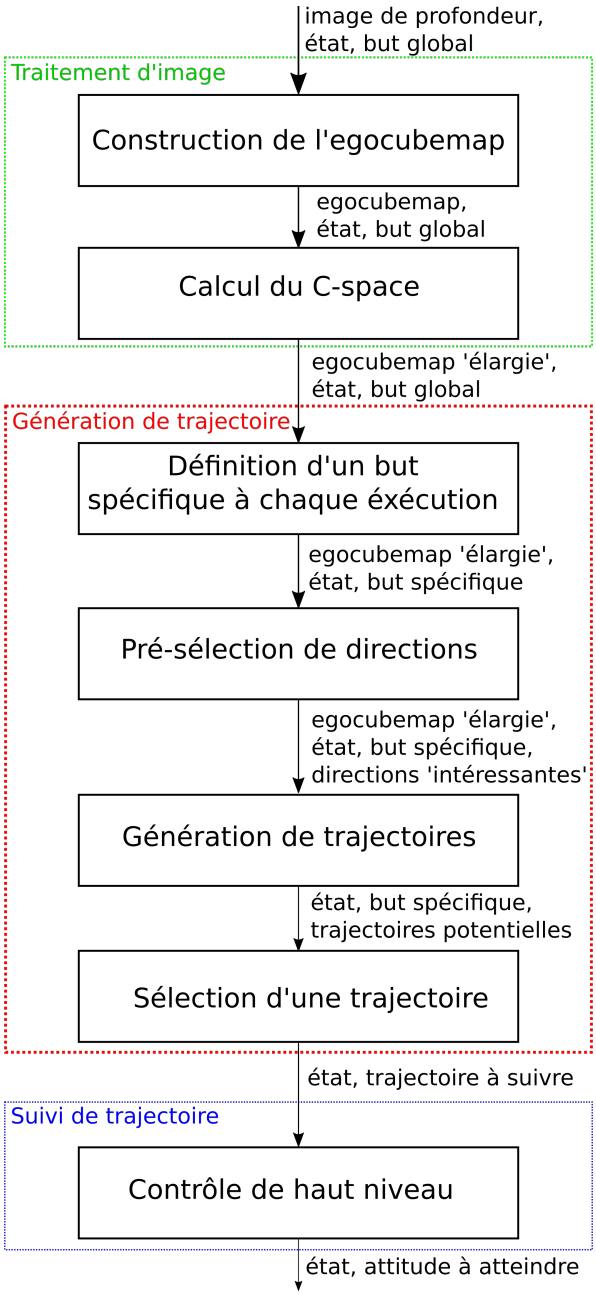


FIGURE 3.13 – Architecture de l'algorithme SOTA.

dernière image disponible. La génération de trajectoire utilise le même genre de mécanisme. Si une nouvelle egocubemap est disponible, dès que sa précédente exécution est terminée, la partie génération de trajectoire se ré-exécute sur la dernière egocubemap disponible. Cette partie de génération de trajectoire peut être plus rapide ou plus lente que la génération de l'egocubemap selon les versions des algorithmes développés. Les détails sur les temps d'exécution des algorithmes sont disponibles dans la partie 5.5. Finalement, le suivi de trajectoire est effectué à la fréquence fixe de 100Hz et ce quel que soit le contrôleur utilisé.

3.6.1 Construction de l'egocubemap

L'egocubemap est construite selon les principes présentés en 3.3.4. On rappelle qu'une transformation de type C-Space, qui augmente artificiellement la taille des éléments observés par la caméra de profondeur, est appliquée durant la construction de l'egocubemap.

Remarque. *Par soucis de simplicité d'utilisation, une fois construite, l'egocubemap est convertie et représentée en coordonnés sphériques. Cette transformation impacte négativement les performances de l'algorithme (voir 3.3.4) mais a le double avantage de simplifier l'écriture des différents algorithmes (ne nécessite plus la gestion de 6 matrices avec des sauts en mémoire pour des points proches géographiquement mais d'une seule avec une organisation relativement simple) et de correspondre à une représentation que l'on a l'habitude d'utiliser ce qui simplifie l'analyse des résultats et leur communication.*

3.6.2 Calcul du but temporaire

Une fois la nouvelle egocubemap générée, on calcule un but, une position à atteindre, temporaire, valide uniquement pendant l'exécution courante de l'algorithme. Il y a plusieurs cas de figure pour ce but temporaire.

Si on est à proximité du point de passage P_{i-1}, on calcule l'angle de lacet nécessaire pour être aligné vers le point de passage P_i. Si on n'est pas en direction de P_i, ce qui est le cas le plus fréquent quand on arrive à P_{i-1}, on transmet directement une commande composée de la position actuelle et de l'angle de lacet nécessaire pour aller en direction de P_i et l'exécution de l'algorithme s'interrompt à cette étape.

Si on n'est pas à proximité de P_{i-1} ou si on est à proximité de P_{i-1} mais en direction de P_i, on fait un second test. Si, lors de l'exécution précédente, aucune trajectoire n'a été générée, par exemple parce que la commande consistait en un nouveau lacet, le but temporaire est P_i. Si, au contraire, une trajectoire a été générée, le but temporaire est une moyenne pondérée entre la position finale de la trajectoire précédente et P_i. En notant Pf_{k-1} la position finale de la trajectoire précédente et G_k le but temporaire courant, on a donc $G_k = \alpha Pf_{k-1} + (1-\alpha)P_i$ avec α un paramètre qu'il conviendra de régler.

Ce but temporaire sert à réduire l'instabilité des trajectoires ce qui tend a créer un vol plus harmonieux, avec moins de changement de directions brusques. L'effet de ce but

temporaire est en particulier visible lorsqu'un obstacle apparaît pile dans l'alignement entre le drone le point de passage P_i. Sans but temporaire, les trajectoires oscillent d'une part et d'autre de l'obstacle, pendant une dizaine d'itérations au maximum, ce qui demande au drone de légers ajustements d'attitudes qui ne sont pas nécessaires. La présence d'un but temporaire permet d'éviter ce comportement mais peut être problématique s'il prend le pas sur le point de passage global. On a ainsi pu constater que l'utilisation de très forts α (> 0.95) générait un suivi trop prononcé vers le but temporaire précédent et réduisait l'utilisation de passages plus efficaces pour aller au point de passage mais invisibles à l'exécution précédente de l'algorithme.

3.6.3 Pré-sélection de directions

On vient de définir la position que l'on cherche à atteindre pour l'exécution courante de l'algorithme G_k. On souhaite maintenant définir des directions qui nous semblent intéressantes pour arriver à cette position.

Pour cela, on va commencer par tronquer l'egocubemap par la distance entre G_k et le quadrirotor. On rappelle que l'egocubemap contient la distance au premier obstacle dans un certain nombre de directions autour du drone. Tronquer l'egocubemap par une distance, revient simplement à vérifier que la valeur de chaque pixel de la cubemap est inférieur à cette distance.

Sur cette egocubemap tronquée, on va calculer la distance de chaque point à G_k. Pour cela, chaque point est reprojeté dans l'espace. Cela nous permet de sélectionner un ensemble de points de la cubemap qui sont les plus proches dans l'espace réel de G_k. Dans la figure 3.14 qui correspond à une représentation sphérique d'une egocubemap, cette zone correspond à 10% de l'ensemble des points et est teintée de bleu alors que la projection de G_k dans la cubemap est représenté par le point rouge.

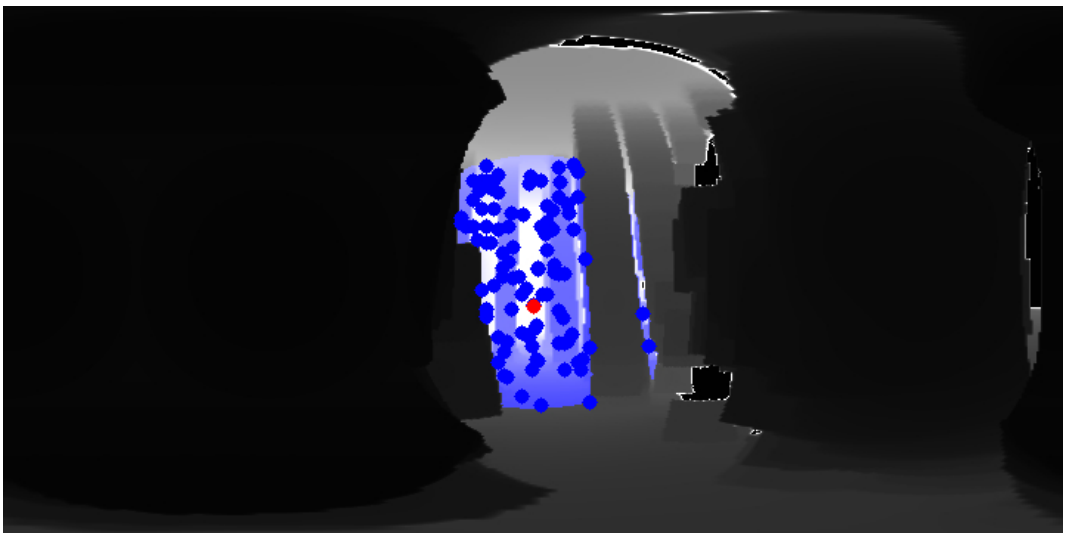


FIGURE 3.14 – Pré-sélection des directions pour $ND = 100$.

Finalement, ND directions seront sélectionnées aléatoirement dans cette zone bleu. Ces directions sont représentées dans la figure 3.14 par des points bleus.

Cette étape permet, d'une manière relativement simple, de sélectionner des directions allant globalement vers le but temporaire G_k tout en conservant une certaine diversité dans la sélection de trajectoire. Sur l'exemple présenté, il y a ainsi plus de 90°d'écart entre les directions les plus éloignées les unes des autres.

Une méthode légèrement plus élégante et fréquemment employée consiste à faire varier la probabilité de sélection des directions en fonction de leur proximité à G_k. Il est ensuite possible de faire varier cette loi de probabilité en fonction de nombreux paramètres (densité de l'environnement, qualité de la progression du drone vers le point de passage P_i sur un horizon temporel plus long ...). La méthode employée ici correspond en pratique à l'utilisation d'une loi de probabilité binaire et il est possible de modifier la taille de la zone d'intérêt en fonction des mêmes paramètres que lors de la modification de la loi de probabilité.

3.6.4 Génération de trajectoires

Commençons par noter D_l avec $l \in [1, ND]$, les directions choisies précédemment. On va maintenant chercher la distance maximale à laquelle il est possible de générer une trajectoire valide dans chacune des directions. Pour qu'une trajectoire soit valide, il faut à la fois quelle soit dynamiquement réalisable par le drone et quelle ne donne pas lieu à des collisions avec l'environnement.

Remarque. *Cette recherche est effectuée direction par direction et est indépendante pour chaque direction, il est donc possible et conseillé de paralléliser les différentes directions.*

On va maintenant décrire le processus de recherche de trajectoire pour une direction D_l. Par construction de l'egocubemap, on sait que la distance maximale à laquelle il est possible d'aller sans rencontrer d'obstacle dans cette direction correspond à la distance stockée dans la cubemap. On obtient, grâce aux transformées permettant de passer des différentes représentation de l'environnement aux coordonnées spatiales présentées dans la partie 3.3.4, la position du point correspondant dans le repère du drone Pf_{ld}.

État final de la trajectoire : L'objectif annoncé étant de générer les trajectoires les plus longues possibles, on devrait donc tenter d'atteindre Pf_{ld}. On souhaite cependant conserver une marge supplémentaire pour que le point final de la trajectoire ne soit jamais collé au C-Space. Ceci est illustré sur la figure 3.15 avec la marge de distance notée ΔD et la position que l'on vise finalement Pf_{l1} L'intérêt principal de cette distance supplémentaire par rapport à l'obstacle est de limiter la proximité du dernier point de la trajectoire aux obstacles car ce point sera un point d'arrêt si le drone l'atteint. Cela permet aussi d'être plus loin de la zone de sécurité autour des obstacles, où le vol est interdit, ce qui laisse plus de chance au drone de percevoir des passages alternatifs. Fixer une marge de distance ΔD nulle crée un phénomène de vision tubulaire et le drone va plus souvent se bloquer dans des minimums locaux qu'avec

une marge non nulle. Pour le reste de l'état final, ou fait le choix de définir une vitesse et une accélération nulle. Ce choix, d'un état final statique, a été fait pour être sûr qu'en cas de crash logiciel, d'absence de solution de la part de l'algorithme ou de crash physique de la carte de calcul, les dernières informations transmises à la carte de vol permettent, à chaque instant, d'amener le drone a un état stable et qui donc permettent d'éviter les collisions.

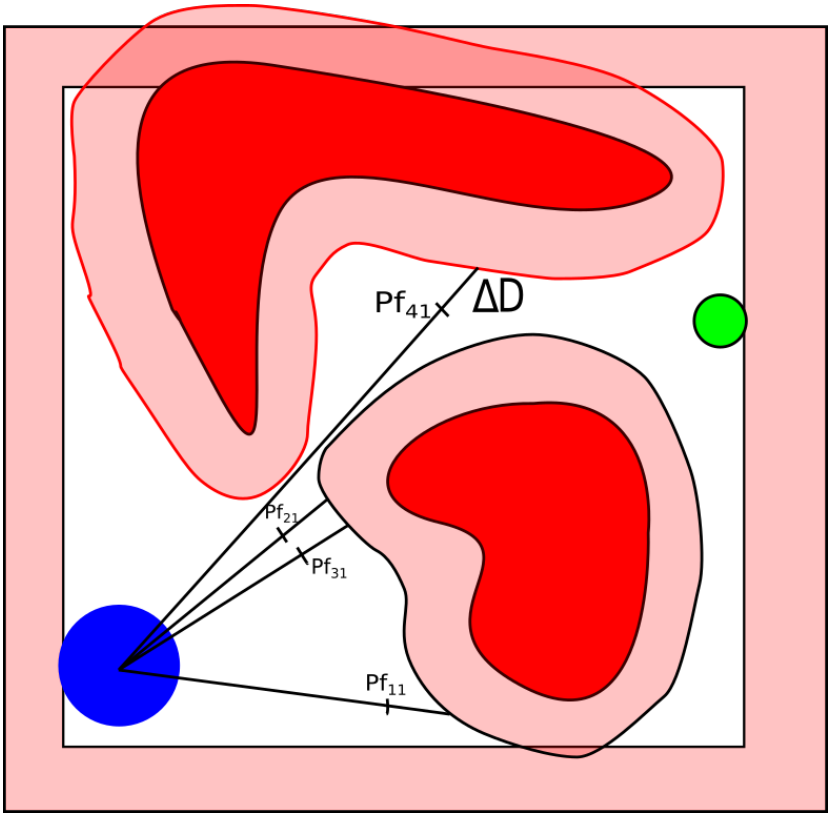


FIGURE 3.15 – Distance tampon par rapport aux obstacles.

Remarque. *On notera rapidement qu'utiliser un état final statique n'est pas la seule solution pour obtenir des garanties en cas de crash logiciel. Quand certains papiers décident de ne pas traiter ce problème parce que la contrainte supplémentaire est très importante et réduit grandement les vitesses de vol atteignables, au moins deux autres travaux prennent en compte cette problématique.* LIU *et al. [Liu+16] utilise une marge de sécurité identique à la marge ΔD présentée précédemment pour créer une zone de freinage s'ils ne sont pas capable d'identifier de nouvelles trajectoires valides. On notera que cela a l'inconvénient principal de devoir laisser une marge importante (cette marge est de l'ordre de 50cm dans nos travaux et de 5m dans leurs travaux).* WATTERSON *et* KUMAR *[WK15] ont de leur coté proposé une double génération de trajectoires. Leur algorithme génère une trajectoire qui permet de s'approcher du but et une seconde trajectoire, d'urgence, qui permet de s'arrêter s'il n'est plus possible de continuer sur la trajectoire principale.*

État initial de la trajectoire : On a donc défini l'état final de la trajectoire à générer. Il reste à définir l'état initial et son coût, donc son agressivité (que l'on peut obtenir par

dichotomie sur la durée de la trajectoire comme précisé dans la partie 3.4.4). Si le drone est en train de suivre une trajectoire lors de l'exécution de l'algorithme, l'état initial est défini comme l'état théorique dans lequel serait le drone s'il avait parfaitement suivi la trajectoire qu'il a en référence. On a fait ce choix pour limiter les dérives que peuvent générer les incertitudes sur le modèle du drone ainsi que l'impact des perturbations. Ce choix permet aussi d'augmenter la continuité des trajectoires générées et de ne pas être influencé par le bruit sur l'estimation de l'état du drone. Si le drone n'est pas en train de suivre une trajectoire, l'état initial est simplement l'état du drone.

Agressivité de la trajectoire : Dernier paramètre à définir, l'agressivité des trajectoires. On souhaite la faire varier pour autoriser différents régimes de vol en fonction de l'environnement. On souhaite que le drone puisse avoir un vol qui paraisse relativement doux s'il ne perçoit que peu d'obstacles et qu'il peut les éviter grâce à des mouvements sans à-coup mais on souhaite dans le même temps qu'il puisse effectuer des trajectoires au plus proche de ses limites physiques si nécessaire pour éviter un obstacle. On va donc définir une agressivité maximale $Aggr_{max}$ proche des limites de ce que le drone peut effectuer et une agressivité minimale $Aggr_{min}$ qui correspond au régime de vol que l'on souhaite obtenir. Avant de détailler le processus de génération de trajectoires, il est intéressant de remarquer que plus l'agressivité des trajectoires générées augmente, plus la trajectoire sera rectiligne. Pour comprendre de manière très schématique cet effet, on peut considérer un instant que l'agressivité défini une accélération maximale. Si l'on diverge du chemin optimal entre deux points, qui est la droite qui les lie, on va pouvoir plus rapidement retourner sur ce chemin si l'on se soumet à de fortes accélérations que si on se limite à des accélérations plus faibles.

Première génération : Lors de la génération des trajectoires, on va donc commencer par voir s'il est possible d'atteindre Pf_{l1} à $Aggr_{max}$. Pour cela, on va vérifier d'une part s'il est possible de générer une trajectoire avec l'agressivité désirée tout en restant dans les limites de faisabilité du drone et d'autre part si la trajectoire générée ne rentre pas en collision avec l'environnement. La vérification de faisabilité est réalisée directement lors de la génération de la trajectoire et les collisions seront vérifiées en projetant des points de la trajectoire dans l'egocubemap. On notera simplement que pour améliorer la performance de l'algorithme, la vérification des collisions est effectuée en commençant par les points les plus éloignés de la trajectoire.

Recherche de la meilleure agressivité : Si la trajectoire est réalisable, on va chercher par dichotomie quelle est l'agressivité la plus faible $Aggr_l$ avec la laquelle il est possible d'atteindre Pf_{l1}. La trajectoire Pf_{l1} à $Aggr_l$ est alors conservée en mémoire comme la meilleure trajectoire pour la direction D_l.

Modification du point final : S'il est impossible d'atteindre Pf_{l1} à $Aggr_{max}$, il sera impossible d'atteindre Pf_{l1} à toute agressivité de l'intervalle $[Aggr_{min}, Aggr_{max}]$. On va alors

tenter d'atteindre Pf_{l2} avec $d(Pf_{l2}) = \beta\, d(Pf_{l1})$. La même recherche que pour Pf_{l1} est effectuée pour savoir s'il sera possible d'atteindre Pf_{l2} ou non. Ce processus est répété jusqu'à ce qu'une trajectoire soit faisable ou jusqu'à ce que la longueur de la trajectoire soit inférieur à une longueur de trajectoire minimale L_{min}. Ce processus est illustré pour les points Pf_{4*} grâce à la figure 3.16.

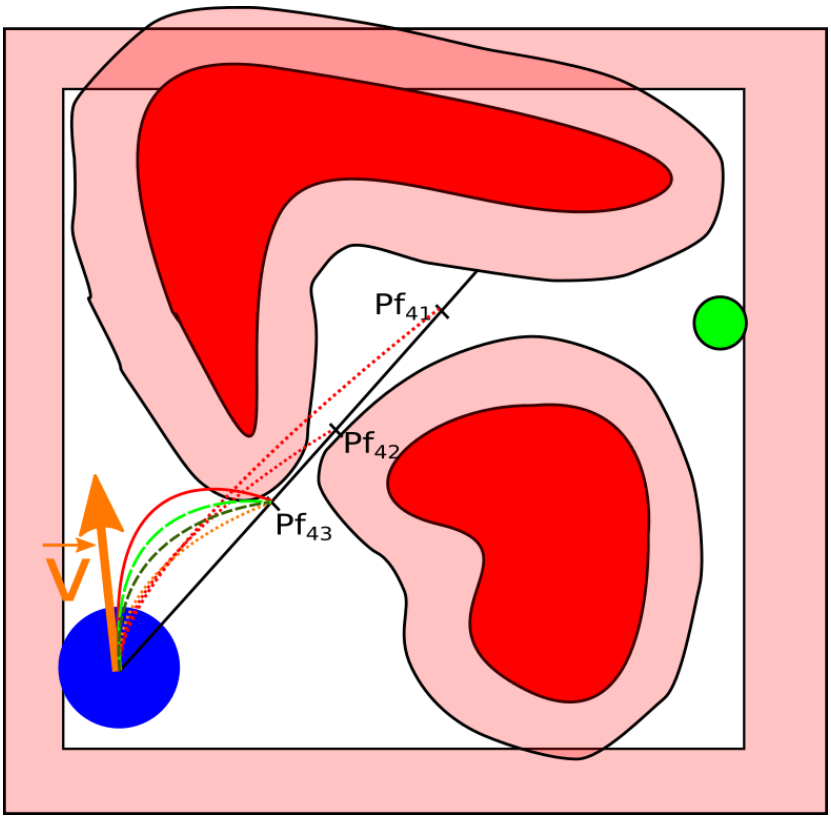


FIGURE 3.16 – Schéma de génération des trajectoires selon une direction.

3.6.5 Sélection de la meilleure trajectoire

Suite à la génération de trajectoires présentée précédemment, on a maintenant jusqu'à ND trajectoires à évaluer pour ensuite en sélectionner une unique. On va simplement associer un coût C_l à chaque trajectoire et la trajectoire ayant le coût le plus faible sera la trajectoire que l'on suivra. Ce coût dépendra de la distance du point final de la trajectoire Pf_l au but temporaire G_k, de l'agressivité de la trajectoire $Aggr_l$ et de la longueur de la trajectoire $d(Pf_l)$. La formule exacte de calcul du coût proposée en 3.30 a été obtenue empiriquement et permet de sélectionner une trajectoire qui propose un bon compromis entre ces différents éléments. On cherche à minimiser les termes du numérateur et à maximiser les termes du dénominateur. On signalera rapidement que l'ajout de la longueur de trajectoire dans cette formule à pour idée de privilégier une trajectoire longue qui dévie du but plutôt qu'un très léger gain dans l'axe du but. En pratique, on constate que cela permet d'éviter de se bloquer dans des minimums locaux.

$$C_l = \frac{Aggr_l * ||Pf_l - G_k||^2}{d(Pf_l)^{0.5}} \tag{3.30}$$

Dans le cas où aucune trajectoire valide n'est transmise par le bloc logiciel précédent, on vérifie que la trajectoire précédente soit toujours faisable. Si c'est toujours le cas, ce qui est le cas le plus courant, on quitte l'algorithme sans envoyer de nouvelle commande. Si ce n'est pas le cas, ce qui peut arriver lorsqu'une perturbation a fait dériver le drone de sa trajectoire de référence ou en présence d'objets dynamiques, une commande d'arrêt d'urgence dont l'objectif est de stopper le drone le plus rapidement possible est transmise au bloc de contrôle.

3.6.6 Suivi de trajectoire

Dans le cas où une trajectoire a été trouvée dans le reste de l'algorithme, il reste à la suivre. Le suivi est réalisé grâce à l'un ou l'autre des contrôleurs proposés en 3.5.3. On signalera simplement que la totalité de la trajectoire est transmise au contrôleur. Cela plus le fait que le dernier état de la trajectoire soit statique permet d'ajouter un élément de sécurité supplémentaire à l'algorithme.

Dans le cas où aucune commande n'a été trouvé, l'arrêt d'urgence correspond à une commande en vitesse qui est l'opposée de la vitesse courante du drone. On a donc :

$$\vec{v_c} = -\vec{v} \tag{3.31}$$

3.7 Algorithme basé sur le filtrage spatial des trajectoires

Si on reprend le schéma de fonctionnement de l'algorithme précédent, disponible en figure 3.13, et qu'on cherche à évaluer le nombre de travaux proposés sur chacun des blocs logiciels ou sur des fonctionnalités équivalentes aux blocs décrits précédemment, on se rend compte que certains de ces blocs font l'objet de nombreuses publications tandis que d'autres ne sont quasiment jamais traités. Les blocs isolés dans les sous parties 3.3, 3.4 et 3.5, qui correspondent à la représentation du monde, à la génération de trajectoire et au suivi de trajectoire font l'objet de nombreux papiers. Au contraire, la stratégie de génération des trajectoires et l'étude des fonctions de coût sont beaucoup moins traités. L'idée que j'ai développée en fin de thèse est d'utiliser les mêmes blocs que dans l'algorithme précédent pour les parties très discutées, de me concentrer sur l'amélioration des points habituellement peu discutés et de voir quels gains en performances il est possible d'obtenir.

J'ai dans un premier temps tenté de créer un algorithme avec une stratégie de génération de trajectoire basée sur des techniques de type monte-carlo [RC13]. J'ai tenté de sélectionner un premier groupe de points au hasard dans tout l'espace vide défini par la cubemap, de calculer le coût de chacune des trajectoires qui permettent d'aller à ces points puis de répéter ces opérations en sélectionnant de nouveaux points en fonction des coûts calculés. Ce

type d'approche a déjà été appliqué à d'autres type de navigation autonome notamment à travers des filtres à particules [Gus+02] mais jamais en vol sur un quadrirotor. L'émergence de cartes graphiques embarquées devrait pourtant permettre d'utiliser ce type de méthode avec un drone. Alors même que ce type de techniques fonctionne pour d'autres robots, assez étonnamment, je n'ai pas réussi à obtenir dans un temps correct de résultats suffisamment encourageants pour explorer cette option plus en détails. J'estime que ce qui m'a posé problème a été de trouver un compromis entre l'exploration de l'espace et la vitesse de convergence vers une solution. Je pense tout de même qu'en y investissant plus de temps, une telle méthode devrait fonctionner. Il faudrait alors la comparer à la méthode basique présentée plus tôt pour estimer l'amélioration qu'elle apporte.

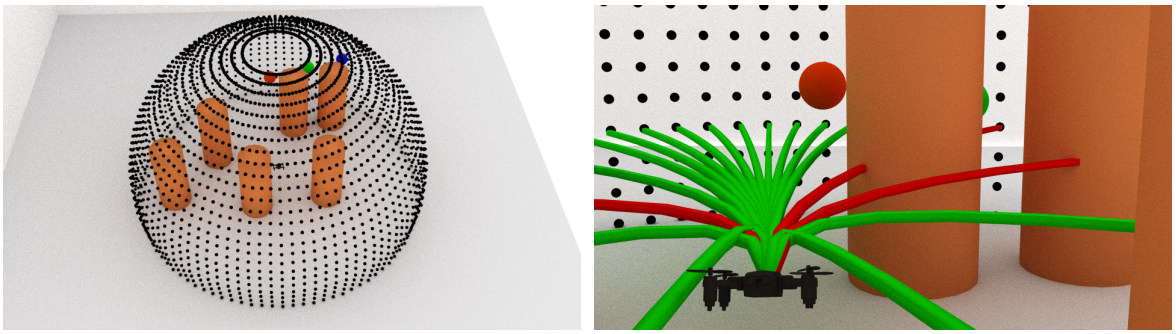
Suite à cet échec, j'ai tenté une autre stratégie de génération de trajectoires. Comme cela m'avait été suggéré, j'ai essayé de créer une représentation de l'espace qui prenne en compte la dynamique du drone . Si je n'ai pas vu comment le faire, cela m'a par contre inspiré pour proposer l'espace des trajectoires proposé. L'idée est de générer des trajectoires qu'il sera possible de remapper sur une représentation sur laquelle il sera possible d'appliquer un post-traitement. Ce post traitement sera en quelque sorte une fonction de coût avancée des trajectoires générées. On va donc créer une représentation qui permette de résoudre un des problèmes constaté sur l'algorithme précédent.

Analysons justement l'algorithme précédent. On constate que celui-ci fait passer le drone au ras des obstacles agrandis par le C-space. C'est l'un des défauts principaux de l'algorithme précédent. Pourquoi prendre le risque de raser un obstacle et de le toucher en cas de perturbations si c'est le seul obstacle de l'environnement ? N'est-il pas souhaitable de prendre plus de marge ? Il faudrait donc augmenter la talle du C-space dans ce cas. Mais au contraire, dans les environnements très denses, on souhaiterait réduire la taille du C-space et réduire la vitesse de vol au strict-minimum pour que le drone puisse doucement passer proche des obstacles. Au premier abord, une solution serait de simplement moduler la taille du C-space et l'agressivité. Seulement, quelles que soient les modifications que l'on fait sur la taille du C-space, avec la méthode précédente, la trajectoire choisie continuera de frôler les obstacles **perçus** donc agrandis. Cela nous amène à la deuxième conséquence de passer au plus proche des obstacles perçus : de toutes petites erreurs de suivi de trajectoire à proximité d'obstacles vont pousser le drone à l'intérieur d'une zone qu'il considère comme interdite et il va alors chercher à s'arrêter. Ce comportement crée des à-coups qui donnent au vol une allure heurté et qui vont perturber les estimations issues des différents capteurs. On a donc voulu créer un système de génération de trajectoire qui permet de s'éloigner des obstacles **perçus** donc s'éloigner des obstacles après agrandissement par le C-space.

Pour résoudre le problème précédent, on a décidé de générer des trajectoires équiréparties sur une surface de l'espace. En appliquant un post-traitement qui prendra en compte les trajectoires voisines à chaque trajectoire, cela revient à effectuer un filtrage spatial des trajectoires générées et cela permet d'obtenir, à moindre coût, de l'information sur l'espace à proximité de chaque trajectoire.

Pour tester le concept, la génération de trajectoires a été effectuée sur une sphère en la décrivant à l'aide des coordonnées sphériques comme il est possible de le voir sur les figures 3.17

avec en (a) une vue de loin et en (b) une vue de juste derrière le drone. La résolution angulaire que l'on a choisie est de 4°. Pour explorer toute la sphère, il faudra donc générer des trajectoires dans environ 4000 directions et il est possible d'agréger toutes ces générations dans une matrice dont les lignes correspondent à une latitude constante et les colonnes à une longitude constante. On obtient alors une matrice de 90 pixels par 45 pixels que l'on appellera matrice de coûts. Notons R le rayon de la sphère, dans le référentiel du drone et en coordonnées sphériques, on va donc tester toutes les trajectoires dont les positions finales Pf seront égales à : (D, θ, ϕ) avec $\theta \in [-\Pi, \Pi/45, \Pi[$ et $\phi \in [-\Pi/2, \Pi/45, \Pi/2[$. On replace ces coefficients dans la matrice de coûts dont on note les coordonnées (x, y) en posant : $x = 45 + \frac{45\theta}{\Pi}$ et $y = 22 + \frac{45\theta}{\Pi}$.



(a) (b)

FIGURE 3.17 – Répartition des trajectoires générées dans l'espace.

Remarque. *On notera que pour obtenir une meilleure répartition dans l'espace des trajectoires générées, une cubemap, ou encore mieux, une cubemap avec coordonnées sphériques, sont plus adaptées qu'une simple répartition sphérique [Bou16] et permettent de réduire de près de moitié le nombre de trajectoires nécessaires pour couvrir tout l'espace. Cela améliorerait aussi l'homogénéité dans l'espace du post-traitement et les résultats du filtrage spatial qui suivra ne pourront aussi qu'être meilleurs. Cela complexifie cependant les post-traitements ce qui ralentirait le développement pour un gain qui est plus de l'ordre de l'ingénierie que de la recherche.*

Pour chaque point final, on recherchera la meilleure agressivité par dichotomie exactement comme dans l'algorithme précèdent. Le coût associé à chaque point final sera finalement calculé en utilisant l'inverse de la formule utilisée dans l'algorithme précédent (équation 3.30) et en associant un coût nul aux trajectoires impossibles. Pour simplifier la lecture, cela donne donc la formule de coût suivante :

$$C_l = \frac{d(Pf_l)^{0.5}}{Aggr_l * ||Pf_l - G_k||^2} \tag{3.32}$$

Au numérateur figure un terme que l'on cherche à maximiser et au dénominateur des termes que l'on cherche à minimiser. On prendra soin de s'assurer que la distance au goal, $||Pf_l - G_k||$, ne soit pas nul et on la substituera par un terme proche de zéro si tel est le cas. On peut remarquer que le terme se basant sur la longueur des trajectoires, $d(Pf_l)$ semble inutile vu

l'algorithme décrit jusqu'à ce point du manuscrit mais son utilité apparaîtra clairement dans quelques lignes.

Après avoir effectué les générations de trajectoires pour tous les points précédemment définis, on dispose, d'une matrice de coûts sur laquelle on va souhaiter effectuer des post-traitements. Une telle matrice de coûts est représentée en bas à gauche dans la figure 3.18 qui servira de visuel pour l'explication du post-traitement et de son impact sur la trajectoire sélectionnée. Sur les matrices de coûts, le pixel le plus blanc correspond au coût le plus élevé, c'est à dire à la meilleure trajectoire. En haut à droite de cette figure, une vue schématique et de dessus de la situation du drone est présentée. Sur cette vue, le drone est représenté par un cercle gris foncé et son orientation est indiquée par un trait rouge. Le but global est entouré en rouge et labelisé *target*, les obstacles sont représentés par des cercles gris clair. Les trajectoires anciennes et courante du drone sont représentées par des traits bleu, plus clair pour les anciennes trajectoires et foncé pour la trajectoire courante, et l'egocubemap dans le plan du drone par les traits marrons. Trois annotations supplémentaires ont été ajoutées. Un obstacle est entourée de rouge, il s'agit d'un obstacle situé entre le drone et le but global et il s'agit d'un obstacle que l'on souhaite à présent contourner. Cet obstacle est surligné en rouge dans la représentation équirectangulaire de l'egocubemap visible en bas à droite de la figure.

Le post-traitement consiste tout simplement à appliquer un filtre gaussien à la matrice de coûts. Sur la figure 3.18, on obtient alors la matrice de coûts filtrée qui est représentée en haut à gauche. Pour rappel, en notant $C[x, y]$ le coefficient de la ligne x et de la colonne y de la matrice de coûts non filtrée, en notant $Cf[x, y]$ ce même coefficient dans la matrice filtrée, en notant Ks la taille du noyau gaussien utilisé et $K[i, j]$ le coefficient du noyau de la ligne i et de la colonne j, on a :

$$K[i,j] = \alpha * \exp \frac{-(i-(Ks-1)/2)^2}{2\sigma^2} * \frac{-(j-(Ks_1)/2)^2}{2\sigma^2}, \quad \forall i,j \in [0, Ks] \tag{3.33}$$

$$Cf[x,y] = \sum_{i=0}^{Ks} \sum_{j=0}^{Ks} C[x-i, y-j]\, K[i,j] \tag{3.34}$$

α est un facteur de normalisation qui permet que la somme de tous les éléments de K soit égale à 1 et σ est la déviation standard de la distribution gaussienne. On notera finalement que la gestion des bords lors de ce filtrage par convolution est effectué par repliement (wrapping en anglais).

La figure 3.18 permet de mettre en évidence la marge de sécurité supplémentaire par rapport aux obstacles qu'apporte le filtrage spatial. Dans la matrice de coûts non filtré, le coût maximum est entouré en orage et est à proximité directe des trajectoires impossibles dues à l'obstacle surligné en rouge dans la représentation de l'egocubemap. Dans la matrice de coûts filtrée, le coût maximum est entouré en vert et les traits verts et oranges tracés permettent d'observer clairement que ce coût correspond à une trajectoire qui ira dans une direction plus éloignée de l'obstacle que la direction qui aurait été sélectionnée dans la matrice de coûts non filtrée. Sur la vue de dessus de la situation, en haut à droite de la figure, les zones surlignées en vert et orange permettent de se rendre compte de cette marge de sécurité supplémentaire. Sans filtrage spatial, la trajectoire sélectionnée serait la trajectoire qui passe au plus proche

de la sécurité issue du C-Space ne laissant comme marge que la zone surligné en orange. Avec le filtrage spatial, toute la zone verte correspond à une marge de sécurité supplémentaire.

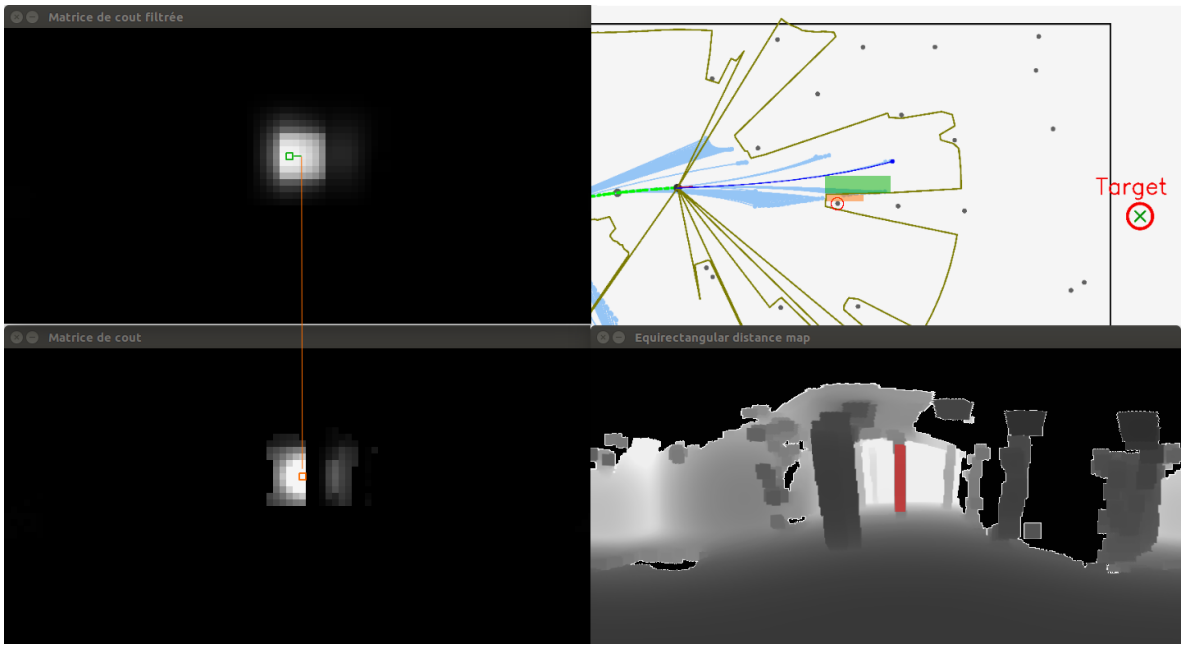


FIGURE 3.18 – Matrice de coûts filtrée (haut gauche) et non filtrée (bas gauche) pour une situation décrite par une vue schématique de dessus (haut droite) et la représentation équirectangulaire de l'egocubemap correspondante (bas droite).

L'algorithme décrit ci-dessus pourrait être suffisant mais n'utiliser qu'une unique longueur de trajectoire semble restrictif. Le procédé décrit ci-dessus est donc appliqué deux fois : une fois à 0.8 fois la distance maximale de perception et une seconde fois, à 0.3 fois la distance maximale. Cela permettra de gérer de la navigation dans des environnements denses. On prendra soin de diminuer la résolution angulaire de notre génération sphérique pour cette longueur plus courte ce qui permettra de réduire le nombre total de trajectoires à générer. En conséquence, on réduira la taille du filtre gaussien sur la matrice de coûts issue de ces générations de trajectoires plus courtes. On notera que, dans le calcul de coût, le terme se basant sur la longueur des trajectoires retrouve son utilité et permettra une nouvelle fois de privilégier les trajectoires les plus longues.

Finalement, la trajectoire choisie correspondra à celle dont le coût sera maximal après prise en compte des coûts présents sur les deux matrices de coûts filtrées et dont le coût avant filtrage est non nul. En reprenant les notations et constantes définies dans les paragraphes précédents et en notant $\wedge$ l'opérateur **et**, l'équation 3.35 permet de trouver les coordonnées $[x_{max}, y_{max}]$ correspondants à la meilleure trajectoire issue d'une matrice de coûts. Pour deux, ou plus, matrices de coûts, il suffit de comparer la meilleure trajectoire de chaque matrice et de conserver celle qui possède le coût le plus élevé.

$$\forall x \in [1, 90]\,, \quad y \in [1, 45]\,, \quad [x_{max}, y_{max}] = Arg_{[x,y]}\left(Max(Cf\,[x, y]) \wedge C\,[x, y] \neq 0\right) \quad (3.35)$$

3.8 Extension n°1 : Gestion des minimums locaux

Lors de la phase de tests sur le benchmark proposé dans le chapitre 4, on a remarqué la présence de nombreux minimums locaux. Lorsque la taille de ces minimums locaux est inférieure à la longueur des trajectoires générées, l'algorithme présenté ci-dessus est suffisant. Au contraire lorsque ces minimums locaux sont de mêmes tailles ou plus grands que les trajectoires générées, il est nativement impossible de les gérer. C'est un problème qui est commun à toutes les méthodes de navigation se basant sur une représentation locale de l'environnement à horizon fini. L'idée de cette extension a été de gérer ces minimums locaux en appliquant directement **un filtre sur la matrice de coûts**.

Dans l'algorithme présenté précédemment, il y a toujours sélection d'une nouvelle trajectoire. Cependant, si aucune trajectoire n'est faisable vers le but, le drone va revenir sur ses pas. Il est possible de repérer ce comportement en comparant la distance entre le point final de la trajectoire précédente au point de passage et le point final de la dernière trajectoire calculée au point de passage. Si le résultat s'est trop détérioré, cela signifie que le drone n'a pas trouvé de passage et qu'il y a un donc un minimum local à proximité du dernier point de la trajectoire précédente.

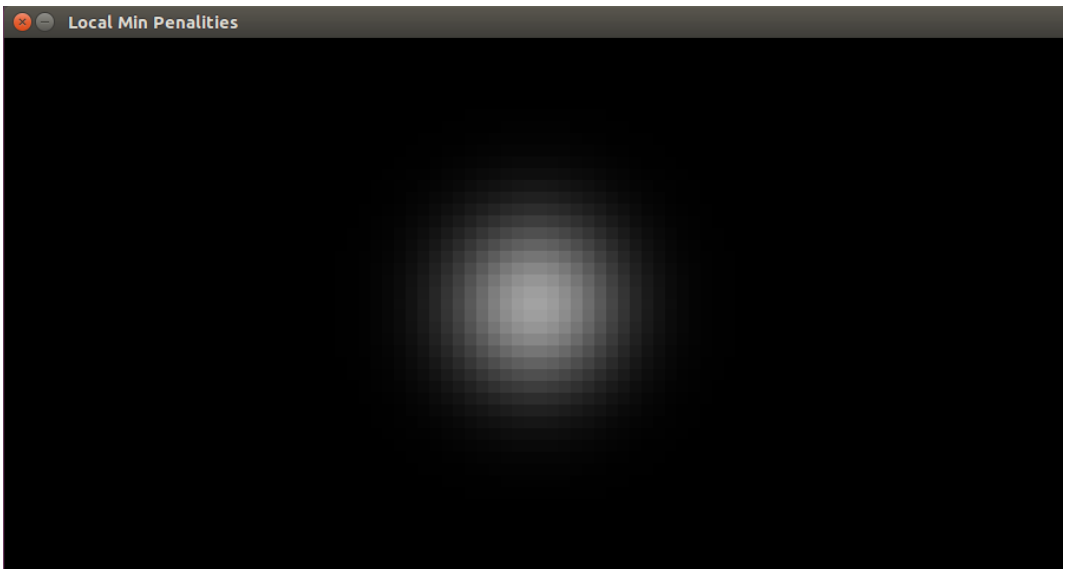


FIGURE 3.19 – Masque pour prise en compte des minimums locaux.

Une fois ce minimum local repéré, il reste à ajouter l'information aux matrices de coûts. On fait cela en soustrayant la distribution gaussienne visible en figure 3.19 aux matrices de coûts. On prend soin à bien centrer cette distribution dans la direction du minimum local. On notera que la variance de cette distribution influe sur la largeur du cône de l'espace que l'on cherche à éviter. Il pourrait être intéressant de la faire varier en fonction de la distance au minimum-local mais pour gagner en temps de calcul, la distribution gaussienne est pré-calculée et il est donc impossible de faire varier cette variance. On multiple par contre l'ensemble de la gaussienne par un facteur qui dépend de la distance au minimum local.

3.9 Extension n°2 : Gestion des obstacles dynamiques

Au vue des performances de l'algorithme proposé et de la facilité d'y ajouter des extensions comme l'extension des minimums locaux permet de le constater, on a souhaité étendre encore les capacités de l'algorithme via une nouvelle extension. Puisque l'évitement d'obstacles dynamiques semble être une des extensions fréquemment considérée ces dernières années pour les drones, c'est l'extension que l'on a essayé d'ajouter.

L'idée de cette extension est très simple : on commence par estimer si des éléments de l'environnement sont en mouvement puis on pénalise les trajectoires dans les directions où des mouvements ont été détectés et dans les directions où il devrait y avoir du mouvement dans les secondes à venir.

L'estimation des mouvements de l'environnement est effectué en calculant le flux optique sur des images de couleur issue d'une caméra dirigée vers l'avant du drone. Dans notre cas, et puisqu'on utilise un capteur RGBD pour obtenir nos images de profondeur, ces images sont issues de la caméra associée au capteur de profondeur utilisé dans le reste de l'algorithme. On rappelle que le calcul du flux optique prend deux images ordonnées en entrée et correspond à l'estimation des déplacements des pixels pour passer d'une image à l'autre. Mathématiquement, en notant $F_o(x, y) \in \mathbb{R}^2$ le flux optique du pixel de la ligne x et de la colonne y, $prev(x, y)$ le pixel situé aux même coordonnées dans l'image de référence la plus ancienne et $next(x, y)$ le pixel situé sur la ligne x et de la colonne y dans l'image de référence la plus récente, on a :

$$prev(x, y) \sim next(x + F_o(x, y)[0], y + F_o(x, y)[1]) \tag{3.36}$$

On utilise la méthode de Farnebäck[Far03] pour calculer le flux optique. Cette méthode est la méthode la plus classique pour obtenir un flux optique dense. Elle est directement disponible dans la librairie de traitement d'image OpenCV. On prendra soin de sous-échantillonner les images utilisées lors du calcul de flux optique pour ne pas trop impacter le temps de calcul total de l'algorithme. L'algorithme de Farnebäck comprend un filtre moyenneur que l'on a choisi de grande taille ce qui diminue le bruit sur le flux optique calculé mais donne un flux optique peu précis. Ce compromis nous convient puisque notre objectif est d'éviter les directions générales dans lesquelles il y a du mouvement et non de détecter un mouvement précis.

Grâce à ce flux optique, il est à présent possible d'extraire les zones de l'image dans lesquelles il y a du mouvement. Pour cela, il suffit de filtrer le flux optique en fonction de sa norme. Une norme faible signifie qu'il n'y a pas eu de mouvements entre les deux images d'entrée du flux optique pour le pixel considéré. Au contraire, une norme élevée signifie que l'algorithme de flux optique estime que le pixel considéré à fortement bougé entre les deux images d'entrée. La matrice $\|F_o\|$ permet donc de mettre en évidence, sur l'image la plus ancienne, les pixels qui ont le plus bougé. Ce n'est cependant pas l'information que l'on recherche : on souhaite créer des trajectoires sur plusieurs secondes, généralement trois secondes, et on cherche les directions dans lesquelles on estime probable qu'il y aura du mouvement *dans le futur*. Pour cela, connaissant le temps entre les deux images utilisées en entrée, on estime les déplacements

des objets au cours des trois prochaines secondes en supposant que ceux-ci seront linéaires. En faisant cette hypothèse forte et en notant ΔT le temps entre les deux images d'entrée et $D_{Xs}(x, y) \in \mathbb{R}^2$ le déplacement après X secondes, on a tout simplement :

$$D_{Xs}(x, y) = \frac{F_o(x, y) * X}{\Delta T} \tag{3.37}$$

On peut alors obtenir les positions estimées des différents pixels Pos_{Xs} après X secondes dans l'espace image avec la formule suivante, presque identique à celle du flux optique mais ajustée pour le déplacement correspondant au temps souhaité :

$$prev(x, y) \sim Pos_{Xs}(x + D_{Xs}(x, y)[0], y + D_{Xs}(x, y)[1]) \tag{3.38}$$

On est maintenant capable d'obtenir les positions estimées dans le futur des différents pixels de notre environnement. On peux alors créer différentes matrices de pénalisation en fonction du temps de déplacement qui vont pénaliser les directions dans lesquelles on estime qu'il y aura des obstacles dans le futur. En notant P_{Xs} une telle matrice de pénalisation après X secondes, on construit à partir d'une matrice nulle, la matrice P_{Xs} en fonction du flux optique F_o de la manière suivante :

$$\begin{aligned} &\forall (x, y) \in [1, F_o.rows] \times [1, F_o.cols], \quad si\, \|F_o(x, y)\| > Seuil, \\ &\qquad P_{Xs}(x + D_{Xs}(x, y)[0], y + D_{Xs}(x, y)[1]) = \|F_o(x, y)\| \end{aligned} \tag{3.39}$$

Finalement, on calcule ces matrices de pénalisation à différents timings pour obtenir, par une somme pondérée, une matrice de pénalisation P qu'il ne restera plus qu'à soustraire aux matrices de coûts comme on a pu le faire pour les minimums locaux. L'équation 3.40 est un exemple de ce procédé avec des matrices intermédiaires calculées à une, deux et trois secondes. On notera que l'on conseille de prendre $\alpha > \beta > \gamma$ parce que l'estimation à un horizon temporel proche ne peut être que plus précise que l'estimation à un horizon temporel lointain.

$$P = \alpha P_{1s} + \beta P_{2s} + \gamma P_{3s} \tag{3.40}$$

3.10 Quelques choix techniques

La communauté du contrôle qui a été la première à travailler sur les quadrirotors travaille principalement avec Matlab et c'est tout naturellement que les premiers papiers qui se concentraient sur le contrôle des quadrirotors ont mis en avant des outils utilisant Matlab. Ce langage est pourtant relativement lent et limite la quantité de calculs réalisables en temps réel sur la structure limitée en calcul qu'est le quadrirotor. Pour faire de la vision et donc de l'évitement d'obstacles, c'est très problématique et d'autres pistes ont été envisagées. Des FPGA ont par exemple été utilisés [Bar16] ; [OHP15]. Plus récemment, ce sont des cartes graphiques

dédiées à l'embarqué qui sont de plus en plus utilisées et c'est cette dernière option que l'on a choisie.

L'utilisation d'une telle carte graphique permet d'obtenir une puissance de calcul équivalente à un ordinateur portable et permet l'utilisation de systèmes d'exploitations classiques tel qu'Ubuntu. Il devient donc possible d'utiliser des outils classiques de robotique en embarqué sur les drones. C'est ainsi qu'on a choisi d'utiliser le middleware ROS pour gérer la communication entre les différentes fonctionnalités présente sur le drone. ROS est nativement utilisable en python et en C++. Nos codes sont donc interfacés dans l'un ou l'autre de ces langages.

Comme cela a été précisé dans la partie 3.3.4, pour conserver un temps d'exécution de la totalité de l'algorithme de la centaine de millisecondes, il nous paraissait nécessaire de paralléliser la construction du système de vision. Pour cette raison, on a aussi choisi de réaliser le système de vision en OpenGL et cette partie de notre algorithme sera donc effectuée sur le processeur graphique embarqué.

Le reste de l'algorithme a été réalisé en c++, un langage qui fait parti des langages qui s'exécute rapidement sur CPU. Cela va nous permettre d'utiliser en parallèle le GPU et le CPU. Il s'agit aussi d'un des langages les plus utilisés ce qui offre la possibilité d'utiliser des librairies offrant de très nombreuses fonctionnalités tel que OpenCV pour la vision ou Eigen pour le calcul vectoriel et matriciel.

Chapitre 4

Les environnements de test et le benchmark

Au cours de cette thèse, de très nombreux tests ont été effectués. Certains ont eu lieu en simulation, d'autres en alliant environnement de simulation et quadrirotor réel et les derniers en condition réelle, en extérieur avec un quadrirotor en autonomie complète.

On présentera dans un premier temps les tests en simulation. En 4.1.1, les deux simulateurs de quadrirotors utilisés seront ainsi mis en avant avec leurs forces et faiblesses. On se concentrera ensuite sur les trois environnements de tests simulés qui ont été créés. Deux de ces environnements ont étés conçus dans le but d'être aussi légers que possible pour pouvoir multiplier les tests. L'environnement le plus simple constitué de cylindres sur un sol plat sera présenté en 4.1.2. On présentera ensuite en 4.1.3 des environnements toujours aussi légers spécialement dédiés aux tests d'algorithmes d'évitement dynamique. Le dernier environnement de test simulé, décrit en 4.1.4, plus complexe, a pour but de simuler un environnement proposant la même difficulté qu'un vol réel dans une forêt.

Les tests alliant simulation et quadrirotor réel, habituellement nommés hardware-in-the-loop, seront ensuite présentés en 4.2 puis en 4.3, l'environnement dans lequel les tests réels ont étés effectués sera mis en avant. On profitera de cette section pour présenter le quadrirotor avec lequel les vols en extérieurs ont été effectués.

Finalement, dans le but de simplifier et d'uniformiser les tests pour toutes les équipes travaillant sur des problématiques d'évitement d'obstacles pour quadrirotor, un benchmark a été créé en utilisant les forêts simulées précédemment décrites. Celui-ci sera présenté en 4.4.

4.1 Les environnements simulés

4.1.1 Les simulateurs de quadrirotor utilisés

L'étude de la littérature montre qu'à l'exception des simulateurs qui reposent sur MATLAB/Simulink qui gèrent très difficilement la création d'environnements complexes, deux simulateurs open sources sont principalement utilisés pour modéliser des quadrirotors. Il s'agit de Microsoft AirSim [Sha+18] pour les études sur les fonctionnalités avancées et de RotorS [Fur+16] pour les problèmes de contrôle. Si ces deux simulateurs possèdent des interfaces

avec ROS, ce qui est indispensable pour notre projet, RotorS, qui fonctionne directement dans l'environnement par défaut de simulation de ROS, Gazebo, est beaucoup plus pratique à utiliser.

Microsoft AirSim, qui est basé sur Unity, permet de voler dans des environnements qui sont graphiquement beaucoup plus réalistes que ce que RotorS, qui est basé sur Gazebo [Gaz], peut offrir. L'avantage principal de ces environnements réalistes est dans la qualité des rendus des caméras. Cependant, les contreparties sont assez importantes : d'une part, Microsoft AirSim utilise Unity ou l'Unreal engine qui sont tous deux à sources fermées et d'autre part, ces deux environnements sont beaucoup plus lourds que Gazebo ce qui peut donner lieu à des temps de chargements de plusieurs minutes à l'initialisation des tests. En plus de cette liste d'inconvénient, il faut ajouter que l'intérêt dans notre projet d'avoir des rendus graphiques de très bonne qualité est assez faible. On préférera privilégier l'ajout de bruits et de perturbations d'intensités et de natures équivalentes aux bruits et perturbations observés en situation réelle à une tentative de reproduire un visuel fidèle à une situation réelle ce qui coûte beaucoup de calculs pour un gain difficilement quantifiable.

Si l'objectif primaire de tous les algorithmes d'évitement d'obstacles est de garantir l'évitement, un objectif secondaire est très souvent d'aller le plus vite possible. Ainsi, Barry a réalisé un comparatif de la vitesse de vol de plus d'une dizaine de références[Bar16]. Cette course à un évitement d'obstacles toujours plus rapide augmente d'année en année la dynamique des vols autonomes. Il devient donc nécessaire de modéliser la dynamique des quadrirotors le plus fidèlement possible. C'est justement l'un des points sur lequel RotorS se concentre avec attention. C'est pour cette raison et pour sa facilité d'utilisation que l'on a finalement choisi RotorS plutôt que Microsoft AirSim.

RotorS utilise ROS Kinetic et Gazebo 7.0.0, les dernières versions à support long terme (LTS) disponibles au début de cette thèse. C'est donc naturellement que l'on a aussi utilisé ces versions. RotorS permet de simuler différents châssis de quadrirotors. Pour choisir notre châssis, on s'est basé sur la taille moyenne des châssis utilisés en évitement d'obstacles au cours des dernières années. Une relecture rapide des références données en 3.1 permet de constater que la majorité des papiers utilisent des drones compris entre 30 cm de diamètre (e.g. Parrot Bebop 2) et 1 m de diamètre (e.g. Astec Pelican). On observe aussi ces dernières années une tendance à utiliser des châssis de plus en plus petits. L'un des premiers algorithmes d'évitement d'obstacles autonome publié était ainsi dédié à un hélicoptère de 95 kg [Sch+08] quand en 2015/2016, l'un des châssis les plus utilisés était le Asctec Pelican [Liu+16] ; [RBR16] et que sur les deux dernières années, de multiples papiers se sont concentrés sur des châssis modifiés à partir de Bebop [Cha+17] ; [Yan+17] ; [ACP17] ; [Näg+17]. Finalement, pour se rapprocher d'un maximum de plates-formes en terme de poids et de taille, on a choisi un Astec Hummingbird de 55 cm de diamètre et de 710 g. Cette taille et ce poids se situent dans la moyenne basse de la littérature ce qui correspond à un compromis entre la littérature des cinq dernières années et la tendance à privilégier les plates-formes plus petites.

RotorS propose plusieurs modes de contrôle et contrôleurs. Il est ainsi possible de contrôler le drone en commandant les vitesses moteurs, l'attitude ou la position. Lorsque le simulateur est contrôlé en attitude, un régulateur PID gère alors le calcul des vitesses moteurs.

En position, c'est le contrôleur de référence de Lee, Leok et McClamroch [LLM10] qui permet de calculer les vitesses moteurs. On notera que l'une des principales critiques qu'il est possible d'adresser à RotorS concerne la qualité du contrôleur de position proposé. Il est notamment possible, avec ce contrôleur, de créer des crashs en commandant un cap opposé au cap courant. Ceci est très troublant lors des premières utilisations du simulateur, en particulier lorsque la volonté de tester des fonctions de navigation sans contrainte dynamique forte donne rapidement lieu à un crash.

Remarque. *Si tous les résultats présentés dans le chapitre suivant utilisent RotorS, ce serait injuste de ne pas présenter RosCopter_Sim, anciennement fcu_sim. Ce simulateur n'a pas pour but d'être aussi précis que RotorS mais a le mérite d'être très stable et très facile d'utilisation. Il s'agit d'un simulateur avec lequel il est très facile de travailler et avec lequel j'ai très longtemps testé mes algorithmes. Toute la dynamique d'un modèle simple est comprise dans un unique fichier ce qui rend les modifications très aisées. J'ai notamment pu modifier la dynamique pour tester la vision lors de vol à très hautes vitesses avec un suivi de trajectoire parfait en quelques minutes grâce à ce simulateur.*

4.1.2 L'environnement simple

Il s'agit de l'environnement de test le plus simple généré au cours de la thèse. Il est composé de cylindres positionnés aléatoirement dans un carré de $40\,\mathrm{m} \times 40\,\mathrm{m}$ entouré de murs pour contraindre le quadrirotor à rester dans la zone dans laquelle des obstacles sont générés. Il s'agit d'un environnement extrêmement léger dans lequel il est possible, en répétant les tests et en utilisant le positionnement aléatoire des obstacles, d'effectuer des analyses statistiques pour estimer l'efficacité des différentes modifications effectuées sur les algorithmes *en cours de développement.*

Cet environnement met à profit la fonction population proposée dans les spécifications des mondes Gazebo dont une utilisation est présentée ci-dessous :

```
<population name="cylinder_forest">
 <model name="cylinder">
    <!--details supprimes par soucis de concision-->
  </model>
  <box>
    <size>40 40 0.01</size>
  </box>
  <model_count>100</model_count>
  <distribution>
    <type>random</type>
  </distribution>
</population>
```

La figure 4.1 présente différents environnements ainsi générés. Si l'on considère que l'environnement (a) est l'environnement de référence, on voit qu'il est possible de faire varier

la taille et le nombre des obstacles (b) mais aussi leurs natures (c) et leurs orientations (d).

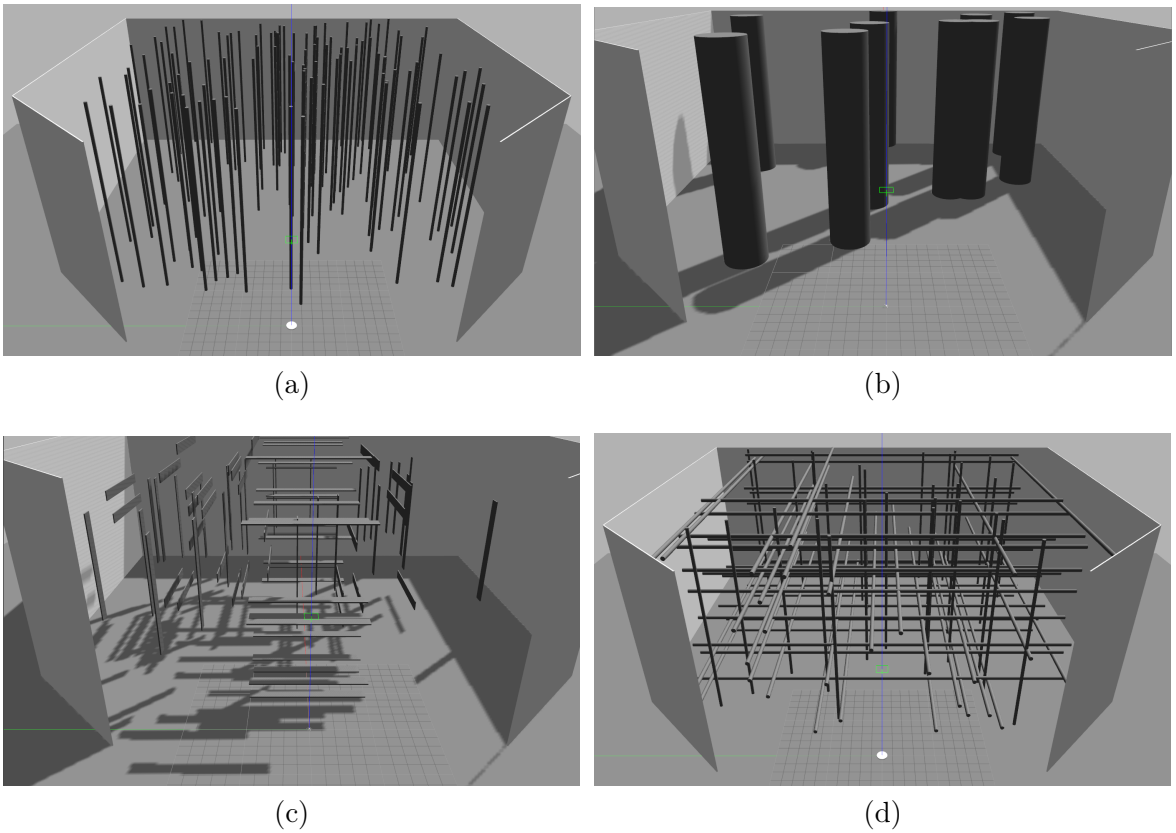


FIGURE 4.1 – Environnements légers générés pour les tests en phase de développement.

Pour chaque environnement, on prendra soin de détailler les entrées de l'algorithme d'évitement. On précisera notamment si les entrées sont bruitées ou non et si les entrées correspondent à la sortie de capteurs simulés ou si elles correspondent à l'entrée de la partie sur laquelle je me suis concentré c'est à dire simplement à l'état du drone et une image de profondeur.

Dans cet environnement simplifié, les entrées sont non bruitées et ne correspondent qu'aux entrées de la partie spécifique sur laquelle je me suis concentré. L'état du drone provient donc d'un message d'odométrie et d'un message d'IMU à partir des valeurs exactes extraites de Gazebo. Ces messages couvrent les 12 états du quadrirotor définis comme les entrées de la partie spécifique de l'algorithme d'évitement d'obstacles sur laquelle je me suis concentré. L'image de profondeur est une simulation non bruitée d'une Intel® RealSense™ D435. Son champ de vue est donc défini à 69° horizontal et 42° vertical quand les valeurs de profondeur qui composent cette image de profondeur sont parfaites dans [0.15 m, 10 m] et indéfinies en dehors de cet intervalle.

Pour permettre d'estimer les performances de l'algorithme d'évitement d'obstacles, il faut ajouter un dernier élément, un détecteur de collision. Dans Gazebo, un détecteur de

collision est dédié à la détection de volume entre un unique *volume de collision* et le reste du monde. Pour simplifier la gestion de la détection des collisions du quadrirotor, on définit un cylindre englobant le drone. Cela permet d'avoir un unique détecteur de collision entre ce cylindre et le reste de l'environnement. Ce détecteur publie ensuite un message quand il détecte une collision. Cela permet de ne lire qu'un seul capteur et qu'un seul topic ROS pour gérer les collisions. Le *volume de collision* est légèrement plus gros que le quadrirotor comme la figure 4.2 le montre. Pour des raisons de sécurité cela est acceptable, un quadrirotor qui volerait à moins de 2 ou 3 cm des obstacles serait vulnérable à des rafales de vent et ne pourrait être considéré comme en sécurité.

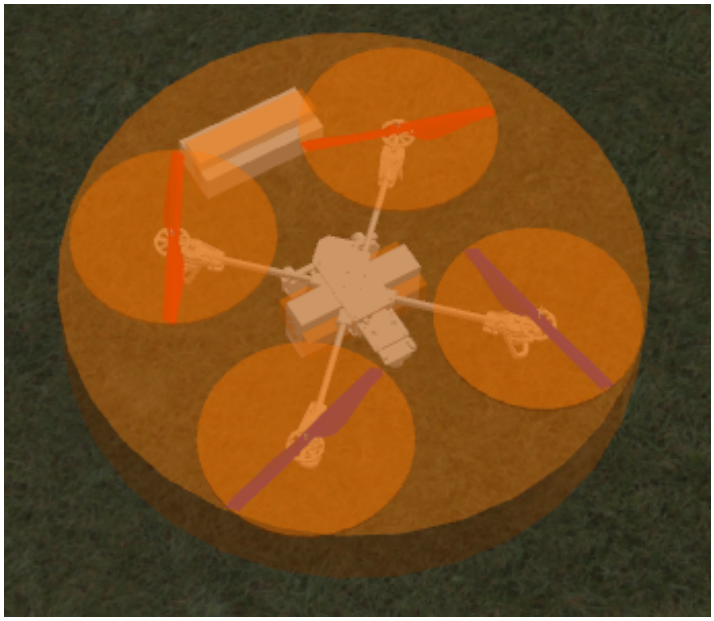
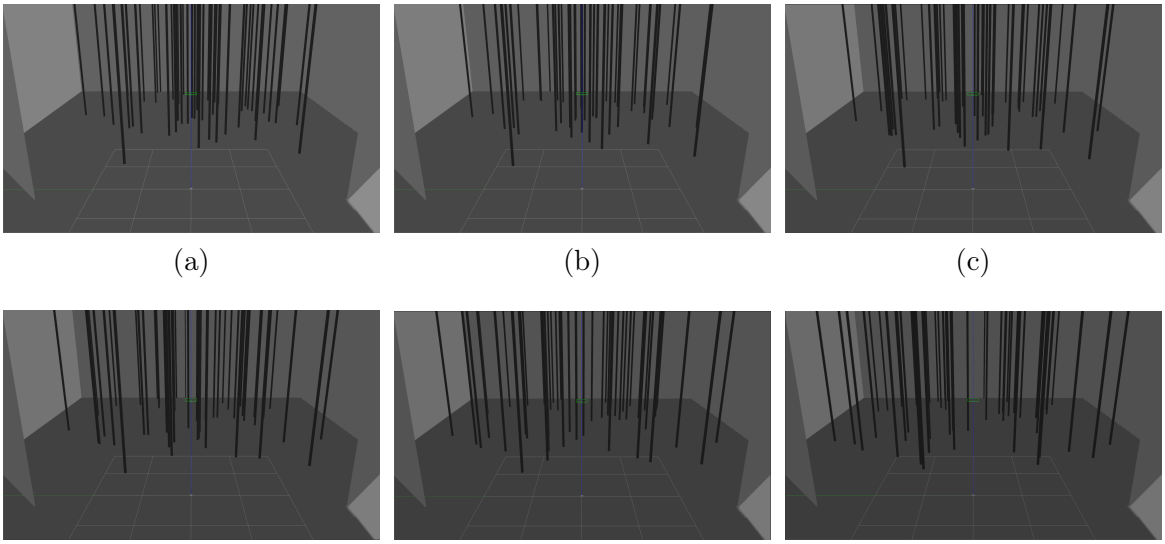


FIGURE 4.2 – Visualisation du cylindre utilisé pour détecter les collisions du quadrirotor simulé.

4.1.3 Les environnements dynamiques

Le premier environnement dynamique ressemble fortement à l'environnement simplifié. Il s'agit d'un cube de 40 m × 40 m dans lequel des cylindres verticaux sont générés puis bougent à vitesse constante selon des trajectoires définies aléatoirement. Ces mondes sont générés automatiquement via un script python qui permet de les paramétrer en fonction des tests réalisés. Avant de détailler les différentes caractéristiques et paramètres de ces mondes dynamiques, on présente en figure 4.3, six captures d'écrans à 1 s d'intervalle d'un monde ainsi généré.

L'environnement de test ainsi généré ressemble fortement à l'environnement simple présenté en 4.1.2, il est pourtant construit totalement différemment. Un plugin Gazebo a dû en effet être généré pour gérer les mouvements de chaque objet mobile. Le script python évoqué



(a) (b) (c)

(d) (e) (f)

FIGURE 4.3 – Évolution sur 5 secondes d'un environnement dynamique aléatoire.

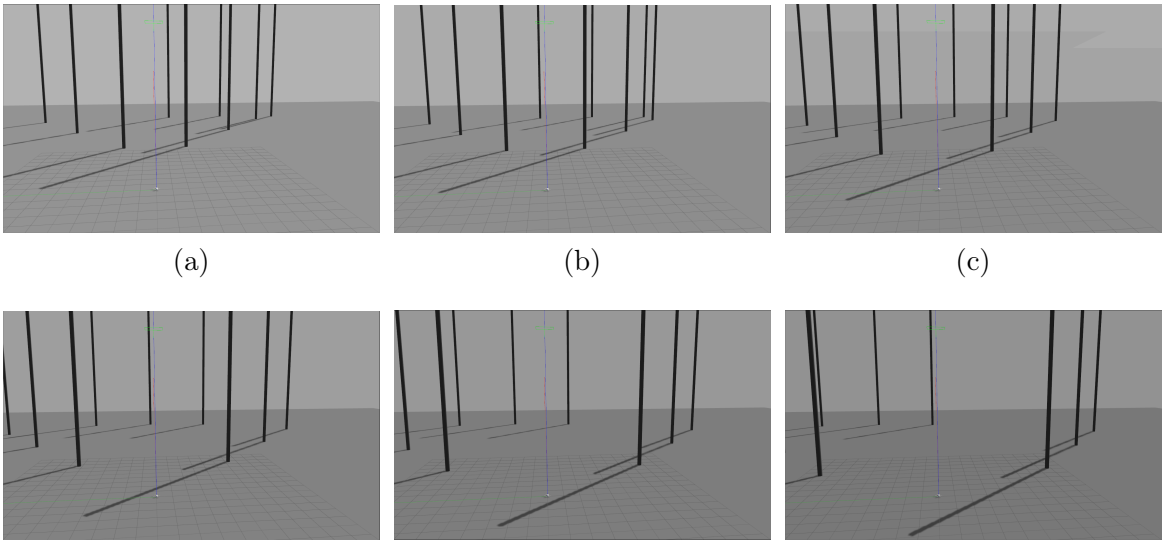
précédemment automatise non seulement la création du fichier world mais aussi la création des plugins Gazebo et la création du fichier CMakeLists.txt nécessaire pour compiler tous ces plugins. Un script bash a finalement été créé pour automatiser la génération de multiples mondes, il est ainsi possible de simplement :

— définir le nombre de mondes à générer
— définir la zone dans laquelle les obstacles se déplacent
— définir le nombre d'obstacles
— définir la vitesse des obstacles.

Les plugins sont automatiquement créés et compilés et les mondes générés sont ensuite utilisables sur un simple appel comme pour tout autre monde Gazebo.

Remarque. *Depuis la version de Gazebo 8.0.0, il est possible de ne générer qu'un fichier .world dans lequel, pour chaque solide mobile, une trajectoire est définie sous forme de points de passages associés à un temps de passage. Cette possibilité n'existait pas dans la version 7.0.0 d'où la méthode proposé générant un plugin pour chaque solide mobile.*

Les tests dans le premier monde dynamique précédemment défini ont permis de faire apparaître des cas particuliers problématiques pour obtenir un évitement dynamique fiable. Il apparaissait notamment que lorsque des obstacles se déplaçaient dans certaines directions particulières, cela pouvait systématiquement causer des collisions. Ces directions sont relativement intuitives puisqu'il s'agit des obstacles se déplaçant à $\pm 90°$ par rapport à la direction du drone et de ceux qui se déplacent dans la même direction que le drone mais en sens inverse. Un monde spécifique permettant de se confronter à chacune de ces situations problématiques a donc été défini, on peut voir son évolution sur 5 s en figure 4.4. Parce que cet environnement réduit l'évitement dynamique à chaque instant à l'évitement d'un unique obstacle, on l'appellera par la suite l'environnement dynamique *épars*.



(a) (b) (c)

(d) (e) (f)

FIGURE 4.4 – Évolution sur 5 secondes d'un environnement dynamique pour tester des cas d'évitement spécifiques.

Dans ces environnements dynamiques, les entrées sont les mêmes que dans l'environnement simple. On rappelle donc que l'état du drone parfait sera directement accessible et que l'image de profondeur en entrée correspond à la sortie d'une Intel® RealSense™ non bruitée.

4.1.4 La forêt simulée

Le troisième environnement de test est une simulation de forêt de 80 m × 80 m.

Les sols sont extraits de fichiers de type Digital Elevation Model (DEM). Ils sont extraits de localités Etats-Uniennes et ont été téléchargés grâce à The National Map [Usg]. On s'est pour l'instant limité à cinq profils de sols qui suffisent pour créer des terrains très divers. Un profil de sol est en pente constante, un autre est plat quand les trois derniers profils correspondent à des terrains mixtes avec des lits de rivières et de petites collines.

Les arbres sont générés par un générateur d'arbre procédural [jar]. Ce générateur crée des modèles *low poly*, qui possèdent peu de polygones, ce qui est nécessaire pour permettre à Gazebo de conserver sa fluidité. Vingt modèles d'arbres ont été générés. Ces arbres sont aléatoirement légèrement inclinés par rapport à l'axe vertical et aléatoirement orientés selon l'axe vertical. Grâce à ces modifications d'orientation, ces vingt modèles sont suffisants pour créer des forêts réalistes sans qu'il soit possible de percevoir de répétitions de motifs comme on peut le constater sur la figure 4.5.

A partir des profils de sols et des modèles d'arbres, les forêts sont générées en sélectionnant au hasard 100 modèles d'arbres et en plaçant chacun des arbres sur le profil de sol

sélectionné. La répartition des arbres sur la surface de 80 m × 80 m suit trois distributions gaussiennes possédant des déviations standards différentes ce qui permet d'obtenir des mondes qui possèdent des zones de densité d'arbres diverses. Une des métriques utilisable pour quantifier la densité d'une foret est l'occupation au sol, cela revient à estimer le pourcentage de la surface recouverte par des feuillages lors d'une observation par satellite. Les paramètres choisis lors de la génération permettent d'obtenir à la fois des zones à faible densité d'arbres (moins de 40% d'occupation au sol), des zones de densité moyennes (entre 40 et 70% d'occupation) et une zone de haut densité (plus de 70% d'occupation). Il nous paraissait crucial de créer des forêts possédant des pourcentages d'occupations variés. D'une part parce que cela se rapproche de forêts réelles et d'autre part, parce que cela nous permet d'obtenir des informations précieuses sur le comportant de nos algorithmes. Par exemple, l'algorithme va-t-il plutôt mener le drone à travers la zone dense à faible vitesse ou va-t-il au contraire générer des détours pour voler plus vite ?

FIGURE 4.5 – Vue de dessus d'une forêt générée aléatoirement.

Le but de cet environnement étant de fournir des conditions proches des conditions en extérieur, on a fait le choix de fournir des données simulées des principaux capteurs que l'on aura sur notre drone au lieu de directement fournir l'état du drone comme cela a été

le cas dans les environnements précédents. C'est une différence relativement importante avec les environnements précédents puisque cela impose d'effectuer l'estimation d'état alors que ce n'était pas le cas précédemment. Il est tout de même possible, comme pour les autres mondes simulés, de directement récupérer l'état parfait du drone issu des données brutes de Gazebo mais cela augmente alors la distance entre la simulation et les vols réels.

On a ainsi simulé un GPS, une IMU et un magnétomètre en se basant sur la librairie de capteurs proposée par RotorS et en adaptant les bruits de ces capteurs en fonction des modèles que l'on embarquera sur notre drone. Il est intéressant de noter que ces bruits sont plus ou moins réalistes en fonction des capteurs. Les bruits des gyroscopes et accéléromètres de l'IMU sont ainsi composés d'un bruit gaussien, d'un bruit à variation lente assimilable à un biais et d'une marche aléatoire. La somme de ces différents éléments permet d'obtenir un bruit largement plus réaliste que le bruit du GPS qui est un simple bruit gaussien IID [1]. L'étude de la littérature montre que la modélisation du bruit des GPS est un problème ouvert ce qui explique ce modèle très peu réaliste. Quelque soit le modèle choisi, la modélisation resterait très imprécise dans notre environnement puisqu'on s'attend à ce que le drone passe **sous** les arbres ce qui aurait tendance à complexifier le bruit du GPS. Pour limiter l'impact de cette importante différence par rapport aux conditions réelles, on a fait le choix de modéliser un GPS RTK dont les erreurs d'estimations sont relativement faibles, de l'ordre de la dizaine de centimètres.

Le capteur de profondeur simulant une Intel® RealSense™ est lui aussi bruité via un plugin développé en basant sur l'estimation du bruit d'Intel [Kes+17]. Ce plugin fait intervenir cinq procédés différents pour bruiter les images de profondeur parfaites que Gazebo fournit. Pour détailler ces procédés, on note $P(x, y)$ le coefficient de la ligne x et de la colonne y de l'image de profondeur et $P_b(x, y)$ le coefficient situé aux mêmes coordonnées dans l'image bruitée. L'équation 4.1 correspond à l'équation du décalage horizontal d'un pixel dans l'image bruitée avec $\mathcal{U}(1, N)$ la loi de probabilité uniforme entre 1 et N. Cette équation est appliqué à un nombre de pixels N_{decal} et a été implémenté pour simuler la difficulté à replacer les pixels dans la direction de la baseline des caméras de stéréo-vision.

$$P_b(x, y) = P(x, y - Decal), \quad Decal \hookrightarrow \mathcal{U}(1, Decal_{max}) \tag{4.1}$$

Les caméras ayant des positions différentes, certains pixels ne sont visibles que sur une seule caméra et il est alors impossible de calculer leur profondeur. Cela est particulièrement le cas lorsque la profondeur est discontinue dans la direction de la baseline des caméras de stéréo-vision. L'équation 4.2 est une manière relativement simple de représenter la perte d'information qui en résulte : lorsque le gradient de profondeur horizontal est supérieur à un $Seuil$, on définit l'un des pixels associé à ce gradient comme de valeur inconnue ce qui est noté NaN. On notera que, dans l'implémentation finale, le simple gradient entre deux pixels voisins a été remplacé par un filtre de Sobel mais que le principe est le même.

$$Si|P(x, y) - P(x - 1, y - 1)| > Seuil, \quad P_b(x, y) = NaN \tag{4.2}$$

1. Indépendants et Identiquement distribués

En plus de ces problèmes de perception sur l'axe horizontal, la totalité des valeurs de profondeurs estimées sont imprécises. On a modélisé cela sous deux formes. D'une part, un léger flou est appliqué à la totalité de l'image au moyen d'un filtre gaussien dont les équations ont été données dans la partie 3.7. D'autre part, le kernel gaussien présenté dans cette même partie grâce à l'équation 3.33 est ajouté comme un bruit additif. Cela permet d'obtenir un bruit qui n'est indépendant d'un pixel à l'autre mais qui est localisé sur différentes régions de l'image comme cela peut être le cas avec la caméra réelle. L'intensité de ce bruit dépend du carré de la profondeur des pixels considérés comme estimé dans les caractéristiques publiées par Intel [Kes+17].

Finalement deux opérations de seuillages sont effectuées. Les pixels lointains sont tronqués à une profondeur maximale et les pixels proches sont définis comme inconnus NaN comme le fait l'Intel® RealSense™.

La figure 4.6(a) présente l'image de couleur d'une scène simulée. La figure 4.6(b) montre la profondeur perçue par ce qui serait une Intel® RealSense™ parfaite. La figure 4.6(d) est la carte de profondeur artificiellement bruitée. La Figure 4.6(c) présente l'image couleur d'une scène réelle d'un environnement comparable à (a) et (e) expose une image de profondeur telle que perçue par une réelle Intel® RealSense™.

Finalement, du vent a été ajouté à l'environnement. Celui-ci est généré à partir d'un plugin Gazebo et permet de simuler un vent continu et bruité sur lequel s'ajoute, par moment, des rafales récurrentes. Avec l'un comme l'autre de nos contrôleurs, l'intensité de ce vent donne lieu à des perturbations maximales sur le plan horizontal d'une quarantaine de centimètres soit 80% du diamètre du drone utilisé.

4.2 L'environnement hardware-in-the-loop

Pour minimiser les risques et faire de premiers tests réels sur des petits drones pour lesquels les crashs ont souvent moins de conséquences qu'avec le drone qui embarque une carte graphique et sur lequel les algorithmes sont embarqués, on a effectué des tests hardware-in-the-loop (HITL), avec un drone réel mais avec des calculs déportés et un état parfait estimé par capture de mouvement.

4.2.1 L'arène de vol avec capture de mouvement

Ces tests ont étés effectués dans la salle de capture de mouvement du Gipsa-Lab qui est visible en figure 4.7. Cette salle est équipée de 12 caméras Vicon T40S pour un volume utile de $3\,\mathrm{m} \times 2.5\,\mathrm{m} \times 2\,\mathrm{m}$. Ces caméras émettent de la lumière infrarouge modulée. Des marqueurs fixés sur les objets que l'on souhaite observer reflètent cette lumière infrarouge ce qui permet de calculer avec environ 10 ms de délai leurs positions et orientations.

Étant le premier à travailler dans la salle de capture de mouvements avec ROS, des

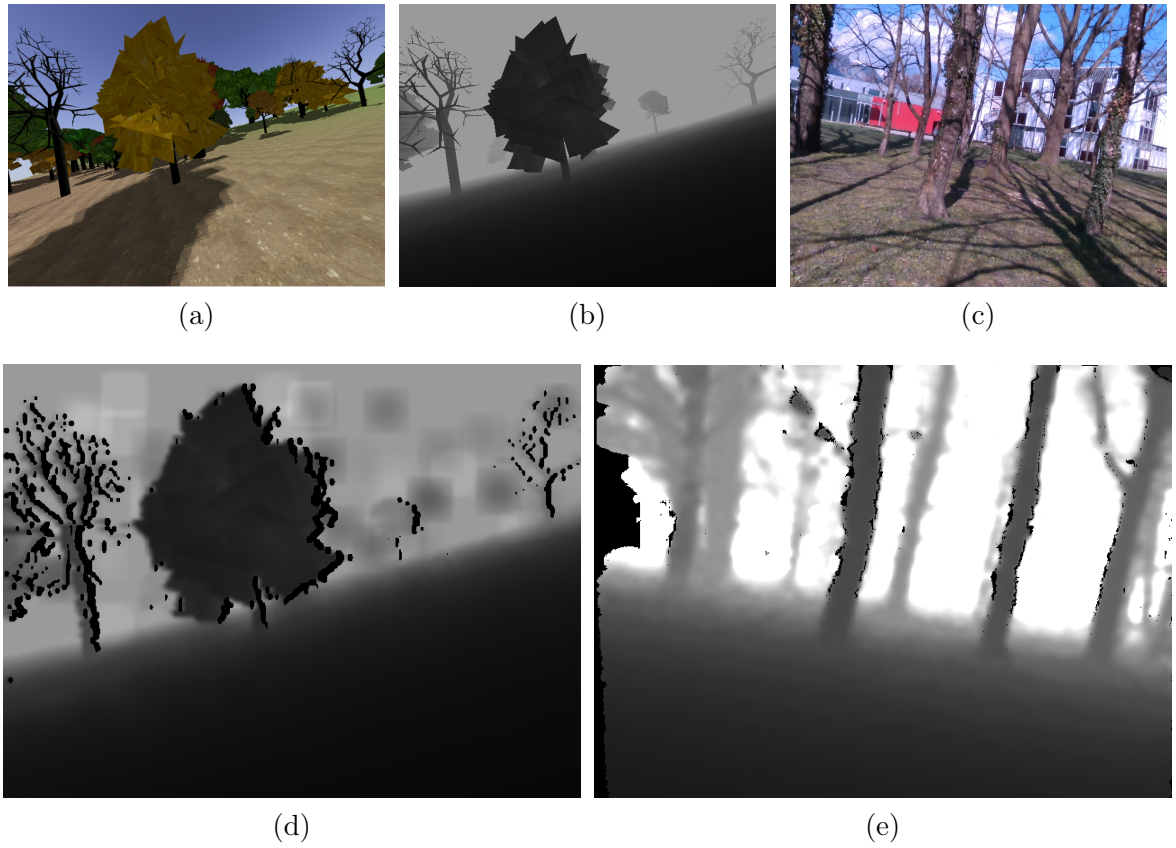


FIGURE 4.6 – Comparaison d'une carte de profondeur bruitée simulée et d'une image de profondeur perçue par une Intel® RealSense™.

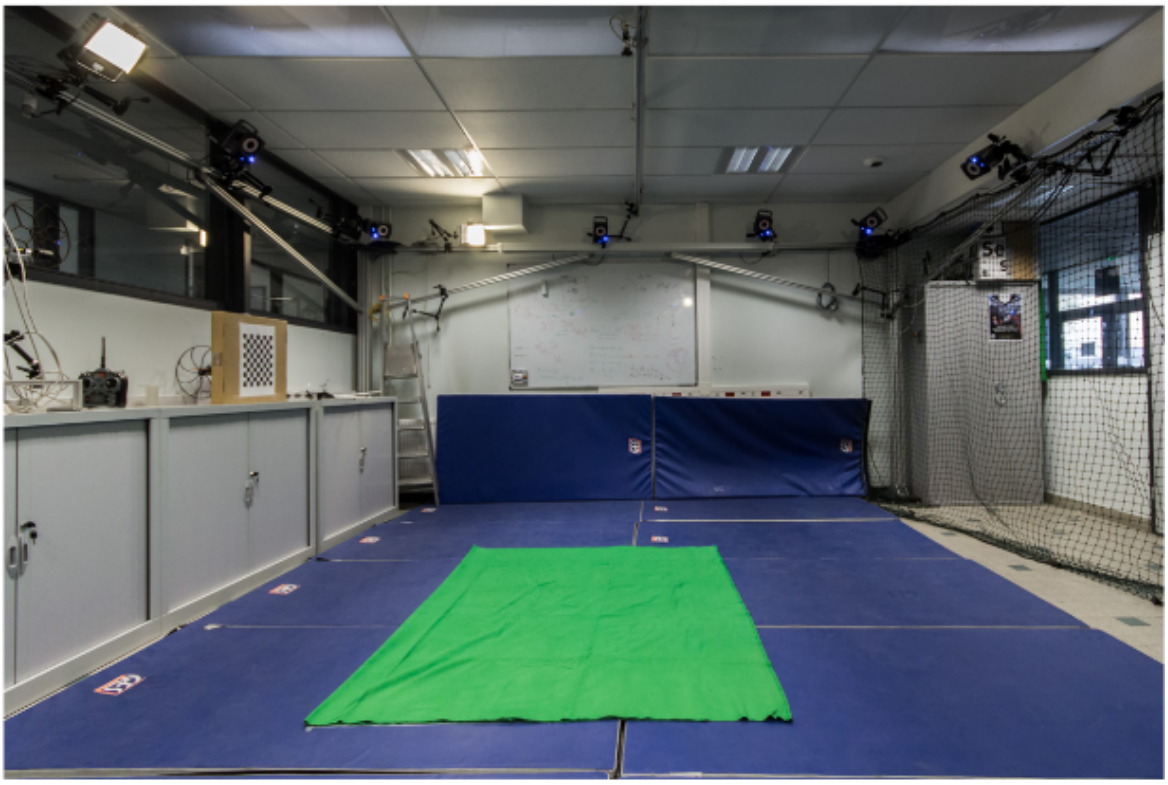


FIGURE 4.7 – Photo de l'arène de vol du Gipsa-Lab.

noeuds ROS ont été soit développés soit adaptés aux besoins du laboratoire pour être capable d'y faire voler un drone en utilisant ROS. On a par exemple réutilisé le *ros_vrpn_client* qui permet de récupérer la position mesurée par les caméras Vicon et de la convertir en un message ROS. Il a par contre fallu développer un noeud qui convertit les positions successives mesurées par le système de capture de mouvement en messages d'odométrie et d'IMU grâce des filtres de Savistky-Bolay. On a aussi développé un noeud pour gérer l'IRC, l'émetteur RC utilisé pour piloter certains drones, un autre noeud pour interpréter les commandes de la télécommande, un autre encore pour gérer le régime de vol (automatique ou manuel) pour certains drones. On finira par évoquer deux noeuds essentiels qui ont du être écrits pour bien correspondre à nos besoins, l'un sert à tester et évaluer les performances de contrôleurs et l'autre gère toute la création de fichier de logs. La majorité de ces noeuds ont depuis resservi dans d'autres projets menés au laboratoire.

4.2.2 Les environnements simulés

La partie simulée des environnements HITL est gérée par Gazebo. Le drone est replacé à chaque pas du simulateur Gazebo à la dernière position et attitude mesurée par le VICON. Par définition l'environnement simulé fait la même taille que l'espace utile de la salle de capture de mouvement. Cette taille, très faible pour effectuer de l'évitement d'obstacles, va fortement nous limiter. On générera donc un environnement relativement simple avec des murs correspondants aux limites de capture du système de capture de mouvement et quelques cylindres positionnés aléatoirement qui feront office d'obstacles. La figure 4.8(a) montre un tel environnement simulé et le met en parallèle avec l'espace réel dans lequel on replacé les mêmes obstacles en (b).

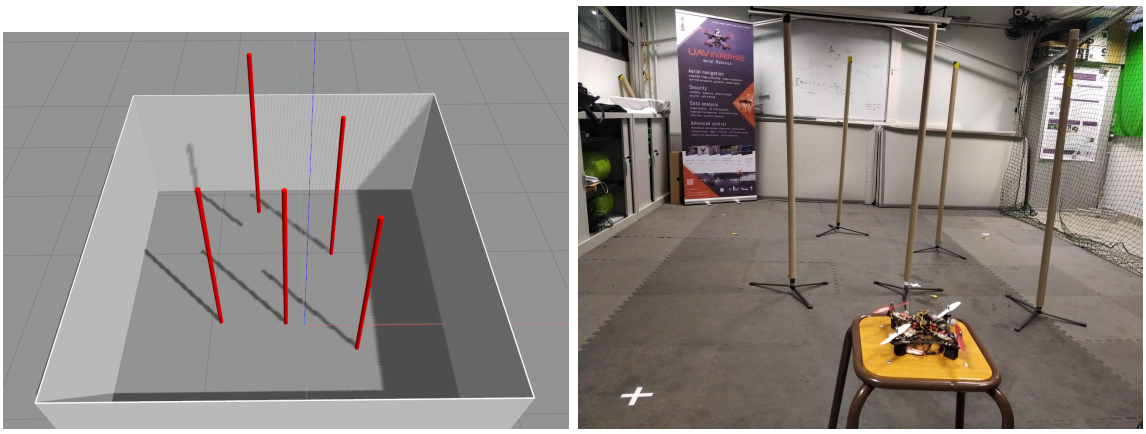


FIGURE 4.8 – Vue en parallèle d'un test HITL pour lequel on a placé des obstacles physiques et de l'environnement de simulation correspondant.

4.3 L'environnement de vol extérieur

Si les tests simples en extérieurs ne sont pas, par nature, garants de la performance d'algorithmes d'évitement d'obstacles, ils sont néanmoins un passage obligé qui fait office de démonstrateur du travail réalisé.

On ne reviendra pas dans cette partie sur les contraintes administratives (voir 2.4.1) et on se contentera de présenter le terrain d'expérimentation et le quadrirotor utilisé.

4.3.1 Le terrain d'expérimentation

Celui-ci a été gracieusement mis à disposition par la mairie de la commune de Revel (38420), Isère, France. Il s'agit d'un terrain communal qui se trouve aux coordonnées GPS suivantes : 45.1779N, 5.9018E. Il se compose à la fois d'une clairière sans obstacles délimitée en vert et d'un bosquet d'arbre entouré en rouge sur la vue satellite du terrain proposée en figure 4.9(a). Les figures 4.9(b), (c) et (d) sont des photos des différentes parties du terrain. En (b), une vue de la partie du terrain sans obstacles. En (c) et en (d), le bosquet dans lequel les vols d'évitements ont été effectués.

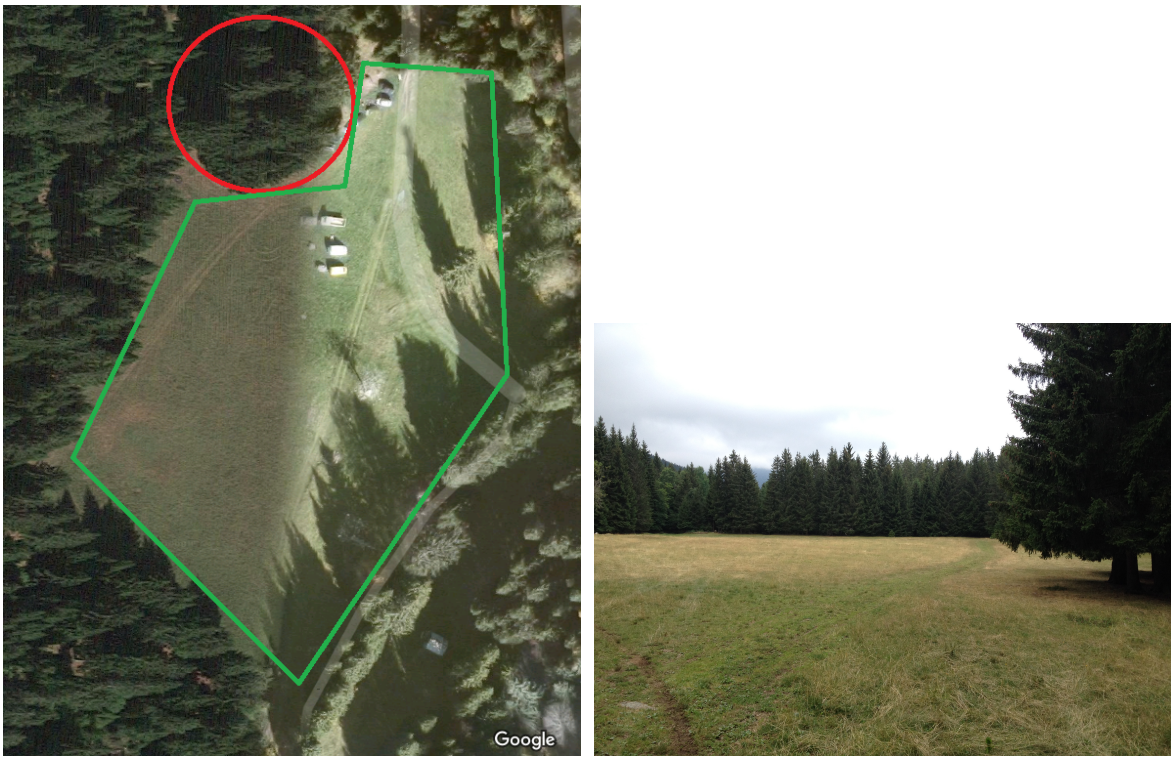
4.3.2 Le quadrirotor expérimental

On va présenter à la fois les éléments physiques constitutif du quadrirotor et les blocs logiciels sur lesquels viennent s'insérer les algorithmes présentés dans le chapitre 3. On rappelle qu'au cours du développement de l'algorithme d'évitement d'obstacles, on a considéré directement l'état du drone en entrée des algorithmes, le bloc d'estimation permettant d'obtenir cet état à partir des données capteurs sera donc présenté ici.

4.3.2.1 Les éléments physiques

La liste détaillée des éléments essentiels de notre quadrirotor est la suivante :
- Châssis : en carbone, distance entre centre des moteurs en diagonale 28cm
- Contrôleur de vol : CC3D Revo F4 Emax comprenant une centrale inertielle (IMU) MPU6000
- Batteries 4S à 5000mAh
- Contrôleur Électronique de Vitesse (ESC) : BLHeli-32 30A
- Moteurs : X-Nova RM2206-2300kV
- Hélices : GemFan 6040 Bullnose
- Récepteur radio : Spektrum AR7700, communication PPM liée à une télécommande Spektrum DX6i

Comme on l'a vu dans la partie 2.4.6, pour voler de manière autonome et suivre une consigne de trajectoire, il est néanmoins nécessaire d'ajouter un certain nombre d'éléments. Sur notre



(a) (b)

(c) (d)

FIGURE 4.9 – Vue satellite et photos du terrain d'expérimentation.

drone, cela se traduit par :

- un gps UBLOX NEO-M8P assemblé par DROTEK, compatible RTK, qui comprend un magnétomètre LIS3MDL.
- une Intel® RealSense™ D435 qui aura deux usages : elle fournira d'une part une information de profondeur qui sera une des entrées de l'algorithme d'évitement et d'autre part ses images seront utilisées pour affiner le positionnement dans l'espace
- un TeraRanger Evo 60m qui pointe vers le sol pour avoir une information sur la hauteur du quadrirotor.
- une carte graphique embarquée de type NVIDIA Jetson pour avoir la capacité de calcul suffisante pour exécuter tous les codes développés. Elle offre la possibilité de communiquer en Wifi et possède une IMU propre. Elle est montée sur une AUVIDEA J120 qui permet de gérer les connexions entre les autres éléments du drone et la Jetson.

L'ensemble de ces éléments une fois correctement liés au châssis, correctement alimentés et correctement configurés permettent d'obtenir un quadrirotor de 1750 g pour une diagonale de 47 cm. Le ratio poussée sur poids de ce quadrirotor est de 2 et et la durée de vol est estimée entre 10 et 15 minutes. Une photo de ce quadrirotor est présentée en figure 4.10.

FIGURE 4.10 – Photo du quadrirotor construit pour effectuer les tests en extérieur.

Ajouter tous les éléments listés précédemment sur une structure aussi dense est un

vrai challenge d'autant que certains éléments peuvent interférer avec d'autres. On notera notamment qu'on a deux circuits électriques, un circuit de puissance sur lequel sont branchés les ESCs et sur lequel transite de forts courant (jusqu'à 100A) et un second circuit sur lequel sont branchés tous les capteurs. L'utilisation d'un seul circuit est problématique parce que les pics et variations de courant perturbent le fonctionnement des capteurs.

Remarque (Focus sur le blindage électromagnétique)**.** *Il y a deux sources importantes de perturbations électromagnétiques sur un drone. Il s'agit d'une part du fort courant qui alimente les moteurs et d'autre part des données qui transitent par USB3. Il est impératif de prendre des mesures pour limiter l'impact de ces perturbations sur les capteurs, notamment sur le magnétomètre et le GPS. Sur le quadrirotor expérimental, cela se traduit par l'éloignement de ces deux capteurs du reste des éléments du drone et par le blindage qui entoure la liaison USB3 ainsi que la carte NVIDIA Jetson.*

4.3.2.2 La gestion des communication

Il s'agit d'un des éléments essentiels au bon fonctionnement du drone : la capacité à communiquer correctement, c'est à dire en satisfaisant certains critères de qualité, par exemple sur la fréquence de communication et le taux de perte, entre les différents éléments physique du drone.

Le schéma 4.11 reprend la totalité des éléments présents sur le quadrirotor et détaille les différentes communications. Les symboles + signifient que de légères modifications ont été effectuées pour que la liaison soit fonctionnelle. Les symboles signifie qu'une part significative des drivers nécessaires à la communication ont été réécrits.

4.3.2.3 Le firmware

Par définition, il est responsable des fonctionnalités minimales, celles qui sont requises pour effectuer les vols manuels. On a choisi d'utiliser le firmware ROSflight. On rappelle que ROSflight est un firmware minimal qui ne comporte donc que ce qui est nécessaire pour faire voler un quadrirotor et assurer une communication avec le middleware ROS. Dans la partie de présentation des firmwares 2.4.3, ROSflight a déjà été brièvement décrit. Pour plus d'informations, on renvoie le lecteur vers la documentation de ROSflight [JK16].

On rappelle les trois principales caractéristiques du firmware ROSflight :
- commandes possibles via radio ou via ROS et gestion du passage entre les deux types de commandes
- estimation d'un état composé de l'attitude et des vitesses angulaires grâce à un filtre complémentaire
- contrôle d'attitude grâce à un PID

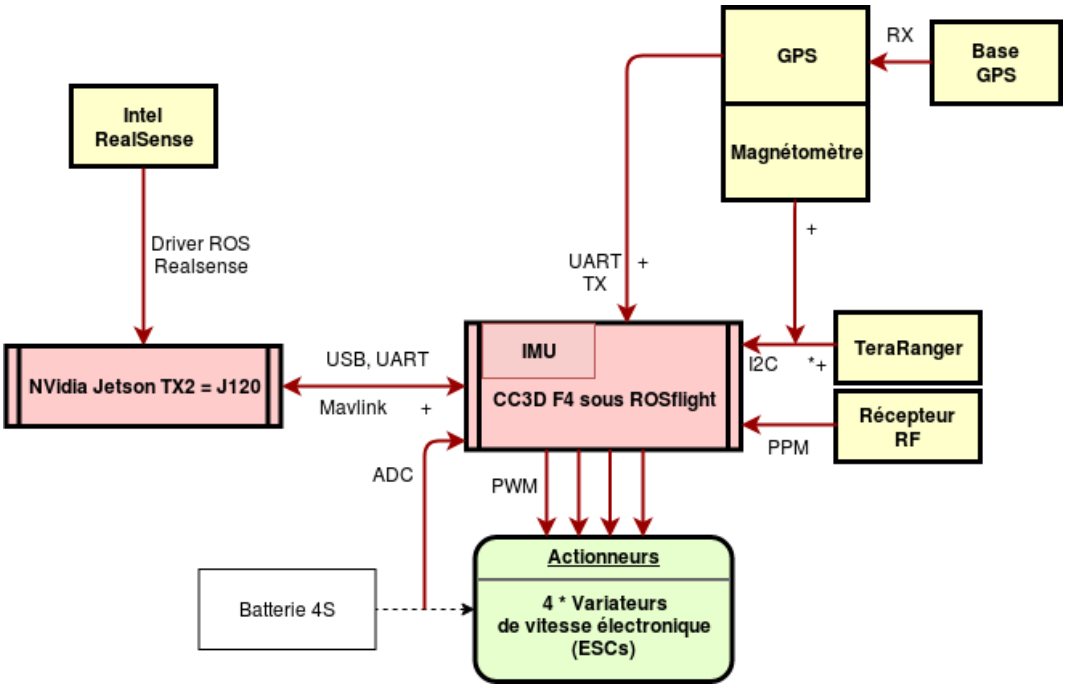


FIGURE 4.11 – Schéma de communication du quadrirotor d'expérimentation.

4.3.2.4 Estimation de l'état étendu

En 2.4.6, on expliquait que pour pouvoir ajouter des fonctionnalités avancées au drone, il était souvent nécessaire d'étendre l'état estimé en y incluant sa position dans l'espace, sa vitesse et son accélération linéaire. Cette extension de l'état du drone est nécessaire dans notre cas, en particulier lors de la mise à jour de l'egocubemap et du suivi de trajectoire. Étendre l'état du drone n'est cependant pas évident et requiert de pré-traiter puis de fuser les informations de tous les capteurs embarqués. Cette opération requiert plus de puissance de calcul que ce que propose la CC3D et sera donc effectuée sur la carte de calcul embarquée, la NVIDIA Jetson TX2.

L'outil principal pour ajouter ces états est un filtre de Kalman étendu [Asc13] dont le rôle est de fusionner les données des multiples capteurs. Dans notre cas, l'EKF fusionne 7 sources de données, 2 pour obtenir une estimation de l'attitude du drone dans un repère absolu, l'attitude estimée de ROSflight et le cap estimé par le magnétomètre, et 5 pour estimation les positions, vitesses et accélération linéaires :

— les accélérations issues de l'IMU
— les accélérations issues d'une estimation à partir des commandes moteurs et de l'attitude du quadrirotor
— la position issue du GPS
— le déplacement issu d'un algorithme d'odométrie visuelle
— la hauteur par rapport au sol issue du TeraRanger.

Sur ces 7 sources de données, seul l'attitude estimée de ROSflight, les accélérations de

l'IMU et les données GPS ne passent par une boucle de prétraitement.

Le magnétomètre est corrigé en fonction du courant qui est transmis aux moteurs. Il est en effet possible de montrer que ce courant génère des perturbations électromagnétiques qui perturbent les mesures. Pour corriger cet effet, on l'identifie tout simplement en faisant varier la vitesse des moteurs de leur minimum à leur maximum tout en maintenant le drone au sol. Cela nous permet de modéliser l'erreur en fonction de la vitesse moteur et il ne reste alors plus qu'à soustraire cette erreur en vol en fonction de la vitesse moteur instantanée.

De son coté, le TeraRanger est aligné avec $\vec{b_3}$. S'il mesure donc la hauteur au sol lorsque l'orientation du drone est nulle, il faut corriger la valeur qu'il mesure z_{mes} lorsque le drone ce n'est pas le cas. Cette correction est effectuée grâce à l'équation 4.3. On remarquera le terme z_{off} qui correspond à la différence de hauteur entre le TeraRanger, situé sous le drone et le centre de gravité du drone. On notera aussi que le TeraRanger a une portée limitée que ce soit pour la distance minimale de perception (50cm) ou la distance maximale (60m) et que certaines mesures sont nulles ce qui représente son incapacité à estimer une distance. Un noeud ROS a donc été créé pour effectuer la correction en fonction de l'attitude et pour gérer les limites de perception et les mesures nulles.

$$\hat{z} = \frac{z_{mes} + z_{off}}{cos(\phi) * cos(\theta)} \tag{4.3}$$

On a aussi ajouté un noeud d'odométrie visuelle qui permet d'estimer le déplacement du drone à partir des changements sur le flux vidéo. Réalisé par Amaury Nègre, ce noeud estime le déplacement entre deux images caméra. Il utilise une méthode d'odométrie visuelle dense et basée sur des images clés. Cette méthode ajoute à la méthode dense proposée par Steinbrücker, Sturm et Cremers [SSC11] l'utilisation d'image clés proposées par Kerl, Sturm et Cremers [KSC13].

Finalement, on a souhaité ajouter une information supplémentaire. Cette information permet, en revenant à la dynamique de notre robot de faire le lien entre attitude et position. On a vu dans la partie 2.3.3, qu'en utilisant une dynamique simplifiée du quadrirotor, il est possible de connaître les accélérations à partir de la poussée et de l'attitude du drone. Il serait dommage de se passer de cette possibilité pour obtenir une estimation supplémentaire de l'accélération et on va donc ajouter en entrée de l'EKF cette information supplémentaire qui provient de notre connaissance du robot utilisé.

4.4 Le benchmark

Un benchmark en simulation d'évitement d'obstacles a été réalisé au cours de la thèse et va maintenant être présenté. La première partie sera l'occasion d'évoquer les motivations derrière cette création ainsi que les apports de ce benchmark. Dans un second temps, le choix du simulateur, du châssis du quadrirotor, des environnements de test et des capteurs sera discuté. La partie suivante sera consacrée à une présentation rapide de l'usage, étape par

étape, du benchmark. Cette partie permettra entre autre de mettre en avant les tests effectués lors de l'exécution du benchmark. Finalement, en 4.4.4, l'analyse statistique sur laquelle se base l'indicateur primaire du benchmark sera présentée. On en profitera pour présenter des indicateurs secondaires qui pourraient permettre à un concepteur d'application pour drone qui cherche un algorithme d'évitement d'obstacles d'affiner son choix en fonction de ses besoins spécifiques.

4.4.1 Les motivations

Les benchmarks sont moins communs en robotique que dans d'autres domaines tel que l'apprentissage automatique ou la vision par ordinateur. Cela s'explique très facilement quand on considère la difficulté d'obtenir des conditions qui sont suffisamment proches d'un endroit à l'autre pour avoir des résultats d'expérimentations comparables. Même la création d'environnements contrôlés est tellement complexe que cela ne peut donner lieu qu'à des environnements très simplifiés qui au final deviennent assez éloignés de conditions non standardisées. Il n'est, par ailleurs, pas possible de demander à une équipe de recherche d'exécuter les codes des autres équipes, ne serait-ce que pour des questions de responsabilité en cas de défaillance lors des tests.

Une alternative aux benchmarks est l'organisation de challenges tel le DARPA robotics challenge ou le Autonomous Drone Racing Competition [Dro]. Cependant la participation à ces challenges est souvent coûteuse et est souvent considérée comme le but d'un projet. Elle offrira la possibilité de comparer des projets finis mais pas celle de comparer différents projets en cours de développement.

Les benchmarks ont l'avantage d'être à la fois utile en cours de projet et lors du partage des résultats. Ils autorisent aussi la comparaison directe entre différentes solutions pour une même tâche. Peut être plus important encore, utiliser un benchmark commun est un pas vers la compatibilité entre algorithmes et par conséquent vers une vérification simplifié des résultats par des équipes indépendantes. Pour toutes ces raisons et malgré les difficultés techniques, certains benchmarks ont été proposés en robotique. Certains d'entre eux testent une unique fonction, souvent affiliée à de l'estimation d'état, tel KITTI [GLU12] qui teste l'estimation de profondeur pour des voitures autonomes ou le RGB-D dataset qui permet de tester des techniques de SLAM [Stu+12]. D'autres papiers se revendiquant comme des benchmarks proposent des protocoles de tests très précis pour s'assurer que les tâches demandés à différents robots soient strictement identiques [Spr+16] ; [Lei+17] ; [MRB08].

Finalement, un dernier type de benchmark a déjà été proposé en robotique, il s'agit des benchmarks en simulation. Malgré des résultats qui ne peuvent être qu'approximatifs si on les compare à des résultats obtenus lors de tests réels, les simulations peuvent tout de même donner une idée des avantages et inconvénients des algorithmes proposés. On notera de plus qu'un benchmark simulé est souvent d'une grande aide pendant la phase initiale d'un projet. Finalement, les tests en simulations sont beaucoup moins contraignants que les tests réels, notamment en temps humain. Cette facilité de test permet donc de multiplier les tests et

de faire de l'analyse statistique [Qui+18] sur les résultats. Une analyse très peu réalisée en conditions réelles à cause de contraintes de coût et temporelles.

A la connaissance de l'auteur, il n'y a aucun autre benchmark, en simulation ou non, d'évitement d'obstacles pour quadrirotor d'où la création durant la thèse du benchmark BOARR disponible à `https://github.com/Gipsa-lab-PFP/BOARR`.

4.4.2 Spécifications des environnements, capteurs et tests

Remarque. *Il me semble important de préciser ici que si certains éléments ont déjà été présentés dans les environnements de tests de la thèse, ils ont, en majorité, d'abord été créés pour le benchmark, en se concentrant sur la réalisation d'un outil le plus générique possible, puis ont été utilisés dans la thèse et non créés pour tester les algorithmes présentés dans la thèse puis adapté au benchmark. L'utilisation du simulateur RotorS, non utilisé dans la thèse avant la création du benchmark en est l'illustration première.*

Commençons cette partie en précisant que son but n'est absolument pas de présenter les détails techniques du benchmark mais uniquement quelques points qui permettent de mettre en avant certaines caractéristiques que l'on juge intéressantes du travail réalisé. Si un lecteur est intéressé par les détails du benchmark, il est invité à lire le papier de présentation de ce dernier [DM+19]. Pour plus de détails encore, la documentation du benchmark est disponible à l'url suivante `https://github.com/Gipsa-lab-PFP/BOARR`.

Le benchmark proposé est open source et basé sur un simulateur communément utilisé. En s'inspirant des capteurs utilisés dans les papiers publiés ces dernières années, il propose plusieurs capteurs de profondeur pour s'adapter aux besoins des différentes équipes travaillant sur le sujet. Les tests sont réalisés dans une forêt car il s'agit de l'environnement de test le plus utilisé dans la littérature.

4.4.2.1 Choix du simulateur et du châssis

Le benchmark utilise le simulateur RotorS présenté en 4.1.1 ainsi que le châssis de l'Asctec Hummingbird tel que présenté dans la même partie. Une fonctionnalité de RotorS à mettre en avant pour ce benchmark est la possibilité d'utiliser différents types de contrôle. Si le contrôle par défaut ne me semble pas très performant, en particulier en position, comme signalé en 4.1.1, d'autres contrôleurs plus performants ont aussi été créés par les auteurs de RotorS [ASL19]. Parmi ces contrôleurs open-source, on peut évoquer un MPC linéaire et un MPC non linéaire qui sont accompagnés par un observateur de perturbations. Ces deux contrôleurs sont des contrôleurs de positions et sont utilisés en cascade avec un PID en attitude qui est aussi disponible dans le code partagé. Pour faciliter l'utilisation du benchmark, tous les contrôleurs par défaut du paquet RotorS sont tous actifs et un système de multiplexage pour sélectionner le contrôle approprié à son algorithme a été mis en place.

4.4.2.2 Environnements de tests

Deux environnements de tests sont proposés. Ils ont tous deux été présentés en 4.1.2 et en 4.1.4. Le premier correspond à l'environnement simple avec 100 cylindres verticaux, exactement comme sur la figure 4.1(a) et le second est la forêt mise en avant au travers de la figure 4.5. Pour pouvoir découvrir les environnements, un script est fourni par environnement. Chaque script permet de lancer un test avec un algorithme basique, qui avance en ligne droite vers les points de passages, dans son environnement associé. Ces scripts, `dummy_perfect_sensing_benchmark` et `dummy_noisy_benchmark`, permettent à la fois de voir un quadrirotor évoluer dans les environnements et de voir la vidéo, éditée automatiquement et en direct, qui illustre le vol et en résume ses caractéristiques.

4.4.2.3 Capteurs

La plupart des capteurs utilisés proviennent de la librairie de capteurs de RotorS. Il s'agit des capteurs présentés dans la partie 4.1.4 auxquels ont été ajoutés des capteurs qui ne seront pas utilisés au cours de cette thèse mais qui peuvent être utilisés par d'autres algorithmes d'évitement d'obstacles. Tous les capteurs, exceptés les cameras RGB, sont proposés dans une version parfaite et dans une version bruité. Les versions parfaites, qui donnent, comme leur nom l'indique, des valeurs mesurées parfaitement égales aux valeurs du simulateur, sont utilisées lors des vols dans l'environnement simple quand les versions bruitées sont utilisées lors des vols dans la forêt.

Le quadrirotor simulé est toujours équipé avec tous les capteurs qui, pour des raisons pratiques, sont considérés de masse nulle (ou de masse déjà comprise dans la masse du châssis). Chaque capteur émet des messages ROS du type le plus commun associé à ses données et ce à une fréquence qui correspond à une fréquence utilisable lors de vols réels.

GPS, IMU, magnétomètre : Il s'agit des capteurs qui ont été présentés dans la partie 4.1.4. On peut signaler que le GPS simule un GPS RTK, avec des erreurs en position de l'ordre de la dizaine de centimètre et émet des messages à 5 Hz quand l'IMU est de son côté configurée à 500 Hz.

Caméras : Deux caméras sont proposées, toutes deux a 30Hz. L'une est orientée vers le bas puisque de nombreux algorithmes utilisent une telle caméra pour estimer une information de position à partir d'algorithmes d'odométrie visuelle comme SVO [FPS14]. L'autre caméra est orientée vers l'avant ce qui est la direction appropriée pour éviter les obstacles pour les algorithmes basés sur une perception monoculaire. C'est aussi l'autre direction privilégiée par des algorithmes d'odométrie visuelle tel que PTAM [KM07]. Ces deux caméras ne sont disponibles que dans leur version parfaite. Proposer une version bruitée de caméra alors la qualité du rendu graphique est très limitée dans Gazebo semble ne pas avoir de sens. Ceci est une limite de ce benchmark qui aura du mal à donner une idée claire des performances

d'algorithmes se basant sur la vision.

Détecteur de collision : Le détecteur de collision a été présenté en 4.1.2. Pour rappel, il se base sur une représentation des quadrirotors comme un unique cylindre. Évidemment, ce capteur n'est pas bruité et ce quelque soit la configuration de vol choisie.

Capteurs de profondeurs : Deux capteurs de profondeurs sont proposés. D'une part la caméra stéréo Intel® RealSense™ D435 précédemment présentée en 4.1.4 et d'autre part un capteur LIDAR qui tente de se rapprocher d'un Velodyne HDL-32E. Ce capteur propose une information de profondeur avec un champ de vue de 41° vertical par 360° horizontal à 10 Hz avec une profondeur minimale de 1 m et une profondeur maximale de 100 m. Une version bruité par un bruit gaussien point à point (σ =2 cm), tel que défini dans la documentation du capteur est aussi proposé même si ce bruit est quasiment négligeable.

On notera qu'il est possible et relativement rapide de personnaliser l'ensemble des capteurs montés sur le drone. Tout choix de capteur, notamment bruité ou non, et toute modification devrait systématiquement accompagner la publication des résultats statistiques du benchmark.

4.4.3 Utilisation détaillée étape par étape

On propose une structure en trois étapes. La première étape permet de se familiariser avec l'environnement et de vérifier la compatibilité des algorithmes. La deuxième étape permet de vérifier les performances dans les conditions du test statistique et la dernière étape est l'analyse statistique elle même.

4.4.3.1 Test de compatibilité

Il s'agit d'un test dans l'environnement simple, non bruité et léger présenté en 4.1.2. Après téléchargement du code du benchmark depuis `https://github.com/Gipsa-lab-PFP/BOARR` et des sources de RotorS, s'en suit la phase de compilation. Une fois celle-ci complétée, on va pouvoir tester la compatibilité entre un algorithme et le benchmark. Le test est relativement simple, il s'agit de parcourir 100 m entre trois points de passages prédéfinis qui vont naturellement faire évoluer le drone dans la zone occupée par les cylindres. La simplicité de l'environnement permet de redémarrer très rapidement de nouveaux tests tant que le quadrirotor ne vole pas comme souhaité.

Pour qu'un algorithme utilisant un autre simulateur que RotorS soit compatible, il n'y aura que quatre étapes très simples à réaliser. Ces quatre étapes sont détaillés ci-dessous pour que leur simplicité soit apparente. Il faudra :

— modifier le type du topic de commande, si besoin, et le remapper

— éditer les fichiers launch pour remapper les topics standard d'entrée des différents noeuds testés et les placer dans un dossier spécifique
— éditer l'unique fichier de configuration dans lequel figure un unique paramètre qui est la sélection du type de contrôle
— adapter les référentiels des algorithmes si besoin

Ces actions, très simples, sont bien entendu décrites en détail dans un fichier de documentation (le fichier *UseYourOwnAlgorithm.md* dans le dossier **doc**). Une fois ces actions effectuées, lancer chaque test de compatibilité requiert simplement d'exécuter un unique script bash, les détails sont encore une fois disponibles dans le fichier de documentation.

4.4.3.2 Un test en forêt, base des tests statistiques

Après quelques vols réussis dans l'environnement parfait, une transition vers un test en forêt, l'un des tests à la base des tests statistiques, constitue la dernière étape intermédiaire avant de lancer la totalité du test statistique. Ce test, que l'on appellera *test de base*, est une tentative de voler pour 1km dans en environnement de type forêt sans que le capteur de collision ne détecte de collision. Ce test est effectué en présence de vent et avec du bruit sur les capteurs.

Seize points de passages sont définis, aux mêmes coordonnées x et y, dans chaque forêt générée. Ils sont représentés sous forme de croix en noir, rouge, bleu ou vert sur la figure 4.12. En commençant du coin en bas à droite et en se déplaçant dans le sens des aiguilles d'une montre, on donne à chacun de ces points de passage un identifiant entre 0 et 15. Chaque test commence, de manière aléatoire à un point de passage. Notons L l'identifiant de ce point de passage, le prochain point de passage est sélectionné aléatoirement entre les points de passages dont les identifiants sont $(L+5 \vee 6 \vee 7)\%16$ avec $\vee$ le symbole mathématique *ou* et le % le symbole représentant l'opération *modulo*. Ce processus de sélection du point de passage suivant est répété jusqu'à ce que les trajets générés dépassent le kilomètre qui constitue, on le rappelle, notre objectif de vol. Ce processus de sélection a été choisi après de multiples tests comme étant une solution efficace pour couvrir la totalité des terrains générés tout en évitant de faire des aller-retours. La figure 4.12 illustre le processus pour les deux premiers points de passage d'un test avec en bleu tous les points de passages considérés pour sélectionner W_2 et en vert les points de passages considérés pour W_3. Le point de passage en couleur pleine étant finalement le point de passage choisi, aléatoirement, entre les points considérés.

Pour éviter de valoriser les détours effectués par les algorithmes testés, on définit alors la *distance linéaire*. Cette *distance linéaire* parcourue par le quadrirotor est définie comme la distance entre tous les points de passage préalablement atteints par le drone (la distance entre W_1 et W_2 sur la figure 4.12) à laquelle on ajoute la distance entre le dernier point de passage atteint (W_2 sur la figure 4.12) et la projection perpendiculaire de la position du quadrirotor sur la droite formée par le dernier point de passage atteint (W_2 sur la figure 4.12) et le point de passage visé (W_3 sur la figure 4.12). Sur la figure 4.12, on donc une distance linéaire égale à $d_1 + d_2$. Chaque test est stoppé automatiquement, soit lorsqu'une collision est détectée et le

test alors considéré comme un échec, soit lorsque la *distance linéaire* atteint 1 km ce qui est le signe d'un test réussi.

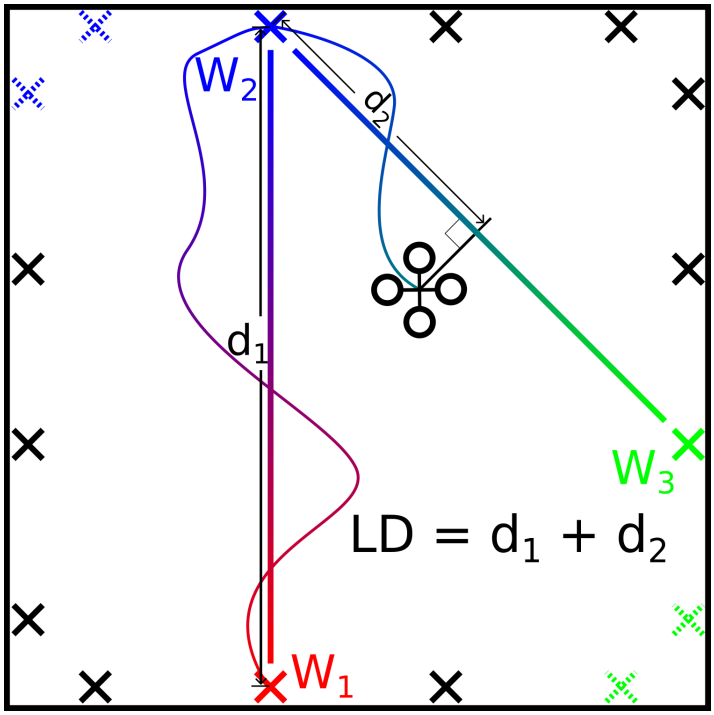


FIGURE 4.12 – Vue schématique de dessus des points de passage et du calcul de la *distance linéaire*.

Comme signalé dans le paragraphe précédent, pour tous les tests, une vidéo peut être automatiquement générée et est disponible en direct. Cette vidéo peut être particulièrement révélatrice durant cette phase de test pour visualiser le comportement du drone et le corriger si nécessaire. Comme pour les tests présentés précédemment, ce test peut être lancé en utilisant un unique script bash.

4.4.3.3 Tests multiples dans la forêt

Comme annoncé, il s'agit de la dernière étape. Elle consiste simplement à effectuer tous les tests qui seront statistiquement analysés pour caractériser l'algorithme testé. Comme pour les autres étapes, il est possible d'exécuter la totalité de l'étape en lançant un unique script. On notera que RotorS n'est pas toujours stable, en particulier au démarrage, mais que le script créé permet de vérifier que la sortie de chacun des tests correspond à une sortie attendue. Si ce n'est pas le cas, le test est effectué à nouveau. Cette étape prend environ une semaine si les tests sont effectués en temps réel ce qui doit être possible en utilisant un ordinateur à 500$ si l'algorithme testé n'est pas trop coûteux en calcul. L'étape peut prendre moins longtemps en utilisant le facteur d'accélération de Gazebo si la machine utilisée pour exécuter les tests le permet. Chaque *test de base* ajoutera une ligne à un fichier texte qui servira ensuite comme source pour l'analyse statistique présentée dans la partie suivante.

4.4.4 Analyse statistique des résultats

Le but principal de cette analyse est d'extraire un certain nombre d'indicateurs pour décrire les performances des algorithmes testés. Un indicateur primaire, basé sur la probabilité de collision pour une certaine distance, sera calculé. C'est en définissant la précision souhaitée sur cet indicateur que l'on déterminera le nombre de tests nécessaires à l'analyse statistique. D'autres indicateurs sont aussi proposés pour aider à distinguer les avantages et inconvénients de chaque algorithme testé.

4.4.4.1 Probabilité de collision et nombre de tests

Comme expliqué précédemment, chaque *test de base* consiste en une tentative de voler un kilomètre linéaire entre des points de passage et sans collision. En utilisant une approche probabiliste, il est possible, pour N tests, de calculer une estimation $\hat{p}$ de la probabilité p de voler un kilomètre linéaire sans collision comme :

$$\hat{p} = \frac{1}{N}\sum_{i=1}^{n} X_i \quad \text{with } X_i = \begin{cases} 1, & \text{Si le test } i \text{ est réussi} \\ 0, & \text{Sinon} \end{cases} \tag{4.4}$$

Il est ensuite naturel de vouloir borner l'erreur entre la probabilité estimée et la probabilité réelle de compléter un kilomètre de vol sans collisions. Dans une formulation probabiliste, avec ϵ l'erreur d'estimation et λ la probabilité que l'estimation soit en dehors de la borne d'erreur, on a :

$$Pr(|p - \hat{p}| \leq \epsilon) \geq 1 - \lambda \tag{4.5}$$

La borne de Chernoff nous donne l'expression d'une borne minimum N_{min} sur le nombre nécessaire de tests N pour s'assurer d'atteindre tout λ et ϵ :

$$N > \frac{\ln\left(\frac{2}{\lambda}\right)}{2\epsilon^2} \tag{4.6}$$

Un choix arbitraire a ensuite besoin d'être effectué pour définir un compromis entre la précision de la borne ϵ, la probabilité d'être à l'intérieur de cette borne λ et le nombre de tests N. La table 4.1 détaille le nombre de tests nécessaires pour obtenir différentes précisions et bornes d'incertitudes.

ϵ	0.01	0.02	0.05	0.05	0.1
λ	0.01	0.01	0.01	0.05	0.01
N_{min}	26 492	6 623	1 060	738	265

Table 4.1 – Nombre de test pour différentes précisions et bornes d'incertitudes.

On choisit, arbitrairement, un borne de probabilité à 5% avec une probabilité de 1% d'être en dehors de la borne. La borne de Chernoff nous donne alors un nombre de test minimal de 1 060 tests pour pouvoir garantir ces bornes.

Pour que tous les algorithmes soient testés dans les même conditions, les 1 060 mondes Gazebo nécessaires ont tous été générés et sont disponibles dans le répertoire `gazebo/benchmark_v1.0_worlds` du paquet publié. Pour la même raison, une séquence de point de passage est aussi générée et sauvée dans un fichier `yaml` pour chaque monde généré.

4.4.4.2 Autres indicateurs

La probabilité de collision est évidemment l'indicateur principal de la performance des algorithmes d'évitement d'obstacles proposés. D'autres indicateurs peuvent cependant aussi donner des informations importantes sur les performances d'un algorithme.

Sur les vols réussis, les indicateurs suivants sont calculés :

Le temps de complétion moyen : Diminuer le temps de parcours d'une tâche est souvent d'une importance cruciale, soit pour pouvoir faire plus d'actions avec une autonomie limité soit simplement pour augmenter l'efficacité de l'action courante.

La vitesse de vol moyenne : Contrairement à ce que l'on pourrait penser, ceci n'est pas directement lié au temps de complétion moyen puisque des algorithmes peuvent faire plus de détour que d'autres. Cet indicateur peut être particulièrement important pour les vols dans des environnements épars où de possibles détours ont moins d'importance qu'une vitesse moyenne élevée.

La distance moyenne parcourue : Cet indicateur donne une information sur les détours effectués pour éviter les obstacles. Il mettra en avant les algorithmes qui tendent à couper au plus court et à voler dans des zones à haute densité d'obstacles, quitte à réduire la vitesse du quadrirotor pour ce faire.

L'énergie moyenne utilisée lors d'un test : Les algorithmes ayant les plus faibles temps de complétion devraient bien figurer au niveau de cet indicateur puisque la compensation de la gravité est la cause principale de consommation d'énergie pour les quadrirotors. Cet indicateur devrait tout de même privilégier les algorithmes offrant une progression fluide puisque des accélérations fortes augmentent la consommation énergétique.

Sur les vols qui ont donné lieu à une collision (échec), la distance linéaire moyenne collision est calculée. Il s'agit d'un indicateur très intéressant lors de tests avec un fort taux d'échec. Enfin, lors des vols qui ont excédé le temps maximal de test sans réaliser un kilomètre de vol et sans collisions, la distance linéaire moyenne est aussi calculée. Il est possible de se servir de cette donnée pour analyser la durée des tests et voir si celle-ci est trop courte ou si ce cas correspond à un drone bloqué dans un minimum local.

Pour conclure sur ce benchmark, celui-ci sert un double but. D'une part, il nous permettra, au cours de cette thèse, d'effectuer des tests très complets de nos algorithmes. D'autre part, et même principalement vu le temps investi pour le rendre aussi générique que possible, on espère, grâce à ce benchmark pouvoir comparer les algorithmes présentés depuis des années. De nombreux algorithmes d'évitement d'obstacles ont été proposés ces dernières années

et il est à présent pratiquement impossible de comparer une contribution aux algorithmes existants. Cette impossibilité à comparer les différents algorithmes est amplifiée par le fait que la majorité des algorithmes d'évitement d'obstacles sont à sources fermées en raison de leur potentiel commercial. Même sans cela, comparer son code à un code existant requiert de programmer une interface entre le code existant et les outils de test personnels. Cela peut impliquer un temps important et être source d'erreur. En créant ce benchmark, qui est aussi générique et facile d'usage que possible, on aide les créateurs d'algorithme d'évitement d'obstacles en leur fournissant un environnement de test complexe et totalement automatisé. En contrepartie, on espère que cela va les inciter à publier leurs résultats sur ce benchmark ce qui permettra d'établir des points de comparaisons entre les différents algorithmes.

Le benchmark proposé étant open source, il peut évoluer. La modélisation des capteurs, voir le nombre de capteurs, ainsi que la simulation du quadrirotor et l'ensemble des blocs proposés peuvent ainsi être améliorés par la communauté qui peut, à son tour ajouter des tests pour que ceux-ci correspondent mieux à ses besoins.

On notera finalement que ce type de test statistique, en simulation, peut être étendu à de nombreuses autres tâches en robotique. Ne serait-ce que pour les quadrirotors, il est possible d'imaginer le même type de benchmark pour des tâches comme la saisie d'objets ou encore l'inspection à courte distance.

Chapitre 5

Résultats de contrôle et d'évitement

On va, dans un premier temps, se concentrer sur les résultats obtenus autour des problématiques de contrôle. La partie 5.1 sera ainsi l'occasion de comparer plusieurs méthodes de suivi de trajectoires à la fois en simulation et dans une salle de capture de mouvement.

Dans la partie suivante (5.2), on présentera les résultats obtenus avec des quadrirotors réels. Ce sera l'occasion de s'attarder sur les difficultés que l'on a eu en cours de projet pour obtenir un drone qui soit capable de se positionner en extérieur et sur les résultats obtenus dans la salle de capture de mouvements.

La partie 5.3 marquera le retour à la simulation. On présentera en effet dans cette partie les résultats du benchmark ce qui nous permettra de caractériser plus précisément les forces et faiblesses des algorithmes d'évitement d'obstacles présentés.

L'avant dernière partie (5.4) sera dédiée à la présentation des résultats d'évitement dynamique. Ce sera l'occasion de comparer l'évitement dynamique d'obstacles éparses et l'évitement dynamique dans un environnement où tous les obstacles se déplacent.

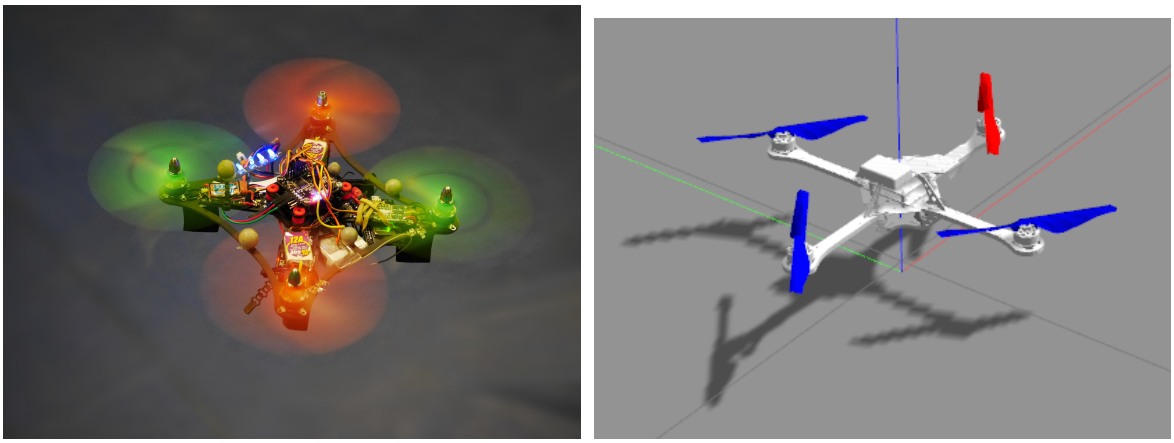
Finalement, la partie 5.5 sera consacrée au détail des temps de calculs de chacun des algorithmes proposés. Il s'agit d'une transition naturelle vers les conclusions et perspectives puisque discuter des temps de calculs des différentes parties des algorithmes permettra de pointer un certain nombres d'améliorations envisageables.

5.1 Étude de la qualité de suivi de trajectoires

Remarque. *Si dans la partie 3.5.3, j'ai clairement expliqué que je n'ai pas développé le contrôleur à retour d'état utilisé en fin de thèse, j'ai créé la totalité des outils nécessaires à la comparaison des différents modes de contrôle et j'ai effectué les tests présentés dans la suite de cette partie. Quoi qu'il en soit, et même si je n'ai pas proposé d'amélioration au niveau du suivi de trajectoire au cours de cette thèse, la qualité du contrôle étant un élément déterminant dans les performances finales de l'algorithme, il m'apparaissait obligatoire de présenter les résultats de suivi de trajectoires obtenus au laboratoire.*

Les tests de suivi de trajectoires ont été réalisés à la fois en simulation et dans la salle de capture de mouvement en utilisant les données de position issues du système de capture

de mouvements. La figure 5.1 montre une image des deux drones impliqués dans ces tests. Le munchkin42, le drone visible en figure (a) pèse 232 g sans sa batterie 2S, pour une diagonale de 33 cm quand le drone simulé en (b) est un AscTec Hummingbird et pèse 680 g pour 57 cm de diamètre.



(a) (b)

FIGURE 5.1 – Vues des deux drones impliqués dans les tests sur la qualité de contrôle.

Les trajectoires sont générées grâce à la méthode de génération de trajectoires utilisée dans les différents algorithmes, la méthode de MUELLER, HEHN et D'ANDREA[MHD15] présentée dans la partie 3.4.4. Deux types de trajectoires différentes ont été testées.

La première trajectoire est triangulaire et invariante en z. La vitesse à chaque coin est nécessairement nulle pour que la trajectoire soit faisable, chaque arête mesure deux mètres et il est possible de régler la vitesse souhaitée au milieu des arêtes. Des tests ont été réalisés avec des vitesses de 1.5m/s et 3m/s ce qui correspond à des accélérations moyennes de $1.5\,\mathrm{m\,s^{-2}}$ et $3\,\mathrm{m\,s^{-2}}$. A titre indicatif, atteindre ces vitesses nécessite de générer des trajectoires dont l'agressivité vaut respectivement 0.7 et 5.5. On rappelle que l'agressivité permet d'obtenir une estimation de la complexité de la trajectoire à suivre comme cela a été expliqué dans la partie 3.4.2 et qu'elle est, dans notre cas, définie comme la valeur moyenne du jerk élevé au carré comme cela a été détaillé dans la partie 3.4.4.

La seconde trajectoire forme un huit de 2 mètres de long parcouru deux fois en 3 dimensions. La trajectoire est générée pour que la vitesse soit à peu près constante sur toute la trajectoire. Encore une fois, des tests ont étés réalisés à 1.5 et 3m/s ce qui correspond pour cette trajectoire à des agressivités de 1.2 et 9.5.

On a testé un total de 3 contrôleurs : un contrôleur en cascade composé du retour d'état créé au sein de laboratoire et d'un PID en attitude, un contrôleur en cascade composé de deux PID et le *lee position contrôleur* [LLM10].

Avant de donner des résultats quantitatifs, la figure 5.2 permet de se faire une idée des performances de ces différents contrôleurs. Seuls les tests sur les huits à 1.5m/s sont présentés

sur cette figure. La figure (a) correspond à l'exécution de la trajectoire avec le munchkin42 et le contrôleur à retour d'état, (b) correspond au munchkin42 avec deux PIDs en cascade, (c) au hummingbird simulé avec le lee contrôleur, (d) au hummingbird avec le retour d'état et (e) au hummingbird avec les PIDs en cascade. Cette figure permet de constater visuellement que le suivi de trajectoire semble moins précis avec les PIDs en cascade qu'avec les deux autres contrôleurs. On va cependant chercher à vérifier cela en quantifiant les erreurs de suivi.

Pour quantifier ces tests de suivi de trajectoire, on définit six distances : la distance moyenne à la trajectoire définie d'après l'équation 5.1, la distance maximale à la trajectoire définie à partir de l'équation 5.2, la distance tangente moyenne à la trajectoire (équation 5.3), la distance tangente maximale (équation 5.4), la distance normale moyenne à la trajectoire (équation 5.5) et enfin la distance normale maximale à la trajectoire (équation 5.6). Dans ces équations, $x(t)$ correspond à la position du drone à l'instant t et $x_{ref}(t)$ correspond à la trajectoire de commande à l'instant t. t_0 correspond au temps du début de la trajectoire, t_f au temps à la fin de la trajectoire et Δt correspond au temps entre deux points de la trajectoire de commande.

$$D_{moy} = \int_{t_0}^{t_f} D(t)\,\mathrm{d}t = \int_{t_0}^{t_f} \|x(t) - x_{ref}(t)\|\,\mathrm{d}t \tag{5.1}$$

$$D_{max} = Max\,\|x(t) - x_{ref}(t)\|\,, \quad t \in [t_0, t_f] \tag{5.2}$$

$$TD_{moy} = \int_{t_0}^{t_f} TD(t)\,\mathrm{d}t = \int_{t_0}^{t_f} \frac{\|(x(t) - x_{ref}(t)).(x_{ref}(t-\Delta t) - x_{ref}(t))\|}{\|x_{ref}(t-\Delta t) - x_{ref}(t)\|}\,\mathrm{d}t \tag{5.3}$$

$$TD_{max} = Max\frac{\|(x(t) - x_{ref}(t)).(x_{ref}(t-\Delta t) - x_{ref}(t))\|}{\|x_{ref}(t-\Delta t) - x_{ref}(t)\|}, \quad t \in [t_0, t_f] \tag{5.4}$$

$$ND_{moy} = \int_{t_0}^{t_f} \frac{\|(x(t) - x_{ref}(t)) \times (x_{ref}(t-\Delta t) - x_{ref}(t))\|}{\|x_{ref}(t-\Delta t) - x_{ref}(t)\|}\,\mathrm{d}t = \int_{t_0}^{t_f} \sqrt{D(t)^2 - TD(t)^2}\,\mathrm{d}t \tag{5.5}$$

$$ND_{max} = Max\frac{\|(x(t) - x_{ref}(t)) \times (x_{ref}(t-\Delta t) - x_{ref}(t))\|}{\|x_{ref}(t-\Delta t) - x_{ref}(t)\|}, \quad t \in [t_0, t_f] \tag{5.6}$$

La table 5.1 synthétise les résultats des tests de contrôle et donne toutes les erreurs précédemment définies dans les différentes configurations précisée par les quatre premières colonnes de la table.

Remarque. *Plusieurs exécutions du même scénario en simulation permettent de constater le suivi de trajectoire en simulation est quasiment déterministe puisque l'écart maximal entre les*

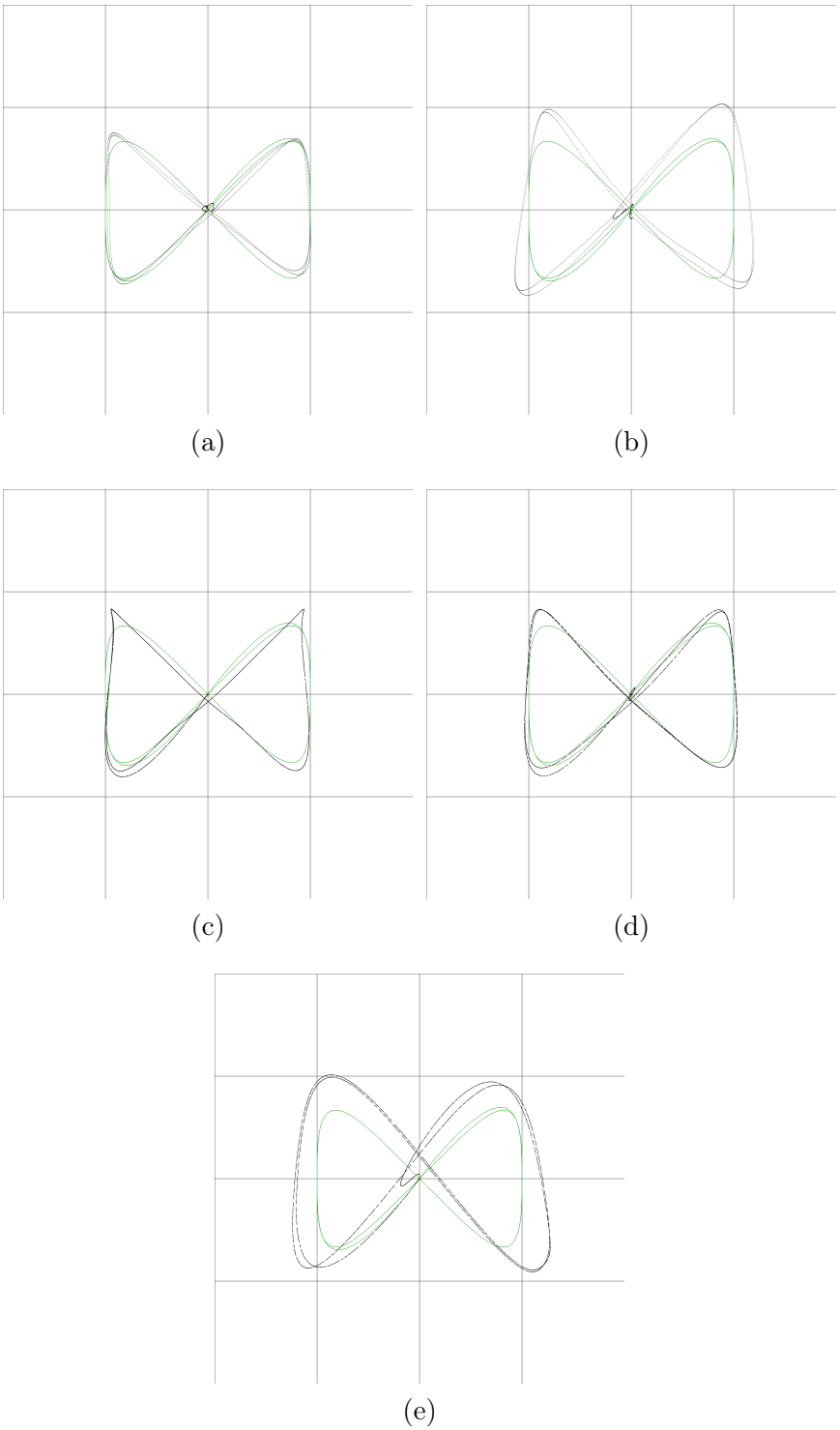


FIGURE 5.2 – Vue de dessus des trajectoires effectuées avec différentes options de contrôle. (a) munchkin42 et retour d'état. (b) munchkin42 et PIDs en cascade. (c) hummingbird simulé et lee contrôleur. (d) hummingbird simulé et retour d'état. (e) hummingbird simulé et PIDs en cascade.

Drone	Contrôleur	Type de traj	Vitesse	D_{moy}	D_{max}	TD_{moy}	TD_{max}	ND_{moy}	ND_{max}
Munchkin42	retour d'état	Triangle	$1.5\,\mathrm{m\,s^{-1}}$	0.13	0.31	0.12	0.31	0.029	0.076
Munchkin42	PID	Triangle	$1.5\,\mathrm{m\,s^{-1}}$	0.15	0.33	0.12	0.33	0.06	0.22
Humingbird	lee	Triangle	$1.5\,\mathrm{m\,s^{-1}}$	1.05	1.94	0.76	1.93	0.49	1.93
Humingbird	retour d'état	Triangle	$1.5\,\mathrm{m\,s^{-1}}$	0.05	0.15	0.04	0.15	0.01	0.05
Humingbird	PID	Triangle	$1.5\,\mathrm{m\,s^{-1}}$	0.25	0.44	0.22	0.44	0.07	0.36
Munchkin42	retour d'état	Huit	$1.5\,\mathrm{m\,s^{-1}}$	0.20	0.37	0.19	0.36	0.06	0.12
Munchkin42	PID	Huit	$1.5\,\mathrm{m\,s^{-1}}$	0.21	0.38	0.12	0.28	0.15	0.37
Humingbird	lee	Huit	$1.5\,\mathrm{m\,s^{-1}}$	1.20	2.70	0.70	2.42	0.75	2.70
Humingbird	retour d'état	Huit	$1.5\,\mathrm{m\,s^{-1}}$	0.06	0.14	0.05	0.14	0.03	0.11
Humingbird	PID	Huit	$1.5\,\mathrm{m\,s^{-1}}$	0.41	1.05	0.36	1.03	0.17	0.35
Humingbird	retour d'état	Triangle	$3.0\,\mathrm{m\,s^{-1}}$	0.21	0.45	0.18	0.41	0.08	0.26
Humingbird	PID	Triangle	$3.0\,\mathrm{m\,s^{-1}}$	0.59	1.23	0.53	1.19	0.20	0.47
Humingbird	retour d'état	Huit	$3.0\,\mathrm{m\,s^{-1}}$	Crash					
Humingbird	PID	Huit	$3.0\,\mathrm{m\,s^{-1}}$	Crash					

TABLE 5.1 – Résultats des tests de suivi de trajectoire.

trajectoires suivies lors de ces multiples exécutions est resté inférieur au centimètre. Les résultats proviennent donc d'un unique test en simulation. Au contraire, pour les tests HITL, il est possible de constater visuellement des variations de la trajectoire suivie pour une même trajectoire de commande. Pour cette raison, les résultats communiqués sont des résultats moyennés sur plusieurs exécutions. La distance maximale à la trajectoire est la plus grande distance tous tests confondus.

On notera que les tests des différents contrôleurs ont été réalisés sans modifier les contrôleurs d'attitude lorsque cela a du sens (retour d'état et PIDs). De même, les gains des contrôleurs n'ont pas été modifiés lors du passage d'un drone à l'autre. Ces tests n'ont pas été présentés ici mais les mêmes gains ont servi à contrôler des tous petits quadrirotors ($<100\,\mathrm{g}$) et des quadrirotors plus gros ($<1.7\,\mathrm{kg}$) avec des performances relativement proches de celles détaillées ici.

Première constatation, les erreurs associées au lee contrôleur sont beaucoup plus élevées que les erreurs associées aux deux autres contrôleurs et cela est en contradiction avec les observations que l'on avait fait sur la figure 5.2. Une étude plus précise des vols avec le lee contrôleur permettent de montrer que ces vols sont en fait réalisés à une vitesse plus faible que la vitesse demandée. Le huit à $1.5\,\mathrm{m\,s^{-1}}$ est par exemple réalisé en 28 s au lieu de 15 s. L'erreur importante est donc principalement due au retard du drone par rapport à l'état commandé. Cette impossibilité de commander précisément une suite d'état est problématique pour éviter un obstacle rapidement ce qui peut être nécessaire dans le cas d'obstacles dynamiques et il ne sera donc pas possible d'utiliser ce contrôleur dans notre projet.

Deuxième constatation, l'erreur augmente bien avec l'agressivité et ce dans toutes les configurations (modèle de quadrirotors + contrôleur) testées. C'est un résultat attendu qu'il était important de vérifier

On observe ensuite que le retour d'état est toujours plus performant que les PIDs. C'est relativement clair en simulation mais beaucoup moins marqué si l'on regarde les erreurs moyenne avec le Munchkin42. Ces erreurs ne sont cependant pas les erreurs les plus importantes, l'erreur que l'on souhaite le plus minimiser est l'erreur normale qui est l'erreur la plus à même de causer des collisions vu notre méthode de sélection de trajectoire. Or, pour cette erreur normale, l'erreur moyenne lors du suivi par retour d'état est toujours au minimum deux fois plus faible que l'erreur lors du suivi en utilisant les PIDs. En considérant que, pour une utilisation dans cette thèse, la performance des contrôleurs est corrélée à cette erreur normale, le contrôleur par retour d'état est donc le contrôleur testé le plus performant. Il s'agira donc du contrôleur utilisé pour effectuer le suivi de trajectoire dans cette thèse.

On constate aussi que le suivi de trajectoires est plus précis avec le Hummingbird simulé qu'avec le munchkin42, jusqu'à trois fois plus précis avec le contrôleur par retour d'état. Il s'agit également d'un résultat attendu puisque les conditions sont bien mieux contrôlées en simulation qu'avec un drone réel. Le centre du gravité du drone est en particulier situé exactement au centre géométrique en simulation ce qui est loin d'être le cas pour le munchkin42.

Si l'on se replace dans le cadre de l'évitement d'obstacles, cette étude permet d'établir une marge de sécurité en fonction des contrôleurs et drones utilisés. On constate, par exemple, qu'avec le contrôleur à retour d'état, quelque soit la trajectoire à $1.5\,\mathrm{m\,s^{-1}}$ ou à agressivité inférieure ou égale à 1.2, l'erreur normale est toujours inférieure au rayon du drone. On peut donc raisonnablement penser qu'un C-Space de rayon égale au rayon du drone permettra de compenser les erreurs en suivi de trajectoire. Un tel C-Space permettrait au drone de passer entre deux obstacles dès lors que l'espace qui les sépare est supérieur à deux fois le diamètre du drone.

5.2 Résultats d'évitement d'obstacles avec quadrirotors réels

On entame cette partie en discutant des tests en extérieur et, pour cela, on commence par signaler que tous les vols ont été effectués conformément à la réglementation en vigueur.

Maintenant que cette précision a été donnée, il faut mettre en avant le point le plus décevant de cette thèse : on a été dans l'incapacité de conduire des tests d'évitement d'obstacles en extérieur. La raison principale de cet échec étant qu'il n'y a eu, durant le projet CAP2018, aucun vol à pilotage automatique réalisé en extérieur qui a donné lieu à un contrôle satisfaisant alors qu'il s'agissait de ce qui avait été considéré comme un pré-requis lors de développement de l'algorithme l'évitement d'obstacles.

Il serait possible de discuter des raisons de cet échec pendant des pages mais celui-ci est en réalité la conséquence de problèmes organisationnels. Des algorithmes d'évitement d'obstacles ont été validés dans la salle de capture de mouvement moins d'un an et demi après le début de la thèse lors des tests HITL présentés dans la suite quand les premiers tests de vol en extérieur n'ont pas eu lieu avant fin mars 2019 soit plus d'un an après et ce essentiellement pour des raisons administratives et financières. Entre temps, constatant que l'entreprise

membre du projet CAP2018 étant sensée créer le quadrirotor sur lequel notre solution viendrai s'ajouter prenait du retard sur l'estimation d'état, on a récupéré cette fonctionnalité au sein du laboratoire. Cela aurait du entraîner une charge de travail supplémentaire mais cette charge n'a jamais été attribuée à un ou des membres du projet au sein du laboratoire et n'a donc pas été réalisée. Cette situation nous à conduit à régler des problèmes d'ordres matériels sur le drone entre fin mars et fin juin 2019 puis à être confronté à des problèmes d'estimation d'état depuis cette date. On estime à présent être proche d'obtenir une estimation d'état suffisante pour effectuer un vol stationnaire en extérieur en se basant sur un contrôle en position mais cela ne garantit en rien que les algorithmes d'évitement d'obstacles seront alors fonctionnels.

Comme cela a été précédemment dit, des tests ont tout de même été réalisés avec un quadrirotor réel dans la salle de capture de mouvements. Ces tests ont été réalisés avec le munchkin42, le drone présenté lors des résultats de contrôle. La figure 5.3 montre une vue de dessus d'un des tests réalisés. Les murs et les obstacles sont visibles en noir ou en gris foncé. Le volume occupé par le drone est représenté en gris clair avec en rouge la direction de la caméra. Les trajectoires en bleu sont toutes les trajectoires qui ont été générées au cours du test et la trajectoire en verte correspond à la trajectoire suivie par le drone. Finalement, la forme géométrique marron délimite l'espace libre tel que perçu par le drone et les trois croix vertes représentent les trois points de passages qui ont été atteints.

Plus de cinquante tests et probablement plus de cent tests ont été réalisés depuis janvier 2018 dans l'environnement HITL. Tous ces tests n'ont pas nécessairement donné lieu à des prises de notes rigoureuses puisqu'après une phase de test initiale d'environ 50 tests, la majorité de ces tests ont eu pour objectifs de présenter le laboratoire à des extérieurs en visite.

Le résultat principal que l'on a pu obtenir est le suivant : sur la totalité des tests effectués, il n'y a aucune collision lorsque le rayon du C-space était supérieur ou égal à 1.5 * le rayon du drone et lorsque l'agressivité était inférieure à 1.5.

Ces résultats sont évidement satisfaisants mais difficilement généralisables à des vols en extérieurs à cause de la taille de la pièce qui force les vols effectués a être très différents des vols extérieur. On a, entre autre, une profondeur maximale dans l'environnement HITL de 3m au lieu des 15m que l'on considèrent habituellement et cela a un impact fort sur les caractéristiques des vols, en particulier sur les vitesses de vol et sur la capacité à éviter les minimums locaux.

5.3 Caractérisation des algorithmes en simulation

Commençons cette partie en analysant les résultats obtenus sur l'environnement de test simple (présenté dans la partie 4.1.2, environnement sans perturbations avec entrées de l'algorithme non bruitées) avec 100 cylindres verticaux. On a obtenu, avec le premier algorithme présenté, l'algorithme SOTA, un taux de succès de 99.4% sur 1000 tests et ce avec une perception limitée au champ de vision de l'Intel® RealSense™ avec une profondeur maximale de 20 m, une agressivité de 1.2 et un C-space de 56 de rayon soit deux fois le rayon du drone.

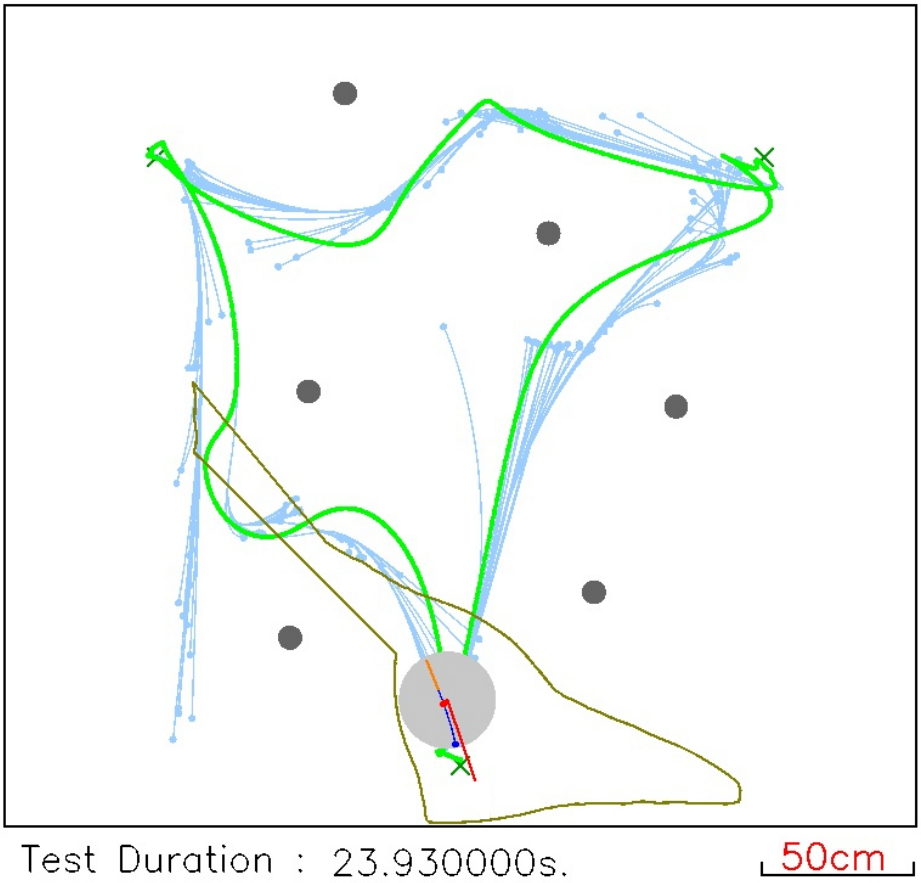


FIGURE 5.3 – Vue de dessus du résultat d'un test HITL.

La vitesse moyenne sur ces vols a été de $2.42\,\mathrm{m\,s^{-1}}$ en prenant en compte le temps de rotation à l'arrêt entre chaque point de passage pour des vitesses typiques en cours de trajectoires comprises entre $4\,\mathrm{m\,s^{-1}}$ et $5.5\,\mathrm{m\,s^{-1}}$. Le détour moyen à quant à lui été de 3% de la distance linéaire parcourue soit 3.59 m pour un trajet de 117.5 m.

Ces résultats sont très satisfaisants puisqu'ils constituent les meilleurs résultats disponibles dans la littérature sur de l'évitement d'obstacles en simulation et sur un aussi grand échantillon de tests.

L'autre information que donne ces résultats est la nécessité de passer à des tests plus complexes. L'incertitude sur la probabilité de succès sur 1000 tests est en effet légèrement supérieure à 5% (pour conserver un intervalle de confiance sur la probabilité de 99%) ce qui veut dire que l'incertitude sur la probabilité de collision est d'un ordre de magnitude supérieur à la probabilité de collision obtenue (0.6%). Il ne sera donc plus possible d'utiliser ces tests pour comparer des solutions entre elles ou plus précisément, il ne sera plus possible d'utiliser ces tests pour comparer des solutions lorsque utilisées dans leurs meilleures plages de fonctionnement. Or s'il est intéressant de comparer les limites de ces solutions (en réduisant les marges de sécurité, augmentation des agressivités des vols, réduction de la portée des capteurs de profondeur...), on souhaite tout de même pouvoir comparer les solutions quand elles fonctionnement le mieux.

C'est le moment où le benchmark rentre en scène avec son environnement beaucoup plus complexe et ses perturbations récurrentes. On note que l'on utilise toujours une seule caméra de profondeur, la camera bruitée correspondant à une Intel® RealSense™ D435 ainsi que le contrôle par retour d'état développé au sein du laboratoire 3.5.3. L'état utilisé sera **l'état parfait**, non bruité, puisque comme cela a été expliqué dans la partie précédente, l'estimation d'état n'est pas fonctionnelle. On finit par rappeler que la perturbation qui simule du vent donne lieu à un déplacement constant d'une trentaine de centimètres selon l'axe du repère inertiel dirigé par $\vec{i_1}$ plus un déplacement à haute fréquence d'environ 1 cm et que toutes les 15 s, une bourrasque de même intensité tend à déplacer le drone d'environ 30 cm plus 1 cm à haute fréquence selon l'axe dirigé par $\vec{i_2}$. On est donc dans un environnement très largement perturbé avec pour seule, mais importante, divergence avec une situation réelle dans l'utilisation d'un état exact et non estimé.

Le tableau 5.2 synthétise les résultats obtenus. La durée maximale d'un test est de 800 secondes. La colonne LocMin indique l'utilisation de méthodes destinées à s'extraire des minimums locaux. Pour l'algorithme basé sur le filtrage spatial des trajectoires générées, il s'agit du plugin présenté dans la partie 3.8 et pour l'algorithme SOTA, il s'agit d'un simple manager qui ajoute un point de passage s'il le drone reste immobile pendant plus de 20s.

Première constatation, les résultats sont très loin des 0.6% de collision des tests que l'on appelle dans cette thèse simple. Cela montre qu'il nous reste du travail à accomplir pour obtenir un résultat parfait mais cela montre aussi que la grande majorité des tests en simulation

Algo	Agr	R_{Cspace}	LocMin	Test Nb	Succès	Col	OOT	LD_{col}	LD_{oot}	D_{suc}	V_{suc}
SOTA	1.2	43	N	103	13.6%	56.3%	30.1%	331	346	1068	2.54
SOTA	1.2	56	O	175	17.5%	55.4%	27.1%	362	411	1113	2.48
SOTA	0.12	56	O	132	14.4%	63.6%	22.0%	476	413	1153	1.62
Filt Spa	1.2	56	N	20	25%	70%	5%	322	964	1370	2.96
Filt Spa	0.12	56	O	417	**60.8%**	37.6%	1.4%	446	870	1520	2.31
Filt Spa	0.12	49	O	1060	50.8%	48.6%	0.7%	424	809	1469	2.31
Filt Spa	0.12	42	O	172	36.6%	62.8%	0.6%	422	159	1419	2.32
Filt Spa	0.12	29	O	259	1.9%	98.1%	0.0%	260		1308	2.37

TABLE 5.2 – Résultats du benchmark pour différents algorithmes d'évitement d'obstacles.

présentés dans la littérature, qui sont plus simples que nos *tests simples*, sont insuffisants pour pouvoir juger des performances d'un algorithme dans des environnements complexes.

Deuxième constatation, le nouvel algorithme développé performe bien mieux que l'algorithme SOTA. Le taux de succès des vols passe en effet d'un maximum de 17.5% avec l'algorithme SOTA à 60.8% avec l'algorithme utilisant des filtres spatiaux sur les trajectoires générées. Pour mettre ces résultats en perspectives, c'est bien avec l'algorithme SOTA que l'on avait atteint les 0.6% de collisions sur les tests simples.

On constate cependant un défaut majeur de l'algorithme utilisant du filtrage spatial, la distance effective parcourue pour compléter les tests (>1400m) est très supérieure à la distance parcourue avec l'algorithme SOTA (<1150m). Cela s'explique par le moyennage des coûts qui ont tendance à faire dévier le drone plutôt qu'à le faire passer dans un passage étroit. On espère cependant qu'une meilleure paramétrisation permette de limiter en partie cet effet.

Autre constatation, pour l'algorithme avec filtrage spatial, les performances s'améliorent avec un C-space plus large alors que l'on espérait que le filtrage permettent de réduire la taille du C-space. Il est en particulier intéressant de noter qu'entre un C-space de 1.75 * le rayon du drone (49cm) et un C-space de deux fois le rayon du drone (56cm), il y a toujours une amélioration signification des performances. Il serait intéressant de voir jusqu'à où ce facteur influe positivement.

Autre point sur lequel on souhaite insister, la distance linéaire moyenne parcourue avant collision permet de mettre en évidence que certains environnements sont naturellement plus complexes que d'autres. Si la probabilité de crash était linéaire, la distance de collision moyenne tendrait vers 500m alors que celle-ci est bien plus faible. Ceci est particulièrement visible lorsque l'on a fait des statistiques sur 1060 tests où plus de 130 tests ont donné lieu à des crashs au cours des 200 premiers mètres linéaires pour moins de 65 tests ayant donné lieu à une collision sur les 200 derniers mètres de vol. Cela met une fois de plus en avant la difficulté des environnements dans lesquels les tests sont effectués. Certaines des forêts sont si denses qu'un humain ne pourrait y passer et certaines des forêts créent de long couloirs desquels il est très difficile de sortir lorsqu'ils se terminent par un cul-de-sac. De notre point de vue, l'environnement proposé est plus complexe que les divers chemins de trails qui ont pu être suivis en extérieur et on est très curieux de savoir quelles performances pourraient obtenir les

différents algorithmes précédemment proposés dans la littérature.

On estime que c'est la complexité de l'environnement, ainsi que les importantes perturbations s'y ajoutent, qui sont la cause de la majorité des erreurs. On pense que vu la complexité de l'environnement, on atteint les limites de ce qu'il est possible d'effectuer avec des algorithmes réalisant une planification uniquement locale et qu'une planification globale devient nécessaire pour éviter de se retrouver bloqué au milieu de la forêt.

5.4 Résultats des tests avec obstacles dynamiques

L'environnement dynamique *éparse*, présenté notamment via la figure 4.4, dans lequel quelques obstacles se déplacent selon des directions spécifiques a été utilisé lors de la création de l'algorithme présenté dans la partie 3.9. Il permet de vérifier que l'algorithme proposé est capable d'éviter des obstacles provenant de directions identifiées comme problématiques dans les phases initiales du développement de l'algorithme d'évitement d'obstacles dynamiques.

La figure 5.4 présente des vues de dessus des positions du drone et des trajectoires commandées à une seconde d'intervalle sur huit secondes lors d'un test mené dans l'environnement dynamique plus complexe comprenant un nombre arbitraire d'obstacles se déplaçant à une vitesse paramétrable. Dans cet exemple, l'agressivité du drone a été réglée de sorte à ce que celui-ci vole environ à $4\,\mathrm{m\,s^{-1}}$ quand les cinquante obstacles ont des vitesses d'environ $2\,\mathrm{m\,s^{-1}}$. Les tests sont effectués en utilisant un état parfait, une perception parfaite mais limitée au champ de vision d'une Intel® RealSense™, le contrôleur à retour d'état et le simulateur RotorS présenté en 4.1.1.

Remarque. *De par la nature dynamique de ces tests, leur illustration avec des images statiques reste limitée. Comme pour la majorité des tests, des vidéos ont été réalisées et seront présentées lors de la soutenance de cette thèse.*

Si l'on regarde ces résultats non plus dans le cas particulier du test illustré par la figure 5.4 mais pour de multiple tests dans les mêmes conditions, l'algorithme proposé permet d'obtenir un taux de succès d'environ 30%. Ce taux de succès est donné de manière approximative parce qu'il est difficile de lui donner du sens. Dans les différents environnements complexes dans lesquels des tests ont été effectués, il est en effet possible d'obtenir des tests positifs ou des tests négatifs simplement en faisant varier le début du mouvement du drone par rapport aux déplacements des obstacles. Ce taux de succès est néanmoins à mettre en parallèle avec ceux des deux algorithmes ne prenant pas en compte la dynamique de l'environnement : 0%.

On attribue la majorité des échecs à deux facteurs principaux. Le premier de ces facteurs est le champ de vision réduit. Assez fréquemment, l'algorithme génère des trajectoires qui entrent en collision avec des obstacles qui n'ont jamais été observés mais qui se sont rapprochés du drone au cours de leurs déplacements. Deuxième facteur de collision, la multiplication des obstacles se déplaçant dans des directions différentes. Ces mouvements génèrent des informations contradictoires et il est alors difficile d'évaluer la pertinence de la trajectoire choisie par

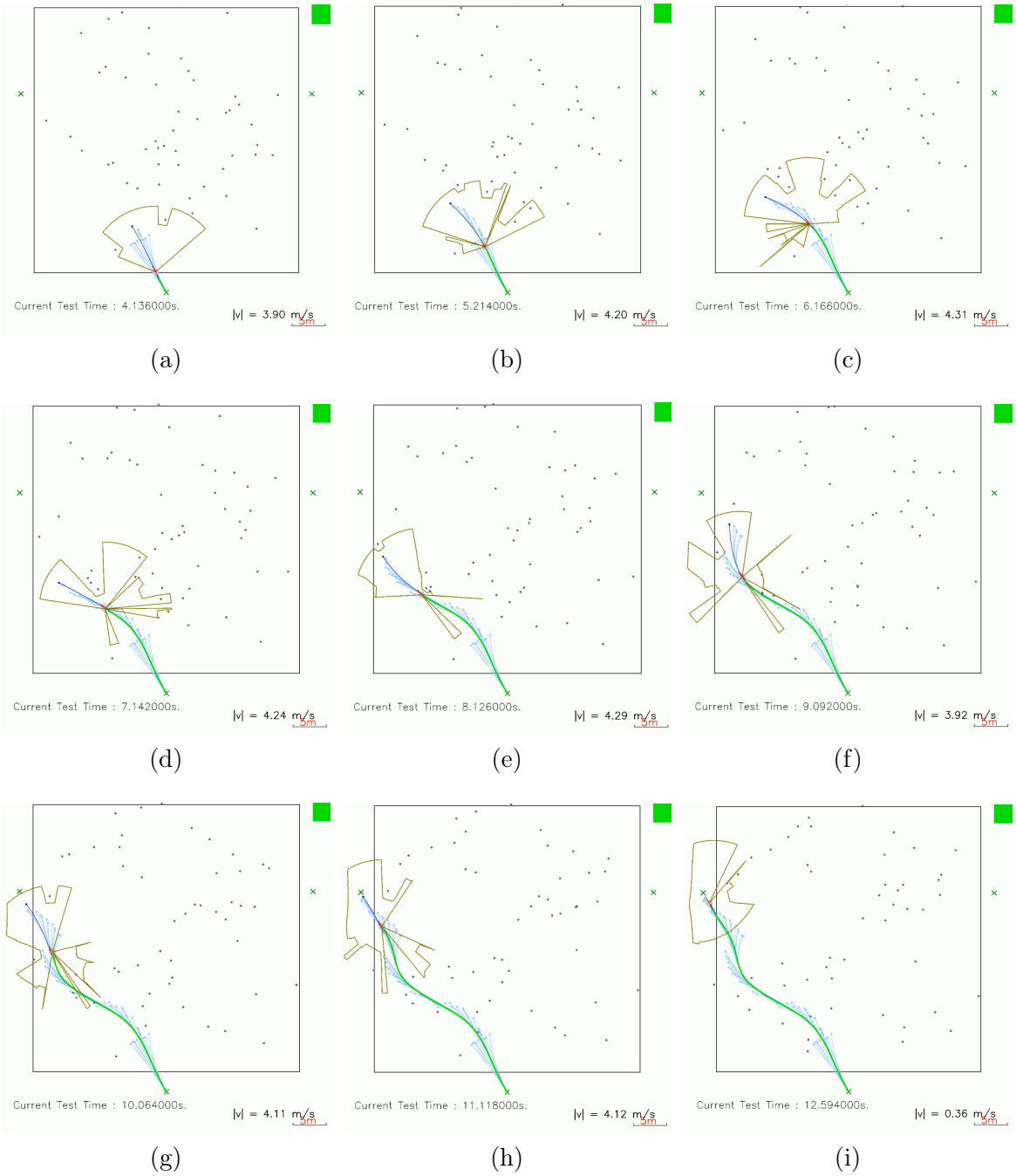


FIGURE 5.4 – Évolution sur 8 secondes d'un quadrirotor dans l'environnement dynamique aléatoire.

l'algorithme. On peut cependant constater que c'est principalement dans ces conditions que les trajectoires choisies sont en dehors du champ de vision de la caméra de profondeur ce qui augmente la probabilité de collision avec un obstacle non observé.

Si les résultats obtenus sont très largement perfectibles et méritent une étude plus approfondie, il est clair que l'algorithme performe mieux que les algorithmes ne prenant pas en compte la dynamique de l'environnement. Cela est confirmé par les tests dans l'environnement dynamique épars pour lesquels les algorithmes réalisés pour des environnements statiques échouent aussi. Cela signifie qu'on a effectivement réussi à ajouter une information sur la dynamique de la scène à notre algorithme de filtrage spatial.

Si l'on prend du recul sur ces résultats et qu'on les met en parallèle avec le plugin réalisé pour éviter les minimums locaux, on se rend compte que l'algorithme développé autorise l'ajout d'informations permettant de satisfaire des buts secondaires en plus d'effectuer de l'évitement d'obstacle vers un point de l'espace. Pour reprendre des exemples de buts secondaires qui ont fait l'objet de publications, cela permettrait par exemple de privilégier certaines directions pour améliorer les prises de vues depuis le drone ou encore de privilégier des directions permettant d'améliorer les performances d'un algorithme visuo-inertiel.

5.5 Temps de calcul des algorithmes

Pour finir sur les résultats obtenus, la table 5.3 détaille les temps d'exécutions des différents algorithmes sur mon ordinateur portable de travail pour toutes les versions de l'algorithme et pour comparaison, sur une NVIDIA Jetson TX1 pour l'algorithme SOTA. On en profite pour préciser que, lors de ces tests comme pour toute la thèse, l'egocubemap est de taille 128x128x6 pixels.

Phase de l'algorithme	Lap SOTA	Lap Filt Spa	Lap Dyna	Jet SOTA
Création de l'egocubemap		38.2 ms		35 ms
dont propagation		24.8 ms		23 ms
dont calcul du Cspace		9.4 ms		9 ms
Sélection d'une Trajectoire	25.5 ms	67.5 ms	69.1 ms	55 ms
dont sélection de directions	2.5 ms	-	-	5 ms
dont générations de trajectoires	20.9 ms	65.7 ms	65.7 ms	42 ms
dont sélection d'une trajectoire	0.1 ms	0.2 ms	1.8 ms	0.1 ms
Contrôle PID ou retour d'état		<1 ms		
Total	64.7 ms	106.7 ms	108.5 ms	105 ms

Table 5.3 – Temps moyen d'exécution des algorithmes proposés.

Première remarque, centrée sur les algorithmes proposés, les choix économes en calcul effectués (vision 2,5D et utilisation d'une cubemap, génération de trajectoires quasi-instantanée, méthode de Flux optique rapide..) ont pour conséquences des temps d'exécutions des algorithmes relativement rapides sans avoir fait de réel effort d'accélération des codes. En effet,

seule la partie vision, écrite en OpenGL, s'éxécute sur GPU. Les autres fonctionnalités des algorithmes s'exécutent sur CPU et n'ont pas été parallélisé pour les tests sur ordinateur portable. Sur la Jetson TX1, ces fonctionnalités ont simplement bénéficié d'une parallélisation des boucles via OpenMP lorsque celle-ci était directe.

Plus généralement, alors qu'un temps d'exécution faible apparaissait en début de thèse comme une nécessité pour effectuer de l'évitement d'obstacles, la génération de commandes sous forme de trajectoires réduit grandement l'impact du temps d'exécution sur les performances. On voit par exemple que l'algorithme basé sur le filtrage spatial dure 67% plus longtemps que l'algorithme SOTA et que cela ne l'empêche de posséder de bien meilleures performances. Le temps d'exécution reste un facteur important dans l'efficacité des algorithmes d'évitement, comme on a pu le constater qualitativement en ralentissant et accélérant artificiellement les différents algorithmes créés, mais son importance est légèrement moindre que celle que l'on prévoyait en début de thèse, en particulier lorsque les environnements sont statiques.

Chapitre 6

Conclusion et Perspectives

6.1 Conclusions

Cette thèse a été consacrée à l'évitement d'obstacles pour quadrirotors. Des contributions théoriques, pratiques et méthodologiques ont été proposées.

Deux algorithmes d'évitement d'obstacles utilisant une unique caméra de profondeur et facilement adaptables sur de nombreux drones ont été proposés. Ces algorithmes se basent sur une carte de distance locale stockée dans une cubemap et de multiples générations de trajectoires en cours d'exécution grâce une méthode basée sur la minimisation du jerk. Ces algorithmes permettent d'éviter efficacement des obstacles aux profils géométriques complexes à des vitesses d'environ 4 m/s comme cela a pu être vérifié lors de simulations dans des environnements bruités et comportant des perturbations. L'adaptabilité de ces algorithmes à différents quadrirotors provient de l'utilisation d'une génération de trajectoire basée sur un modèle simplifié des quadrirotors, sur l'utilisation d'une représentation de l'environnement acceptant différentes sources et, surtout, sur l'utilisation d'un contrôle en cascade qui permet l'utilisation du contrôleur d'attitude natif de chaque quadrirotor.

Le second algorithme proposé utilise une nouvelle stratégie de génération et de sélection de trajectoires. Cette méthode permet d'ajouter une marge supplémentaire entre les trajectoires générées et les limites de sécurité issues de l'utilisation du Configuration-space. Cette marge supplémentaire est donc dé-corrélée de la sécurité issue de la perception ce qui rend le vol plus fluide et limite les comportements hésitants, composés de nombreux arrêts, qui sont courants sur les algorithmes de navigation et qui sont dues à la proximité avec les limites de sécurité. Cette méthode est plus sûre que la méthode la plus classique et est aussi la porte ouverte à de nombreuses extensions comme cela a pu être montré au cours de la thèse en effectuant de l'évitement dynamique avec une simple extension de l'algorithme proposé.

Les limites de ces algorithmes ont également pu être étudiées. L'une de ces limites est dans la qualité du contrôle qui induit une vitesse maximale d'environ $5\,\mathrm{m\,s^{-1}}$ lors de l'exécution de l'algorithme avec une Intel® RealSense™. Les tests effectués ne permettent pas de déterminer si c'est le suivi ou la génération de trajectoire qui montre ses limites à cette vitesse mais on ne peut que signaler que, d'une part comme de l'autre, des approximations ont été faites lors de la conception de ces méthodes. L'autre limite est dans l'utilisation d'une carte locale. Si cela permet une exécution rapide des algorithmes ce qui autorise, par exemple, l'évitement d'obstacles dynamiques, cette représentation locale de l'environnement montre ses

limites lorsque la tâche demandée n'est plus de l'évitement d'obstacles mais de la navigation autonome comme cela peut être le cas dans des forêts denses.

Le benchmark BOARR développé pour effectuer des tests d'évitement d'obstacles en simulation constitue l'autre contribution importante de la thèse. Ce benchmark a été construit pour tenter de simuler au mieux un des problèmes présents à de multiples reprises dans la littérature : la navigation dans une forêt dense avec un quadrirotor d'environ 500 g. Il a été développé sans chercher à prédire et maximiser les performances des algorithmes créés et en utilisant les librairies les plus communément utilisées dans le domaine, librairies qui n'étaient pas celles utilisées au sein du laboratoire avant la création du benchmark. Ce benchmark est maintenant disponible en open-source. Par sa mise à disposition, l'auteur de cette thèse espère pousser la communauté à tester leurs algorithmes avec plus d'exigence que ce qui est pour le moment réalisé. Comme le projet CAP2018 qui a financé cette thèse le met en avant, les quadrirotors peuvent être dangereux et il est important que tous les éléments critiques et que toutes les fonctions liées à la sécurité soient testées rigoureusement.

6.2 Perspectives

Si certains points ont pu être montrés au cours de la thèse, il reste néanmoins de très nombreuses questions autour des travaux effectués.

Commençons par l'algorithme d'évitement d'obstacles développé. Un travail de recherche de quelques mois pourrait se concentrer sur la modification de l'agressivité en fonction de l'environnement. L'idée serait de ralentir le drone, de réduire l'agressivité, lorsque l'environnement est perçu comme complexe, par exemple, lorsque celui-ci est dense ou lorsque des perturbations sont détectées. Cela permettrait d'augmenter la sécurité du drone dans les zones difficiles et au contraire d'augmenter ses performances dans les environnements perçus comme simples par le quadrirotor. Un travail de réflexion serait aussi le bienvenu sur le coût associé à chaque trajectoire. Le travail réalisé a permis de montrer l'intérêt d'effectuer un post-traitement sur l'ensemble des coûts calculés mais l'étude du calcul de chaque coût est restée superficielle. L'extension d'évitement pourrait aussi bénéficier d'amélioration ou au minimum d'un petit peu d'exploration. Que donnerait cette extension si elle ne se basait non plus sur le flux optique issu du capteur de profondeur mais sur l'évolution de la matrice de coût des trajectoires générées ? Dernières idées sur les algorithmes, plus lointaines, il serait très intéressant d'avoir à disposition un outil qui permette d'identifier les capacités de suivi de trajectoire de chaque quadrirotor sur lequel les algorithmes seraient exécutés. Cet outil d'identification permettrait d'adapter automatiquement les gains des contrôleurs et de génération de trajectoires à chaque quadrirotor. Finalement, il pourrait être intéressant de coupler les algorithmes d'évitement créés à un algorithme de navigation plus complexe. Vu la nature de l'algorithme d'évitement qui est totalement capable de gérer la navigation locale, un algorithme de navigation léger modélisant l'environnement grâce à des graphes semble être complémentaire et le couplage des deux algorithmes devrait donner des résultats intéressants.

Passons maintenant aux perspectives sur le benchmark BOARR qui est une proposition

d'outil qui a pour vocation d'évoluer et à terme d'être remplacé par une solution alternative de test. Il s'agit, lors de l'écriture de ce manuscript et à la connaissance de l'auteur de la thèse, de l'outil de test le plus complet pour évaluer les algorithmes de navigation et d'évitement d'obstacles pour quadrirotors. Pour une efficacité maximale, la logique voudrait que le benchmark BOARR continue d'évoluer tant que l'architecture générale utilisée en évitement d'obstacles, ROS/RotorS et tests en forêts, reste la même. Il est en effet plus facile de faire évoluer un outil que d'en recréer un. Les pistes d'améliorations du benchmark sont multiples. Tout est améliorable ! Que ce soit la simulation du drone, la simulation des capteurs, la génération de l'environnement, le principe des tests, la simulation du vent ... tout est améliorable ! Si je devais choisir une seule amélioration, je me concentrerai sur les modélisations des capteurs mais seule l'utilisation du benchmark par différentes équipes pourra permettre d'identifier quel travail est prioritaire. Au moment de l'écriture de ce manuscript, une autre équipe s'est déjà saisie du projet. A elle(s) et non à moi qui ai fait tous les choix initiaux de choisir les directions futures !

Bibliographie

[ACP17] Wilbert G Aguilar, Verónica P Casaliglla et José L Pólit. « Obstacle avoidance based-visual navigation for micro aerial vehicles ». In : *Electronics* 6.1 (2017), p. 10 (cf. p. 76).

[And+18] Olov Andersson et al. « Receding-horizon lattice-based motion planning with dynamic obstacle avoidance ». In : *2018 IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, p. 4467-4474 (cf. p. 31).

[AP19] Ross E Allen et Marco Pavone. « A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance ». In : *Robotics and Autonomous Systems* 115 (2019), p. 174-193 (cf. p. 31, 32).

[Ard17] Ardupilot. *Ardupilot Object Avoidance*. 2017. url : `http://ardupilot.org/dev/docs/code-overview-object-avoidance.html#code-overview-object-avoidance` (cf. p. 22).

[Arg+13] Lucas M Argentim et al. « PID, LQR and LQR-PID on a quadcopter platform ». In : *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*. IEEE. 2013, p. 1-6 (cf. p. 24).

[Asc13] Leonardo Ascorti. « An application of the extended Kalman filter to the attitude control of a quadrotor ». In : (2013) (cf. p. 25, 92).

[ASL19] ETHZ ASL. *Control strategies for rotary wing Micro Aerial Vehicles using ROS*. 2019. url : `https://github.com/ethz-asl/mav_control_rw` (cf. p. 95).

[Bar16] Andrew James Barry. « High-speed autonomous obstacle avoidance with pushbroom stereo ». Thèse de doct. Massachusetts Institute of Technology, 2016 (cf. p. 36, 47, 73, 76).

[BHD13] Dario Brescianini, Markus Hehn et Raffaello D'Andrea. *Nonlinear quadrocopter attitude control : Technical report*. Rapp. tech. ETH Zurich, 2013 (cf. p. 24).

[Bou+11] Hakim Bouadi et al. « Adaptive sliding mode control for quadrotor attitude stabilization and altitude tracking ». In : *2011 IEEE 12th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE. 2011, p. 449-455 (cf. p. 24).

[Bou16] Paul Bourke. *Converting to/from cubemaps*. 2016. url : `http://paulbourke.net/miscellaneous/cubemaps/` (cf. p. 68).

[Bro+16] Roland Brockers et al. « Vision-Based Obstacle Avoidance for Micro Air Vehicles Using an Egocylindrical Depth Map ». In : *International Symposium on Experimental Robotics*. Springer. 2016, p. 505-514 (cf. p. 37).

[BS07] Samir Bouabdallah et Roland Siegwart. « Full control of a quadrotor ». In : *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Ieee. 2007, p. 153-158 (cf. p. 16).

[BS11] John Brandt et Michael Selig. « Propeller performance data at low reynolds numbers ». In : *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. 2011, p. 1255 (cf. p. 20).

[Can+19] Bruce Canovas et al. « A Coarse and Relevant 3D Representation for Fast and Lightweight RGB-D Mapping ». In : *VISAPP 2019-International Conference on Computer Vision Theory and Applications*. 2019 (cf. p. 36).

[Cas+13] Robert T Casey et al. « Attitude representation and kinematic propagation for low-cost UAVs ». In : *AIAA Guidance, Navigation, and Control (GNC) Conference*. 2013, p. 4615 (cf. p. 24).

[Cha+17] Punarjay Chakravarty et al. « CNN-based single image obstacle avoidance on a quadrotor ». In : *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, p. 6369-6374 (cf. p. 76).

[civ18a] Direction de la sécurité de l'aviation civile. *AÉRONEFS TÉLÉPILOTÉS – ACTIVITÉS PARTICULIÈRES, Synthèse des démarches*. 2018. URL : `https://www.ecologique-solidaire.gouv.fr/sites/default/files/Synthese_demarches_activites_particulieres_drones.pdf` (cf. p. 19).

[civ18b] Direction de la sécurité de l'aviation civile. *Guide DSAC, Aéromodélisme : modèles réduits et drones de loisirs*. 2018. URL : `https://www.ecologique-solidaire.gouv.fr/sites/default/files/Guide_aeromodelisme_modeles_reduits_drones_de_loisir.pdf` (cf. p. 18).

[civ18c] Direction de la sécurité de l'aviation civile. *Guide DSAC, Aéronefs circulant sans personne à bord : activités particulières*. 2018. URL : `https://www.ecologique-solidaire.gouv.fr/sites/default/files/Guide_drones_activites_particulieres.pdf` (cf. p. 18, 19).

[DM+19] Thibaut Tezenas Du Montcel et al. « BOARR : A Benchmark for quadrotor Obstacle Avoidance based on ROS and RotorS ». In : (2019) (cf. p. 95).

[Dro] *IROS 2018 Autonomous Drone Racing Competition*. URL : `http://rise.skku.edu/iros2018racing/index.php/iros-2018-adr/` (cf. p. 94).

[Dro18] Dronecode. *PX4 Obstacle Avoidance*. 2018. URL : `https://docs.px4.io/en/computer_vision/obstacle_avoidance.html` (cf. p. 22).

[dro19] dronedj.com. *Drones at London Gatwick and Heathrow*. 2019. URL : `https://dronedj.com/2019/01/08/heathrow-airport-runway-closed-drone/` (cf. p. 18).

[eca19] ecalc.ch. *Multicopter Calculator*. 2019. URL : `https://www.ecalc.ch/xcoptercalc.php` (cf. p. 21).

[Far03] Gunnar Farnebäck. « Two-frame motion estimation based on polynomial expansion ». In : *Scandinavian conference on Image analysis*. Springer. 2003, p. 363-370 (cf. p. 72).

[FCT16] Pete Florence, John Carter et Russ Tedrake. « Integrated perception and control at high speed : Evaluating collision avoidance maneuvers without maps ». In : *Workshop on the Algorithmic Foundations of Robotics (WAFR)*. 2016 (cf. p. 47).

[FFS17] Matthias Faessler, Antonio Franchi et Davide Scaramuzza. « Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories ». In : *IEEE Robotics and Automation Letters* 3.2 (2017), p. 620-626 (cf. p. 48).

[FKS19] Davide Falanga, Suseong Kim et Davide Scaramuzza. « How Fast Is Too Fast ? The Role of Perception Latency in High-Speed Sense and Avoid ». In : *IEEE Robotics and Automation Letters* 4.2 (2019), p. 1884-1891 (cf. p. 31, 45).

[FPS14] Christian Forster, Matia Pizzoli et Davide Scaramuzza. « SVO : Fast semi-direct monocular visual odometry ». In : *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, p. 15-22 (cf. p. 25, 96).

[Fra19] Gouvernement Français. *Carte des restrictions de vol pour drone de Loisir*. 2019. url : `http://www.geoportail.gouv.fr/donnees/restrictions-pour-drones-de-loisir/` (cf. p. 18).

[Fur+16] Fadri Furrer et al. « Rotors—A modular gazebo mav simulator framework ». In : *Robot Operating System (ROS)*. Springer, 2016, p. 595-625 (cf. p. 75).

[Gaz] Gazebo. url : `http://gazebosim.org/` (cf. p. 76).

[GK06] Colin J Green et Alonzo Kelly. « Optimal sampling in the space of paths : Preliminary results ». In : (2006) (cf. p. 47).

[GKM10] Chad Goerzen, Zhaodan Kong et Bernard Mettler. « A survey of motion planning algorithms from the perspective of autonomous UAV guidance ». In : *Journal of Intelligent and Robotic Systems* 57.1-4 (2010), p. 65 (cf. p. 27).

[GLU12] Andreas Geiger, Philip Lenz et Raquel Urtasun. « Are we ready for autonomous driving ? the KITTI vision benchmark suite ». In : *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012, p. 3354-3361 (cf. p. 94).

[GS17] Fei Gao et Shaojie Shen. « Quadrotor trajectory generation in dynamic environments using semi-definite relaxation on nonconvex QCQP ». In : *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, p. 6354-6361 (cf. p. 32).

[Gus+02] Fredrik Gustafsson et al. « Particle filters for positioning, navigation, and tracking ». In : *IEEE Transactions on signal processing* 50.2 (2002), p. 425-437 (cf. p. 67).

[HC19] Yi Hsuan Hsiao et Pakpong Chirarattananon. « Ceiling Effects for Hybrid Aerial-Surface Locomotion of Small Rotorcraft ». In : *arXiv preprint arXiv :1905.04632* (2019) (cf. p. 16, 26).

[HD11] Markus Hehn et Raffaello D'Andrea. « Quadrocopter trajectory generation and control ». In : *IFAC proceedings Volumes* 44.1 (2011), p. 1485-1491 (cf. p. 48).

[HNR68] Peter E Hart, Nils J Nilsson et Bertram Raphael. « A formal basis for the heuristic determination of minimum cost paths ». In : *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), p. 100-107 (cf. p. 28).

[Hor+13] Armin Hornung et al. « OctoMap : An efficient probabilistic 3D mapping framework based on octrees ». In : *Autonomous robots* 34.3 (2013), p. 189-206 (cf. p. 36, 38).

[Hua+09] Haomiao Huang et al. « Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering ». In : *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, p. 3277-3282 (cf. p. 14).

[HWT08] Gabriel Hoffmann, Steven Waslander et Claire Tomlin. « Quadrotor helicopter trajectory tracking control ». In : *AIAA guidance, navigation and control conference and exhibit*. 2008, p. 7410 (cf. p. 48).

[jar] jarikomppa. url : `https://github.com/jarikomppa/proctree/` (cf. p. 81).

[JK16] James Jackson et Daniel Koch. *ROSflight*. 2016. url : `http://rosflight.org/` (visité le 20/02/2018) (cf. p. 21, 91).

[Kam+17] Mina Kamel et al. « Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System ». In : *Robot Operating System (ROS) The Complete Reference, Volume 2*. Sous la dir. d'Anis Koubaa. Springer, 2017 (cf. p. 56).

[Kau+18] Elia Kaufmann et al. « Deep drone racing : Learning agile flight in dynamic environments ». In : *arXiv preprint arXiv :1806.08548* (2018) (cf. p. 30).

[KBS16] M. Kamel, M. Burri et R. Siegwart. « Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles ». In : *ArXiv e-prints* (nov. 2016). arXiv : `1611.09240 [cs.RO]` (cf. p. 56).

[KCK13] Suseong Kim, Seungwon Choi et H Jin Kim. « Aerial manipulation using a quadrotor with a two dof robotic arm ». In : *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, p. 4990-4995 (cf. p. 26).

[Kes+17] Leonid Keselman et al. « Intel realsense stereoscopic depth cameras ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, p. 1-10 (cf. p. 83, 84).

[KM07] Georg Klein et David Murray. « Parallel tracking and mapping for small AR workspaces ». In : *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society. 2007, p. 1-10 (cf. p. 96).

[KSC13] C. Kerl, J. Sturm et D. Cremers. « Dense visual SLAM for RGB-D cameras ». In : *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013, p. 2100-2106 (cf. p. 93).

[LaV98] Steven M LaValle. « Rapidly-exploring random trees : A new tool for path planning ». In : (1998) (cf. p. 28).

[Lei+17] Jürgen Leitner et al. « The ACRV picking benchmark : A robotic shelf picking benchmark to foster reproducible research ». In : *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, p. 4705-4712 (cf. p. 94).

[Lem16] Wim Lemkens. *Dynamic obstacle avoidance*. 2016. URL : `https://www.youtube.com/watch?v=7KPSFFFyjVE` (cf. p. 32).

[Liu+16] Sikang Liu et al. « High speed navigation for quadrotors with limited onboard sensing ». In : *IEEE International Conference on Robotics and Automation (ICRA)*. 2016, p. 1484-1491 (cf. p. 63, 76).

[LLM10] Taeyoung Lee, Melvin Leok et N Harris McClamroch. « Geometric tracking control of a quadrotor UAV on SE (3) ». In : *49th IEEE conference on decision and control (CDC)*. IEEE. 2010, p. 5420-5425 (cf. p. 55, 77, 104).

[LP90] Tomas Lozano-Perez. « Spatial planning : A configuration space approach ». In : *Autonomous robot vehicles*. Springer, 1990, p. 259-271 (cf. p. 41).

[LWZ15] Hao Liu, Xiafu Wang et Yisheng Zhong. « Quaternion-based robust attitude control for uncertain robotic quadrotors ». In : *IEEE Transactions on Industrial Informatics* 11.2 (2015), p. 406-415 (cf. p. 24).

[Man15] Augustin Manecy. « Stratégies de guidage visuel bio-inspirées : application à la stabilisation d'un micro-drone et à la poursuite de cibles. » Thèse de doct. EEATS, 2015 (cf. p. 16).

[MB04] Abdellah Mokhtari et Abdelaziz Benallegue. « Dynamic feedback controller of Euler angles and wind parameters estimation for a quadrotor unmanned aerial vehicle ». In : *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. T. 3. IEEE. 2004, p. 2359-2366 (cf. p. 16).

[Mej+10] Luis Mejias et al. « Vision-based detection and tracking of aerial targets for UAV collision avoidance ». In : *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, p. 87-92 (cf. p. 31).

[MHD15] M. W. Mueller, M. Hehn et R. D'Andrea. « A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation ». In : *IEEE Transactions on Robotics* 31.6 (2015), p. 1294-1310 (cf. p. 49, 104).

[MHP08] Robert Mahony, Tarek Hamel et Jean-Michel Pflimlin. « Nonlinear complementary filters on the special orthogonal group ». In : *IEEE Transactions on automatic control* 53.5 (2008), p. 1203-1217 (cf. p. 24).

[MK11] D. Mellinger et V. Kumar. « Minimum snap trajectory generation and control for quadrotors ». In : *IEEE International Conference on Robotics and Automation (ICRA)*. 2011, p. 2520-2525 (cf. p. 48).

[MRB08] Olivier Michel, Fabien Rohrer et Yvan Bourquin. « Rat's life : A cognitive robotics benchmark ». In : *European Robotics Symposium 2008*. Springer. 2008, p. 223-232 (cf. p. 94).

[MT17] Anirudha Majumdar et Russ Tedrake. « Funnel libraries for real-time robust feedback motion planning ». In : *The International Journal of Robotics Research* 36.8 (2017), p. 947-982. eprint : `https://doi.org/10.1177/0278364917712421` (cf. p. 47).

[Näg+17] Tobias Nägeli et al. « Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization ». In : *IEEE Robotics and Automation Letters* 2.3 (2017), p. 1696-1703 (cf. p. 76).

[Neu+16] M. Neunert et al. « Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking ». In : *IEEE International Conference on Robotics and Automation (ICRA)*. 2016, p. 1398-1404 (cf. p. 48).

[OHP15] H. Oleynikova, D. Honegger et M. Pollefeys. « Reactive avoidance using embedded stereo vision for MAV flight ». In : *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, p. 50-56 (cf. p. 28, 45, 55, 73).

[Ole+17] Helen Oleynikova et al. « Voxblox : Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning ». In : *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 (cf. p. 36).

[Ott+09] Michael W Otte et al. « Path planning in image space for autonomous robot navigation in unstructured environments ». In : *Journal of Field Robotics* 26.2 (2009), p. 212-240 (cf. p. 37).

[PC16] Fabio Poiesi et Andrea Cavallaro. « Detection of fast incoming objects with a moving camera. » In : *BMVC*. 2016 (cf. p. 31).

[pen16] e penser. *Porte et moment cinétique*. 2016. url : `https://www.youtube.com/watch?v=vNOuNlJNzhI/` (cf. p. 5).

[PGC18] Bryan Penin, Paolo Robuffo Giordano et François Chaumette. « Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions ». In : *IEEE Robotics and Automation Letters* 3.4 (2018), p. 3725-3732 (cf. p. 32).

[Qui+09] Morgan Quigley et al. « ROS : an open-source Robot Operating System ». In : *ICRA workshop on open source software*. T. 3. 3.2. Kobe, Japan. 2009, p. 5 (cf. p. 10, 21).

[Qui+18] Jean Quilbeuf et al. « Statistical Model Checking Applied on Perception and Decision-making Systems for Autonomous Driving ». In : *10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles at the IEEE International Conference on Intelligent Robots and Systems, october 2018*. 2018 (cf. p. 95).

[Rat+09] Nathan Ratliff et al. « CHOMP : Gradient optimization techniques for efficient motion planning ». In : (2009) (cf. p. 36).

[RBR16] Charles Richter, Adam Bry et Nicholas Roy. « Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments ». In : *Robotics Research*. Springer, 2016, p. 649-666 (cf. p. 28, 32, 48, 76).

[RC13] Christian Robert et George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013 (cf. p. 66).

[San+19] Nitin J Sanket et al. « EVDodge : Embodied AI For High-Speed Dodging On A Quadrotor Using Event Cameras ». In : *arXiv preprint arXiv :1906.02919* (2019) (cf. p. 31, 45).

[Sch+08] Sebastian Scherer et al. « Flying Fast and Low Among Obstacles : Methodology and Experiments ». In : *International Journal of Robotics Research* 27.5 (2008), p. 549-574 (cf. p. 28, 76).

[Sha+18] Shital Shah et al. « Airsim : High-fidelity visual and physical simulation for autonomous vehicles ». In : *Field and service robotics*. Springer. 2018, p. 621-635 (cf. p. 75).

[Sin+17] S. Singh et al. « Robust online motion planning via contraction theory and convex optimization ». In : *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, p. 5883-5890 (cf. p. 47).

[SL16] Fereshteh Sadeghi et Sergey Levine. « Cad2rl : Real single-image flight without a single real image ». In : *arXiv preprint arXiv :1611.04201* (2016) (cf. p. 45, 55).

[Smo+17] Nikolai Smolyanskiy et al. « Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness ». In : *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, p. 4241-4247 (cf. p. 45, 55).

[Son+17] In Hyuk Son et al. « Graphene balls for lithium rechargeable batteries with fast charging and high volumetric energy densities ». In : *Nature communications* 8.1 (2017), p. 1561 (cf. p. 20).

[Spr+16] Christoph Sprunk et al. « An experimental protocol for benchmarking robotic indoor navigation ». In : *Experimental Robotics*. Springer. 2016, p. 487-504 (cf. p. 94).

[SSC11] F. Steinbrücker, J. Sturm et D. Cremers. « Real-time visual odometry from dense RGB-D images ». In : *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, p. 719-722 (cf. p. 93).

[SSP13] Nitin Sydney, Brendan Smyth et Derek A Paley. « Dynamic control of autonomous quadrotor flight in an estimated wind field ». In : *52nd IEEE Conference on Decision and Control*. IEEE. 2013, p. 3609-3616 (cf. p. 16).

[Stu+12] Jürgen Sturm et al. « A benchmark for the evaluation of RGB-D SLAM systems ». In : *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012, p. 573-580 (cf. p. 94).

[Sun+18] Sihao Sun et al. « High-Speed Flight of Quadrotor Despite Loss of Single Rotor ». In : *IEEE Robotics and Automation Letters* 3.4 (2018), p. 3201-3207 (cf. p. 17).

[TK18] Sarah Tang et Vijay Kumar. « Autonomous flight ». In : *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), p. 29-52 (cf. p. 30).

[Usg] *USGS : The National Map*. URL : `https://viewer.nationalmap.gov/basic/` (cf. p. 81).

[Val+14] Roberto G Valenti et al. « Autonomous quadrotor flight using onboard RGB-D visual odometry ». In : *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, p. 5233-5238 (cf. p. 36).

[VZT12] Michael P Vitus, Wei Zhang et Claire J Tomlin. « A hierarchical method for stochastic motion planning in uncertain environments ». In : *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, p. 2263-2268 (cf. p. 48).

[WK15] Michael Watterson et Vijay Kumar. « Safe receding horizon control for aggressive mav flight with limited range sensing ». In : *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, p. 3235-3240 (cf. p. 63).

[WW09] Steven Waslander et Carlos Wang. « Wind disturbance estimation and rejection for quadrotor position control ». In : *AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference*. 2009, p. 1983 (cf. p. 16).

[Yan+17] Shichao Yang et al. « Obstacle avoidance through deep networks based intermediate perception ». In : *arXiv preprint arXiv :1704.08759* (2017) (cf. p. 76).

[ZF05] J-C Zufferey et Dario Floreano. « Toward 30-gram autonomous indoor aircraft : Vision-based obstacle avoidance and altitude control ». In : *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE. 2005, p. 2594-2599 (cf. p. 28).

[Zha+18] Ji Zhang et al. « P-CAP : Pre-computed Alternative Paths to Enable Aggressive Aerial Maneuvers in Cluttered Environments ». In : *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, p. 8456-8463 (cf. p. 28).

Évitement d'obstacles pour quadrirotors en utilisant une caméra de profondeur

Résumé — La sécurité est l'une des préoccupations majeures en robotique et ce autant dans le but de protéger le robot que dans le but de protéger son environnement. Parmi les fonctionnalités relevant de la sécurité, l'évitement d'obstacles est prisée car elle permet de faire évoluer un robot de manière autonome dans un environnement non contrôlé. Cette thèse s'articule autour de cette fonctionnalité d'évitement d'obstacles pour un type de robot spécifique, le quadrirotor. Deux algorithmes d'évitement d'obstacles seront ainsi présentés. Le premier sera quasiment exclusivement l'agrégat d'éléments présentés indépendamment les uns des autres dans la littérature récente dédiée aux quadrirotors. Le deuxième algorithme présentera une nouveauté, un filtrage spatial sur les coûts associés aux différentes trajectoires générées, qui permet d'améliorer les performances d'évitement. Ces deux algorithmes utiliseront un unique capteur de profondeur et se baseront, notamment, sur la platitude des quadrirotors et sur l'utilisation d'une carte de profondeur locale donnant une information sur 360°. L'autre contribution de cette thèse réside dans la création du benchmark BOARR qui a été développé pour effectuer des tests d'évitement d'obstacles en simulation. Ce benchmark a été construit pour tenter de simuler au mieux un des problèmes présent à de multiples reprises dans la littérature : la navigation dans une forêt dense avec un quadrirotor d'environ 500 g. Ce benchmark est disponible en open-source et l'auteur de cette thèse espère, grâce à cet outil, augmenter la qualité des tests effectués sur la capacité d'évitement d'obstacles qui est, on le rappelle, une fonctionnalité de sécurité et qui nécessite donc des tests rigoureux.

Mots clés : Robotique aérienne, Drones, Quadrirotor, Évitement d'obstacles, Benchmark en simulation, Évaluation quantitative

GIPSA-Lab, 11 rue des Mathématiques
38400 Saint-Martin d'Hères, France

Obstacle avoidance for quadrotors using a RGBD camera

Abstract — Safety is a major concern in robotics to protect both the robot and its environment. Among the fonctionalities related to safety, obstacle avoidance is looked for since it allows autonomous navigation in unknown environments. This thesis is built around this obstacle avoidance functionnality for a specific type of robots, quadrotors. Two obstacle avoidance algorithms are presented. The first one is the aggregation of recent researchs dedicated to quadrotors but presented in various contexts. The second algorithm includes a novel feature, a spatial filter on costs associated to generated trajectories, which improves the overall performances in avoiding obstacles. Both algorithms use a single RGBD sensor et are based, among other things, on differential flatness and on a representation of the environment that we call an egocubemap. During the thesis, the quality of the tests there were used to cheracterize the algorithms was one of the major concern. It led to the creation of the BOARR benchmark. This benchmark has been developped as an attempt to simulate the most common test environment for quadrotor obstacle avoidance : a forest. This benchmark is available online and is open source. It aims at increasing the quality of the tests used when characterizing the performances of obstacle avoidance algorithms which are related to safety and therefore deserve extended testings.

Keywords : Aerial robotics, UAV, Quadrotors, Obstacle avoidance, Simulated benchmark, Quantitative evaluation

GIPSA-Lab, 11 rue des Mathématiques
38400 Saint-Martin d'Hères, France