REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
École Nationale Polytechnique

Departement d'Automatique
Laboratoire de Commande des Processus
Mémoire de projet de fin d'études
pour l'obtention du diplôme d'ingénieur d'état en Automatique

Perception et Navigation visuelle d'un quadrirotor en utilisant des techniques d'apprentissage profond

Réalisé par:

Merouane GUETTACHE
Yacine BEN AMEUR

Présenté et Soutenu publiquement le 01 Juillet 2019 devant le jury composé de :

Président	**M.S. BOUCHERIT**	Professeur	ENP Alger
Encadreur	**Z. MOUSSAOUI**	Ingénieur chercheur	Airbus Groupe-ECE
Co-encadreur	**M. TADJINE**	Professeur	ENP Alger
Examinateur	**M. CHAKIR**	Enseignant chercheur	ENP Alger

ENP 2019

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
École Nationale Polytechnique

Departement d'Automatique
Laboratoire de Commande des Processus
Mémoire de projet de fin d'études
pour l'obtention du diplôme d'ingénieur d'état en Automatique

Perception et Navigation visuelle d'un quadrirotor en utilisant des techniques d'apprentissage profond

Réalisé par:

Merouane GUETTACHE
Yacine BEN AMEUR

Présenté et Soutenu publiquement le 01 Juillet 2019 devant le jury composé de :

Président	**M.S. BOUCHERIT**	Professeur	ENP Alger
Encadreur	**Z. MOUSSAOUI**	Ingénieur chercheur	Airbus Groupe-ECE
Co-encadreur	**M. TADJINE**	Professeur	ENP Alger
Examinateur	**M. CHAKIR**	Enseignant chercheur	ENP Alger

ENP 2019

Dédicace

A celle et celui qui ont toujours garni mon chemin avec force et lumière, mes très chers parents.

A toute ma famille pour leur amour et le respect qu'ils m'ont toujours accordé,
A ma deuxième famille: Yacine, Nabil, Daoud, Nounou, Ishak, Houssem,
A toute personne qui m'a aidé à franchir un horizon dans ma vie,

je dédie ce travail.

Merouane GUETTACHE

Dédicace

À mes très chers parents, pour tous leurs sacrifices, leur dévouement, et leur soutien tout au long de mes études. Que Dieu les protège.

À Nawel, Abdenour, Walid, Luna,
à Sandra, Merouane, Nabil, Daoud, Nounou, Ishak, Houssem,
et à tous ceux que j'aime,

Je dédie ce mémoire

Yacine BEN AMEUR

Remerciements

Nous remercions ***Dieu***, le tout puissant de nous avoir donnés le courage, la patience et la force durant toutes ces années d'étude.

Les travaux présentés dans ce mémoire ont été effectués au sein du Laboratoire de Commande des Processus (LCP) de l'École Nationale Polytechnique.

Ce travail que nous présentons a été effectué sous la direction de Monsieur *M. Tadjine*, professeur à l'Ecole Nationale Polytechnique, et Monsieur *Z. Moussaoui*, ingénieur chercheur chez Airbus, qui ont suivi de très près ce travail. Nous les remercions également pour leur orientation pédagogique dans l'élaboration de ce mémoire.

Nous tenons à remercier Monsieur *M.S. Boucherit*, professeur à l'Ecole Nationale Polytechnique, pour l'honneur qu'il nous fait de présider le jury de notre soutenance.

Que Monsieur *M. Chakir*, enseignant chercheur à l'Ecole Nationale Polytechnique, soit convaincu de notre sincère reconnaissance pour avoir accepté d'examiner et de critiquer ce mémoire.

Enfin, nous remercions vivement nos amis, camarades et co-équipiers lors de la compétition *AlphaPilot*, *Nabil Tchoulak* et *Houssem Meghnoudj*, pour leur aide et leur soutien. Qu'ils trouvent ici l'expression de notre profonde gratitude.

ملخص

تقع سباقات الطائرات بدون طيار ذاتية الحكم في قلب تحديات الروبوتات الهوائية ، وتفيد شعبيتها المتزايدة الأبحاث في مجال الإبحار المستقل المعتمد على الؤية. في هاذا المشروع ، نقدم حلنا الذي يتألف من استراتيجية تنقل مختلطة: نقاط التتبع والتحكم المعتمد على الصورة. يتم إستخدام تحكم هندسي لتحريك الكوادكوبتر إلى نقاط طريق محددة مسبقاًا ، بينما يوفر المرشح التكميلي وORB-SLAM2 تقديراً للحالة من إشارات IMU والرؤية المزدوجة، على التوالي. يتم تدريب شبكة عصبية تلافيفية للكشف عن النقاط الأربع للبوابة المراد عبورها ، ثم يتم استخدامIBVS لتمرير الكوادكوبترعبر هذه البوابة. تم تطبيق هذا النهج على جهاز محاكاة الطيران FlightGoggles . استطاع الكوادكوبتر المرور بأمان بـ ١١ بوابة (٠٤٢ م) في ٤٦ ثانية والوصول إلى سرعة قصوى تبلغ ٦ م \ ث.

كلمات مفتاحية سباق الطائرات بدون طيار المستقل ، الملاحة البصرية ، هههه، التحكم البصري ، الروبوتات الجوية

Abstract

Autonomous drone racing crystallizes some of the core challenges in aerial robotics, and its growing is benefiting the autonomous vision based navigation research community.
In this thesis, we present our solutions which consists of a hybrid navigation strategy: waypoints following and image-based visual servoing. Geometric Tracking Control is deployed to move the quadcopter to pre defined waypoints while a complementary filter and ORB-SLAM2 are providing state estimate from IMU signals and stereo camera respectively. A Convolutional Neural Network is trained to detect the 4 corners of the next gate to pass, IBVS is then used to navigate the quadcopter through this gate.
This approach was implemented on the photo-realistic simulator FlightGoggles. The quadcopter was able to navigate safely through 11 gates (240 m) in 50 s reaching a peak speed of 6.5 m/s.

Keywords autonomous drone race, visual navigation, CNN, visual servoing, aerial robotics

---

Résumé

Les courses de drones autonomes sont au cœur des défis de la robotique aérienne, et leur popularité croissante profite à la communauté de recherche sur la navigation autonome basée sur la vision.
Dans cette thèse, nous présentons notre solution qui consiste en une stratégie de navigation hybride : suivi des points de cheminements et asservissement visuel basé sur l'image. Une commande géométrique est déployée pour déplacer le quadrirotor vers des points de cheminement prédéfinis tandis qu'un filtre complémentaire et ORB-SLAM2 fournissent une estimation de l'état à partir des signaux IMU et une caméra stéréoscopique respectivement . Un réseau neuronal convolutionnel est entraîné pour détecter les 4 coins de la porte à franchir, l'IBVS est ensuite utilisé pour faire passer le quadcoptère par cette porte.
Cette approche a été implémentée sur le simulateur photoréaliste FlightGoggles. Le quadcoptère a pu franchir en toute sécurité 11 portes (240 m) en 50 s et atteindre une vitesse de pointe de 6.5 m/s.

Mots clés course de drones autonome, navigation visuelle, CNN, asservissement visuel, robotique aérienne

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Ces dernières années, les robots volants tels que les hélicoptères miniatures ou les quadrirotors ont connu une notoriété grandissante. Les applications potentielles vont de l'exploration et l'inspection aérienne à la livraison en passant par la reconstruction 3D d'environnements difficiles d'accès. Néanmoins, la conduite manuelle d'un quadrirotor nécessite un pilote qualifié et une supervision humaine constante.

Il existe donc un fort intérêt scientifique à développer des solutions permettant aux quadrirotors de voler de façon autonome : C'est ce qu'on appelle la navigation et c'est un problème fondamental de la robotique mobile et de l'intelligence artificielle. Il s'agit d'un problème de recherche difficile car la charge utile d'un quadrotor est très limitée et donc la qualité des capteurs embarqués et la puissance de calcul disponible sont fortement limitées.

Les méthodes de navigation pour la robotique mobile se focalisent,entre autres, sur les scanners laser, l'ultrason, et les caméras pour détecter les obstacles et suivre une trajectoire. Des progrès impressionnants ont récemment été faits en utilisant les LIDAR ou les Microsoft Kinect (RGB-D), les deux capteurs sont lourds et actifs et augmentent donc la consommation d'énergie et réduisent l'autonomie. Ceci contraste avec l'être humain qui utilise seulement la vision pour se mouvoir et conduire des voitures et même des drones. La vision passif est donc prometteuse pour produire une solution faisable, à moindre coût et plus adéquate pour la robotique mobile autonome [1].

De plus, l'utilisation de la vision permet d'élargir le champ d'action du quadrirotor en collectant des données plus explicites sur l'environnement. En effet, à l'inverse du GPS, IMU et capteurs de distance (sonar,laser,infrarouge), les caméras produisent la plus grande bande passante de données et de caractéristiques utilisables lors de la modélisation 3D et la navigation (couleur, texture, flux optique, détection d'objets ...). Exploiter ces données est moins explicite que dans le cas d'un GPS ou d'un scanner laser mais bien plus éloquent[2].

Parallèlement, les MAVs, et les robots de manière générale, sont appelés à l'avenir à être déployés dans des environnements dynamiques et non structurés, fournissant une assistance dans de nombreux domaines allant des applications militaires aux secours en situation dangereuse. Il sera difficile de pré-programmer toutes les tâches et tous les scénarios possibles dans de tels robots. Les robots devront être capables d'apprendre par eux-mêmes ou avec la supervision (plus ou moins explicite) de l'homme.

L'apprentissage automatique sera nécessaire pour atteindre ce haut degré d'autonomie et permettra aux robots de percevoir et d'agir adéquatement aux variations de son environnement. De toutes les techniques d'apprentissage automatique, l'apprentissage profond

(*Deep Learning*) a récemment retenu l'intérêt des chercheurs, en particulier dans le domaine de la vision par ordinateur.

Ces deux facteurs sont à l'origine de la naissance d'un nouveau domaine de recherche unique en son genre qu'est la navigation visuelle pour les courses de drones autonomes. Ces algorithmes offrent au drone un cadre et un ensemble d'outils pour l'élaboration de comportements sophistiqués et difficiles à concevoir. Inversement, les défis posés par les courses de drones autonomes fournissent à la fois inspiration, impact et validation pour les développements en matière de navigation visuelle et d'apprentissage automatique.

Objectifs du mémoire

Les travaux présentés dans ce mémoire se situent dans le cadre des objectifs suivants : étude, développement et validation par simulation d'une architecture de commande, estimation d'état et navigation visuelle applicable sur la course de drones autonomes. Le cahier des charges est spécifiée par La compétition internationale de courses de drones autonomes appelées *AlphaPilot* à laquelle nous avons pris part (Section 1.3).

Organisation du mémoire

Ce mémoire est organisé de la manière suivante :

Le premier chapitre servira à décrire de manière concise la problématique des courses de drones autonomes et lister les différents contraintes qu'il faut respecter et les défis à relever. Nous parlerons également de notre participation à la compétition internationale *AlphaPilot* (Section 1.3).

Dans le second chapitre nous exposerons l'état de l'art en matière de courses de drones autonomes. Cet état de l'art nous le subdiviserons en trois sous-problématiques : La commande des quadrirotors, l'estimation de l'état en utilisant la vision et la navigation visuelle autonome. C'est d'ailleurs cette structure que nous garderons pour la suite du mémoire. Cet état de l'art nous permettra d'identifier un certain nombre d'approches candidates à l'application sur les drones autonomes.

Le troisième chapitre sera consacré à la modélisation et la commande du drone quadrirotor. Un modèle dynamique basée sur les equations de Newton-Euler sera établi. Nous élaborerons ensuite un asservissement non linéaire avec deux boucles de commandes en cascade : une pour asservir la position et l'autre l'orientation du quadrirotor. Plusieurs simulations en boucle fermée montreront la stabilité et la robustesse de cette commande.

Dans le quatrième chapitre, nous décrirons la méthode que nous avons retenue pour localiser et estimer l'orientation du quadrirotor à savoir l'utilisation d'un filtre complémentaire sur les signaux bruités de la centralle inertielle, combiné avec un algorithme de Stereo Visual SLAM appelé ORB-SLAM2.

Le cinquième chapitre concernera la navigation réactive basée sur l'image en appliquant la méthode du Visual Servoing. Pour ce faire il est nécessaire en premier lieu de détecter les 4 coins du *gate* dans l'image. Cette tâche a été accomplie à l'aide d'un réseau de neurones

convolutionnel profond.

Finalement, nous présenterons dans le sixième chapitre l'architecture globale du système en recombinant la commande du quadrirotor, l'estimation de l'état et la navigation visuelle autonome en un seul bloc fonctionnel testé sur le simulateur photoréaliste FlightGoggles. Les avantages mais aussi les limites de l'approche ainsi appliquée y seront exposés.

Chapitre 1

Problématique

1.1 Introduction

La **course de drones à vue subjective** (*first person view drone racing*) est un e-sport de nouvelle génération où des drones survolent une série d'obstacles pilotés par des professionnels. Les pilotes de drones utilisent des lunettes de réalité virtuelle reliées à une caméra montée sur le véhicule et qui diffuse un flux vidéo en temps réel.
Dans une course de drone typique, la mission est de faire un circuit spécifique: passer par des portes (des contours circulaires ou rectangulaires que le drone doit traverser appelés *gates*), tout en évitant les collisions et en respectant un ordre prédéfini.

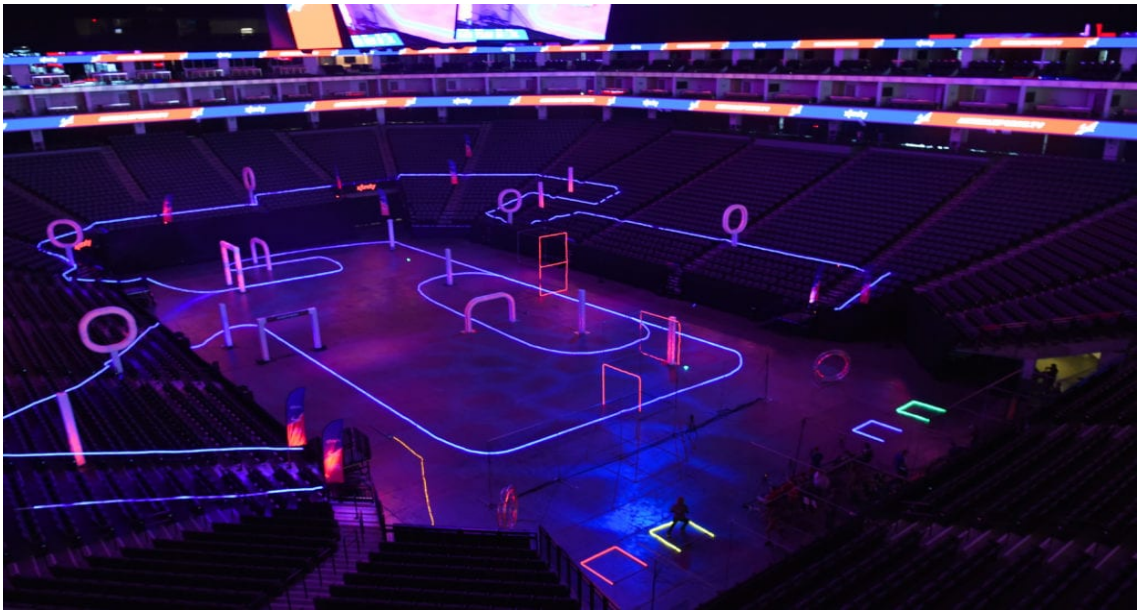


Figure 1.1: Circuit typique d'une course de drone.

L'engouement pour ce *e-sport* est à l'origine des **courses de drones autonomes**. Ces dernières sont des évènements scientifiques dont le but principal est de promouvoir la recherche pas seulement dans le domaine spécifique des UAV mais surtout dans le domaine de la navigation visuelle complètement autonome.

À cet effet, nous avons participé à une compétition internationale de courses de drones autonomes nommée **AlphaPilot** [3]. Accompagnés de notre encadreur, M. Zeryab Moussaoui, de nos camarades Nabil Tchoulak et Houssem Meghnoudj, nous avons réussis à nous hisser à la $23^{ème}$ place du classement des tests de qualification parmi 420 équipes participantes.

Dans ce chapitre nous décrirons la problématique des courses de drones autonomes et listerons les contraintes qu'il faut respecter. Nous parlerons aussi de notre participation à la compétition **AlphaPilot**[3].

1.2 Compétitions internationales

Il existe à l'échelle internationale, deux compétitions de drones autonomes :

1.2.1 Autonomous Drone Racing Challenge [4]

La première version de l'*Autonomous Drone Racing Challenge* a eu lieu lors de la Conférence internationale de 2016 sur les robots et systèmes intelligents (IROS) à Daejeon, en Corée du Sud[5]. Dans cette épreuve, il faut que les drones franchissent les obstacles dans l'ordre correct pour obtenir des scores. La couleur, la géométrie ,l'emplacement et l'orientation des obstacles sont annoncés avant l'événement, ce qui permet aux équipes participantes d'utiliser cette information pour une meilleure localisation (voir figure 1.2).

Les obstacles réels ont été installés avec une marge d'erreur prescrite de 10 cm. Aucune limite n'a été fixée pour le type de drone, mais sa dimension était limitée à $1 \times 1 \times 1$ m. L'intervention humaine n'était pas autorisée après le début de chaque manche par l'arbitre. Le circuit a été divisé en 10 sections, étiquetées de A à J, comme le montre la figure 1.3.

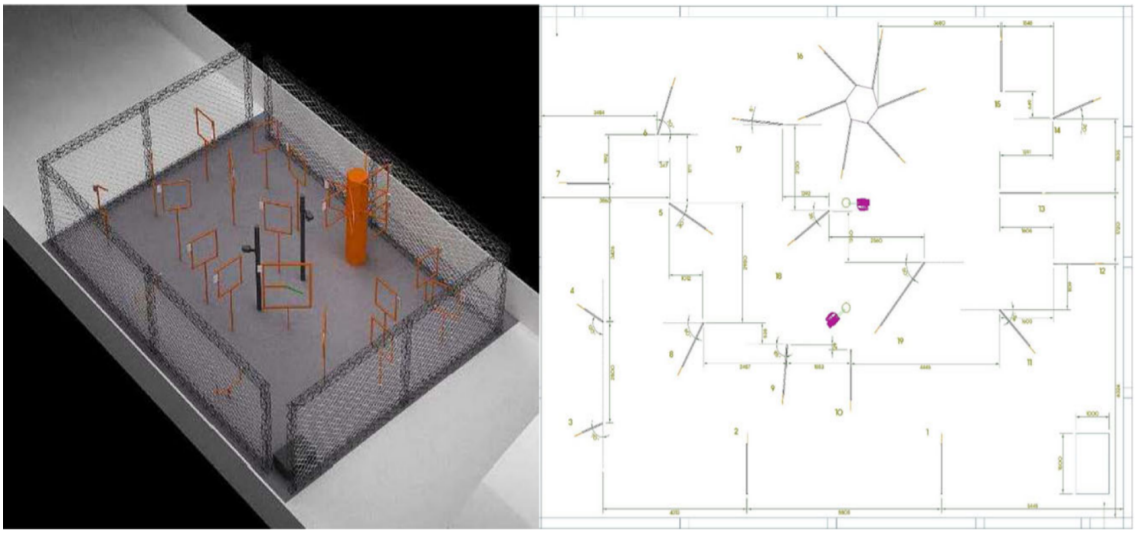


Figure 1.2: Piste de drones autonomes IROS 2016 (à gauche : vue aérienne de l'arène ; à droite : les plans détaillés du cours)

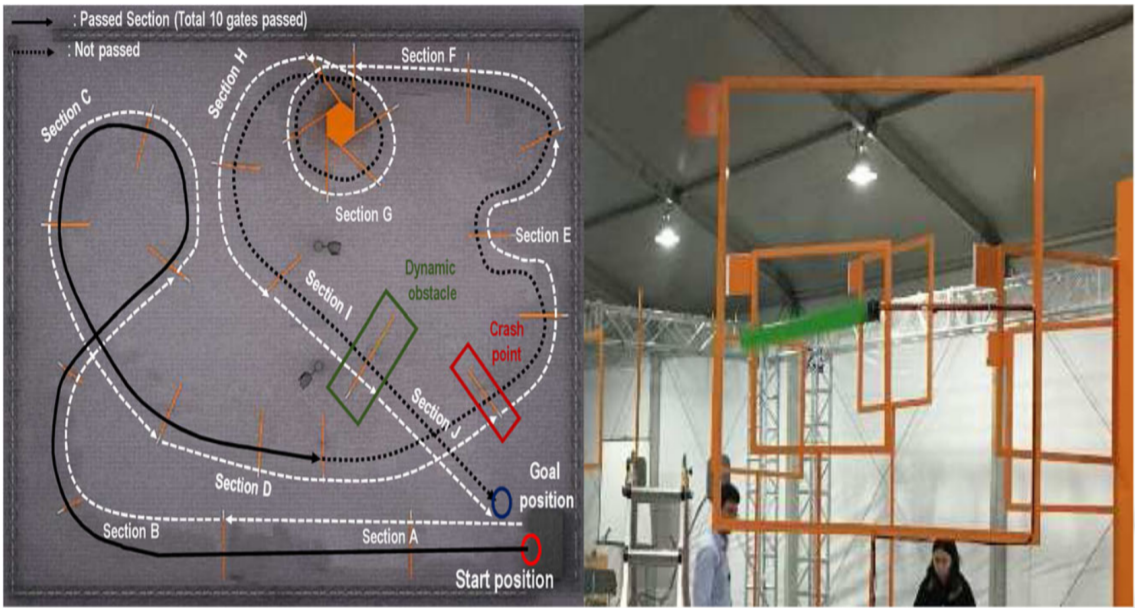


Figure 1.3: Les détails du circuit (à gauche). L'obstacle dynamique (à droite)

1.2.2 AlphaPilot [3]

AlphaPilot est une compétition de drones autonomes organisé par *Lockheed Martin* et la *Drone Racing League* (DRL) avec à la clé un prix de 1.250.000 $!

Des équipes de *drone hobbyists*, d'ingénieurs et de chercheurs de par le monde ont participé à des tests de qualification, dont le troisième est le plus important et porte sur la capacité d'une équipe à concevoir des algorithmes pour le guidage, la navigation et le contrôle (GNC) d'un drone autonome. Ce test est construit sur le simulateur **FlightGoggles** [6] [1] du Massachusetts Institute of Technology (MIT), un environnement de réalité virtuelle basé sur le moteur de jeu *Unity3D*. Ce simulateur fournit aux utilisateurs les outils nécessaires pour tester leurs algorithmes de course de drones en utilisant une dynamique réaliste et des capteurs extéroceptifs.

[1] Site du projet : https://github.com/mit-fast/FlightGoggles

Figure 1.4: Le simulateur FlightGoggles[6] utilisé pour les qualifications de AlphaPilot FlightGoggles

Comme c'est le cas dans les courses de drones réelles, les équipes connaissent l'emplacement approximatif de leur point de départ et l'emplacement des portes et doivent faire face à de légères variations pendant la course. Les équipes sont mises au défi de créer un algorithme unique qui peut être testé sur 25 légères variations du même parcours (voir figure 1.5).

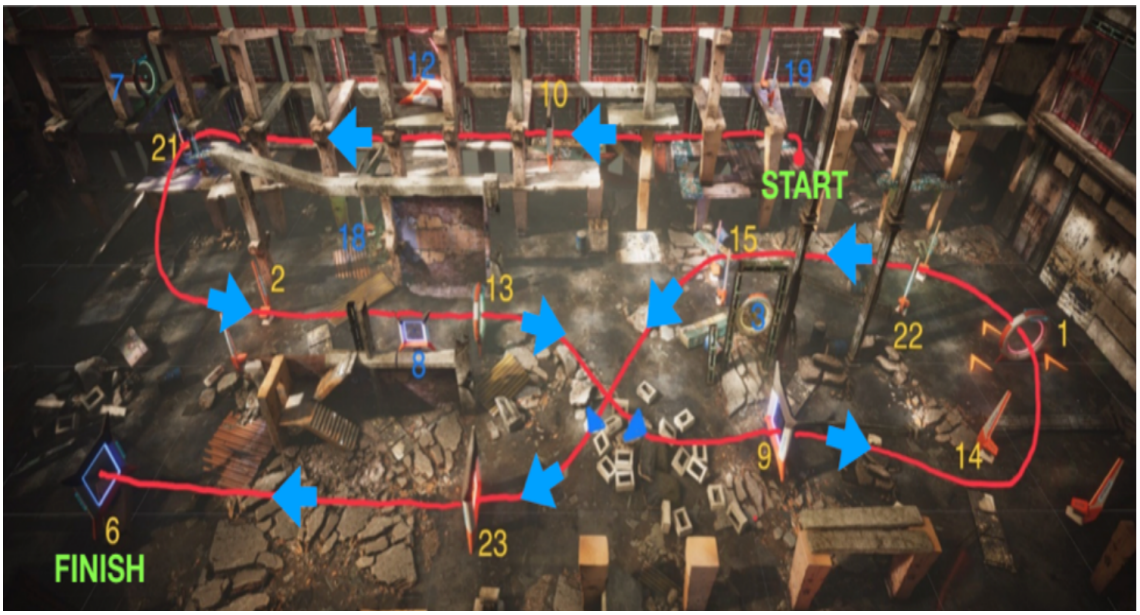


Figure 1.5: Le parcours nominal des qualifications de AlphaPilot sur le simulateur FlightGoggles

1.3 Participation à AlphaPilot

Le tour qualificatif de la compétition AlphaPilot s'est déroulé durant la période allant du 8 février 2019 au 22 mars. 420 équipes y ont participé dont beaucoup de laboratoires de recherche en robotique et intelligence artificielle.
Les membres de l'équipe ainsi que la répartition des tâches est comme suit:

- M. Zeryab Moussaoui: Architecture Système , Management de l'équipe.
- Houssem Meghnoudj: Apprentissage, Rédaction Technique.
- Merouane Guettache: Navigation , Communication de l'équipe.
- Nabil Tchoulak: Intégration Logicielle, Estimation.
- Yacine Ben Ameur: Traitement d'images, Perception.

Ce tour a été séparé en deux problématiques à savoir:

- Conception d'un algorithme de perception capable de détecter les *gates* à partir des images.
- Implémentation d'un algorithme de commande/Navigation sur le simulateur Flight-Goggles.

Nous avons réussi à fournir une solution complète et fonctionnelle dans les délais. Malheureusement, nous n'avons pas pu nous classer dans le top 9 qui nous aurait permis d'accéder à la compétition finale qui se déroule aux États-Unis.

1.4 Contraintes et défis

Les équipes participantes à **AlphaPilot** ont pour mission de:

- Designer un algorithme de vision capable de détecter les 4 coins du prochain *gate* que doit traverser le drone. La précision mais aussi le temps de calcul de l'algorithme sont pris en considération lors de l'évaluation de celui-ci. L'utilisation du *Machine learning* n'est pas obligatoire mais recommandée puisque l'évaluation se fera en utilisant une carte graphique NVIDIA, ces dernières sont connues pour accélerer le calcul des réseaux de neurones profonds.
- Concevoir un algorithme capable de faire franchir au drone 11 *gates* dans un ordre prédéfini en un temps minimal (voir Figure 1.5). Le quadrirotor doit aussi éviter les obstacles sur son chemin. les équipes n'auront aucune connaissance préalable du circuit exacte des courses et devront se servir principalement des données de la caméra pour la navigation.
- Utiliser uniquement les entrées sensorielles suivantes:
  - Flux vidéo photo-réaliste provenant des cameras RGB 60Hz mono et stéréo.
  - Sortie de l'algorithme de détection des *gates* (coordonnées polygonales) dans le repère de la caméra
  - Emplacements des repères IR (*Infra Red*) dans le repère de la caméra.
  - Données bruitées de l'IMU (*Inertial measurement Unit*).
  - Télémètre laser bruité orienté vers le bas pour faciliter l'estimation de l'altitude.

– Fournir la solution de Navigation et de commande sous forme d'un *catkin-workspace*[2] indépendant pour pouvoir être évalué de manière automatique par les examinateurs.

Les courses de drone autonomes soulèvent un nombre important de défis en robotique[7]:

- Le flou de mouvement, des conditions d'éclairage difficiles et *l'aliasing perceptuel* peuvent provoquer de fortes dérives dans tout système d'estimation d'état. De plus, les pipelines d'estimation d'état de pointe peuvent nécessiter des capteurs coûteux [8], avoir des coûts de calcul élevés [9], ou être sujets à la dérive en raison de l'utilisation de cartes compressées [10]. Par conséquent, la performance en temps réel est généralement entravée lorsqu'on utilise des plates-formes à ressources limitées, comme les petits quadrirotors.
- Réagir de manière optimale à des environnements dynamiques changeants [11].
- Coupler la perception et l'action en temps réel sous de fortes contraintes de ressources [12].

1.5 Conclusion

Les courses de drones autonomes peuvent être vues comme un problème de navigation en temps minimal; en respectant les contraintes des portes et d'évitement d'obstacles.

Le drone racing offre un cadre bénéfique pour la recherche dans le domaine de l'aviation autonomes et la robotique en générale grâce à ces différents sous-problématiques: la navigation visuelle, l'estimation de l'état en intérieur et le vol agile.
Dans le prochain chapitre, nous allons présenter l'état de l'art en chacune de ces sous-problématiques pour pouvoir choisir la solution la mieux adaptée aux courses de drones autonomes.

[2]Plus d'informations ici.

Chapitre 2

État de l'art

2.1 Introduction

Les pilotes humains dans les courses de drones sont la preuve que les courses de drones autonomes devraient être possibles avec un minimum de détection: une caméra seulement. Cependant, nous sommes encore loin d'atteindre des performances de niveau humain avec des drones autonomes [13]. La course de drones autonomes offre donc un terrain d'expérimentation naturel pour une navigation autonome basée sur la vision [7].

Le développement de ce sport profite maintenant à la communauté de la recherche [13], avec des projets transversaux tels que la course annuelle de drones autonome de l'IROS [4] [5], ou encore la compétition AlphaPilot[3] qui offre plus d'1 million de dollars à la clé [3].

Plus généralement, la navigation robotique à grande vitesse dans des environnements encombrés et inconnus est actuellement un domaine de recherche très actif [14] [15] [16] [17] [18] [19] [20].

Les drones équipés de capteurs embarqués sont des plateformes idéales pour le vol autonome agile qui soulève des défis fondamentaux en robotique.

Pour ces raisons, repousser les limites de la course de drones est un problème incontournable pour faire avancer le milieu de la recherche. Les algorithmes nécessaires pour obtenir un bon suivi visuel dans le scénario de course de drones seront très utiles à la robotique en général et contribueront à de multiples domaines tels que la navigation autonome dans des environnements complexes et confinés [21], résoudre des tâches telles que l'exploration [22], l'inspection [23][24], la livraison aérienne, la cartographie [25], l'interaction avec l'environnement [26][27] et, la recherche et sauvetage [28].

La problématique de courses de drones autonome, telle que nous la concevons, est subdivisible en trois sous-problématiques : La commande des quadrirotors, l'estimation de l'état en utilisant la vision et la navigation visuelle autonome. Dans ce chapitre, nous explorerons l'état de l'art en chacune de ses sous-problématiques et tenterons d'identifier les solutions qui nous conviennent le mieux.

2.2 Commande des quadrirotors

Le contrôle des quadrirotors est difficile pour plusieurs raisons [29]:

- Les quadrirotors sont sous-actionnés et la conception de la commande doit exploiter les interactions entre les états dynamiques pour contrôler le véhicule.
- Les quadrirotors utilisent des effets aérodynamiques pour produire de la poussée et de la portance, et la régulation de ces forces est par nature approximative, ce qui entraîne des erreurs de modélisation importantes.
- Les effets externes tels que le vent, la turbulence et la génération de tourbillons entraînent des niveaux élevés de perturbation de la charge dans les boucles de régulation.

La plupart des véhicules aériens sont contrôlés à l'aide d'une structure de contrôle hiérarchique avec des boucles de régulation imbriquées basées sur trois niveaux [29]:

- **Planification** : C'est la boucle la plus à l'extérieur dans le contrôle des véhicules aériens et elle est associée à la planification des trajectoires et l'établissement des points de passage.
- **Orientation** : Ce niveau concerne le suivi des trajectoires pour atteindre des objectifs locaux. Cette boucle de régulation est généralement conçue en utilisant la référence d'attitude (attitude = orientation) comme entrée, ce qui conduit à un problème de régulation entièrement actionné.
- **Contrôle** : Il s'agit de la boucle la plus interne et concerne principalement l'attitude et la stabilité en vol du véhicule.

Nous nous intéresserons ici à la commande des quadrirotors et le suivi de trajectoires, ainsi que la difficulté de ces tâches lorsque le véhicule doit effectuer des vols rapides et agiles.

2.2.1 Principaux schémas de commande des quadrirotors

L'architecture de commande la plus courante pour les quadrirotors consiste en deux boucles de commande pour la position et l'attitude [30]. La boucle de régulation de position à haut niveau met généralement en œuvre une loi de régulation de la position et de la vitesse avec des termes de *feed-forward* pour compenser la gravité et les accélérations par rapport à une trajectoire de référence. La sortie de cette boucle de régulation est la poussée collective et l'orientation souhaitées. L'orientation désirée est ensuite contrôlée par une boucle de régulation bas-niveau basée sur l'hypothèse que la dynamique d'attitude d'un quadrirotor est beaucoup plus rapide que sa dynamique de position et de vitesse. Une étude des représentations d'orientation et de leur aptitude à contrôler l'attitude d'un corps rigide est présentée dans [31]. Dans la commande quadrirotor moderne, l'attitude est représentée soit sous forme de quaternion unitaire, soit sous forme de matrice de rotation. Sur la base de ces représentations, il est possible de concevoir des contrôleurs d'attitude presque globalement asymptotiquement stables. Les méthodes de commande du quadrirotor consistant en une boucle de régulation de position PD de haut niveau et une boucle de régulation d'orientation PD de bas niveau basée sur des matrices de rotation sont présentées dans [32] et [33]. Les contrôleurs d'attitude ayant des propriétés similaires basées sur des quaternions unitaires

sont présentés dans [33], [34] et [35].

D'autres techniques peuvent également être appliquées pour la conception du contrôleur d'orientation [36], [37],[38]. [39] a analysé l'application de deux techniques de contrôle différentes - le *sliding mode* et le *Backstepping* - et a montré que ce dernier a des propriétés de stabilisation particulièrement intéressantes. Les régulateurs LQR peuvent également être implémentés en tant que contrôleurs à retour d'état complet pour la commande de position et d'attitude [36]. De plus, les méthodes LQR se sont avérées appropriées pour contrôler les systèmes multi-rotors tels que les quadrirotors portant une charge suspendue [37] ou équilibrer un pendule inversé [38] [39].

Étant donné que pour toutes ces méthodes, la stabilité asymptotique du régulateur de position supérieur et du régulateur d'attitude inférieur peut être montrée, le système interconnecté avec les deux boucles est lui aussi asymptotiquement stable (voir [40] Lemma 5.6).

Souvent, le contrôle d'attitude ne peut pas être réalisé par une seule boucle de contrôle en raison du matériel couramment utilisé. Généralement, deux unités de traitement sont intégrées aux quadrirotors, où un ordinateur de bord fait l'estimation d'état et le contrôle de haut niveau (cette boucle impose les vitesses angulaires désirées) et un micro-contrôleur moins puissant mais en temps réel est utilisé pour le contrôle de bas niveau (commande les vitesses angulaires imposées en utilisant la vitesses des rotors). Pour obtenir un contrôle précis de la vitesse de rotation, des entrées rapides et à faible latence vers les moteurs sont nécessaires, ce qui ne peut être réalisé que par un microcontrôleur en temps réel. Par conséquent, le contrôle d'attitude d'un quadrirotor est souvent obtenu par deux boucles de contrôle en cascade, par exemple [41] et [42], où la première boucle contrôle l'attitude à basse fréquence et la seconde boucle contrôle les vitesses de rotation à haute fréquence.

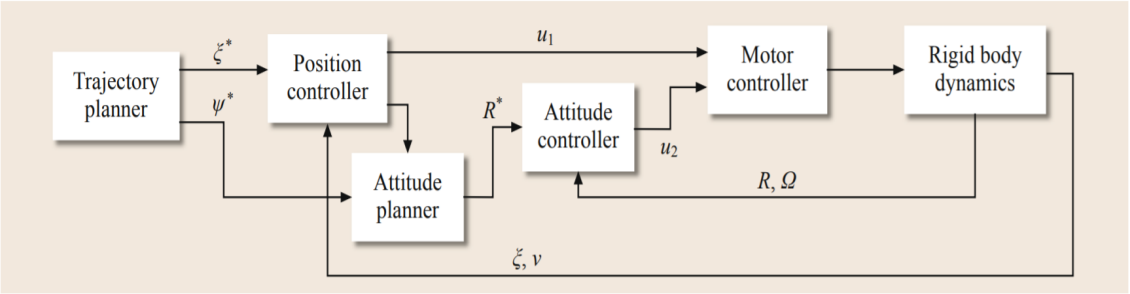


Figure 2.1: Architecture de commande classique des quadrirotors.

2.2.2 Planification de trajectoires et vols agiles

La planification de trajectoire doit générer des trajectoires dynamiquement réalisables qui amèneront le quadrirotor à sa destination [21].

Le quadrirotor est sous-actionné, ce qui rend difficile la planification des trajectoires dans l'espace d'état à 12 dimensions (6 DDL en position et vitesse). Cependant, le problème est considérablement simplifié si l'on utilise un modèle sans tenir compte des effets de traînée du rotor. Dans ce cas, le système est différentiellement plat (*differentially flat*) lorsqu'on choisit sa position et de son cap en sorties plates [32]. En d'autres termes, les états et les entrées peuvent être écrits comme fonctions algébriques des quatre sorties plates et de

leurs dérivés. Ceci facilite la génération automatisée de trajectoires puisque n'importe quelle trajectoire lisse (avec des dérivées raisonnablement limitées) dans l'espace des sorties plates peut être suivie par le quadrirotor. [43] A montré que la dynamique d'un quadrirotor est différentiellement plate même lorsqu'elle est soumise à des effets de traînée du rotor linéaire et a utilisé cette propriété pour calculer des termes de *feed-forward* qui sont ensuite appliqués par un contrôleur de position. Dans [44], la propriété de platitude différentielle d'un hexarotor qui prend la poussée collective désirée et son orientation désirée en entrée a été exploitée. L'orientation désirée était ensuite contrôlée par une boucle de commande séparée de bas-niveau, ce qui permet également d'effectuer des manœuvres de vol à des vitesses de plusieurs mètres par seconde.

La commande à modèle prédictif (*Model Predictive Control* (MPC)) est devenu de plus en plus populaire pour la commande du quadrirotor [45], [46], [47] grâce à sa capacité à gérer simultanément différentes contraintes et objectifs grâce à l'optimisation [12]. Par exemple, pour passer à travers un espace étroit tout en le localisant à l'aide d'une caméra embarquée, il est nécessaire de s'assurer que l'espace est visible à tout moment. Pour tirer pleinement parti de l'agilité des quadrirotors autonomes, il est nécessaire de créer une synergie entre la perception et l'action en les considérant conjointement comme un problème unique [12]. Dans [48], une commande à modèle prédictif (MPC) est proposé, qui, avec un régulateur quadratique linéaire (LQR) de haut niveau, permet de suivre des trajectoires agressives. Les commandes à modèle prédictif ont également été utilisées comme contrôleurs de haut niveau avec une boucle de contrôle d'attitude séparée basée, par exemple [49] [50] [47]. De plus, les méthodes de génération rapide de trajectoires peuvent être exploitées dans un modèle de contrôle prédictif en n'appliquant que la première commande résultant d'une trajectoire planifiée et en reprogrammant ensuite une nouvelle trajectoire pour la prochaine itération de boucle de contrôle [47] [51].

2.3 Estimation de l'état

Définie de façon générale, la détection et la perception comprennent l'obtention de données sur le véhicule et son environnement et l'extraction d'informations utiles à partir de ces données [52]. Dans le contexte de la vision par ordinateur, la perception est le problème d'extraire des éléments pertinents dans un flux d'images.
L'estimation est le processus d'extraction d'informations sur des variables d'intérêt à partir de mesures bruitées et qui peuvent être liées à ces variables par des modèles mathématiques complexes.

Les quadrirotors ont besoin de précision et un faible temps de latence dans leur estimation d'état afin d'obtenir un vol stable et robuste. Cependant, en raison des contraintes de puissance et de charge utile des plates-formes aériennes, les algorithmes d'estimation d'état doivent fournir ces qualités sous les contraintes de calcul du matériel embarqué [53].

Les caméras et les unités de mesure inertielle (IMU) répondent à ces contraintes de puissance et de charge utile, de sorte que les algorithmes d'odométrie visuelle-inertielle (VIO) sont des choix populaires pour l'estimation d'état dans ces scénarios, en plus de leur capacité à fonctionner sans localisation externe par des systèmes de capture de mouvement ou de positionnement global (GPS).

Les unités de mesure inertielle microélectromécaniques (IMU) sont disponibles dans le commerce depuis un certain temps et ont été utilisées pour la détection et la stabilisation dans de nombreuses applications. Leur petite taille et leur faible consommation d'énergie en font des capteurs bien adaptés aux petits drones. Toutefois, deux facteurs empêchent les solutions de navigation purement inertielles : premièrement, les IMU ne fournissent pas d'informations sur les obstacles proches ; deuxièmement, leurs caractéristiques de bruit et de biais conduisent à une dérive rapide sans limites dans la position calculée. Des capteurs supplémentaires sont donc nécessaires.

La caméra donne un flux de données particulièrement riche, adapté à la fois à la perception et à l'estimation de l'état. Les caméras sont devenues très petites et légères, et sont donc adaptées à une utilisation en tant que capteur sur de petits véhicules [52] [53]. Les systèmes de vision peuvent fournir des mesures d'obstacles ou de points de repère dans l'environnement pour permettre d'éviter les obstacles et pour aider à calculer la position, l'orientation et la vitesse du véhicule pour la navigation.

Les caractéristiques de base de l'IMU et de la caméra sont complémentaires et permettent de fournir une estimation précise [54]. En effet, les IMU fournissent les données de mesure à un taux d'échantillonnage élevé, mais avec le temps, les mesures des IMU dérivent et génèrent une erreur accumulée. En contrepartie, la caméra fournit des mesures à faible taux d'échantillonnage en raison du long temps de calcul pour le traitement de l'image capturée et la mesure ne dérive pas avec le temps. Dans l'approche de navigation basée sur la vision, la caméra est montée sur le drone et la caméra se déplace dans son environnement et le mouvement des caractéristiques de l'image peut être utilisé pour estimer la trajectoire du drone.

La plupart des chercheurs suivent les deux techniques suivantes pour cette approche basée sur la vision: Localisation et cartographie simultanées et Odométrie visuelle.

2.3.1 Odométrie visuelle

L'odométrie visuelle (V.O) est une procédure d'estimation incrémentale de la position et de l'orientation du véhicule de navigation en analysant l'écart que le mouvement induit sur la séquence d'images capturées par une ou plusieurs caméras embarquées [55].

La VO calcule la position actuelle du véhicule en suivant les points caractéristiques capturés dans le plan image. Le déplacement est calculé de façon incrémentale à partir des images acquises. Le drone monté d'une caméra vole dans l'environnement et les images sont prises séquentiellement à chaque instant k comme le montre la Figure 2.2 [55]. Les images successives de la transformation des positions du drone (ou Caméra) sont montrées dans l'équation 2.1.

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

Où, $R_{k,k-1}$ représente la matrice de rotation et $t_{k,k-1}$ représente le vecteur de translation. Les poses (position et rotation) de la caméra $C_{0:n} = [C_0 \dots C_n]$ contiennent les transformations de la caméra prises par rapport à l'image à l'instant même $k = 0$.La pose actuelle C_n peut être

estimée en intégrant toutes les transformations de la caméra $T_k, k = 1, ..., n$:

$$C_n = C_{n-1}T_n \tag{2.2}$$

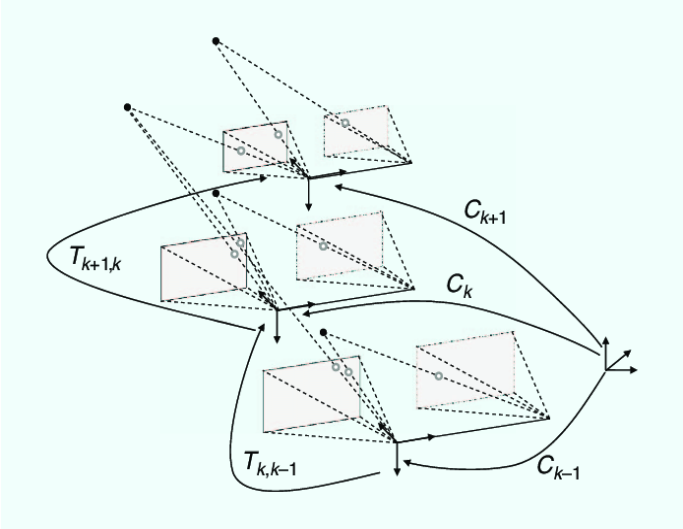


Figure 2.2: Une illustration de l'odométrie visuelle.

L'algorithme rapide d'odométrie visuelle monoculaire semi-directe (Semi-direct Visual Odometry (SVO)) [56] est un exemple d'algorithme VO de pointe. Il est conçu pour être rapide et robuste en agissant directement sur les correctifs d'image, sans dépendre d'une extraction lente des caractéristiques. Au lieu de cela, il utilise des filtres probabilistes de profondeur sur les patchs de l'image elle-même.

2.3.2 Odométrie visuelle inertielle

Quel que soit l'algorithme utilisé, les solutions VO monoculaires traditionnelles sont incapables d'observer l'échelle de la scène et sont sujettes à la dérive d'échelle et à une ambiguïté d'échelle. Des méthodes de fermeture de boucle dédiées [57] sont intégrées pour réduire la dérive de l'échelle. Toutefois, l'ambiguïté d'échelle ne peut être résolue par la fermeture de la boucle et nécessite l'intégration d'informations externes. Il s'agit généralement de détecter les objets à l'échelle de la scène [57], [58],[59] [60] ou de fusionner les informations d'une unité de mesure inertielle (IMU) pour créer un système d'odométrie visuelle-inertielle (VIO).

La fusion de l'information inertielle et visuelle permet non seulement de résoudre l'ambiguïté de l'échelle, mais aussi d'augmenter la précision de l'odométrie visuelle elle-même. En théorie, la complémentarité entre les mesures inertielles d'un IMU et les données visuelles devrait permettre en toutes circonstances une estimation très précise du mouvement: les techniques de localisation visuelle dépendent entièrement de l'observation des particularités de l'environnement et dans les situations intérieures, ces techniques sont affectées par des occlusions intermittentes. D'autre part, la pose d'un dispositif IMU peut être suivie par une double intégration des données d'accélération ou des schémas plus complexes tels que la stratégie de pré-intégration, puis intégrée avec une méthode d'odométrie visuelle telle que SVO [56] pour former un système VIO dans lequel la dérive IMU reste contrainte. Dans les systèmes de pointe, la fusion est ensuite réalisée soit par un procédé basé sur l'optimisation, soit par un procédé basé sur un filtre [61].

2.3.3 localisation et cartographie visuelles simultanées

L'algorithme de localisation et de cartographie simultanées (SLAM) est un point névralgique de la recherche sur les robots et la vision par ordinateur, il est considéré comme l'une des technologies clés de la navigation automatique dans des environnements inconnus [62].

Le but du SLAM en général, et du V-SLAM en particulier, est d'obtenir une estimation globale et cohérente de la trajectoire. Cela implique de garder une trace d'une carte de l'environnement (même dans le cas où la carte n'est pas nécessaire en soi) parce qu'elle est nécessaire pour voir quand le robot revient dans une zone déjà visitée [55]. (C'est ce qu'on appelle la fermeture de boucle. Lorsqu'une fermeture de boucle est détectée, cette information est utilisée pour réduire la dérive de la carte et du trajet de la caméra. Comprendre quand une boucle se ferme et intégrer efficacement cette nouvelle contrainte dans la carte actuelle sont deux des principaux enjeux de SLAM.)

La Figure 2.3 montre un véhicule muni d'une caméra se déplaçant dans l'environnement et capturant les caractéristiques inconnues de l'image.

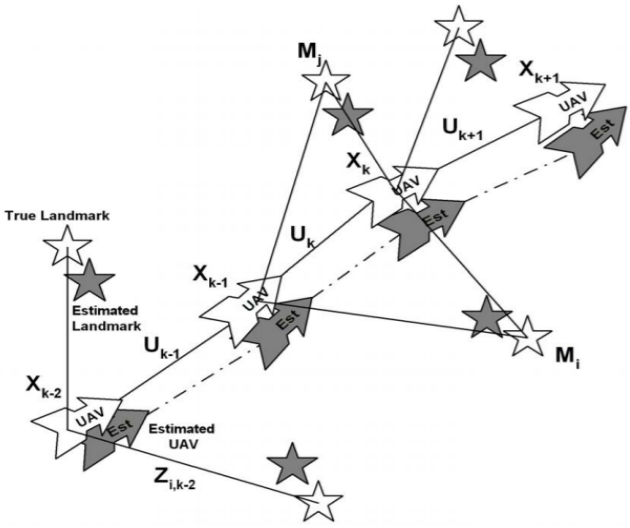


Figure 2.3: Localisation et Cartographie Simultanées

Avec, X_k L'état du véhicule à l'instant k,U_k le vecteur de commande du véhicule pour passer de k à $k-1$, M_i vecteur d'emplacement de la $i^{\text{ème}}$ caractéristique d'image (l'hypothèse est que cette caractéristique est statique) et Z_{ik} à l'instant k est la $i^{\text{ème}}$ image est observée à partir de la caméra du véhicule.

Les étapes du SLAM :

1. Supposons que la position initiale du véhicule X_{k-1} est le point de départ avec certaines caractéristiques préexistantes de l'image sur la carte avec une grande incertitude de la position du véhicule.

2. Lorsque le véhicule commence à se déplacer de X_{k-1} à X_k, le modèle de mouvement fournit de nouvelles estimations de sa nouvelle position et l'incertitude de son emplacement augmente.

3. Pendant le déplacement du véhicule, ajouter de nouvelles caractéristiques M_i à la carte et mettre à jour la mesure des caractéristiques existantes.

4. Calculez la différence entre l'observation prévue et l'observation réelle.

5. Mettre à jour l'estimation de l'état actuel du véhicule ainsi que la carte.

MonoSLAM [63] est le premier système SLAM de vision monoculaire en temps réel, il a été conçu pour étendre le filtre de Kalman pour le *back-end*, en suivant des points de caractéristiques très clairsemés. Le LSD-SLAM (SLAM monoculaire direct à grande échelle) [64] est un algorithme basé sur des caractéristiques et des méthodes directes. Il applique la méthode directe au SLAM monoculaire semi-dense, qui peut réaliser une reconstruction de scène semi-dense sur un CPU. ORB-SLAM2 [65] proposent un système SLAM inspiré de LSD-SLAM et qui a pour innovation l'utilisation des caractéristiques ORB, et peut être appliqué à toutes les scènes en temps réel. L'algorithme est divisé en quatre modules : suivi, construction, déplacement et détection en boucle fermée. Avec l'avènement de la nouvelle caméra événementielle à capteur, de nombreuses études SLAM basées sur des caméras événementielles [66][67] ont vu le jour ces dernières années. Cependant, la caméra événementielle est coûteuse et a une faible résolution spatiale, ce qui limite les performances de l'application.

Au bout de plusieurs décennies de développement, les méthodes de SLAM 2D avec scanner laser sont considérées comme matures. Les algorithmes modernes comme Hector SLAM [68] sont suffisamment robustes pour les applications sur un quadrirotor. En contrepartie, Le SLAM visuel sur drone reste un problème ouvert, principalement parce que les algorithmes sont fragiles au mouvement agile du drone et à l'incertitude inconnue de l'environnement. Des algorithmes récents, dont ORB-SLAM2 et LSD-SLAM, sont des candidats potentiels qui peuvent être appliqués aux drones.

2.3.4 Localisation visuelle par Apprentissage profond

Inspirées par la performance exceptionnelle des réseaux de neurones convolutionnels (CNN) dans une variété de tâches dans divers domaines et dans le but d'éliminer l'ingénierie manuelle des algorithmes pour la sélection de caractéristiques, les architectures CNN qui régressent directement la pose métrique en 6 DDL ont récemment été explorées[69] [70] [71].

Dans [72], une méthode d'odométrie visuelle stéréo est présenté. [73] a étudié la faisabilité d'utiliser un CNN pour extraire l'auto-déplacement des trames de flux optique. Dans [74] la l'utilisation d'un CNN pour extraire la relation homographique entre les paires de trames a été démontrée. Récemment, [74][75] utilisent un réseau de neurones récurrent (RNN: *Recurrent neural network*) pour fusionner la sortie d'un système de pose visuel standard basé sur des marqueurs et d'une IMU, éliminant ainsi la nécessité d'un filtrage statistique.

[76] ont proposé une architecture CNN multitâche entrainable de bout en bout (end-to-end) pour la localisation visuelle en 6 DDL et l'estimation d'odométrie à partir d'images monoculaires ultérieures, et a été le premier à combler l'écart de performance entre les méthodes de localisation locales basées sur les caractéristiques et celles basées sur un CNN.

Cependant, malgré leur capacité à gérer des conditions perceptuelles difficiles et à gérer efficacement de grands environnements, ils sont toujours incapables d'égaler les performances des méthodes de localisation basées sur les caractéristiques (features) les plus récentes. Ceci est dû en partie à leur incapacité à modéliser en interne les contraintes structurelles 3D de l'environnement tout en apprenant à partir d'une seule image monoculaire.

2.4 Navigation visuelle autonome

La navigation peut être décrite comme le processus de détermination d'un chemin approprié et sûr entre un point de départ et un point d'arrivée pour un robot se déplaçant entre eux[77]. La navigation peut-être schématiquement séparée en systèmes utilisant une carte (ou la construise) et des systèmes qui n'utilisent pas de carte (navigation réactive). La partie commande et planification que nous avons présentée s'inscrit dans la première catégorie. Dans cette section, nous nous intéressons à une navigation visuelle réactive [78].

En effet, les approches traditionnelles de la navigation autonome pour les drones effectuent l'apprentissage et la répétition visuels basés sur l'odométrie visuelle inertielle (VIO) [20][77][79][80][81][82] ou la localisation et la cartographie simultanées (SLAM) [65][83] qui sont utilisés pour fournir une estimation de la pose du drone par rapport à une carte interne [21][84]. Ces méthodes ne concernent pas la génération de trajectoire [32][85]. De plus, l'apprentissage et la répétition supposent un monde statique et une estimation précise de la pose : des hypothèses qui sont souvent violées dans la configuration de course de drones.

Dans le contexte des course de drones autonomes, un drone doit pouvoir franchir les portes rapidement et sans collision. Il est donc important de détecter les portes de manière fiable en utilisant la vision [20][77].

La vision par ordinateur s'est avérée très utile pour de nombreuses applications des drones, comme la détection et la poursuite de cibles. Généralement, l'asservissement visuel, un procédé pour générer un mouvement relatif tridimensionnel (3D) basé sur des données d'images bidimensionnelles (2D), est devenu une alternative très prometteuse pour la navigation des drones [85][86].

De nombreuses applications de l'asservissement visuel ont été appliquées aux drones tels que l'atterrissage sur des plates-formes mobiles [87], le vol à travers des portails [88], la préhension d'objet [88][89] et la poursuite de cible [90].

L'avènement de l'apprentissage profond a inspiré des solutions alternatives à la navigation autonome. Ces approches prédisent généralement les actions directement à partir d'images.

Nous allons donc explorer la tâche de navigation dans le contexte spécifique des courses de drones autonomes, en exposant premièrement les techniques détection des portails (*Gates"*) que doit traverser le quadrirotor; pour ensuite explorer deux méthodes de navigations visuelle populaires dans la course de drones autonomes que sont l'asservissement visuel et les approches par apprentissage profond.

2.4.1 Détection des portails (*Gates*)

La détection d'objets est un problème de longue date, fondamental et difficile à résoudre en vision par ordinateur. La détection d'objets est un domaine de recherche actif depuis plusieurs décennies. Le but de la détection d'objets est de déterminer s'il y a ou non des instances d'objets des catégories données (comme les humains, les voitures, les bicyclettes, les chiens et les chats) dans une image donnée et, le cas échéant, de retourner l'emplacement spatial et l'étendue de chaque instance (p. ex. par une case délimitée [91], [92]). La période avant 2012 a été dominé par des caractéristiques images construites "à la main". Depuis 2012 jusqu'à aujourd'hui, les CNN dominent ce domaine (voir figure 2.4).

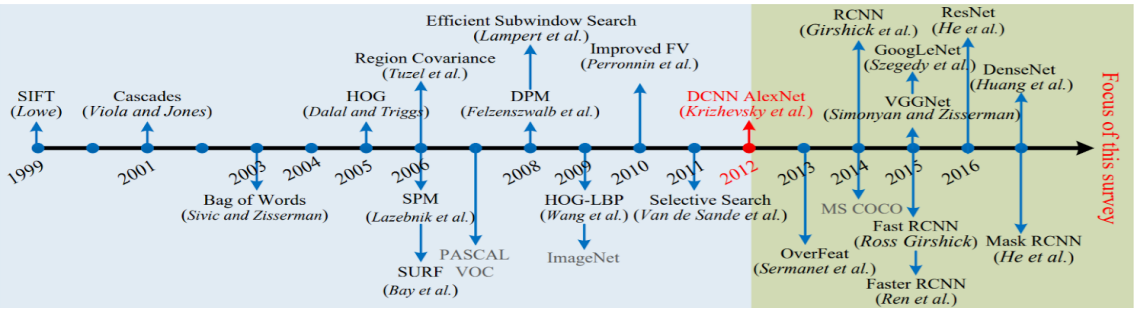


Figure 2.4: Les jalons de la détection et de la reconnaissance d'objets. .

La détection des portes peut être réalisée par de multiples méthodes de vision par ordinateur, telles que Viola Jones [93] et Hough transform [94]. Cependant, ces techniques de traitement d'images basées sur les pixels sont sévèrement affectées par les changements environnementaux, les conditions d'éclairage variables et les portes vues en chevauchement, les algorithmes traditionnels de traitement d'images basés sur la couleur et la géométrie des portes ont tendance à échouer pendant la course réelle, et les paramètres de l'algorithme doivent être ajustés manuellement avant chaque course.

Récemment, la technique de détection d'objets basée sur la méthode d'apprentissage profond s'est avérée très efficace pour résoudre ces problèmes. En particulier, grâce au développement des réseaux neuronaux convolutionnels (CNN) [95], le domaine de la détection d'objets a connu une croissance remarquable. Pour l'informatique embarquée, les algorithmes déployés sont souhaités pour obtenir de meilleures performances tout en consommant moins d'énergie en raison de la charge utile, de la puissance de calcul et de la source d'énergie limitées. AlexNet et GoogLeNet [96][97] ont fait preuve d'une vitesse et d'une performance excellentes dans la compétition ImageNet, et des algorithmes tels que SSD et YOLO [98][99] ont montré une bonne performance dans la partie détection. L'inconvénient de l'utilisation de ces méthodes pour la navigation des drones est que ces algorithmes sont encore insuffisants en termes de puissance de calcul requise pour un système embarqué.

Techniques basées sur la couleur

Les gagnants du défi de course de dronse autonomes de l'IROS 2016 [86] ont utilisé un processus de détection des portes basé sur des informations de couleur. L'image RGB est convertie en un cadre de valeur de saturation de teinte (*Hue Saturation Value* (HSV)) pour capturer la couleur orange des portes. La méthode de détection des bords *Canny* est utilisée, et les bords détectés sont dilatés pour avoir une forme de cadre épais.

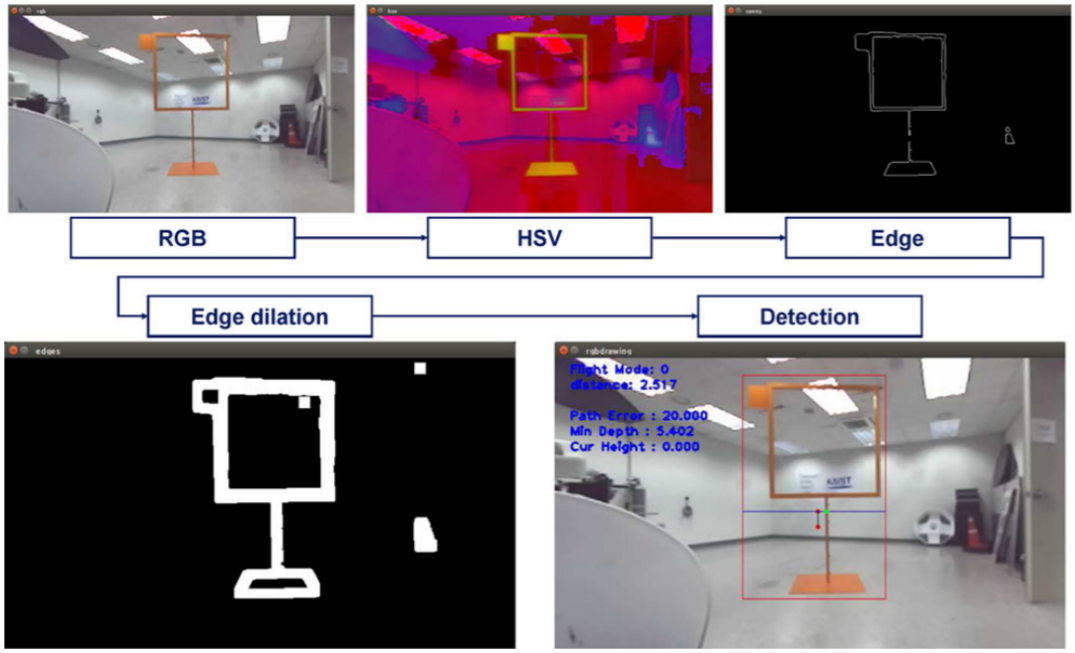


Figure 2.5: Détection de portails par la méthode HSV.

Techniques basées sur l'apprentissage profond

[20] [86] ont présenté un modèle d'apprentissage approfondi pour la reconnaissance de gates en temps réel sur des plates-formes embarquées. Ils ont utilisé un réseau de neurones convolutionnel basé sur le modèle *Single Shot Detctor SSD* [98]. Pour améliorer la vitesse d'inférence avec un meilleur compromis vitesse-précision, ils ont modifié la structure du réseau et appliqué AlexNet [96] comme réseau de base au lieu du VGG-16 [100].

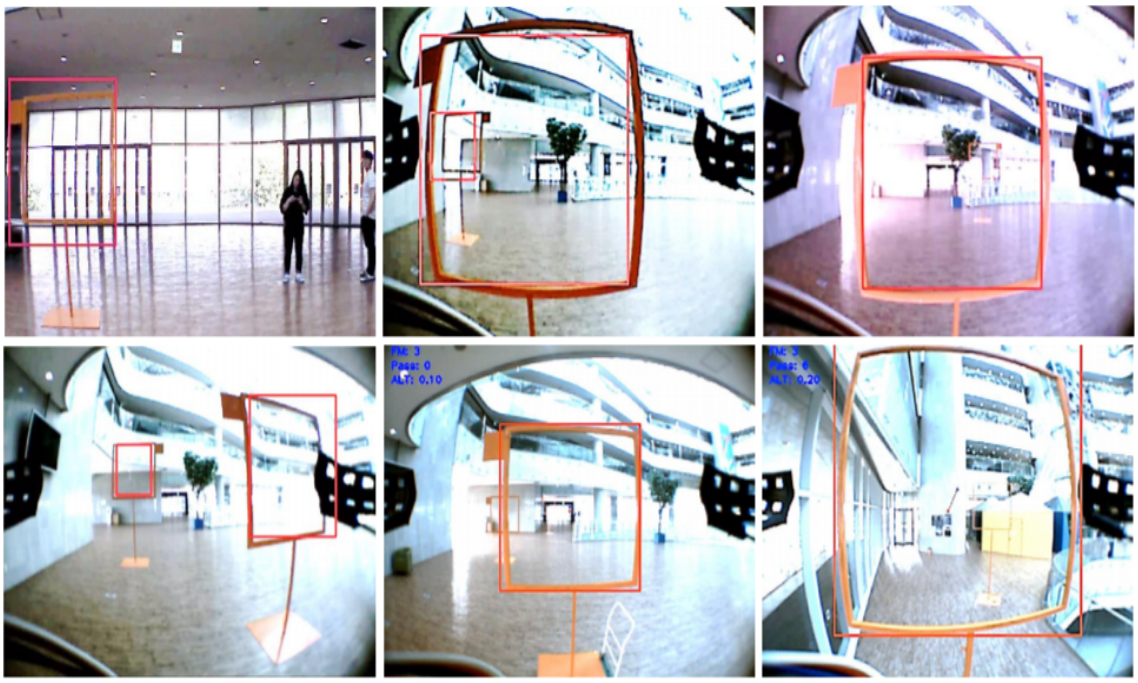


Figure 2.6: Détection de portails grâce au CNN SSD.

2.4.2 L'asservissement visuel

L'asservissement visuel (*Visual servoing* (VS)) fait référence à l'utilisation de données de vision par ordinateur pour contrôler le mouvement d'un robot [101]. Pour les courses de drones autonomes, les données de vision sont acquises à partir d'une caméra montée directement sur le quadrirotor, auquel cas le mouvement du robot induit le mouvement de la caméra : c'est ce qu'on appelle une configuration *œil-dans-la-main* (*eye-in-hand*).

Les techniques de servocommande visuelle peuvent être divisées en deux branches parallèles [101]: Asservissement visuel basé sur la pose (PBVS) et asservissement visuel basé sur l'image (IBVS). Dans le PBVS, la pose de l'objectif est directement obtenue à partir d'une reconstruction 3D complète de l'environnement ou à partir de la pose d'un ou plusieurs repères qui y sont placés. En revanche, l'IBVS formule le problème en termes d'emplacement des caractéristiques dans l'image: la pose désirée est définie par les emplacements souhaités des caractéristiques dans l'image finale. La loi de contrôle vise à minimiser les erreurs de reprojection des caractéristiques. Même si l'IBVS ne nécessite pas d'estimation 3D complète, il a quand même besoin de la profondeur de la cible.

Plusieurs approches IBVS [90][102][103] et PBVS [104][105] ont été appliquées au contrôle des véhicules aériens au cours des dernières décennies. Dans ces solutions, le contrôleur utilise l'information visuelle comme source principale pour le calcul de la pose de la cible, sans tenir compte de l'endroit où la cible est reprojetée dans le plan image le long de la trajectoire. Une solution possible qui atténue généralement cette faiblesse est la limitation cinématique du multirotor en termes d'angles de roulis et de tangage, mais cela pénalise la maniabilité du véhicule. [106] présentent une approche qui tire parti de la dynamique de rotation du véhicule, où un ressort virtuel pénalise la grande rotation par rapport à un cadre aligné avec la gravité. Certaines approches récentes permettent de traduire la dynamique des caractéristiques de la cible dans le plan de l'image et utilise cette information pour compenser les angles de roulis et de tangage et par la suite les maintenir proches de zéro [87], [106][89][107].

Le couplage entre la perception et la planification a également été abordé dans [88][107], où un drone doit à la fois se localiser par rapport à une porte et la franchir. Ils planifient une trajectoire balistique capable de satisfaire à la fois les contraintes dynamiques et de perception en maximisant la distance du véhicule par rapport aux bords du portail à passer.

Les vainqueurs du concours de drones autonomes IROS 2016 [86] ont utilisé l'IBVS comme stratégie de navigation visuelle.

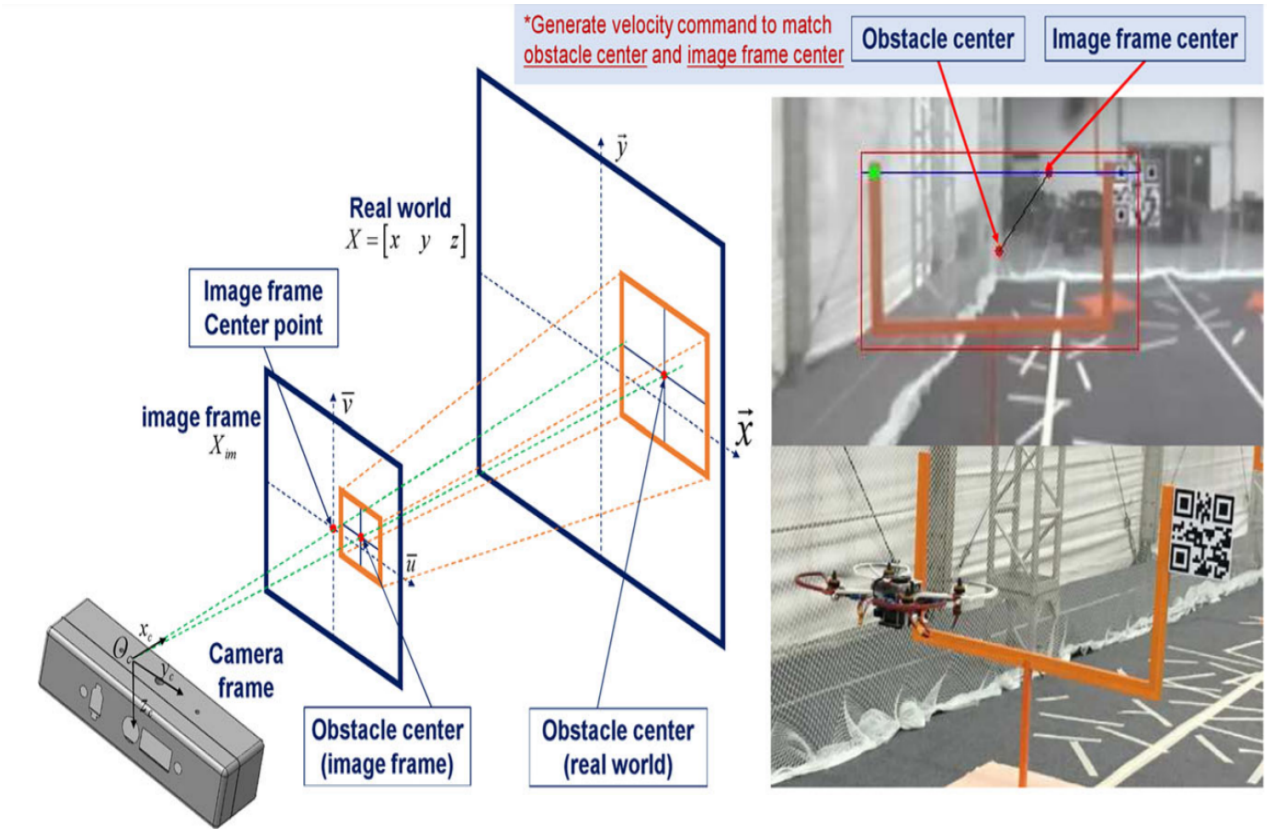


Figure 2.7: Illustration d'une méthode simple de Visual servoing : Alignement du centre du portail avec le centre de l'image.

Ce qui fait du VS un problème difficile pour les multirotors, c'est la dynamique sous-actionnée de ces véhicules, surtout lorsqu'ils effectuent des manœuvres agiles et rapides (c-à-d. à grande vitesse et accélération angulaire) [108]. Lors de ce type de manœuvres, les contrôleurs visuels standard se concentrent uniquement sur l'atteinte de l'objectif sans tenir compte en permanence de la configuration de leur caméra par rapport à l'environnement perçu. En d'autres termes, pendant le vol du drone, ces systèmes peuvent perdre pendant un certain temps la ligne de visée avec une cible d'intérêt, même si cette cible représente l'objectif final du vol. Ce comportement peut empêcher l'applicabilité de tels contrôleurs dans des activités où la relocalisation de la cible d'intérêt n'est pas une tâche triviale.

2.4.3 La navigation par apprentissage profond

L'apprentissage profond (en anglais deep learning, deep structured learning ou encore hierarchical learning) est un ensemble de méthodes d'apprentissage automatique (machine learning) tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires [109].

Des travaux récents ont montré que les réseaux profonds peuvent fournir aux drones des capacités de perception robustes et faciliter une navigation sûre même dans des environnements dynamiques [11][110][20]. Les représentations de sortie vont de la prédiction des commandes de navigation discrètes (classification dans l'espace d'action) [111][110][112] à la régression directe des signaux de commande [113].

[11] ont fait valoir que l'apprentissage *end-to-end* des commandes à bas niveau est sous-optimal pour la tâche de la navigation des drones. En effet, le réseau doit apprendre la notion de base de stabilité à partir de zéro afin de contrôler une plate-forme instable telle qu'un quadrirotor. Cela conduit à une grande complexité des échantillons et ne donne aucune garantie mathématique sur la stabilité des plates-formes.

De plus, comme le réseau n'a pas besoin d'assurer la stabilité de la plate-forme, il peut traiter les images à un débit inférieur à celui du contrôleur de bas niveau, ce qui permet un calcul à bord [11].

Une autre ligne de travail combine les prédictions de réseaux neuronaux avec le contrôle prédictif du modèle en régressant la fonction de coût à partir d'une seule image [114].

Dans le contexte de la course de drones, [11] ont proposé une représentation intermédiaire sous la forme d'une direction de but et d'une vitesse souhaitée. Un CNN est entraîné à imiter une trajectoire optimale[32] à travers la piste, puis le planificateur utilise cette information pour générer un segment de trajectoire court et minimal et les commandes moteur correspondantes pour atteindre l'objectif souhaité. L'un des avantages de cette approche est qu'elle permet de naviguer même lorsqu'il n'y a pas de porte en vue, en exploitant le contexte propre à la voie et les informations de fond.

Dans [115], un drone simulé apprend comment minimiser le temps passé à terminer la piste de course. Les auteurs ont formé un réseau neuronal convolutionnel profond entraîné en utilisant le paradigme de l'*Imitation Learning* sur des contrôleurs PID. Bien qu'intéressante, cette approche ne tient pas compte de plusieurs aspects réels des courses de drones, tels que le calcul embarqué restreint ou la façon de traiter les biais des accéléromètres.

Enfin,[7], les gagnants du concours IROS 2018 Autonomous Drone Racing ont utilisé un CNN qui prédit la pose relative entre le drone et la porte pour passer avec l'incertitude de la prédiction sous forme de covariance des estimations. ces informations sont fusionnées avec la VIO en utilisant un *Extended Kalman Filter* et ensuite une commande à modèle prédictif est déployé pour générer et suivre une trajectoire souhaitée.

2.5 Conclusion

La Navigation visuelle autonome en général, et dans le domaine du Drone racing en particulier, demeure un problème non résolu.

De plus en plus de techniques faisant appel à l'apprentissage automatique sont étudiés avec plus ou moins de succès. Ces techniques nécessitent une grande quantité de données d'entraînement collectées sur le même parcours. Cela contraste avec les pilotes humains, qui peuvent s'adapter rapidement à de nouvelles pistes en tirant parti des compétences acquises dans le passé.

Dans ce chapitre, nous avons présenté l'état de l'art en matière de course de drones autonome en balayant la commande des quadrirotors, ainsi qu'un large éventail de méthodes pour la localisation et la navigation en utilisant principalement une IMU et une caméra

comme capteurs. Nous avons aussi montré que dans le contexte du Drone Racing, ces problématiques sont poussées à l'extrême et sont le plus souvent couplées.

Grâce à l'état de l'art dressé, nous proposons l'architecture suivante pour répondre à la problématique des courses de drones autonomes:

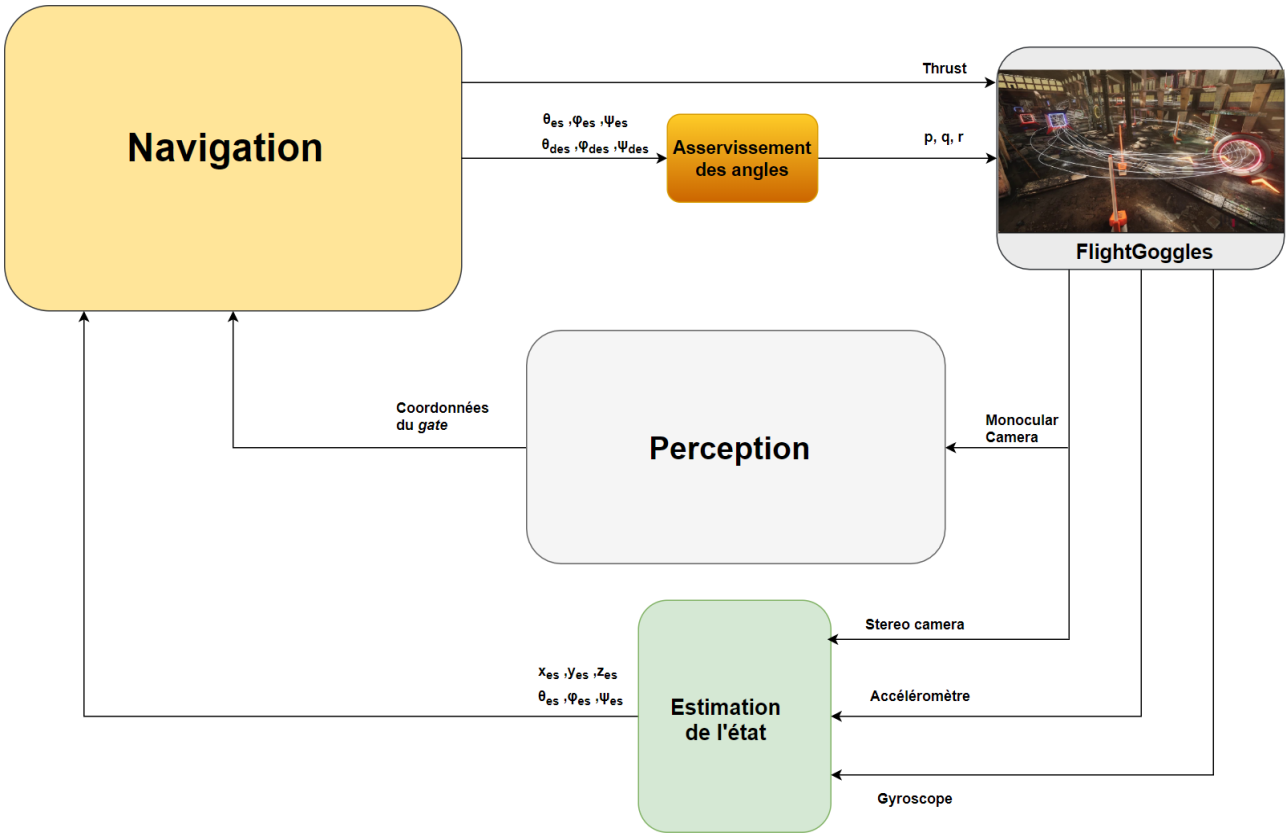


Figure 2.8: Architecture de la solution proposée.

Cet architecture étant modulaire, nous détaillerons dans les trois chapitres suivants les détails de chaque module, puis appliquerons toute l'architecture dans le chapitre implémentation.

Chapitre 3

Modélisation et commande du quadrirotor

3.1 Introduction

Les quadrirotors sont le type le plus populaire de micro-véhicules aériens. Ils ont gagné leur popularité grâce à leur conception mécanique simple et robuste, qui peut être réalisée à partir de composants standard peu coûteux. De plus, ils sont capables de voler en vol stationnaire, de décoller et d'atterrir verticalement tout en étant capables d'effectuer des manœuvres agiles. Les quadrirotors sont déployés dans de nombreuses applications, où les capteurs ou les charges utiles doivent être déplacés vers des endroits difficiles d'accès ou même inaccessibles pour les humains ou les robots au sol. Parmis ces applications, mentionnons l'inspection de diverses infrastructures, le contrôle et l'analyse en agriculture, la surveillance, le transport et la photographie aérienne, qui, ensemble, forment un marché de plusieurs milliards de dollars.

Les premiers travaux de modélisation et de contrôle de la robotique aérienne remontent à la fin des années 1990 et le domaine est resté actif depuis [116][117]. En effet, L'étude des drones quadrirotors, véhicules aériens sans pilote avec quatre moteurs-hélices, a attiré l'attention de chercheurs dans le domaine de l'automatique pendant ces quinze dernières années. En 2012, Mahony et Kumar, deux chercheurs reconnus dans cette communauté, ont même considéré le quadrirotor comme la plateforme fondamentale pour la recherche de la robotique aérienne [118].

La modélisation d'un système dynamique est l'étape initiale avant de passer à la conception d'un contrôleur. Dans le cas des drones quadrirotors, la modélisation a déjà été étudiée par les pionniers dans ce domaine d'application [116][119][120]. Dans ces articles, les auteurs abordent initialement la modélisation dynamique du système en utilisant les équations de Newton-Euler. D'autres auteurs présentent l'analyse de la modélisation en utilisant les équations d'Euler-Lagrange [121].

Nous commencerons donc ce chapitre par décrire de façon concise l'analyse dynamique d'un drone quadrirotor pour l'obtention d'un modèle non linéaire. Dans une première partie, les repères de référence utilisés sont décrits. Puis une analyse cinématique et dynamique du quadrirotor à partir des équations de Newton-Euler est présentée ainsi que la représentation d'état équivalente. Afin d'effectuer une simulation en boucle fermée de notre modèle, nous élaborerons un asservissement non linéaire avec deux boucles de commandes en cascade : une

pour asservir la position et l'autre l'orientation du quadrirotor. Les simulations montreront la stabilité et la robustesse de cette commande.

3.2 Modélisation du système

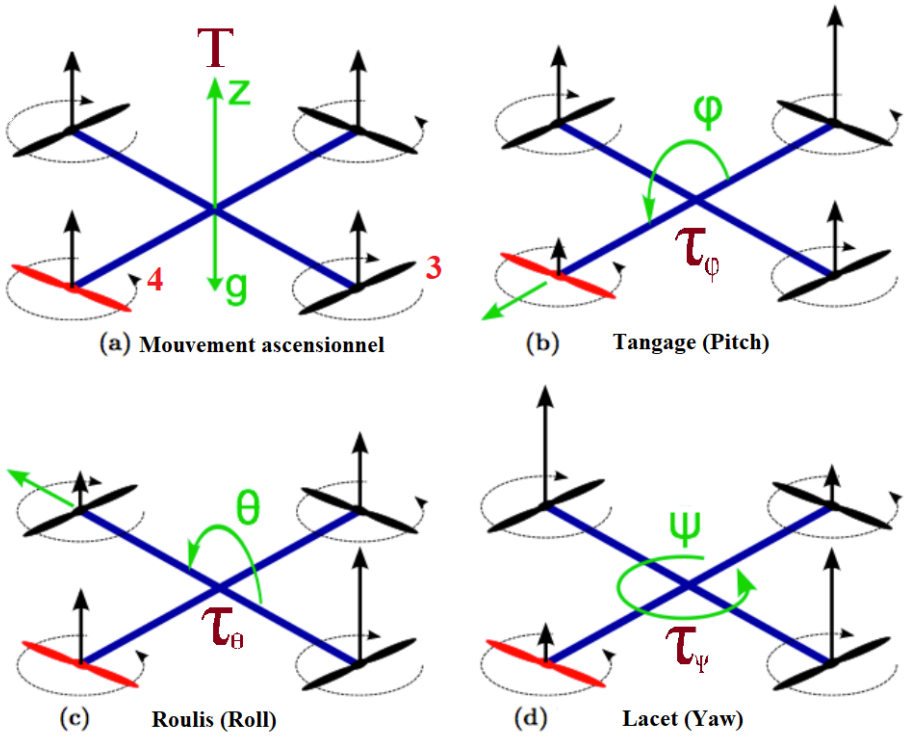


Figure 3.1: Modèle de quadrirotor.

il y a deux configurations typiques des quadrirotors : $'+'$ et $'\times'$. Contrairement aux premiers quadrirotors de l'histoire qui étaient actionnés par un seul moteur et contrôlés par la variation de l'angle d'attaque des hélices, actuellement, sauf certaines exceptions [5], les quadrirotors sont actionnés par quatre moteurs, un pour chaque rotor, et ils n'ont pas la possibilité de varier l'angle d'attaque des hélices. Avec cette configuration, les mouvements sont contrôlés par la variation de la vitesse de chacun des rotors.

Chacune des hélices fournit une force de poussée $\mathbf{F}_i$ et un couple de traînée τ_i . La figure 3.1 montre le sens des forces et des couples induits. On peut constater que le sens de rotation des rotors non opposés est inversé, cela permet d'équilibrer les moments induits par les forces de traîné. Ces forces et couples sont considérés proportionnels au carré de la vitesse de rotation des hélices.

De cette manière, on peut induire les rotations en roulis, tangage et lacet en faisant varier les vitesses de rotation des hélices comme montré dans la figure 3.1.

la force de poussée T est égale à la somme des forces induites par chaque hélice. Dans le cas d'une rotation en roulis positive, le moteur 4 augmente sa vitesse inversement proportionnel avec le moteur 1 tandis que la vitesse des moteurs 2 et 3 reste constante. De manière réciproque on obtient une rotation en tangage. Et dans le cas d'une rotation en lacet positive, ce sont le moteurs 1 et 3 qui tournent plus vite. On peut alors calculer trois couples (τ_ϕ, τ_θ et τ_{psi}) autour des axes du repère du drone en fonction des forces de poussée et des couples générés par les hélices. Pour cela, il faut également considérer la géométrie de la structure du quadrirotor, particulièrement la distance des moteurs aux axes de rotation. Plus de détails sont présents dans la section 3.4.

3.3 Modèle cinématique

Nous considérerons un repère terrestre W avec sa base orthonormale $\{\mathbf{x}_W, \mathbf{y}_W, \mathbf{z}_W\}$ et un repère B lié au corps du quadrirotor muni de sa base $\{\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B\}$. L'origine de B coïncide avec le centre de masse du quadrirotor.

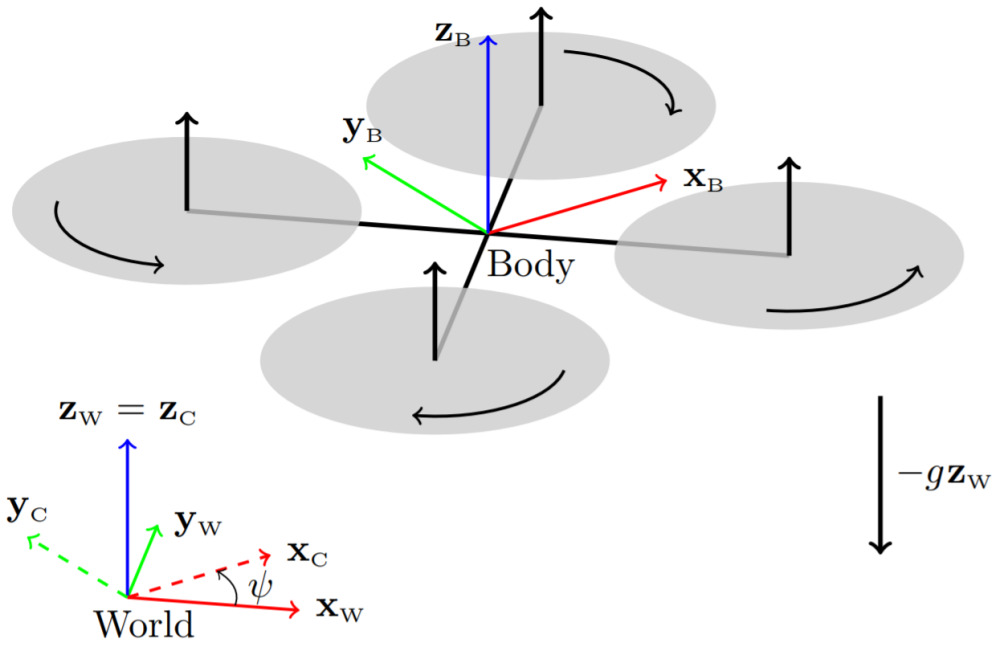


Figure 3.2: Système de coordonnées du quadrirotor.

Les sens de rotation des rotors sont également indiqués sur la Figure 3.2. Le quadrirotor est soumis à une accélération gravitationnelle g dans le sens négatif $\mathbf{z}_W$. Nous désignons la position du centre de masse du quadrirotor comme $\mathbf{p}$, et ses dérivés, la vitesse et l'accélération par $\mathbf{v}$ et $\mathbf{a}$ respectivement.

Matrices de rotation et angles d'Euler

Nous noterons la matrice de rotation $\mathbf{R}_{WB}$ et la translation $\mathbf{t}_{WB}$ qui permettent le passage du repère B au repère W. Pour convertir un point $\mathbf{p}$ du repère B à W on utilise:

$$_W\mathbf{p} = \mathbf{R}_{WB} \cdot {}_B\mathbf{p} + {}_W\mathbf{t}_{WB} \tag{3.1}$$

Nous utiliserons les angles d'Euler pour représenter les rotations. Avec la convention $Z-Y-X$, la rotation entre W et B est définie par l'ordre suivant (Figure 3.1):

1. ψ (yaw) autour de $\mathbf{z}_B$.
2. θ (pitch) autour du nouveau $\mathbf{y}_B$.
3. ϕ (roll) autour du nouveau $\mathbf{x}_B$.

et on a :

$$\mathbf{R}_{WB} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi) \tag{3.2}$$

Où,

$$\mathbf{R}_z(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{3.4}$$

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{3.5}$$

Nous obtenons (avec $c(x) = cos(x)$ et $s(x) = sin(x)$):

$$\mathbf{R}_{WB} = \begin{bmatrix} c(\psi)c(\theta) & c(\psi)s(\theta)s(\phi) - s(\psi)c(\phi) & c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) \\ s(\psi)c(\theta) & s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & s(\psi)s(\theta)c(\phi) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{3.6}$$

$$= [\mathbf{x}_B\, \mathbf{y}_B\, \mathbf{z}_B] \tag{3.7}$$

$\mathbf{x}_B$, $\mathbf{y}_B$ et $\mathbf{z}_B$ étant les vecteurs de la base orthogonale de B représentés dans le repère W comme le montre la Figure 3.2.

Vitesses angulaires

La vitesse angulaire du repère B par rapport au repère inertiel W est dénoté $\boldsymbol{\omega}_{WB}$. Pour la prochaine étape, on définit la matrice antisymétrique (skew-symmetric matrix) suivante.

$$\boldsymbol{\omega}_{WB,X} = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \tag{3.8}$$

$\boldsymbol{\omega}_{WB,X}$ est la représentation antisymétrique de $\boldsymbol{\omega}_{WB}$ tel que:

$$\boldsymbol{\omega}_{WB} \times v = \boldsymbol{\omega}_{WB,X}\, v \ (\forall\, v \in \mathbb{R}^3) \tag{3.9}$$

La vitesse angulaire est définie comme suit:

$$ {}_W\boldsymbol{\omega}_{WB,X} \coloneqq \dot{\boldsymbol{R}}_{WB} \cdot \boldsymbol{R}_{WB}^T \tag{3.10} $$

ou dans le repère mobile B

$$ \begin{aligned} {}_B\boldsymbol{\omega}_{WB,X} &= \boldsymbol{R}_{BW} \cdot {}_W\boldsymbol{\omega}_{WB,X} \cdot \boldsymbol{R}_{BW}^T && (3.11)\\ &= \boldsymbol{R}_{BW} \cdot \dot{\boldsymbol{R}}_{WB} \cdot \boldsymbol{R}_{WB}^T \cdot \boldsymbol{R}_{BW}^T && (3.12)\\ &= \boldsymbol{R}_{WB}^T \cdot \dot{\boldsymbol{R}}_{WB} && (3.13) \end{aligned} $$

En réécrivant 3.10 et 3.13 avec les angles d'Euler on obtient:

$$ {}_W\boldsymbol{\omega}_{WB} = \begin{bmatrix} c(\theta)c(\psi) & -s(\psi) & 0 \\ c(\theta)s(\psi) & c(\psi) & 0 \\ -s(\theta) & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.14} $$

et

$$ {}_B\boldsymbol{\omega}_{WB} = \begin{bmatrix} 1 & 0 & -s(\theta) \\ 0 & c(\phi) & s(\phi)c(\theta) \\ 0 & -s(\phi) & c(\phi)c(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.15} $$

3.4 Modélisation des forces aérodynamiques

L'aérodynamique des rotors a fait l'objet d'études approfondies au milieu des années 1900 avec le développement des hélicoptères et des modèles détaillés de l'aérodynamique des rotors sont disponibles dans la littérature [122][123]. Une grande partie des détails de ces modèles aérodynamiques est utile pour la conception de systèmes de rotors où toute la gamme de paramètres (géométrie du rotor, profil, mécanisme de charnière, et bien plus encore) sont fondamentaux pour la conception du robot. Pour un véhicule à quadrirotor robotique typique, la complexité de la modélisation aérodynamique est mieux ignorée. Néanmoins, un niveau élémentaire de théorie aérodynamique est important pour comprendre les particularités de la conception des commandes.

La poussée en régime permanent générée par un rotor en vol stationnaire (c.-à-d. un rotor qui ne se déplace pas horizontalement ou verticalement) à l'air libre peut être modélisée [124] comme suit:

$$ T_i = C_T \rho A_{r_i} r_i^2 \varpi_i^2 \tag{3.16} $$

Avec pour le rotor i, A_{r_i} est la surface du disque du rotor, r_i est le rayon, ϖ_i est la vitesse angulaire, C_T est le coefficient de poussée qui dépend de la géométrie et du profil du rotor, et ρ est la densité de l'air. En pratique, un modèle plus simple est utilisé :

$$ T_i = c_T \varpi_i^2 \tag{3.17} $$

où $c_T > 0$ représente le coefficient de portance déterminée à partir de tests statiques.

Le couple de réaction (force de traînée créée par la rotation des quatre hélices) agissant sur le quadrirotor est modélisé comme suit :

$$Q_i \coloneqq C_Q \varpi_i^2 \tag{3.18}$$

où le coefficient C_Q (qui dépendent aussi de A_{r_i},r_i et ρ) représente la constante de traînée et peut être déterminée par des essais statiques de poussée.

La poussée totale appliqué au véhicule est la somme des poussées de chaque rotor (Figure 3.1):

$$T = \sum_{i=1}^{4} \mid T_i \mid = c_T \, (\sum_{i=1}^{4} \varpi_i^2) \tag{3.19}$$

Les couples aérodynamiques appliqués au quadrirotor s'écrivent alors (Figure 3.1) :

$$\begin{aligned} \tau_\phi &= c_T \, d(\varpi_4^2 - \varpi_2^2), \\ \tau_\theta &= -c_T \, d(\varpi_1^2 - \varpi_3^2), \\ \tau_\psi &= c_Q \, d(-\varpi_1^2 + \varpi_2^2 - \varpi_3^2 + \varpi_4^2) \end{aligned} \tag{3.20}$$

en combinant 3.19 et 3.20 on obtient la matrice Γ qui relie la poussée et les couples aérodynamiques aux vitesses des rotors:

$$\begin{bmatrix} T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ 0 & -d\,c_T & 0 & d\,c_T \\ -d\,c_T & 0 & d\,c_T & 0 \\ -d\,c_Q & d\,c_Q & -d\,c_Q & d\,c_Q \end{bmatrix} \begin{bmatrix} \varpi_1^2 \\ \varpi_2^2 \\ \varpi_3^2 \\ \varpi_4^2 \end{bmatrix} \tag{3.21}$$

L'inversion de cette matrice fournit une correspondance entre l'entrée de commande souhaitée pour la dynamique du corps rigide et les consignes de vitesse du rotor pour la commande du moteur.

Dans la pratique, d'autres effets aérodynamiques de second ordre sont présents dans la génération de poussée des quadrirotors [125][126]. Les principaux effets aérodynamiques de second ordre présents à basse vitesse sont la variation du débit d'admission, le battement du rotor et la traînée induite. Le premier de ces effets réduit la poussée de soulèvement lorsque le quadrirotor monte et augmente la poussée lorsqu'il descend en raison de la variation de la vitesse d'entrée du flux du rotor causée par le mouvement du quadrirotor. Il s'agit d'un terme d'amortissement dans le sens vertical du mouvement du quadrirotor. Les deux autres effets génèrent des forces qui s'opposent à la translation horizontale du quadrirotor [127]. En pratique, il est difficile de différencier ces effets et il suffit de les modéliser tous par une seule force d'amortissement Δ.

$$\Delta = -Dv \tag{3.22}$$

Cette force assimilant un amortissement est négligée lors de l'établissement du modèle dynamique.

3.5 Modélisation dynamique

D'après la $1^{ère}$ loi d'Euler [128], la quantité de mouvement d'un corps rigide, $\boldsymbol{\mu}$ est définie comme suit:

$$\boldsymbol{\mu} \coloneqq m \cdot \mathbf{v}_s \tag{3.23}$$

où m est la masse du corps et $\mathbf{v}_s$ la vitesse de son centre de masse. La première Loi d'Euler stipule que le changement de la quantité de mouvement est égal à la somme des forces extérieures $\boldsymbol{F}$:

$$\dot{\boldsymbol{\mu}} = \boldsymbol{F} \tag{3.24}$$

On a alors

$$\dot{\boldsymbol{\mu}} = m \cdot \boldsymbol{a}_s = \boldsymbol{F} \tag{3.25}$$

où $\boldsymbol{a}_s$ est l'accélération du centre de masse.
En évaluant 3.25 dans le repère inertiel W on obtient:

$$_W\ddot{\mathbf{p}} = {}_W\boldsymbol{g} + \boldsymbol{R}_{WB}(\phi, \theta, \psi) \cdot \boldsymbol{c} \tag{3.26}$$

$_W\ddot{\mathbf{p}} = [\ddot{x}, \ddot{y}, \ddot{z}]^T$: est l'accélération du centre de masse du quadrirotor.
$_W\boldsymbol{g} = [0, 0, -g]^T$: est la force gravitationnelle.
$\boldsymbol{c} = [0, 0, T]^T$: est la force de portance totale des quatre rotors, exprimée dans B.
$\boldsymbol{R}_{WB}(\phi, \theta, \psi)$: est la matrice de rotation définie dans 3.6.

La $2^{ème}$ loi d'Euler [128] stipule que le moment cinétique d'un corps rigide par rapport à un point stationnaire O est donné par:

$$\boldsymbol{L}_O \coloneqq \boldsymbol{R}_{OP} \times \boldsymbol{\mu} + \boldsymbol{J}_P\dot{\Omega} + m \cdot \boldsymbol{R}_{PS} \times \boldsymbol{v}_P \tag{3.27}$$

où P est un point arbitraire du corps, $\boldsymbol{J}_P$ est la matrice d'inertie par rapport à une rotation autour de P, $\boldsymbol{\Omega}$ est la vitesse angulaire du corps. En remplaçant dans 3.27 le point arbitraire P par le centre de masse $P = S$ on trouve:

$$\boldsymbol{L}_O = \boldsymbol{R}_{OS} \times \boldsymbol{\mu} + \boldsymbol{J}_S\dot{\boldsymbol{\Omega}} \tag{3.28}$$

La $2^{ème}$ loi d'Euler stipule que le changement du moment cinétique est égal à la somme de tous les moments externes $\boldsymbol{M}_O$ par rapport à O:

$$\dot{\boldsymbol{L}}_O = \boldsymbol{M}_O \tag{3.29}$$

Dériver 3.28 et l'injecter dans 3.29 nous donne:

$$\dot{\mathbf{L}}_O = \mathbf{R}_{OS} \times \dot{\mu} + \mathbf{J}_S \cdot \Psi + \boldsymbol{\Omega} \times \mathbf{J}_S \cdot \boldsymbol{\Omega} = \boldsymbol{M}_O \tag{3.30}$$

Avec Ψ l'accélération angulaire du corps. Le développement de 3.30 donne:

$$\begin{aligned} \dot{\mathbf{L}}_O - \mathbf{R}_{OS} \times \dot{\mu} = \mathbf{J}_S \cdot \Psi + \boldsymbol{\Omega} \times \mathbf{J}_S \cdot \boldsymbol{\Omega} &= \mathbf{M}_O - \mathbf{R}_{OS} \times \dot{\mu} \\ &= \mathbf{M}_O + \mathbf{R}_{SO} \times \dot{\mu} \\ &= \mathbf{M}_O + \mathbf{R}_{SO} \times \mathbf{F} \\ \mathbf{J}_S \cdot \Psi + \boldsymbol{\Omega} \times \mathbf{J}_S \cdot \boldsymbol{\Omega} &= \mathbf{M}_S \end{aligned} \tag{3.31}$$

En évaluant 3.31 dans le repère mobile B on obtient:

$$\mathbf{J} \cdot {}_B\dot{\boldsymbol{\omega}}_{\boldsymbol{WB}} + {}_B\boldsymbol{\omega}_{\boldsymbol{WB}} \times \mathbf{J} \cdot {}_B\boldsymbol{\omega}_{\boldsymbol{WB}} = \boldsymbol{\eta} \tag{3.32}$$

$\boldsymbol{J} = diag(J_{xx}, J_{yy}, J_{zz})$: la matrice d'inertie du quadrirotor.
${}_B\boldsymbol{\omega}_{WB} = [p, q, r]^T$: vitesse angulaire du quadrirotor exprimé dans le repère mobile B.
$\boldsymbol{\eta} = [\boldsymbol{\tau}_\phi, \boldsymbol{\tau}_\theta, \boldsymbol{\tau}_\psi]^T$: le vecteur des couples aérodynamiques, exprimé dans B.

Représentation d'état

On choisit le vecteur d'état suivant $s = [\ x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}\ \phi\ \theta\ \psi\ p\ q\ r\ ]^T$.
Le système peut être représenté sous sa forme d'état suivante:

$$\begin{aligned}
\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} &= \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix}\\
\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} &= \begin{bmatrix}0\\ 0\\ -g\end{bmatrix} + \boldsymbol{R}_{WB}(\phi,\theta,\psi)\begin{bmatrix}0\\ 0\\ T\end{bmatrix}\\
\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} &= \begin{bmatrix}1 & t(\theta)s(\phi) & t(\theta)c(\phi)\\ 0 & c(\phi) & -s(\phi)\\ 0 & s(\phi)/c(\theta) & c(\phi)/c(\theta)\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix}\\
\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} &= \begin{bmatrix}\frac{1}{\boldsymbol{J}_{xx}}(\boldsymbol{\tau}_\phi + q\cdot r(\boldsymbol{J}_{yy} - \boldsymbol{J}_{zz}))\\ \frac{1}{\boldsymbol{J}_{yy}}(\boldsymbol{\tau}_\theta + p\cdot r(\boldsymbol{J}_{zz} - \boldsymbol{J}_{xx}))\\ \frac{1}{\boldsymbol{J}_{zz}}(\boldsymbol{\tau}_\psi + p\cdot q(\boldsymbol{J}_{xx} - \boldsymbol{J}_{yy}))\end{bmatrix}
\end{aligned} \tag{3.33}$$

Une autre écriture plus compacte qui se base sur l'écriture vectorielle et la matrice de rotation $\boldsymbol{R}_{WB}$ donne la représentation suivante:

$$\begin{aligned}
{}_W\dot{\mathbf{p}}_{WB} &= {}_W\mathbf{v}_{WB}\\
{}_W\dot{\mathbf{v}}_{WB} &= {}_W\boldsymbol{g} + \boldsymbol{R}_{WB}(\phi,\theta,\psi)\cdot\boldsymbol{c}\\
\dot{\mathbf{R}}_{WB} &= \boldsymbol{R}_{WB}\cdot {}_B\boldsymbol{\omega}_{WB,X}\\
{}_B\dot{\omega}_{WB} &= \boldsymbol{J}^{-1}\cdot(\boldsymbol{\eta} - {}_B\boldsymbol{\omega}_{WB}\times\boldsymbol{J}\cdot{}_B\boldsymbol{\omega}_{WB})
\end{aligned} \tag{3.34}$$

3.6 Fermeture de la boucle

Le quadrirotor est un système sous-actionné, il possède 4 entrées de commande et 6 degrés de liberté et ne peut donc pas être commandé pour suivre des trajectoires arbitraires dans l'espace de configuration.
La structure de contrôle hiérarchique proposée est formée d'une boucle interne qui commande l'orientation en utilisant les couples aérodynamiques comme entrée de contrôle. Le niveau de réglage externe utilise l'orientation et la poussée pour suivre une trajectoire de référence.
Nous désignerons les grandeurs qui peuvent être calculées à partir d'une trajectoire de référence avec un indice *ref* (valeur de référence), et les grandeurs qui sont calculées par une loi de commande en boucle externe et transmises à un contrôleur de boucle interne avec un indice *des* (valeur désirée). Une trajectoire de référence est constituée de la position de référence $\mathbf{p}_{ref}$, de la vitesse de référence $\mathbf{v}_{ref}$, de l'accélération de référence $\mathbf{a}_{ref}$; ainsi que d'un cap de référence ψ_{ref}.

Régulation de la position

Soit : $\mathbf{e}_p = \mathbf{p} - \mathbf{p}_{ref}$ et $\mathbf{e}_v = \mathbf{v} - \mathbf{v}_{ref}$ l'erreur en position et en vitesse respectivement. Nous calculons la force désirée grâce à un régulateur PD de la position et de la vitesse avec deux termes d'anticipation (*feed-forward*) sur l'accélération et la gravité [43]:

$$F_{des} = -\mathbf{K}_p\mathbf{e}_p - \mathbf{K}_v\mathbf{e}_v + mg\mathbf{z}_W + m\boldsymbol{a}_{ref} \tag{3.35}$$

où $\mathbf{K}_p$ et $\mathbf{K}_v$ sont des matrices de gains définies positives. Il est à noter que nous supposons $\| \mathbf{F}_{des} \| \neq 0$.
Projeter cette force sur l'axe $\mathbf{z}_B$ du quadrirotor nous donne la poussée nécessaire qu'il faut appliquer:

$$T_{des} = \mathbf{F}_{des} \cdot \mathbf{z}_B. \tag{3.36}$$

Ensuite, il nous faut obtenir la matrice d'orientation désirée $\mathbf{R}_{des}$ ($\mathbf{R}_{WB_{des}}$). Il suffit d'observer en premier lieu que l'axe $\mathbf{z}_B$ du quadrirotor doit être colinéaire avec la force désirée:

$$\mathbf{z}_{B,des} = \frac{\mathbf{F}_{des}}{\| \mathbf{F}_{des} \|} \tag{3.37}$$

En considérant le vecteur unitaire $\mathbf{x}_{C,des} = [\cos\psi_{ref} \ \sin\psi_{ref} \ 0]$ et on observant la Figure 3.2 on pose:

$$\mathbf{y}_{B,des} = \frac{\mathbf{z}_{B,des} \times \mathbf{x}_{C,des}}{\| \mathbf{z}_{B,des} \times \mathbf{x}_{C,des} \|} \tag{3.38}$$

$$\mathbf{x}_{B,des} = \mathbf{y}_{B,des} \times \mathbf{z}_{B,des} \tag{3.39}$$

Finalement:

$$\mathbf{R}_{des} = [\mathbf{x}_{B,des} \ \mathbf{y}_{B,des} \ \mathbf{z}_{B,des}] \tag{3.40}$$

Cette solution est valable à condition de ne pas rencontrer la singularité $\| \mathbf{z}_{B,des} \times \mathbf{x}_{C,des} \| = 0$. En pratique, s'approcher de cette singularité résulte en des changements majeurs dans les vecteurs unitaires calculées. Pour résoudre ce problème, il faut calculer $\mathbf{y}_{C,des} = [-\sin\psi_{ref} \ \cos\psi_{ref} \ 0]$ pour en déduire $\mathbf{x}_{B,des}$ ensuite $\mathbf{y}_{B,des}$ comme précédemment et voir laquelle des deux solutions s'approche le plus de l'orientation précédente.

Régulation de l'orientation

Le but est d'utiliser les couples aérodynamiques η pour commander l'orientation désirée $\mathbf{R}_{des}$ ou directement calculée par un planificateur de trajectoire $\mathbf{R}_{ref}$.
L'approche que nous allons décrire recherche la stabilité globale en minimisant la distance entre l'orientation du quadrirotor et l'orientation désirée sur $SO(3)$ (l'espace orthogonale des rotations 3D) [129].

Calcul de $\boldsymbol{\omega}_{des}$

Par soucis de clarté, nous désignerons par $\boldsymbol{\omega}$ la vitesse angulaire exprimée dans le repère du quadrirotor ${}_B\boldsymbol{\omega}_{WB}$.
Définissons l'erreur matricielle:

$$\tilde{\mathbf{R}} = \mathbf{R}_{des}^T\mathbf{R} \tag{3.41}$$

Lorsque $\mathbf{R} \to \mathbf{R}_{des}$ on a $\tilde{\mathbf{R}} \to \mathbf{I}_3$.
Définissons aussi la projection antisymétrique de la matrice d'erreur $\tilde{\mathbf{R}}$:

$$\mathbb{P}(\tilde{\mathbf{R}}) \coloneqq \frac{1}{2}(\tilde{\mathbf{R}} - \tilde{\mathbf{R}}^T) \tag{3.42}$$

On peut vérifier [130] que $\mathbb{P}(\tilde{\mathbf{R}}) = sin(\theta)\, a_X$. a est l'axe de la rotation $\tilde{\mathbf{R}}$ et θ son angle.
Soit finalement vex l'opérateur inverse de la projection antisymétrique (c.à.d : $vex(\boldsymbol{\omega}_X) = \boldsymbol{\omega}$).
L'erreur que nous voulons minimiser est la suivante :

$$\begin{aligned}
\mathbf{E} &= \frac{1}{4} \parallel \mathbf{I}_3 - \tilde{\mathbf{R}} \parallel^2, \\
&= \frac{1}{4} tr((\mathbf{I}_3 - \tilde{\mathbf{R}})^T (\mathbf{I}_3 - \tilde{\mathbf{R}})), \\
&= \frac{1}{2} tr((\mathbf{I}_3 - \tilde{\mathbf{R}}))
\end{aligned} \tag{3.43}$$

Pour voir que cette erreur mesure une différence absolue entre $\tilde{\mathbf{R}}$ et $\mathbf{I}_3$, notez que [131]:

$$\begin{aligned}
\mathbf{E} &= \frac{1}{2} tr((\mathbf{I}_3 - \tilde{\mathbf{R}})), \\
&= 1 - \cos\theta, \\
&= 2 \sin^2 \frac{\theta}{2}
\end{aligned} \tag{3.44}$$

Et donc, $\mathbf{E} \to 0$ implique $\theta \to 0$ (lorsque $\theta < 180°$).
La dynamique de l'erreur $\dot{\mathbf{E}}$ est donnée par:

$$\begin{aligned}
\dot{\mathbf{E}} &= -\frac{1}{2} tr(\dot{\tilde{R}}), \\
&= -\frac{1}{2} tr(\tilde{\mathbf{R}}\, \boldsymbol{\omega}_{\boldsymbol{X}}), \text{(voir équation 3.41 et l'équation (3) de 3.34)}, \\
&= \boldsymbol{\omega}^T\, vex(P(\tilde{\mathbf{R}}))
\end{aligned} \tag{3.45}$$

On choisit alors la vitesse angulaire:

$$\boldsymbol{\omega}_{des} = -\mathbf{K}_R vex(\mathbb{P}(\tilde{\mathbf{R}})) \quad (avec\, \mathbf{K}_R > 0) \tag{3.46}$$

pour avoir:

$$\begin{aligned}
\dot{\mathbf{E}} &= -\mathbf{K}_R \parallel vex(P(\tilde{\mathbf{R}})) \parallel^2, \\
&= \mathbf{K}_R \sin^2\theta \ \text{(voir [131])}, \\
&= -4\, \mathbf{K}_R \sin^2 \frac{\theta}{2} \cos^2 \frac{\theta}{2}, \\
&= -2\, \mathbf{K}_R \cos^2 \frac{\theta}{2}\, \mathbf{E}
\end{aligned} \tag{3.47}$$

Ce choix nous assure que $\mathbf{E}$ tend exponentiellement vers 0, et que lorsque $\theta_0 < 180°$ θ tend exponentiellement vers 0 et donc $\mathbf{R}$ tend aussi exponentiellement vers $\mathbf{R}_{des}$.

Calcul de η_{des}

Pour commander la vitesse angulaire désirée $\boldsymbol{\omega}_{des}$, nous appliquons une commande linéarisante *(feedback linearization)* [132] à la dernière équation de 3.34, avec une dynamique en boucle fermée d'un système du premier ordre:

$$\boldsymbol{\eta}_{des} = \boldsymbol{\omega} \times \boldsymbol{J} \cdot \boldsymbol{\omega} + \mathbf{K}_{\Omega} \boldsymbol{J} \cdot (\boldsymbol{\omega}_{des} - \boldsymbol{\omega}) \tag{3.48}$$

En posant $\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega_{des}} - \boldsymbol{\omega}$, il est alors facile de démontrer que la fonction de Lyapunov définie positive suivante:

$$\mathcal{L} \coloneqq \mathbf{K}_R tr(\tilde{\mathbf{R}}^T \tilde{\mathbf{R}}) + \frac{1}{2} \tilde{\omega}^T \tilde{\omega} \tag{3.49}$$

est décroissante :

$$\frac{\boldsymbol{d}}{\boldsymbol{d}t} \boldsymbol{\mathcal{L}} = -\mathbf{K}_{\boldsymbol{\Omega}} \parallel \tilde{\boldsymbol{\omega}} \parallel^{\mathbf{2}} \tag{3.50}$$

En invoquant le lemme de Barbalat, avec une certaine précaution, il s'ensuit que le système est presque globalement asymptotiquement stable. [129] ont démontré qu'avec une commande similaire, la stabilité exponentielle de la dynamique de rotation est assurée lorsque $\theta < 180°$ et tout le système dynamique est stable lorsque $\theta < 90°$.
La commande non-linéaire que nous avons présenté permet des manœuvres agressifs et de grandes excursions à partir de la position de vol stationnaire, comme le montre les résultats en simulation que nous allons voir.

3.7 Simulation du système

Positionnement

Afin de de valider la modélisation effectuée ainsi que de tester la fermeture de la boucle appliquée, nous présentons dans cette partie les résultats des simulations effectuées[1]. En premier lieu nous nous intéressons au positionnement du quadrirotor à une position fixe de l'espace 3D (Figure **??** et 3.7). Un vent latéral constant de $1.0m/s$ est simulé à l'instant $t = 5s$ pour montrer la robustesse de la commande. Les capteurs sont supposés entachés d'un bruit blanc Gaussien et les couples à l'entrée sont limités proportionnellement à la limitation de la vitesse des moteurs.

[1]La simulation du modèle dynamique en boucle fermée peut-être trouvée au lien suivant.

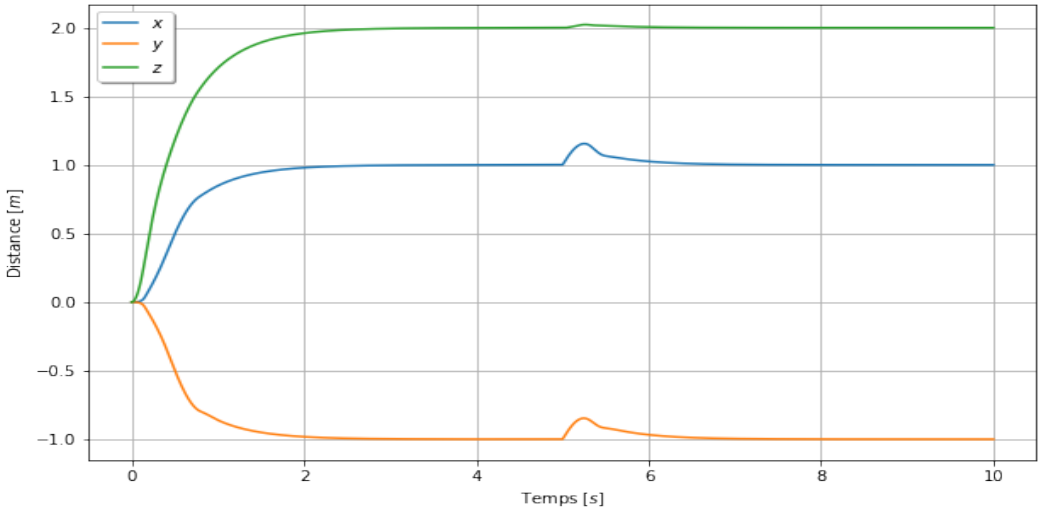


Figure 3.3: Évolution temporelle de la position du quadrirotor.

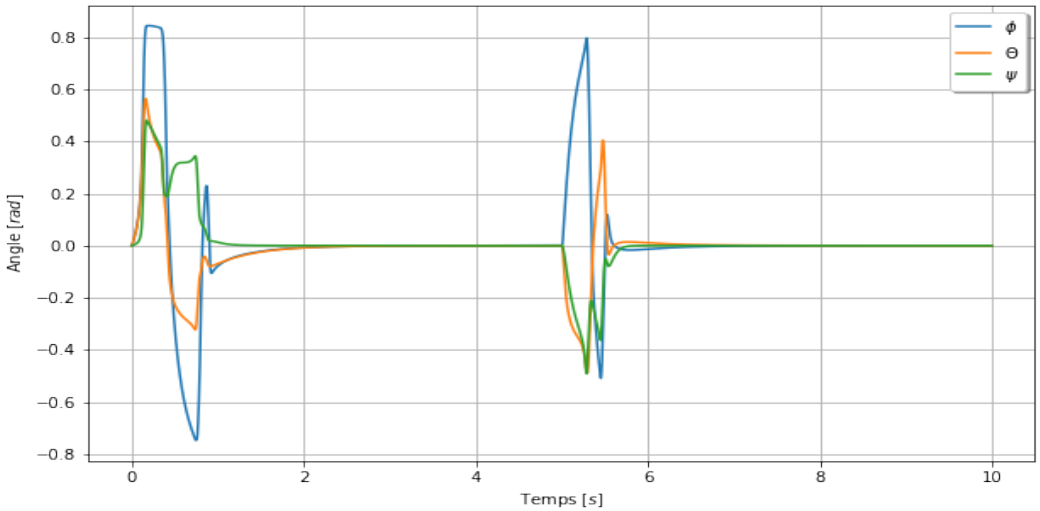


Figure 3.4: Évolution temporelle de l'orientation du quadrirotor.

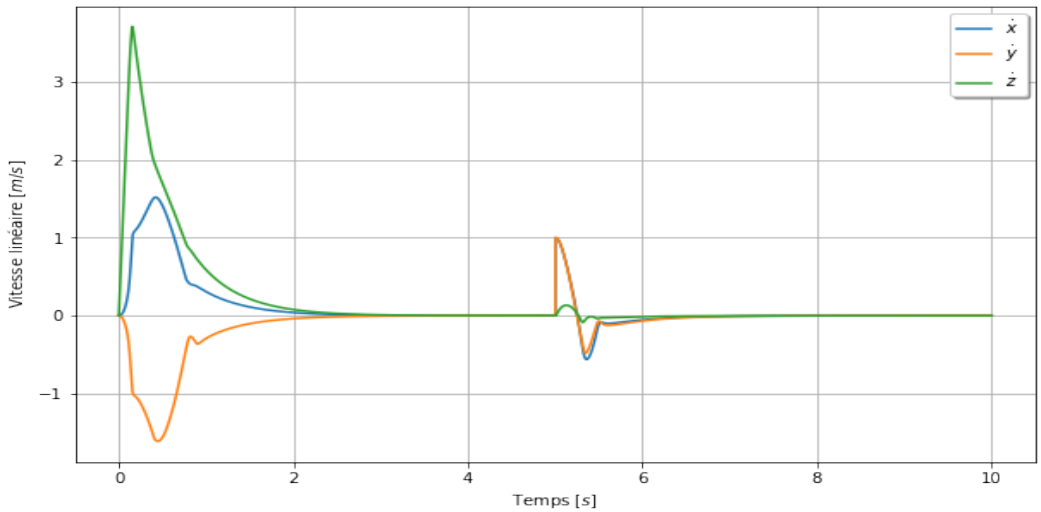


Figure 3.5: Évolution temporelle de la vitesse linéaire du quadrirotor.

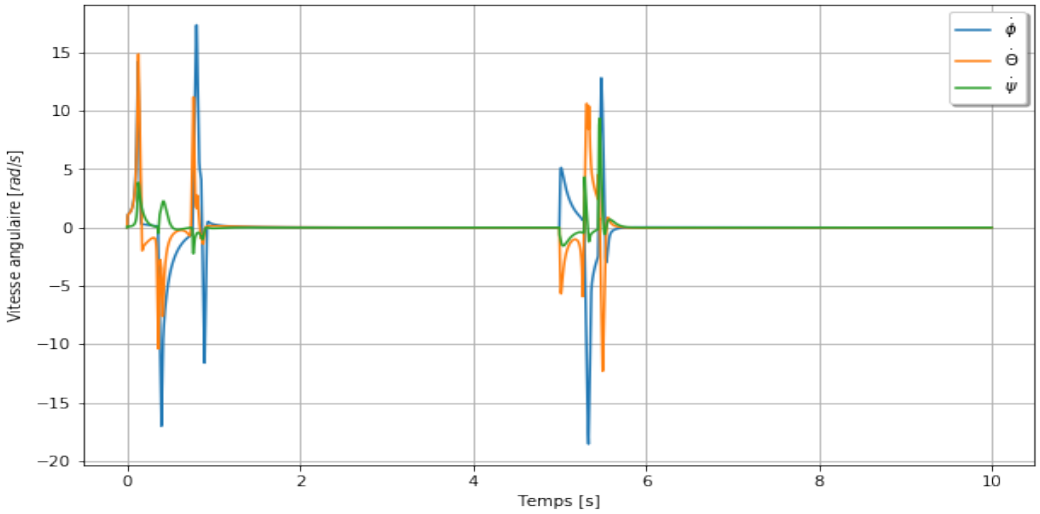


Figure 3.6: Évolution temporelle de la vitesse angulaire du quadrirotor.

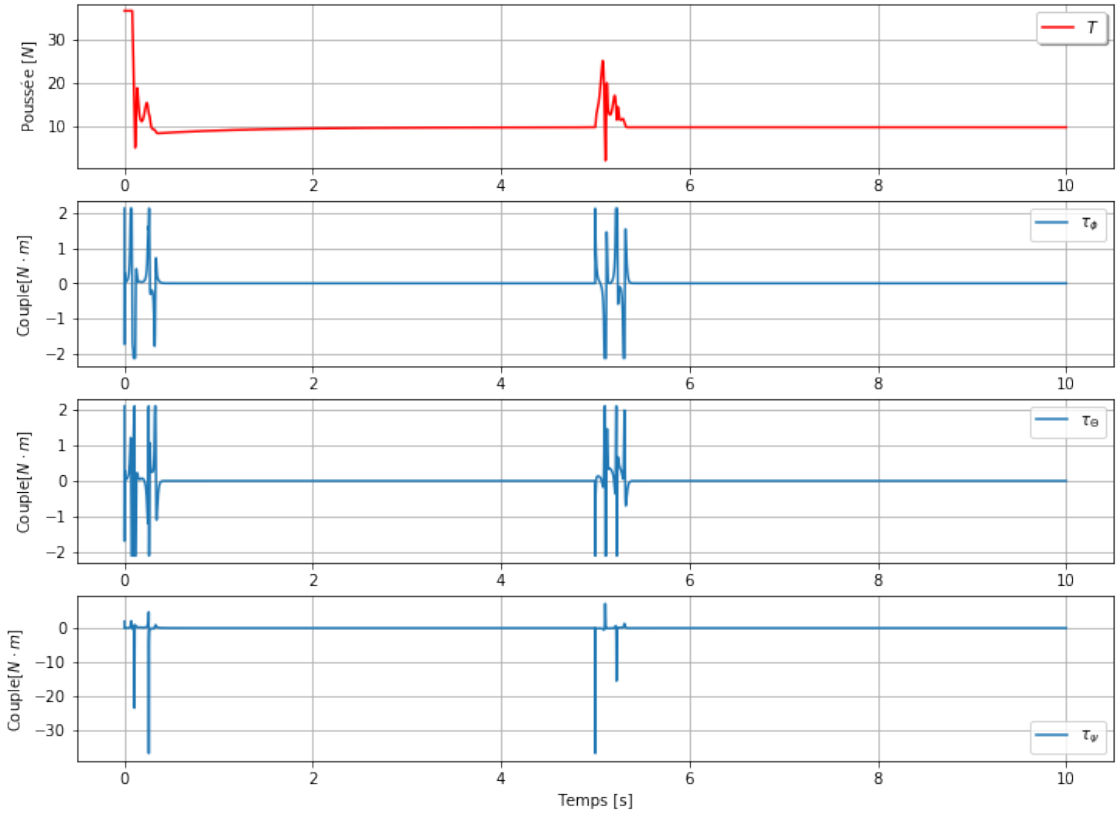


Figure 3.7: Évolution temporelle des forces appliquées au quadrirotor.

Poursuite d'une trajectoire hélicoïdale

La simulation suivante montre la capacité de la commande développée à poursuivre des références énergétiques telles qu'un hélicoïde. L'hélicoïde est définie par les équations suiv-

antes

$$\begin{aligned} x(t) &= -\sin 2t \\ y(t) &= \cos 2t \\ z(t) &= t \end{aligned} \tag{3.51}$$

Les résultats de simulation (figures 3.8,**??**,et **??**) montrent une poursuite de référence en moins de $2sec$, les angles de roulis et de tangage dépassent par moments les 45° sans pour autant déstabiliser le système.

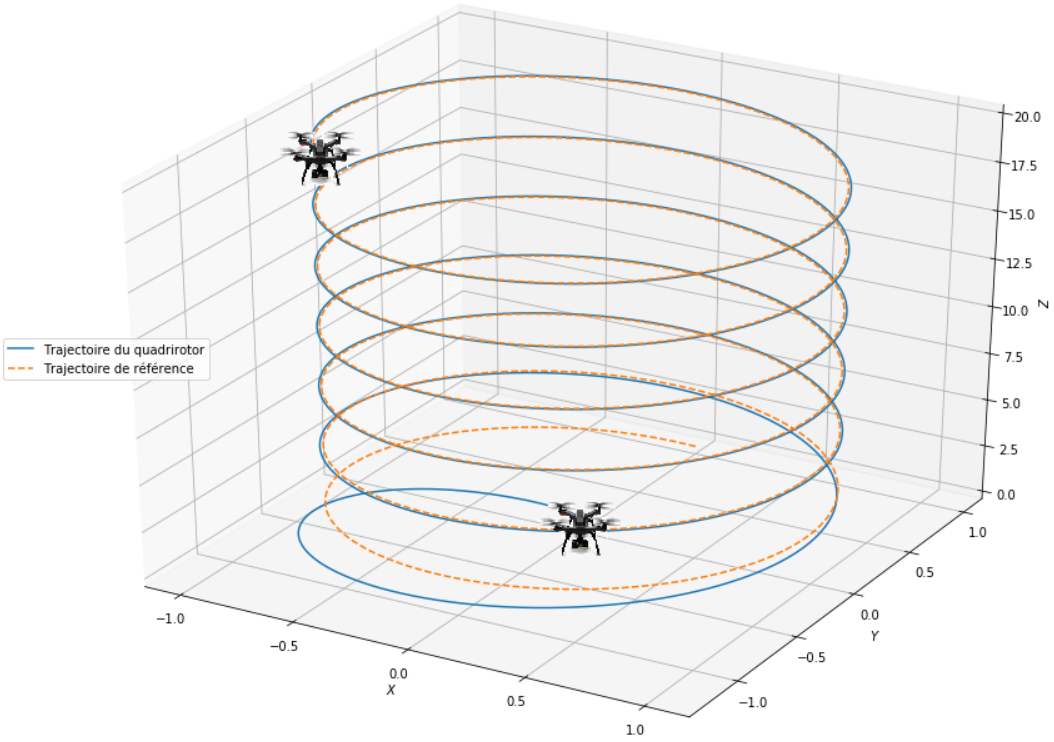


Figure 3.8: Trajectoire 3D du quadrirotor.

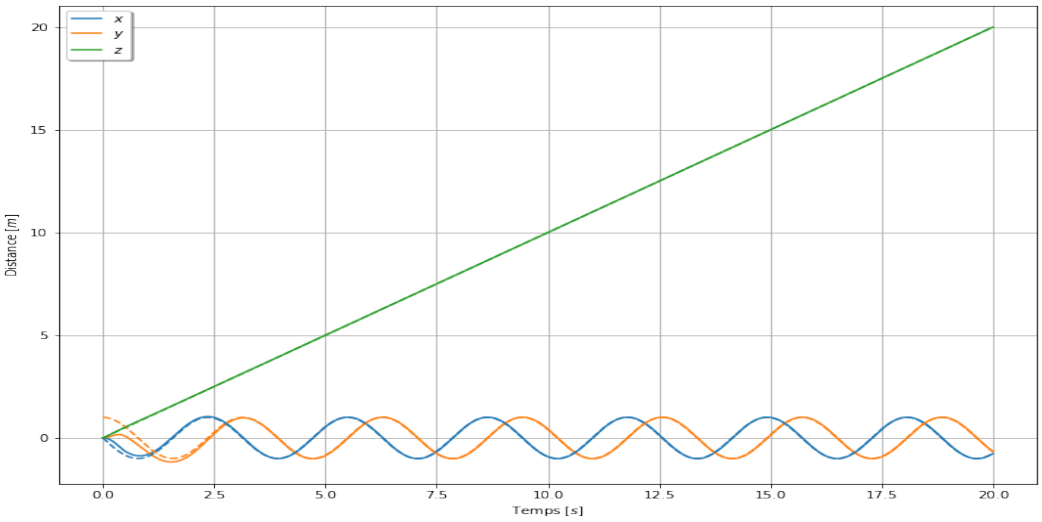


Figure 3.9: Évolution temporelle de la position du quadrirotor.

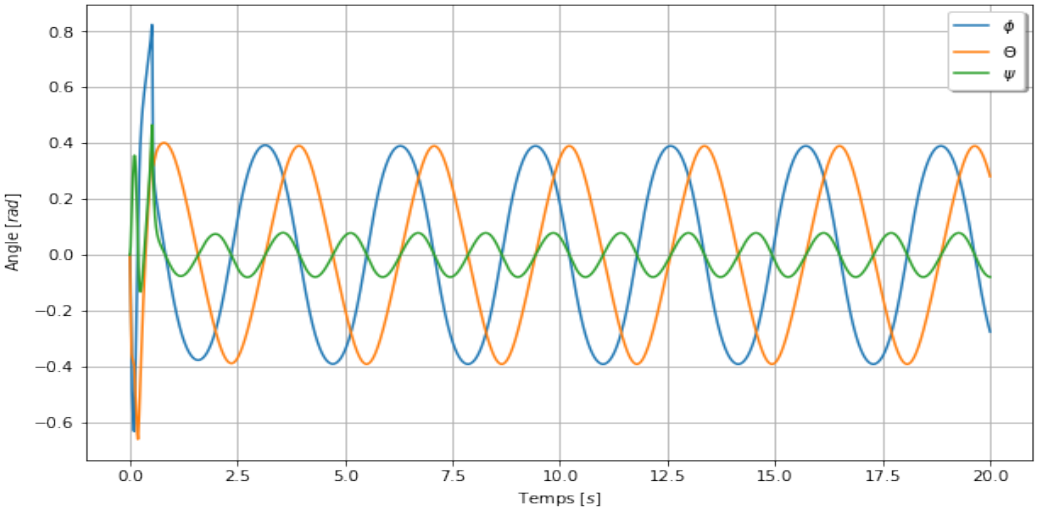


Figure 3.10: Évolution temporelle de l'orientation du quadrirotor.

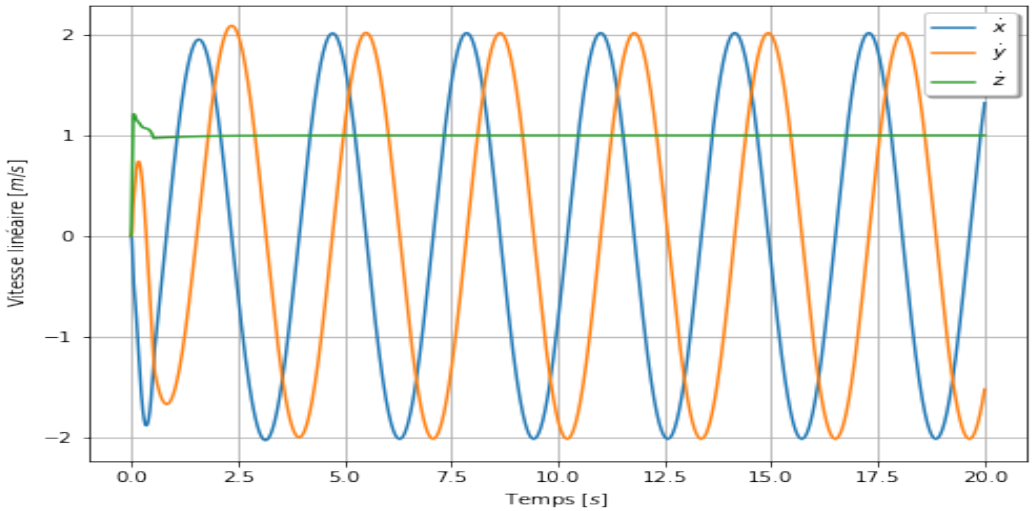


Figure 3.11: Évolution temporelle de la vitesse linéaire du quadrirotor.

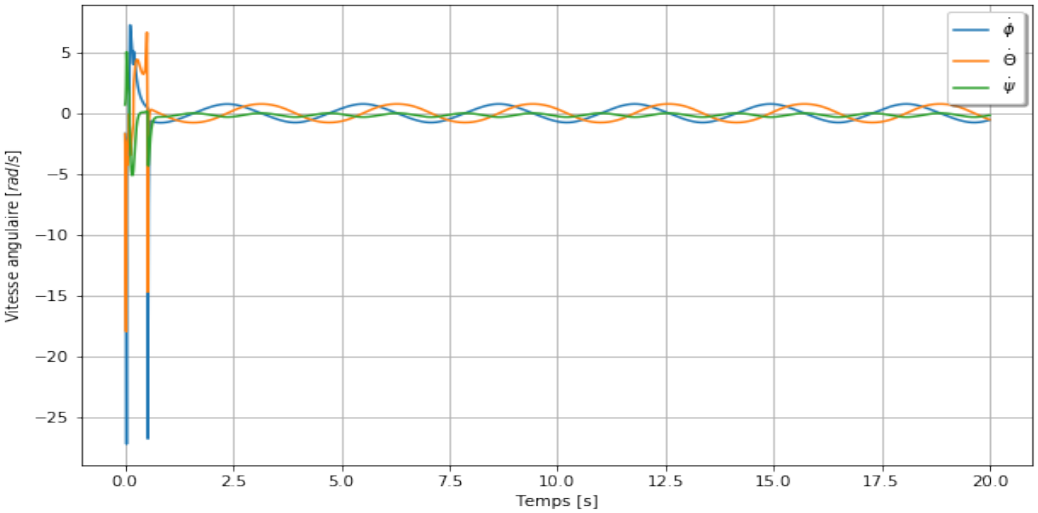


Figure 3.12: Évolution temporelle de la vitesse angulaire du quadrirotor.

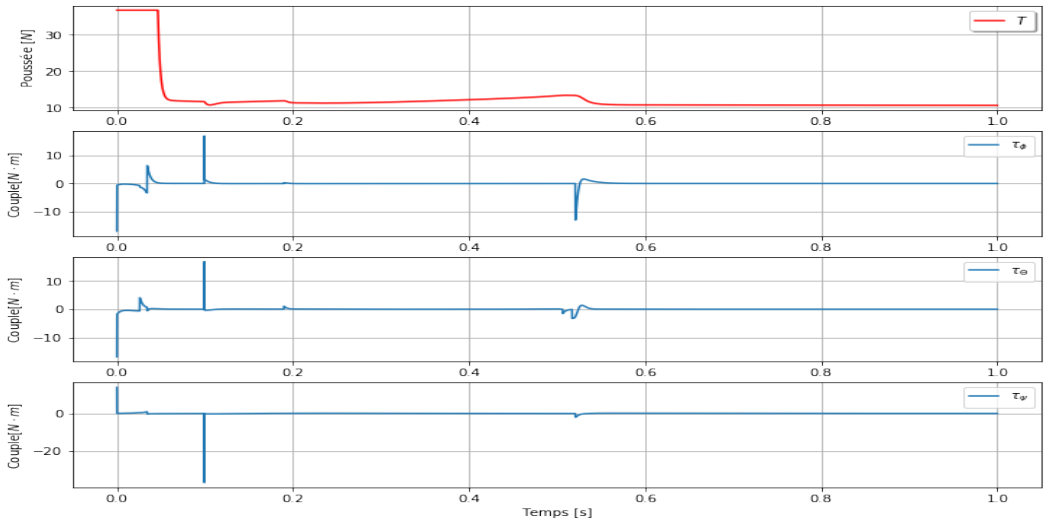


Figure 3.13: Évolution temporelle des forces appliquées au quadrirotor.

Poursuite d'une trajectoire définie par des points de passage

Étant donné une courbe $[x(t), y(t), z(t)]$ (et $\psi(t)$) que nous voulons suivre, nous pouvons calculer la force et le couple nécessaires pour déplacer le quadrirotor le long de celle-ci. Pour ce faire, il faut prendre la 4$^{\text{ème}}$ dérivée de cette courbe comme le montre l'équation d'état du système 3.33.

En effet, la deuxième dérivée de la position est liée à l'orientation, tandis que la deuxième dérivée de l'orientation est liée à la commande. Ainsi, le couple est directement lié à la 4$^{\text{ème}}$ dérivée de position - la *rupture* (*snap* en anglais). Toute discontinuité dans le *snap* nécessiterait un couple infini, nous voulons donc que cette 4$^{\text{ème}}$ dérivée soit *lisse*.

[32] ont montré que lorsqu'on est proche de l'équilibre en vol stationnaire, les entrées de commande du quadrirotor sont une fonction linéaire du *snap*. Cela conduit naturellement à l'idée qu'une bonne trajectoire minimise le carré total du *snap*(comme une approximation pour minimiser l'entrée totale de la commande).

De manière plus formelle, considérons le cas unidimensionnelle où nous voulons suivre une trajectoire $x(t)$ telle que:

$$\begin{aligned} t &= [t_0\, t_1\, t_2 \ldots t_m\,]^T, \\ x &= [x_0\, x_1\, x_2 \ldots x_m\,]^T \end{aligned} \tag{3.52}$$

Nous voulons minimiser:

$$\min_{x(t)} \int_{t_0}^{t_1} (\dddot{x}^2) dt + \ldots + \int_{t_m-1}^{t_m} (\dddot{x}^2) dt \tag{3.53}$$

Nous appliquons donc Euler-Lagrange à chaque terme ($\mathcal{L}(x, \dot{x}, \ddot{x}, \dddot{x}, \ddddot{x}) = \ddddot{x}$):

$$\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) + \frac{d^2}{dt^2}\left(\frac{\partial \mathcal{L}}{\partial \ddot{x}}\right) - \frac{d^3}{dt^3}\left(\frac{\partial \mathcal{L}}{\partial \dddot{x}}\right) + \frac{d^4}{dt^4}\left(\frac{\partial \mathcal{L}}{\partial \ddddot{x}}\right) = 0 \tag{3.54}$$

Seul le dernier terme n'étant pas nul nous obtenons le résultat suivant:

$$\begin{cases} x_1^{(8)} = 0\,, t \in [t_0, t_1] \\ x_2^{(8)} = 0\,, t \in [t_1, t_2] \\ \dots \\ x_m^{(8)} = 0\,, t \in [t_{m-1}, t_M] \end{cases} \tag{3.55}$$

et donc:

$$x(t) = \begin{cases} x_1(t) = c_{1,7}t^7 + c_{1,6}t^6 + \dots + c_{1,1}t + c_{1,0}\,, t \in [t_0, t_1] \\ x_2(t) = c_{2,7}t^7 + c_{2,6}t^6 + \dots + c_{2,1}t + c_{2,0}\,, t \in [t_1, t_2] \\ \dots \\ x_m(t) = c_{m,7}t^7 + c_{m,6}t^6 + \dots + c_{m,1}t + c_{m,0}\,, t \in [t_{m-1}, t_m] \end{cases} \tag{3.56}$$

où les $8m$ constantes $c_{i,j}$ sont à déterminer en considérant les conditions aux limites et la continuité des 4 premières dérivées.

Nous avons donc appliquer cette méthode pour générer une trajectoire $\mathcal{C}^4$ avec comme spécifications:
$t_i = [0, 4, 8, 12, 16, 20]^T$ et $[x_i, y_i, z_i]^T = [[0, 0, 0]^T, [0, 0, 5]^T, [0, 5, 5]^T, [5, 5, 5]^T, [5, 0, 5]^T, [5, 0, 5]]$.
Cette trajectoire représente un carré horizontal dans l'espace 3D.

La figure 3.14 montre les trajectoires générées grâce à cette technique. On voit bien que les fonctions obtenues sont *lisses* ainsi que leurs dérivées jusqu'au *snap*, tout en étant raisonnablement limitées.
La figure 3.15 montre comment le quadrirotor parvient à parfaitement suivre cette trajectoire et la figure 3.16 nous permet de visualiser les signaux importants du système.

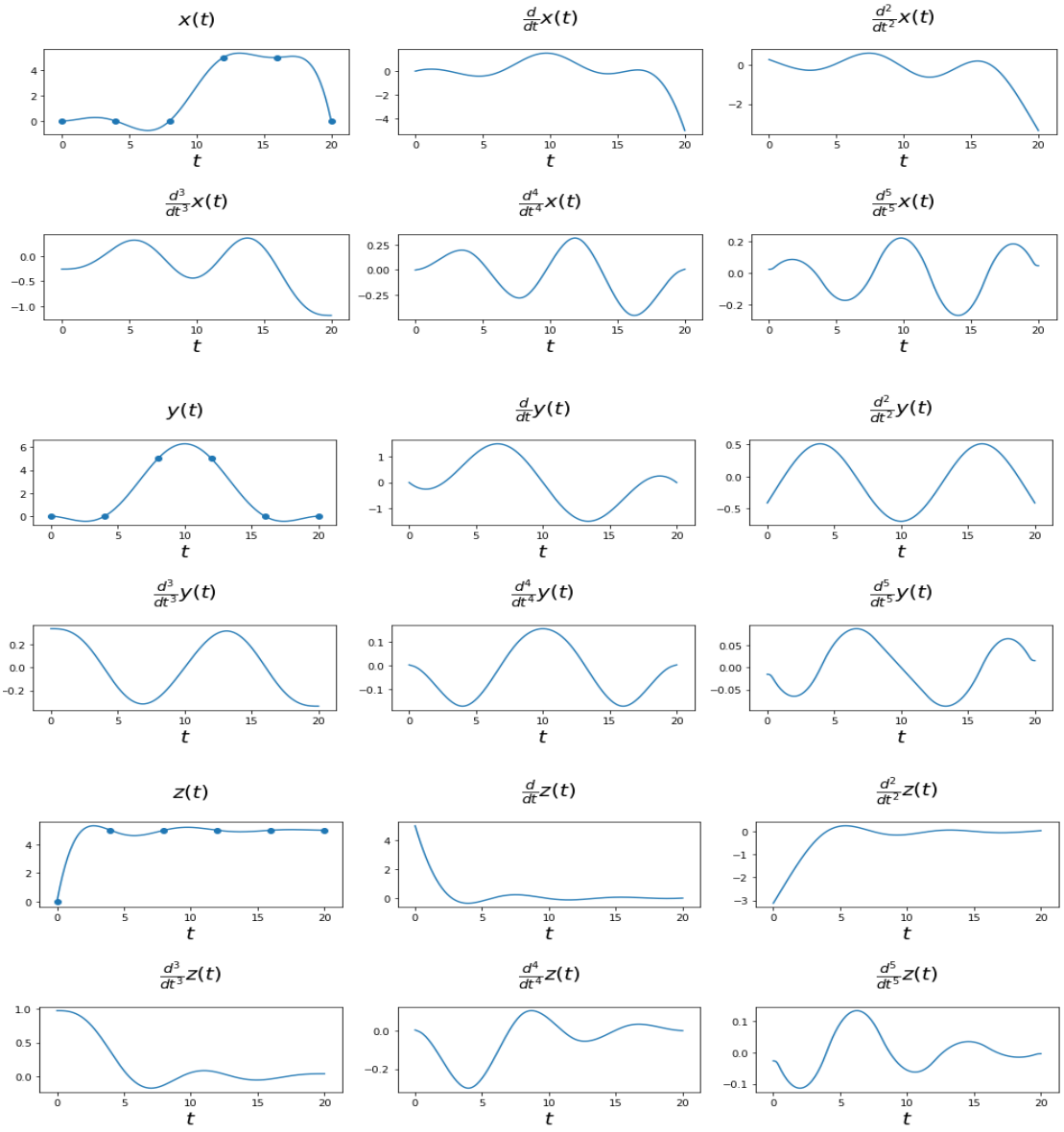


Figure 3.14: Trajectoires minimum-snap générées.

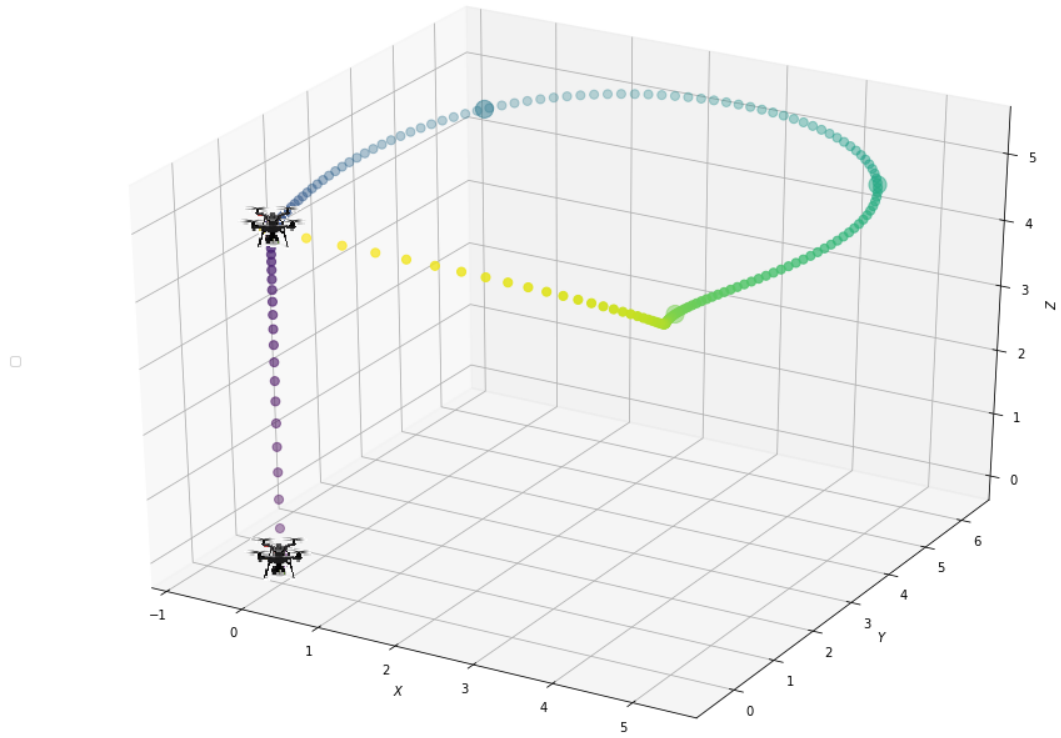


Figure 3.15: Trajectoire 3D du quadrirotor.

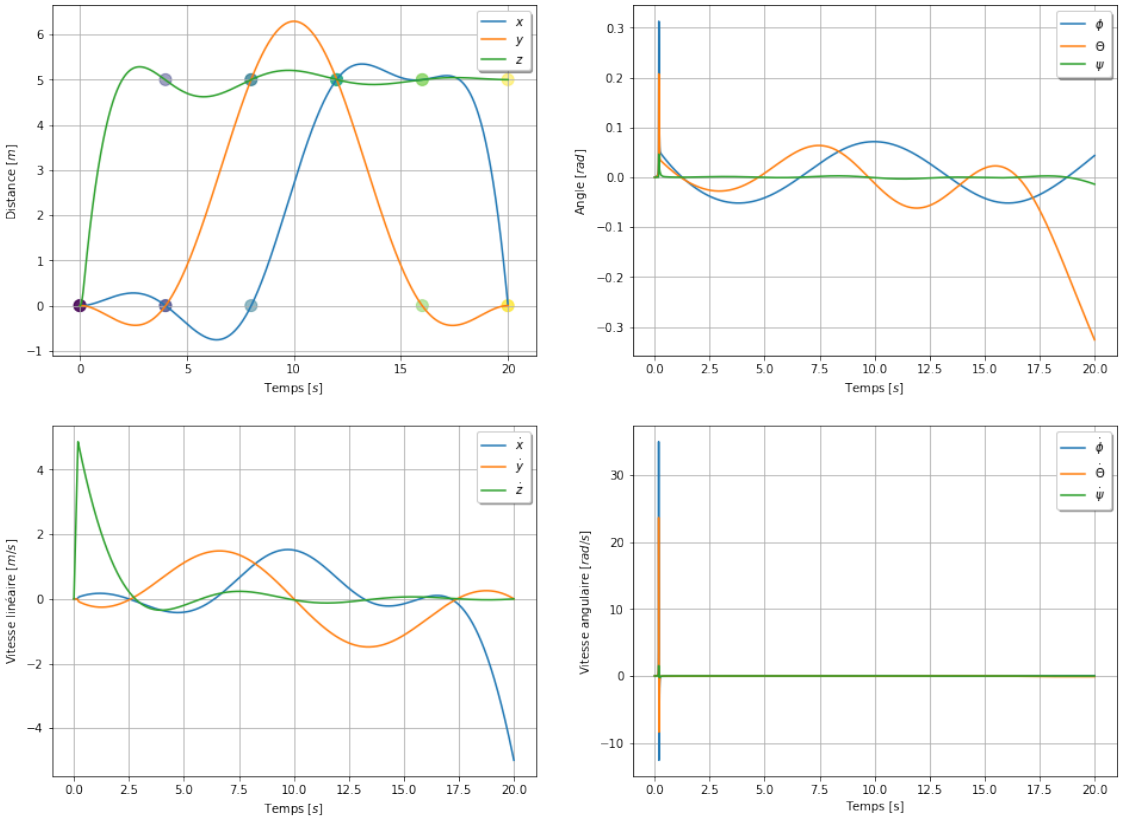


Figure 3.16: Évolution temporelle de l'état du quadrirotor.

3.8 Conclusion

Dans ce chapitre, nous avons établi le modèle cinématique et dynamique du quadrirotor. Une commande non-linéaire qui a la particularité de permettre des vols agiles a été élaborée et a permis de valider par simulation en boucle fermée le modèle ainsi obtenu.

Malgré les nombreuses applications des quadrirotors, leur plein potentiel, notamment en termes d'autonomie et d'agilité, n'a pas encore été exploité. Rendre les quadrirotors plus autonomes et plus agiles présente l'avantage de nécessiter moins d'opérateurs et d'accomplir les tâches plus rapidement, ce qui les rend plus utiles et augmente leur rentabilité. Cependant, rendre les plateformes quadrirotor plus agiles, par exemple en les rendant plus petites, les rend également plus difficiles à contrôler en raison d'une dynamique plus rapide. De plus, en vol agile avec des vitesses et des accélérations élevées, les effets aérodynamiques, difficiles à modéliser et à intégrer dans la commande, deviennent pertinents pour les caractéristiques de vol des quadrirotors. Négliger ces effets n'affecte pas leurs performances en vol stationnaire, mais les réduit significativement en vol agile.

Chapitre 4

Estimation de l'état

4.1 Introduction

Un aspect critique dans la mise en œuvre de quadrirotors réels est de fournir de bonnes estimations de l'état du véhicule. Les principales estimations de l'état requises pour la commande du véhicule sont associées à la dynamique du corps rigide et donc l'altitude, l'orientation, la vitesse angulaire et la vitesse linéaire.

L'instrumentation omniprésente transportée par tout véhicule aérien est une unité de mesure inertielle (IMU: *Inertial Measurement Unit*) pour mesurer l'orientation souvent complétée par une forme de mesure de la hauteur, acoustique, infrarouge, barométrique ou laser.

Dans le contexte du Drone Racing, et dans des environnements intérieurs et encombrés de manière générale, les signaux GPS ne sont pas disponibles. De même pour les systèmes de capturer de mouvement tels que VICON [133] ou Optitrack [134]. De ce fait, un choix populaire ces dernières années est d'utiliser une caméra pour estimer la pose du quadrirotor grâce à l'odométrie visuelle (VO: Visual Odometry) ou la localisation et cartographie visuelles et simultannées (V-SLAM: Visual-Simultaneous localization and mapping).

Nous présenterons dans ce chapitre la solution que nous avons retenue à savoir l'utilisation d'une centrale inertielle combiné avec deux caméras pour la vision stéréoscopique en utilisant l'algorithme ORB-SLAM2.

4.2 Estimation de l'orientation à l'aide de l'IMU

La récente prolifération de la technologie micro-électro-mécanique (MEMS : *Micro electro-mechanical systems*) a conduit au développement d'une gamme d'unités de mesure inertielle à faible coût et de poids léger. Le signal de sortie des IMU à faible coût se caractérise toutefois par des signaux à faible résolution soumis à des niveaux de bruit élevés ainsi que par des biais variant dans le temps. Les signaux bruts doivent être traités pour reconstruire les estimations de l'orientation et les mesures de vitesse angulaire corrigées de leurs biais. Aussi, les algorithmes d'estimation doivent être peu gourmand en mémoire et temps de calcul pour pouvoir être déployés sur les processeurs embarqués.
l'IMU permet d'obtenir l'orientation du quadrirotor à haute fréquence (typiquement 1 à $2Khz$), ceci est primordiale car cette information est nécessaire pour assurer la stabilité du

véhicule. La centrale inertielle est fixée au centre de gravité du quadrirotor. Les mesures disponibles sont les vitesses angulaires à l'aide du gyroscope et les accélérations linéaires à l'aide de l'accéléromètre.
Dans ce qui suit nous considérons le repère de l'IMU confondu avec celui du quadrirotor **B** (voir figure 3.2). L'orientation de ce repère par rapport au repère terrestre **W** est notée $\mathbf{R} = \mathbf{R}_{Wb}$ précédemment défini dans 3.7.

Le gyroscope

Le gyroscope mesure la vitesse de rotation de **B** par rapport à **W**, exprimée dans **B**. Le modèle de mesure du gyroscope est le suivant:

$$\omega_g = {}_B\boldsymbol{\omega}_{WB} + b_g + \mu_g \in \boldsymbol{R}^3 \tag{4.1}$$

${}_B\boldsymbol{\omega}_{WB}$ représente la vitesse angulaire *réelle*. μ_g est un bruit de mesure et b_g est un biais constant (ou lentement variable). Le gyroscope permet donc d'obtenir les vitesses angulaires; mais aussi les angles d'Euler décrivant la rotation du quadrirotor en intégrant ces vitesses (En transformant d'abord ces vitesses à l'aide de la relation 3.15, car les mesures obtenues à partir du gyroscope représente la vitesse angulaire dans le repère **B** et non la dérivée des angles d'Euler).
En notant ΔT le temps d'échantillonnage et $\hat{x}$ l'estimé de x, la procédure à suivre est la suivante:

$$\begin{aligned}
\begin{bmatrix} \dot{\phi}_g \\ \dot{\theta}_g \\ \dot{\psi}_g \end{bmatrix} &\coloneqq \begin{bmatrix} 1 & t(\hat{\theta})s(\hat{\phi}) & t(\hat{\theta})c(\hat{\phi}) \\ 0 & c(\hat{\phi}) & -s(\hat{\phi}) \\ 0 & s(\hat{\phi})/c(\hat{\theta}) & c(\hat{\phi})/c(\hat{\theta}) \end{bmatrix} \omega_g \\
\begin{bmatrix} \hat{\phi}_g \\ \hat{\theta}_g \\ \hat{\psi}_g \end{bmatrix} &\coloneqq \begin{bmatrix} \hat{\phi}_g \\ \hat{\theta}_g \\ \hat{\psi}_g \end{bmatrix} + \Delta T \begin{bmatrix} \dot{\phi}_g \\ \dot{\theta}_g \\ \dot{\psi}_g \end{bmatrix}
\end{aligned} \tag{4.2}$$

Cependant, en raison du bruit de mesure, du biais et de l'intégration dans le temps, les données du gyroscope ne sont fiables qu'à court terme et sont sujettes à la dérive à long terme.

l'accéléromètre

On désigne par $\dot{v}$ l'accélération du repère mobile **B** par rapport à **W**, exprimé dans **W**. Un accéléromètre idéal, attaché au repère **B**, mesure l'accélération linéaire instantanée de **B** *moins* le champs d'accélération gravitationnel g_0 (g_0 exprimé dans **W**), et donne une mesure exprimé dans **B**. S'en suit le modèle de mesure de l'accéléromètre :

$$a_{imu} = \mathbf{R}^T(\dot{v} - g_0) + b_{imu} + \mu_{imu} \in \boldsymbol{R}^3 \tag{4.3}$$

b_{imu} et μ_{imu} représentent le biais et le bruit de mesure respectivement.
Au repos ou pour de faibles accélérations, la gravité domine la valeur de a_{imu}, il est donc très courant d'utiliser [135]:

$$\frac{a_{imu}}{|a_{imu}|} \approx -\mathbf{R}^T\mathbf{z}_W \tag{4.4}$$

comme approximation du vecteur $\mathbf{z}_W$ exprimé dans le repère mobile $\mathbf{B}$. Cette relation s'écrit avec les angles d'Euler comme suit:

$$\frac{a_{imu}}{|a_{imu}|} \approx \begin{bmatrix} s(\theta) \\ -c(\theta)s(\phi) \\ -c(\theta)c(\phi) \end{bmatrix} \tag{4.5}$$

En limitant l'angle de tangage θ entre $-\pi/2$ et $\pi/2$, et l'angle de roulis ϕ entre $-\pi$ et π pour éviter les solutions doubles comme c'est souvent le cas en aéronautique; On peut retrouver de manière unique ces angles en utilisant les deux relations:

$$\phi = atan2(-a_{imu,y}, -a_{imu,z}) \tag{4.6}$$

$$\theta = atan(\frac{a_{imu,x}}{\sqrt{a_{imu,y}^2 + a_{imu,z}^2}}) \tag{4.7}$$

Puisque l'équation 4.5 ne fait pas apparaître l'angle de lacet ψ, on ne peut donc pas calculer cet angle directement à partir de l'accéléromètre. L'absence de dépendance par rapport à l'angle de lacet ψ est facile à comprendre physiquement puisque la première rotation est en lacet autour de l'axe $\mathbf{z}_W$ qui est initialement aligné avec le champ gravitationnel, et l'accéléromètre est insensible à n'importe quelle rotation autour de cet axe.
Les accéléromètres sont très sensibles aux vibrations et, montés sur une plate-forme robotique aérienne typique, ils nécessitent un filtrage mécanique et/ou électrique passe-bas important pour être fiables. La plupart des systèmes avioniques intègrent un filtre anti-repliement analogique sur un accéléromètre avant que le signal ne soit échantillonné.

Le filtre complémentaire

Les filtres complémentaires permettent de fusionner plusieurs mesures bruitées indépendantes d'un même signal ayant des caractéristiques spectrales complémentaires [136].L'idée derrière le filtre complémentaire est de prendre les signaux lents de l'accéléromètre (avec un filtre passe-bas) et les signaux rapides d'un gyroscope (avec un filtre passe-haut) et de les combiner. Soient par exemple les deux mesures $y_1 = x + \mu_1$ et $y_2 = x + \mu_2$ d'un signal x avec μ_1 un bruit à haute fréquence et μ_2 un bruit à basse fréquence. Soit aussi le filtre passe-bas $F_1(s)$ et le filtre passe-haut $F_2(s)$ tels que $F_1(s) + F_2(s) = 1$. l'estimé de x filtré est donné par

$$\begin{aligned} \hat{X}(s) &= F_1(s)Y_1(s) + F_2(s)Y_2(s), \\ &= X(s) + F_1(s)\mu_1(s) + F_2(s)\mu_2(s) \end{aligned} \tag{4.8}$$

le signal $X(s)$ passe complètement à travers le filtre tandis que les bruits sont filtrés en passe-haut passe-bas au choix.

Nous considérons dans ce qui suit le cas unidimensionnelle pour par exemple estimer l'angle de roulis ϕ en fusionnant les données du gyroscope et de l'accéléromètre. La même procédure s'applique à l'angle de tangage θ, mais pas à l'angle de lacet ψ qui sera estimé à l'aide du SLAM dans la section suivante 4.3.
les modèles de mesure de l'accéléromètre et du gyroscope de l'angle de roulis sont donnés respectivement par:

$$\begin{aligned} \phi_a &= \phi + \mu_{\phi,a} \\ \dot{\phi}_g &= \dot{\phi} + \mu_{\phi,g} + b_{\phi,g} \end{aligned} \tag{4.9}$$

En choisissant $F_1(s) = \frac{C(s)}{C(s)+s}$ et $F_1(s) = 1 - F_1(s) = \frac{s}{C(s)+s}$ comme filtres passe-bas et passe-haut respectivement, nous obtenons l'angle de roulis estimé suivant:

$$\begin{aligned}\hat{X}(s) &= F_1(s)\phi_a(s) + F_2(s)\frac{\dot{\phi}_g}{s} \\ \hat{X}(s) &= X(s) + F_1(s)\mu_{\phi,a}(s) + \frac{\mu_{\phi,g} + b_{\phi,g}}{C(s)+s}\end{aligned} \tag{4.10}$$

En pratique, la structure du filtre est implémentée en utilisant la structure de sensibilité complémentaire d'un système de régulation linéaire soumis à des perturbations de charge comme le montre la figure 4.1.

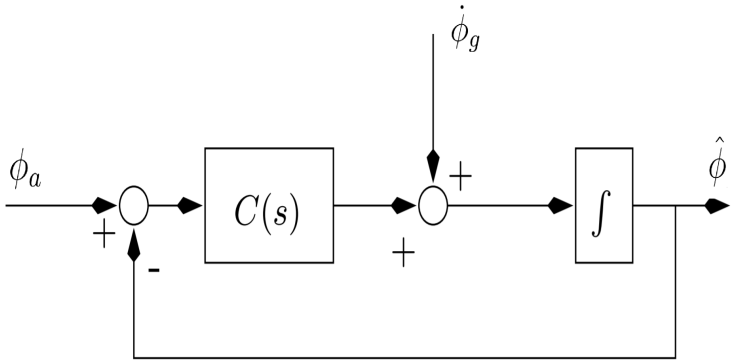


Figure 4.1: Schéma fonctionnel d'un filtre complémentaire classique.

Le choix le plus simple pour $C(s)$ est un gain proportionnel K_p. Dans ce cas la dynamique en boucle fermée de l'estimateur est la suivante:

$$\dot{\hat{\phi}} = \dot{\phi}_g + K_p(\phi_a - \hat{\phi}) \tag{4.11}$$

Cependant, si le biais $b_{\phi,g}$ est constant, un choix naturel de $C(s)$ est un proportionnel intégral $C(s) = K_p + \frac{K_i}{s}$. Dans ce cas nous obtenons la dynamique suivante en représentation d'état:

$$\begin{aligned}\dot{\hat{\phi}} &= \dot{\phi}_g + K_p(\phi_a - \hat{\phi}) - \hat{b}_{\phi,g} \\ \dot{\hat{b}}_{\phi,g} &= -K_i(\phi_a - \hat{\phi})\end{aligned} \tag{4.12}$$

Le signe $(-)$ indique que l'état $\hat{b}_{\phi,g}$ annulera le biais de $\dot{\phi}_g$.
En considérant la fonction de Lyapunov:

$$\mathcal{L} = \frac{1}{2}\left|\phi - \hat{\phi}\right|^2 + \frac{1}{2K_i}\left|b_{\phi,g} - \hat{b}_{\phi,g}\right|^2 \tag{4.13}$$

on a dans ce cas:

$$\frac{d}{dt}\mathcal{L} = -K_p\left|\tilde{\phi}\right|^2 - \mu_{\phi,g}\tilde{\phi} + \mu_{\phi,a}(\tilde{b} - K_i\tilde{\phi}) \tag{4.14}$$

avec $\tilde{\phi} = \phi - \hat{\phi}$ et $\tilde{b}_{\phi,g} = b_{\phi,g} - \hat{b}_{\phi,g}$.
En l'absence de bruits, la méthode directe de Lyapunov permet de prouver la convergence de $\hat{\phi}$, tandis que le principe d'invariance de Lasalle permet de montrer que $\hat{b}_{\phi,g} \to b_{\phi,g}$

4.3 Estimation de la pose par vision stéréoscopique

L'utilisation de la vision pour l'estimation de la pose (position et orientation) a été largement traitée en robotique. Une seule caméra ne suffit pas pour obtenir la profondeur (du moins pas directement), les caméras RGB-D (RGB-profondeur) et les caméras stéréoscopiques ont souvent été utilisées pour palier à ce problème.

Nous avons choisi d'utiliser la vision stéréoscopique, car les caméras RGB-D sont des capteurs actifs et réduisent donc le temps de vol du quadrirotor à cause de leur consommation d'énergie [137].

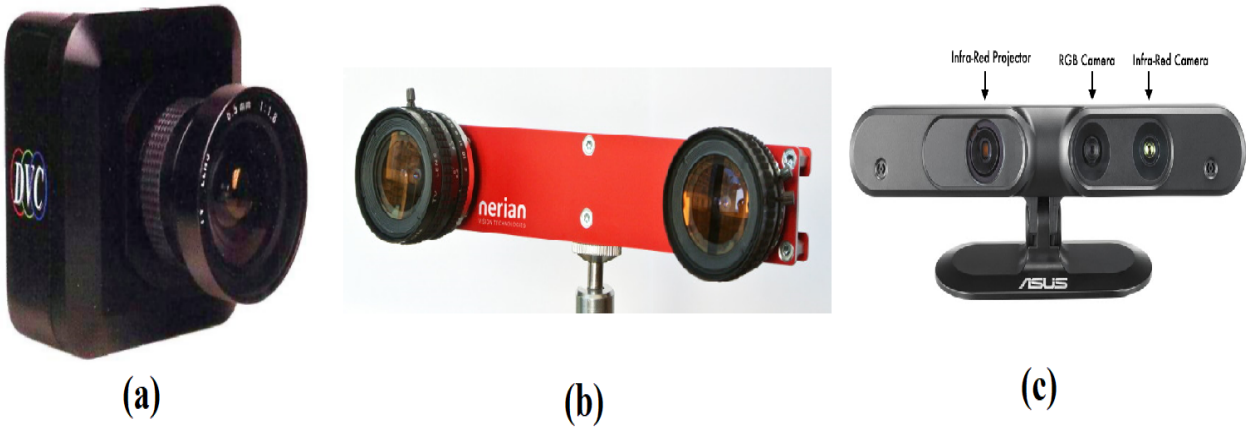


Figure 4.2: Différents types de caméra.(a) monoculaire,(b) stéréoscopique,(c) RGB-D

Puisque nous nous intéressons à l'implémentation d'une solution open-source déjà existante, à savoir ORB-SLAM2 [65], nous ne nous couvrirons dans cette section que les fondements de la vision par ordinateurpour pouvoir ensuite exposer de manière succincte le déroulement de l'algorithme. Plus de détails techniques sont à trouver ici [138][65].

Il est aussi important de noter que la pose ne peut-être estimée par l'algorithme qu'à une fréquence allant de 30 à 60 Hz. Cette fréquence basse risque de déstabiliser le système surtout que la boucle de régulation de l'orientation doit tourner à $\approx 1kHz$, l'orientation est donc donnée par l'IMU.

4.3.1 Notions de vision

Modélisation et calibration de la caméra

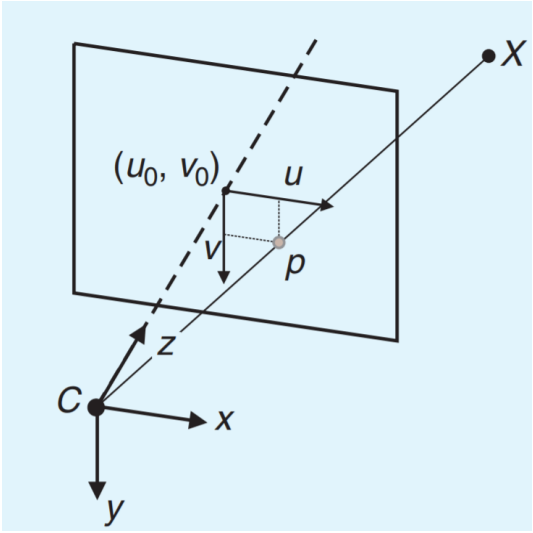


Figure 4.3: Modèle géométrique d'une caméra

Afin de projeter les coordonnées du monde en 3D sur un plan image 2D, nous avons besoin du modèle de projection de la caméra. Le modèle le plus couramment utilisé est celui du sténopé (*Pinhole camera* en Anglais)[139]. L'image est formée par l'intersection des rayons lumineux des objets à travers le centre de la lentille (centre de projection), avec le plan focal (Figure 4.3).
Soit $X = [x, y, z]^T$ un point dans la scène (espace du champs de vision de la caméra) exprimé dans le repère de la caméra et $p = [u, v]^T$ sa projection sur le plan image mesurée en pixels. La correspondance entre le monde 3-D et l'image 2-D est donnée par la projection perspective suivante:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\,X = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4.15}$$

où λ est le facteur d'échelle, α_u et α_v sont les distances focales et u_0, v_0 les coordonnées images du centre de projection. Ces paramètres sont appelés les *paramètres intrinsèques* de la caméra. Quand l'angle de vue d'une caméra est supérieure à 45°, les effets de la distorsion radiale peuvent devenir visibles et peuvent-être modélisés par un polynom de secon -(ou plus)- ordre.
Le but de la calibration de la caméra est de mesurer ses paramètres *intrinsèques* et *extrinsèques*. Dans le cas stéréoscopique les paramètres extrinsèques sont la position et orientation relatives de chaque caméra. La méthode la plus populaire est d'utiliser un plan en forme de damier [1] . La calibration de la caméra est devenue une procédure standard avec des outils tels que le *calibration toolbox* de MATLAB [140], ou la fonction *calibration* d'OpenCV [141].

Il est impossible de retrouver la profondeur d'un point 3-D connaissant seulement sa projection dans le plan image, et c'est là l'une des plus grosses limitations de la vision monoculaire.Or supposons que nous avons la projection d'un même point X exprimé dans le repère

[1] Exemple ici.

terrestre sur deux caméras. Connaissant les paramètres intrinsèques $(\alpha_{u,i}, \alpha_{v,i}, u_{0,i}, v_{0,i})$, extrinsèques (rotation R_i et translation T_i)de ces deux caméras, ainsi que la transformation entre celles ci on peut écrire:

$$\begin{aligned} \lambda_1 \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} &= K_1 \left(R_1 X + T_1 \right) \\ \lambda_2 \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} &= K_2 \left(R_2 X + T_2 \right) \end{aligned} \tag{4.16}$$

Il est alors possible de retrouver X, ce processus est connu sous le nom de *Triangulation*.

Extraction de caractéristique

L'extraction de caractéristiques visuelles (ou *visual features extraction* en anglais) consiste en des transformations mathématiques calculées sur les pixels d'une image numérique. Les caractéristiques visuelles permettent généralement de mieux rendre compte de certaines propriétés visuelles de l'image, utilisées pour des traitements ultérieurs.

On distingue usuellement les caractéristiques globales qui sont calculées sur toute l'image (Figure 4.4) et les caractéristiques locales qui sont calculées autour de points d'intérêt.

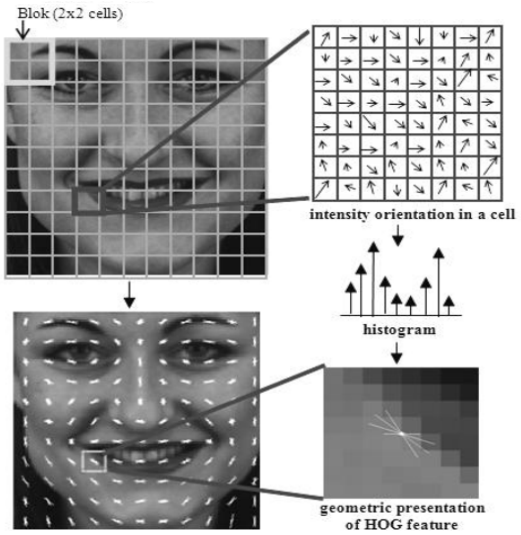


Figure 4.4: Histogramme de gradients orientés

Les caractéristiques locales se distinguent par le fait qu'elles sont distinctes, robustes aux occlusions (car il y en a beaucoup dans une image ou une région) et qu'elles ne nécessitent pas de segmentation.

La caractérisation d'une image nécessite d'abord la détection de zones d'intérêt de l'image puis calculer en chacune de ces zones un vecteur caractéristique. Ces zones d'intérêt sont par exemple les arêtes ou les points saillants de l'image (zones de fort contraste). Il peut aussi s'agir de points pris aléatoirement ou régulièrement dans l'image (échantillonnage dit dense).

Généralement, le vecteur caractéristique en un pixel est calculé sur un voisinage de ce pixel, c'est-à-dire à partir d'une *imagette* centrée sur ce pixel. Certaines méthodes telles que SIFT [142] ou SURF [143] incluent à la fois la détection de zone d'intérêt et le calcul d'un vecteur caractéristique en chacune de ces zones. Concernant le vecteur caractéristique, les SIFT sont grossièrement un histogramme des orientations du gradient et les SURF consistent en le calcul d'approximation d'ondelettes de Haar.

Aggrégation par le *Bag Of Words*

La description d'une image au moyen de caractéristiques locales a généralement une dimension variable, dépendant du nombre de points d'intérêt extraits (en fait, ce nombre multiplié par la dimension du descripteur local). Une telle représentation n'est donc pas adaptée à nourrir des algorithmes d'apprentissage classiquement utilisés (SVM [144], boosting [145]...). Pour se ramener à une représentation dans un espace vectoriel de dimension fixe, il est fait appel à des techniques d'agrégation de descripteurs telles que celles des *sacs de mots* (*bag of words*). Le résultat d'une telle accumulation est donc une caractéristique globale d'une image ou d'une partie d'image (région).

Bundle adjustment

Le *bundle adjustment* consiste à affiner conjointement un ensemble d'estimations initiales des paramètres de la caméra et de la structure pour trouver l'ensemble de paramètres qui prédit le plus précisément les emplacements des points observés dans l'ensemble des images disponibles. Plus formellement [139], supposons que nous avons n points $3D$ dans m *vues* et soit $\mathbf{x}_{ij}$ la projection du point i sur l'image j. soit v_{ij} la variable binaire égale à 1 si le point i est présent dans l'image j et 0 ailleurs. Supposons que chaque caméra j est paramétrée par un vecteur a_j et chaque vecteur $3D$ i par un vecteur b_i. Le *bundle adjustment* minimise l'erreur totale de reprojection par rapport à tous les points 3D et paramètres de la caméra, en particulier:

$$\min_{\mathbf{a}_j,\,\mathbf{b}_i} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij}\, d(\mathbf{Q}(\mathbf{a}_j,\, \mathbf{b}_i),\ \mathbf{x}_{ij})^2 \tag{4.17}$$

où $\mathbf{Q}(\mathbf{a}_j,\, \mathbf{b}_i)$ est la projection pré-dite du point i sur l'image j et $d(\mathbf{x},\, \mathbf{y})$ indique la distance euclidienne entre les points d'image représentés par les vecteurs $\mathbf{x}$ et $\mathbf{y}$.De toute évidence, le bundle adjustment est par définition tolérant aux projections d'images manquantes et minimise un critère physiquement significatif.

4.3.2 L'algorithme d'ORB-SLAM2

ORB-SLAM

ORB-SLAM2 s'appuie sur son prédécesseur ORB-SLAM [9]. La structure contient trois processus qui fonctionnent en parallèle, le *suivi* (*tracking*), la *cartographie locale* (*local mapping*) et *la fermeture de la boucle* (*loop closing*) comme le montre la figure 4.5.

L'entrée dans ORB-SLAM est une image dont la première tâche est d'extraire les caractéristiques ORB. Pour ce faire, il faut trouver les caractéristiques des contours dans la trame d'entrée à l'aide d'une méthode d'extraction de caractéristiques appelée FAST [146],

puis créer un *descripteur* pour les caractéristiques à l'aide d'une méthode appelée BRIEF [147]. Ensuite, l'information est envoyée aux différents processus.

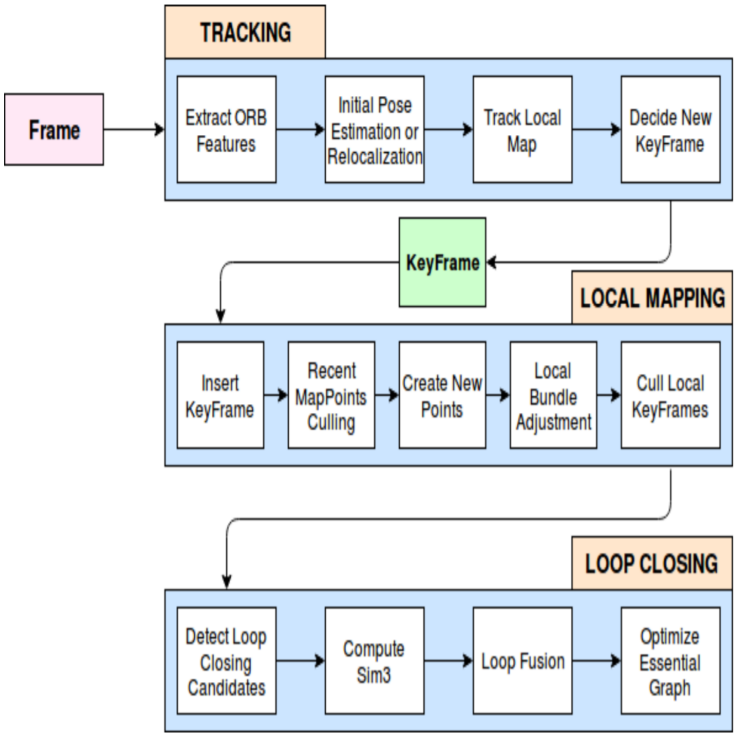


Figure 4.5: Structure de l'algorithme ORB-SLAM

Le tracking Le processus de suivi utilise chaque image pour trouver la localisation de la caméra et décide également quelle image doit être utilisée comme image clé. Les images-clés sont l'ensemble des images sélectionnées utilisées pour construire la carte. Une estimation initiale de la pose est obtenue en utilisant la correspondance des caractéristiques de l'image-clé choisie et de l'image-clé précédente. Une étape de suivi réussie permet de trouver une estimation de la pose de la caméra et une première série de correspondances de caractéristiques à partir desquelles une carte locale peut être projetée [9].

La cartographie locale La carte est constituée par les images-clés acquises et les points de la carte qu'elles contiennent. Lorsqu'une image-clé est acquise, elle est placée dans un *graphe de covisibilité*. Pour éviter une croissance illimitée du graphe, les images-clés sont éliminées si elles sont jugées trop similaires aux autres images-clés [9].

Fermeture de la boucle Le processus final effectue la fermeture de la boucle pour essayer de reconnaître si cet endroit a déjà été visité auparavant. La fermeture de la boucle est essentielle car elle permet au système de mettre à jour les estimations sur l'emplacement et de déterminer la dérive accumulée pendant que la boucle était traversée. Dans ORB-SLAM, cela se fait de telle sorte que pour chaque nouvelle image clé, on tente de fermer la boucle en examinant le *graphede covisibilité* et la similitude entre les images clés. Le graphe de covisibilité est ensuite mis à jour en conséquence [9].

ORB-SLAM2

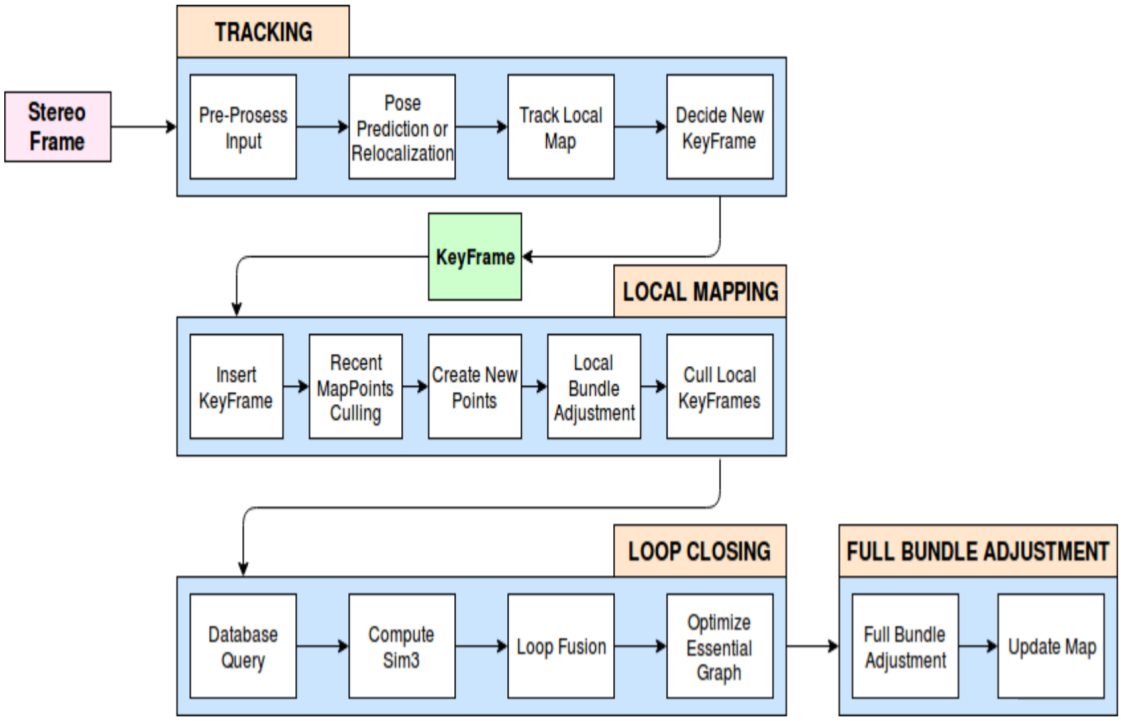


Figure 4.6: Structure de l'algorithme ORB-SLAM2

ORB-SLAM2 est une extension d'ORB-SLAM qui permet également l'utilisation d'une caméra RGB-D ou une caméra stéréo au lieu d'une caméra monoculaire [65]. Comme son prédécesseur, il utilise trois processus qui fonctionnent en parallèle. le suivi, la cartographie locale et la fermeture de la boucle. Une autre innovation d'ORB-SLAM2 est l'initialisation d'un quatrième processus dans celui de fermeture de boucle, comme le montre la figure 4.6. Ce fil effectue un ajustement complet de l'ensemble de la carte pour obtenir une reconstruction optimisée et cohérente de l'environnement.
Pour les caméras stéréoscopiques, les caractéristiques ORB sont extraites des images de gauche et de droite, l'image de gauche sert d'image de référence pour la comparaison des caractéristiques et ses caractéristiques ORB sont ensuite comparées aux caractéristiques ORB de l'image de droite. Il en résulte des fonctions ORB avec information de profondeur permettant une cartographie 3D directe sans triangulation.

4.4 Conclusion

Nous avons présenté dans ce chapitre la technologie fondamentale associée au fonctionnement efficace et performant d'un quadrirotor à savoir le processus d'estimation de l'état.
Nous avons donc vu une méthode populaire pour recouvrir l'orientation du quadrirotor à l'aide de l'IMU en utilisant le filtre complémentaire. Nous avons aussi présenté l'algorithme open-source ORB-SLAM2 pour estimer la pose du véhicule en utilisant la vision stéréoscopique.

Chapitre 5

Navigation visuelle autonome

5.1 Introduction

Depuis que la vision a été introduite aux UAVs, elle a été source de nombreuses contributions à la recherche puisque les stratégies de navigation visuelles autonomes augmentent considérablement le champs d'application de ces plateformes aériennes [78].

Les courses de drones autonomes sont un terrain d'expérimentation et de validation naturel pour mettre au point et développer ces stratégies. Cependant, contrairement aux pilotes humains qui peuvent compléter une course après une poignée de tours d'entraînements, l'état de l'art des algorithme de navigation nécessitent soit une carte exacte de l'environnement ou un nombre exorbitant de données.

Comme solution à cette problématique, nous proposons une stratégie hybride entre la navigation basée sur la carte et une navigation entièrement réactive basée sur l'image. C'est cette dernière que nous expliciterons lors de ce chapitre, tandis que l'architecture globale sera présentée dans le chapitre suivant.

Un réseau de neurones convolutionnel a été entraînée pour identifier les 4 coins du *gate* qu'il faut franchir. Une loi de commande définie sur le plan de l'image est ensuite élaborée pour permettre au quadrirotor de passer à travers ce dernier.
Ce découplage entre la perception et la commande lors de l'entraînement du réseau est particulièrement intéressant puisque il offre l'avantage de pouvoir être entraîné en *offline* (indépendamment de la loi de commande utilisé) et aussi de pouvoir être déployé dans d'autres environnements avec un ré-entraînemnt léger (Il faut seulement fournir les images des nouveaux *gates*).

5.2 Perception visuelle

Les réseaux de neurones convolutionnels (CNNs) ont connu un succès fulgurant lors de ces dernières années, notamment en vision et traitement d'images. C'est donc la solution que nous avons adoptée pour détecter dans le plan de l'image les *gates* que doit traverser le quadrirotor. Après la revue des notions relatives aux CNNs, nous présenterons l'architecture CNN proposée ainsi que les performances observées.

5.2.1 Notions sur les réseaux de neurones convolutionnels

Architecture classique

Les réseaux de neurones convolutionnels (en anglais Convolutional neural networks), aussi connus sous le nom de CNNs, sont un type spécifique de réseaux de neurones qui sont généralement composés des couches suivantes :

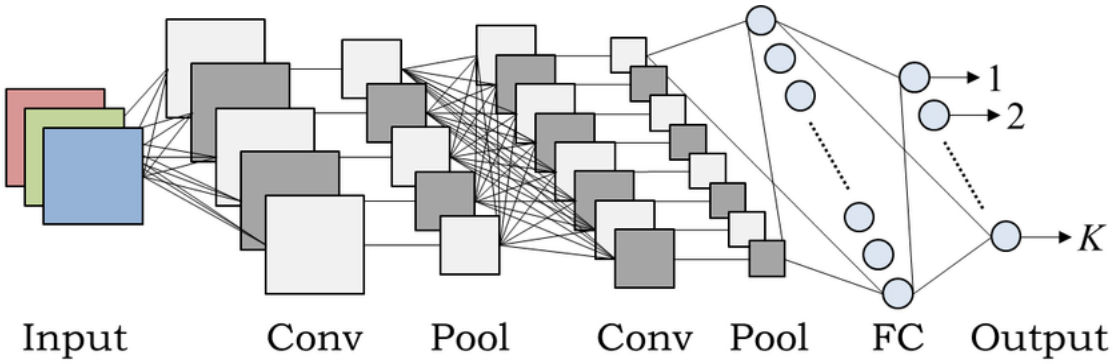


Figure 5.1: Architecture d'un CNN classique

Couche convolutionnelle(CONV) La couche convolutionnelle (en anglais convolution layer) (CONV) utilise des filtres qui *scannent* l'entrée I suivant ses dimensions en effectuant des opérations de convolution. Elle peut être réglée en ajustant la taille du filtre F et le *stride* S. La sortie O de cette opération est appelée *feature map* ou aussi *activation map*.

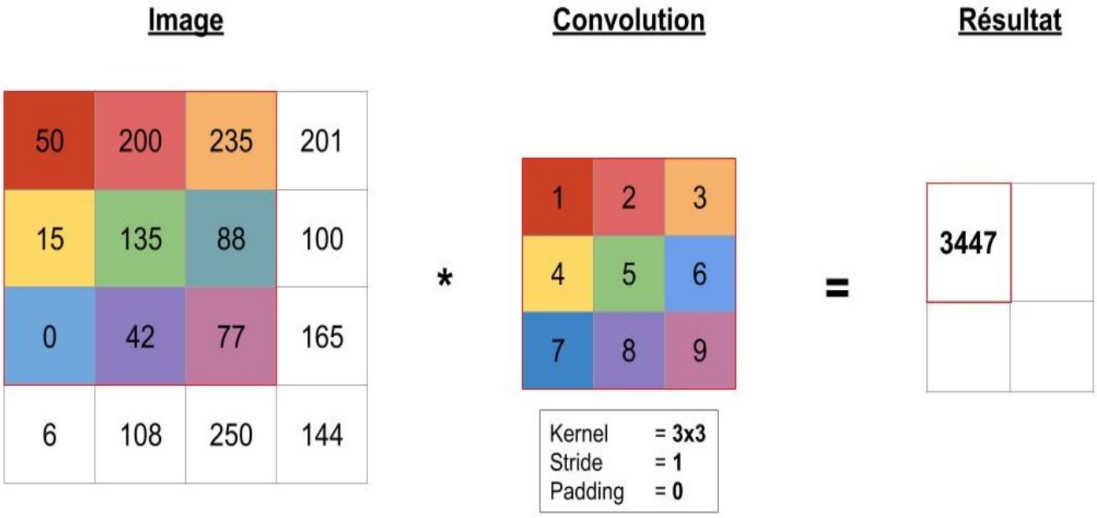


Figure 5.2: Illustration de l'opération convolution 2D

La taille de l'image de sortie est donnée par : $O = \frac{I-K+2P}{S} + 1$ avec:
O : taille de l'image de sortie, I : taille de l'image d'entrée, F : taille du *kernel* (filtre) utilisé, S :*stride* appliqué et P : *padding* appliqué

Couche d'activation La couche d'activation consiste en l'application d'une fonction *non-linéaire* après une opération de convolution. Cette fonction d'activation g est utilisée sur tous les éléments du volume. Elle a pour but d'introduire des complexités non-linéaires au réseau. Ses variantes sont récapitulées dans le tableau suivant:

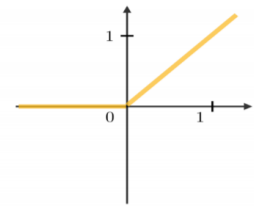
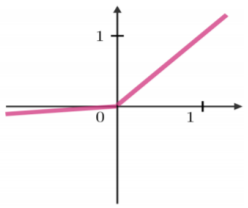
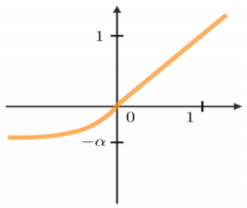
ReLU	Leaky ReLU	ELU
$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z,z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1),z)$ with $\alpha \ll 1$
		
Complexités non-linéaires interprétables d'un point de vue biologique	Répond au problème de *dying ReLU*	Dérivable partout

Tableau 5.1: Fonctions d'activation couramment utilisées

Pooling La couche de *pooling* est une opération de sous-échantillonnage typiquement appliquée après une couche convolutionnelle. Elle consiste à remplacer un carré de pixels (généralement 2×2 ou 3×3) par une valeur unique. De cette manière, l'image diminue en taille et se retrouve simplifiée (lissée). En particulier, les types de pooling les plus populaires sont le *max* et l'*average pooling*, où les valeurs maximales et moyennes sont prises, respectivement.

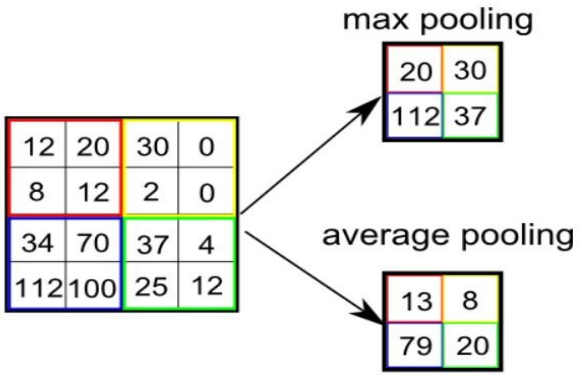


Figure 5.3: Illustration des opérations de pooling "max" et "avg"

Le flattening (ou mise à plat) le flattening consiste simplement à mettre bout à bout toutes les images (matrices) qui émanent des couches de convolutions précédentes pour en faire un (long) vecteur. Les pixels sont simplement récupérés ligne par ligne et ajoutés au vecteur final.

Fully connected (FC) La couche *fully connected* s'applique sur une entrée préalablement aplatie où chaque entrée est connectée à tous les neurones. Les couches de fully connected sont typiquement présentes à la fin des architectures de CNN et peuvent être utilisées pour optimiser des objectifs tels que les scores de classe.

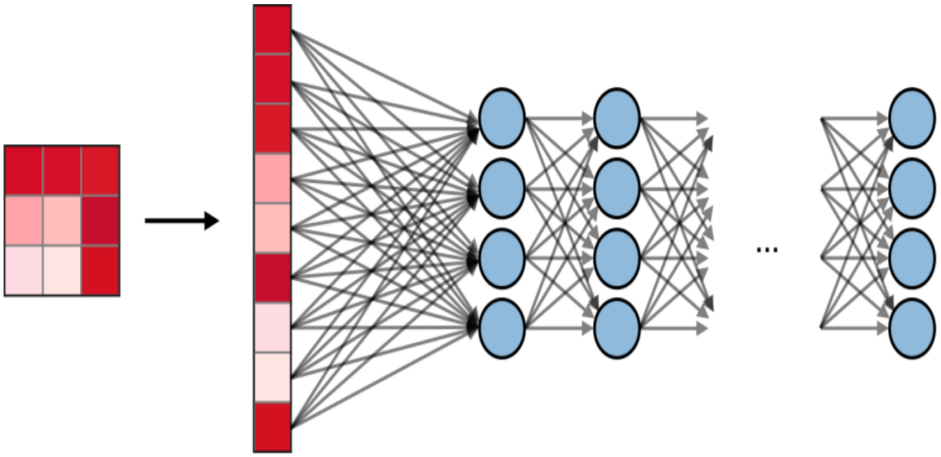


Figure 5.4: Illustration d'une couche FC après aplatissement

Normalisation de lots La normalisation de lot (en anglais *batch normalization*) est une étape qui normalise le lot x_i avec un choix de paramètres γ, β. En notant μ_B, σ_{B^2} la moyenne et la variance de ce que l'on veut corriger du lot on a:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_{B^2} + \epsilon}\beta} \tag{5.1}$$

Couche de sortie La couche de sortie est une couche *fully connected* dont la fonction d'activation ne peut être spécifié arbitrairement puisque celle ci dépend du type de problème. Nous donnons à titre d'exemple le cas de figure suivants:

1. Régression dans R^m: La fonction identité $f(x) = x$.
2. Classification Binaire dans $[0\,1]^2$:la fonction sigmoïde $f(x) = (1 + e^{-x})^{-1}$
3. Classification dans $[0\,1]^m$: la fonction *softmax* définie pour un vecteur $z = [z_1\, z_2 \,...\, z_k]$ comme suit: $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ pour tout $j \in 1, .., k$.

Apprentissage des réseaux de neurones

Fonction objectif (ou *loss function* en anglais) est une fonction $L : (z, y) \in \mathbb{R} \times Y \rightarrow L(z, y) \in \mathbb{R}$ qui sert de critère pour déterminer la meilleure solution à un problème d'optimisation. Le but du problème d'optimisation devient alors une minimisation (ou maximisation) de cette fonction objectif. Les fonctions objectifs les plus courantes sont représenté dans la figure 5.2 suivante:

Fonction de coût La fonction de coût J est communément utilisée pour évaluer la performance d'un modèle (réseau de neurones), et est définie avec la fonction de loss L par:

$$J(\theta) = \sum_{i=1}^{m} L(h_\theta(x^{(i)}), y^{(i)}) \tag{5.2}$$

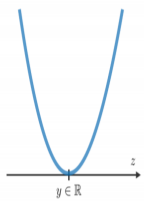
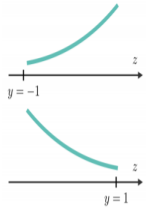
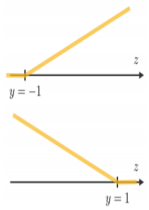
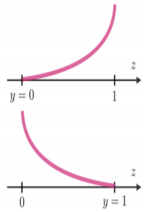
Moindres carrés	Logistique	Hinge loss	Cross-entropie
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0,1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Régression	Classification	Classification	Classification

Tableau 5.2: Exemples de fonctions objectifs

Algorithme du gradient est un algorithme itératif simple et populaire qui permet de mettre à jour des paramètres θ en vue de minimiser une fonction de coût J. En notant $\alpha \in \mathbb{R}$ le taux d'apprentissage (en anglais *learning rate*), la règle de mise à jour de l'algorithme est:

$$\theta \leftarrow \theta - \alpha \nabla J(\theta) \tag{5.3}$$

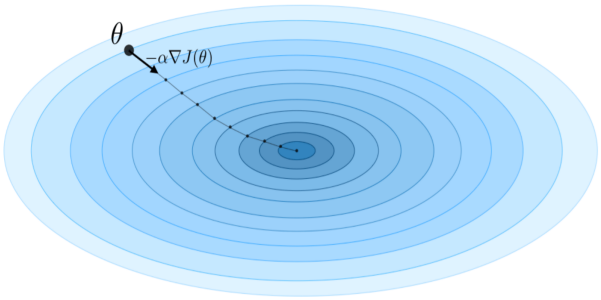


Figure 5.5: Visualisation de l'algorithme du gradient

Le taux d'apprentissage α indique la vitesse à laquelle les coefficients sont mis à jour. Il peut être fixe ou variable. La méthode actuelle la plus populaire est appelée *Adam*, qui est une méthode faisant varier le taux d'apprentissage. Voici une liste non exhaustive des algorithmes d'adaptation du taux d'apprentissage les plus couramment utilisés :

Méthode	Explication	Mise à jour de w	Mise à jour de b
Momentum	- Amortit les oscillations - Amélioration par rapport à la méthode SGD - 2 paramètres à régler	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	- Root Mean Square propagation - Accélère l'algorithme d'apprentissage en contrôlant les oscillations	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \longleftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - Méthode la plus populaire - 4 paramètres à régler	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \longleftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

Tableau 5.3: Principaux algorithmes d'adaptation du taux d'apprentissage

Rétropropagation du gradient (*Backpropagation* en anglais) est une méthode pour calculer le gradient de l'erreur pour chaque neurone d'un réseau de neurones, de la dernière couche vers la première. La dérivée par rapport à chaque coefficient (*poids*) du réseau est calculée en utilisant la règle de la chaîne.

Amélioration de l'apprentissage des réseaux de neurones

Dropout Le dropout est une technique qui est destinée à empêcher le sur-ajustement (*overfitting*) sur les données de training en abandonnant des unités dans un réseau de neurones avec une probabilité $p > 0$. Cela force le modèle à éviter de trop s'appuyer sur un ensemble particulier de *features*.

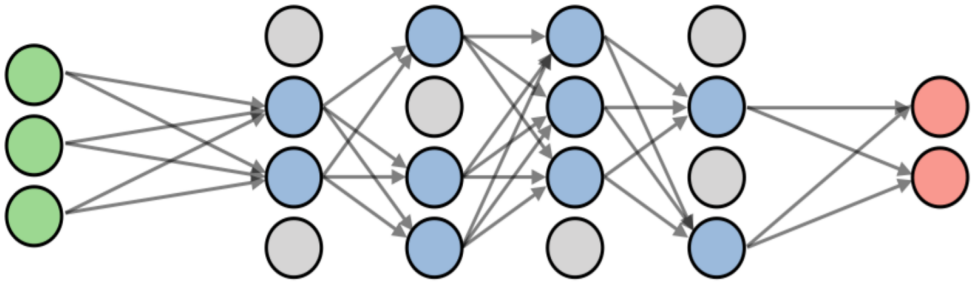


Figure 5.6: Illustration du dropout

Régularisation des coefficients Pour s'assurer que les coefficients ne sont pas trop grands et que le modèle ne sur-ajuste pas sur le training set, on utilise des techniques de régularisation sur les coefficients du modèle. Les techniques principales sont résumées dans le figure suivante:

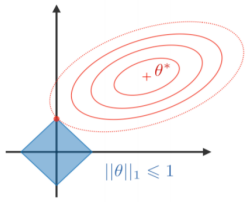
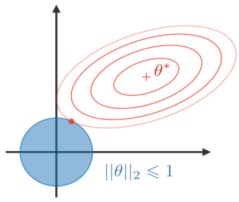
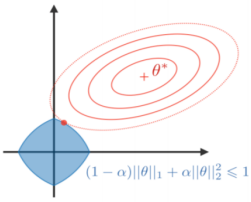
LASSO	**Ridge**	**Elastic Net**
- Réduit les coefficients à 0 - Bon pour la sélection de variables	Rend les coefficients plus petits	Compromis entre la sélection de variables et la réduction de la taille des coefficients
		
$... + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

Tableau 5.4: Différents types de régularisation des coefficients

Arrêt prématuré L'arrêt prématuré (en anglais *early stopping*) est une technique de régularisation qui consiste à stopper l'étape d'entraînement dès que le loss de validation stagne ou commence à augmenter.

Apprentissage par transfert Entraîner un modèle d'apprentissage profond requière beaucoup de données et beaucoup de temps. Il est souvent utile de profiter de coefficients pré-entraînés sur des données énormes qui ont pris des jours/semaines pour être entraînés. l'apprentissage par transfert ou *fine-tuning* est une technique très utilisé pour l'entraînement des CNN, elle consiste à débloquer quelques-unes des dernières couches d'un réseau de base gelée déjà entraînée sur une grande quantité de données. Ce modèle servira donc à extraire les premières et principales caractéistiques et à former conjointement avec la partie ajoutée un nouveau modèle. C'est ce qu'on appelle le *fine-tuning* (réglage fin) parce qu'il ajuste légèrement les représentations plus abstraites du modèle réutilisé, afin de les rendre plus pertinentes pour le problème en question.

Residual Network L'architecture du réseau résiduel (en anglais Residual Network), aussi appelé ResNet, utilise des blocs résiduels avec un nombre élevé de couches et a pour but de réduire l'erreur de training. Son implémention est relativement simple: chaque couche s'alimente dans la couche suivante et directement dans 2-3 couches plus loin.

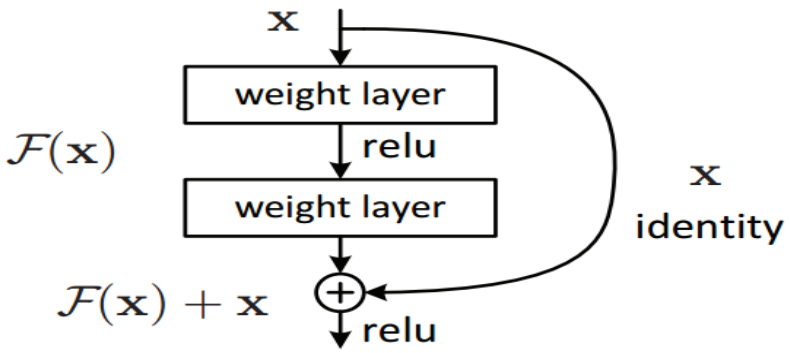


Figure 5.7: Implémentation typique d'un réseau résiduel

[148] ont d'abord démontré le problème de profondeur des réseaux de neurones et ont proposé une solution remarquable qui a depuis lors permis l'entraînement de plus de 2000 couches avec une précision croissante. En appliquant ce principe, les auteurs ont gagné l'édition 2015 de Imagenet [149] et ont atteint un nouvel état de l'art sur tous les benchmarks standard de vision par ordinateur.

5.2.2 Architecture proposée pour la détection de *gates*

Rapidité, efficacité , précision et robustesse ont été nos principaux objectifs lors de la conception du système de perception et de vision du quadrirotor.
Bien qu'ils soient très populaires et très efficaces, les détecteurs d'objets basés sur la région, comme Faster R-CNNN [149] ou les détecteurs d'objets à tir unique comme YOLO V3 [150] et SSD [150], ne sont pas bien adaptés au calcul en temps réel à bord [151], ni les réseaux neuronaux convolutionnels profonds entraînés sur de grands jeux de données (VGG [152], ResNet [148] par ex.). Pour les besoins de la Course Autonome de drones en général où les calculs doivent être exécutés entièrement à bord, nous avons décidé d'adopter une architecture CNN peu profonde que nous avons appelé DroGate.
DroGate est designé sur la base de l'architecture proposé par [153].

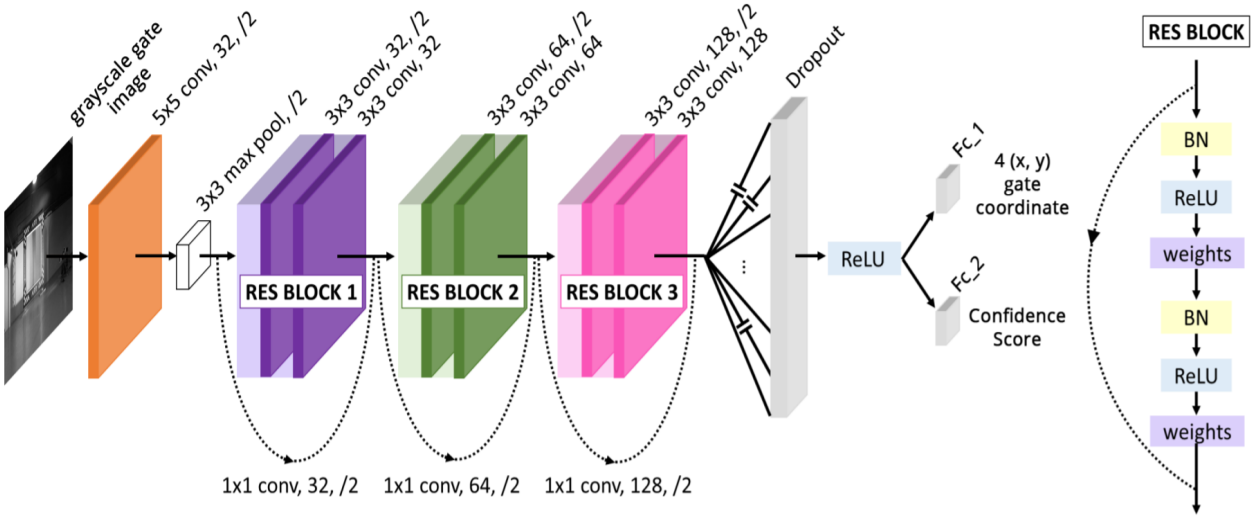


Figure 5.8: Architecture de DroGate

L'entrée du réseau est une image en nuances de gris 300 x 300. La partie convolutionnelle consiste en un bloc rapide ResNet-8 composé de 3 blocs résiduels, suivi d'un *dropout* et d'une non-linéarité ReLu. Les features extraits sont alors traitées par deux *perceptrons* multicouches (MLP : *Multilayer perceptron* en anglais):

1. Couche linéaire qui prédit 4 coordonnées (x,y) du *gate* en pixels.
2. Couche sigmoïde qui renvoie la confiance qu'a le réseau en sa prédiction précédente [154]. Cette information permettra par la suite de se fier ou non à la prédiction du réseau.

L'architecture de DroGate a prouvé son efficacité comme système de perception pour les drones autonomes. En effet, une architecture similaire a été utilisé par [7] et [11] pour la navigation visuelle autonome d'un quadrirotor et a offert un parfait compromis entre précision et rapidité d'exécution.

5.2.3 Collecte et nettoyage des données

Collecte des données

Les données ont été récoltées sur le site du challenge AlphaPilot [3] au lien suivant.
Ces données comportent ≈ 9300 images étiquetées des 4 coordonnées (x,y) des *gates* typiquement présents dans les courses de drones (Figure 5.9). Ces coordonnées, données en pixels, représentent la zone du *gate* que le drone doit traverser.
Ces *gates* sont munies de repères visuels (p. ex. couleurs, motifs, logos) qui fourniront des repères pour faciliter l'orientation tout au long du parcours, et sont présentés sous différentes distances, angles et profils d'éclairage.

Nettoyage des données

Les données contiennent beaucoup de *gates* mal étiquetées et il est crucial de s'en débarrasser, comme pour tout projet d'apprentissage automatique.
Nous avons utilisé plusieurs méthodes pour détecter les données aberrantes:

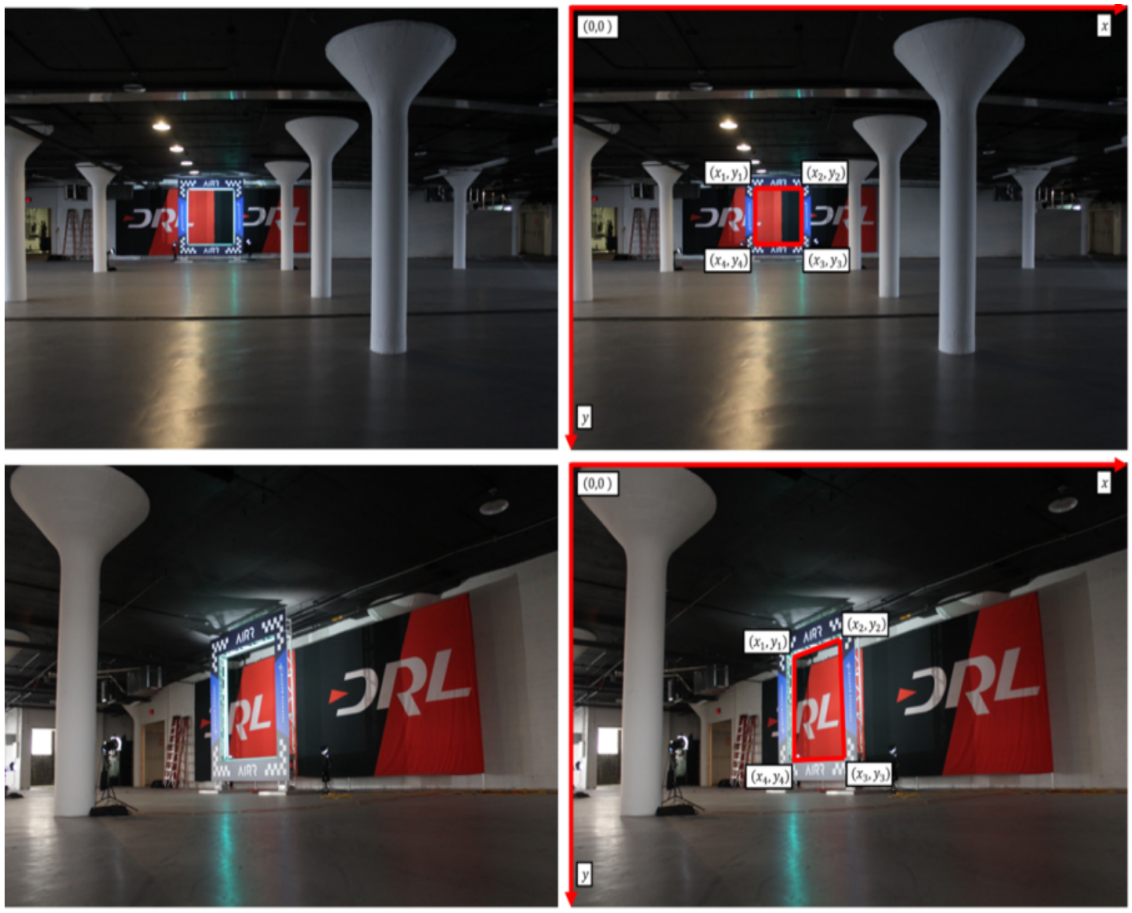


Figure 5.9: Échantillon du dataset utilisé

Heuristique du trapèze Après observation, une hypothèse forte peut être formulée, à savoir que la projection de la porte sur le plan de l'image est dans la plupart des cas un trapèze. Les images qui s'écartent fortement de cette règle sont principalement des valeurs aberrantes.
en notant $\vec{u} = \frac{(y_4 - y_1, x_4 - x_1)}{\|y_4 - y_1, x_4 - x_1\|}$ et $\vec{v} = \frac{(y_3 - y_2, x_3 - x_2)}{\|y_3 - y_2, x_3 - x_2\|}$, le *box plot* suivant permettant de visualiser la distribution de $det(\vec{u}, \vec{v})$ met en avant les valeurs aberrantes:

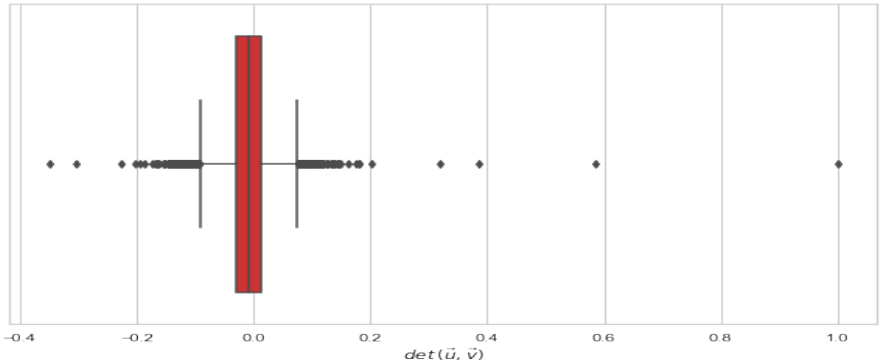


Figure 5.10: Distribution de $det(\vec{u}, \vec{v})$

Cette méthode nous a permis d'identifier les données aberrantes suivantes:

Figure 5.11: Données mal étiquetées identifiées par l'heuristique du trapèze

Auto-encodeur Un auto-encodeur est un réseau de neurones utilisé pour l'apprentissage non supervisé d'une représentation (encodage) d'un ensemble de données.
La forme la plus simple d'un auto-encodeur est un perceptron multicouches - ayant une couche d'entrée, une couche de sortie ainsi qu'une ou plusieurs couches cachées les reliant -, mais avec toutefois une couche de sortie possédant le même nombre de nœuds que la couche d'entrée, car le but est d'avoir en sortie les mêmes données d'entrée.

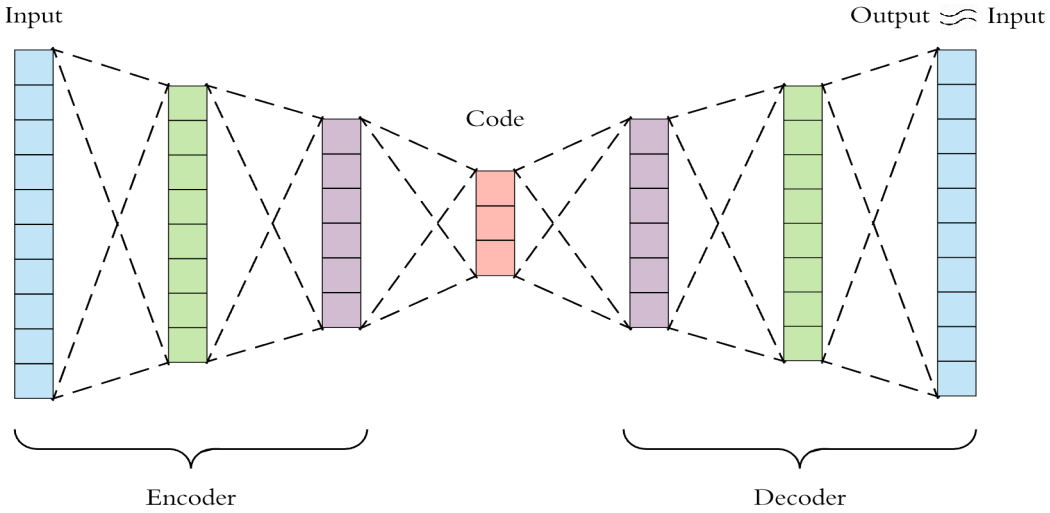


Figure 5.12: Schéma structurelle d'un auto-encodeur

Les données passées à l'auto-encodeur sont les 4 coordonnées (x_i, y_i). en notant $X = [x_1\ y_1\ ...\ x_4\ y_4]$, et $\tilde{X}$ le normalisé de X entre 0 et 1; nous définissons l'erreur de reconstruction comme étant:
$E_{rec} = \| \tilde{X} - autoencoder(\tilde{X}) \|$. La distribution de E_{rec} suivante met en évidence la présence de données aberrantes.

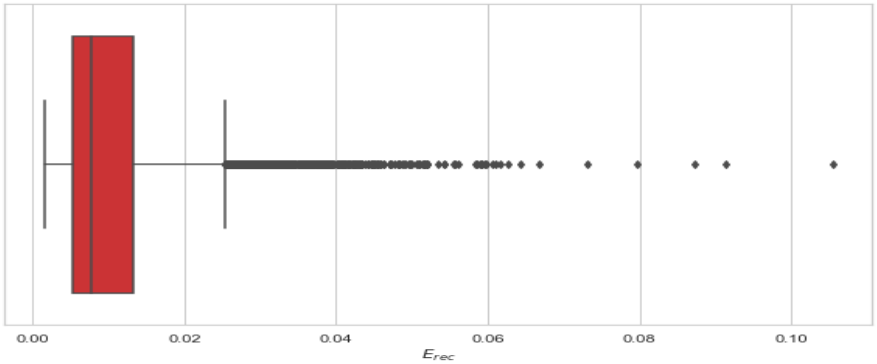


Figure 5.13: Distribution de E_{rec}

Quelques exemples de données mal étiquetées identifiées grâce à l'auto-encodeur sont illustrés ci-dessous:

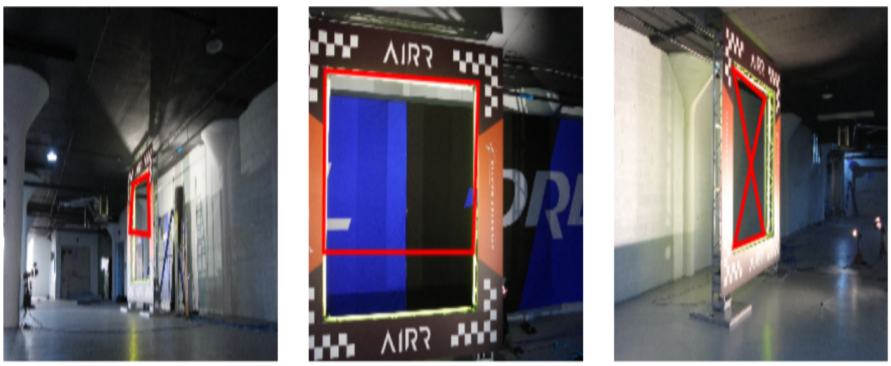


Figure 5.14: Données mal étiquetées identifiées par l'auto-encodeur

5.2.4 Entraînement de DroGate et résultats

Entraînement

Le réseau de neurones convolutionnelle DroGate est entraînée en 2 temps:

1. En premier lieu, le réseau est entraîné pour prédire les 4 coordonnées (x, y) en minimisant l'erreur quadratique moyenne (MSE : *mean squared error*). Ensuite, le score *COCO map* [155] est calculé pour chaque image en comparant les prédictions avec les données. Ces score sont les *score de confiance* que le réseau doit prédire.

2. En second lieu, une couche à activation sigmoïde est ajoutée au réseau. Seuls les poids associés à cette couche sont entraînées lors de cette étape. Cette couche est entraînée à calculer la confiance qu'a le réseau en sa prédiction, en minimisant l'erreur quadratique logaritmique moyenne (MSLE : *mean squared logarithmic error*). le MSLE a montré une meilleure convergence lors de l'entraînement, puisque le MSE fournit de trop faibles gradients au réseau.

Résultats

Performances DroGate a été entraîné en utilisant une carte graphique *Nvidia GTX 1050 Ti*, à l'aide de le framework open-source *TensorFlow*[156] sous Python 3.5. l'algorithme

d'optimisation choisi est *Adam*[157], et a permis un entraînement en $\approx 15\ mn$.
Comme critère de performances nous avons choisi la précision de la prédiction et le temps de calcul du réseau. N.B: La précision de DroGate est évaluée avec des données sur lesquelles le réseau n'a pas été entraînées.

1. **Précision**: Les résultats ont montré qu'en moyenne DroGate rate les coordonnée véritables de $4,6$ pixels (MAE $= 4,6$). Puisque la taille de l'image est de 864×1296, Cette erreur n'est significative que lorsque le quadrirotor est à grande distance du *gate*. Par conséquent, nous estimons que cette précision est suffisante pour la navigation du quadrirotor.

2. **Rapidité**: le tableau suivant montre que DroGate affiche un temps de calcul (par image) très rapide.

Model Hardware	DroGate GTX 1050Ti	SSD [98] GTX 1050Ti
Temps de calcul (ms)	**1.4**	24.16

Tableau 5.5: Temps de calcul de différentes architectures CNN

Visualisation des couches internes La visualisation des couches interne du réseau convolutionnel permet de voir que celui-ci apprend de manière hiérarchique les caractéristiques d'un *gate*. la première et deuxième couche montrent en effet un cadre rectangulaire distingué alors que l'arrière plan de l'image est ignoré. Les couches 3 et 4 (le reste des couches aussi) montrent des caractéristiques plus abstraites et donc plus difficiles à interpréter.

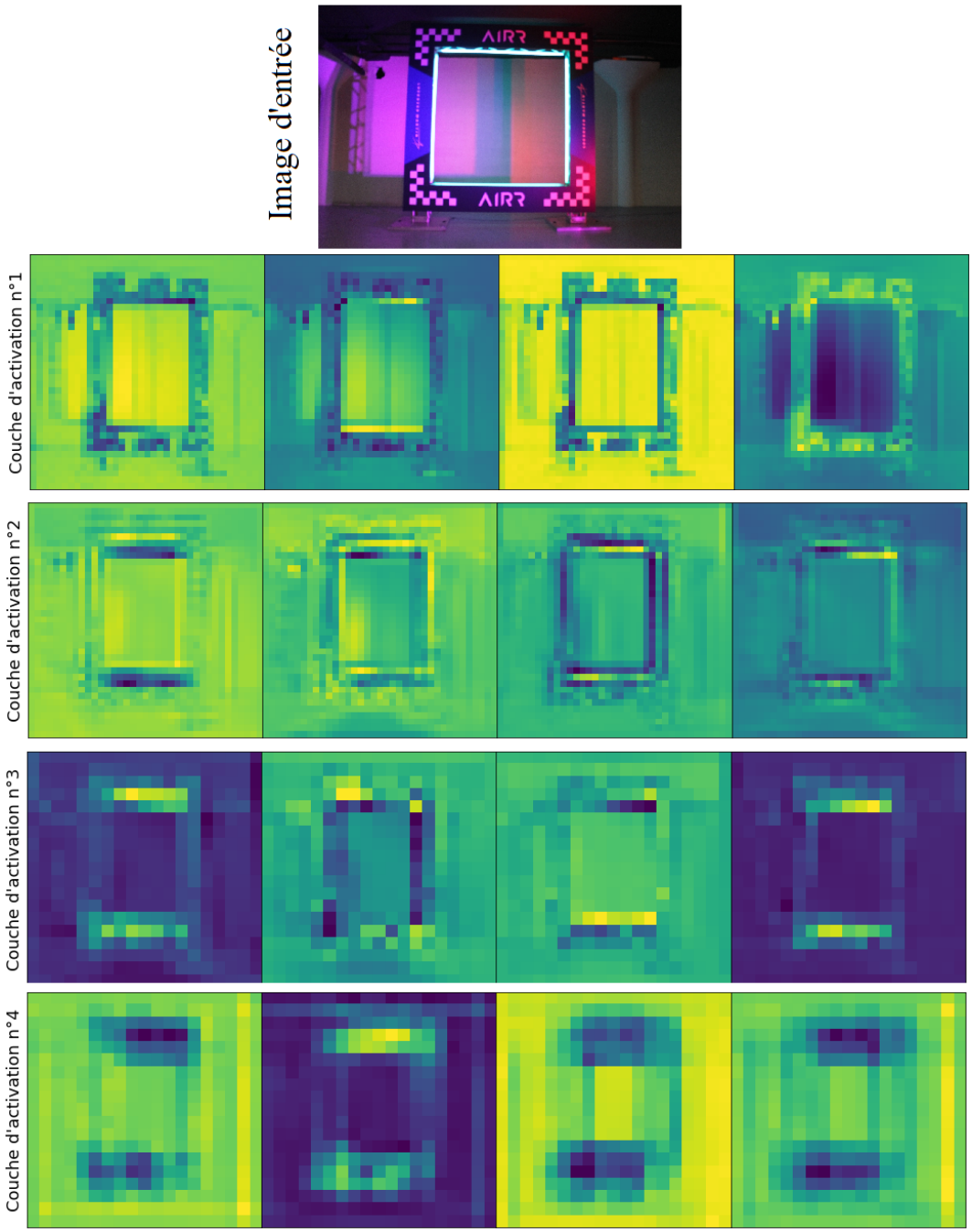


Figure 5.15: Visualisation des couches internes de DroGate

5.3 Asservissement visuel

Les positions des *gates* durant une course n'étant connues qu'approximativement, il est impossible de préprogrammer des points de passage qui assureront au quadrirotor de passer à travers les différents *gates*.
La méthode que nous avons adoptée consiste à appliquer l'asservissement visuel basé sur l'image (*Image-based visual servoing : IBVS*), en utilisant les 4 coordonnées dans le plan image obtenues grâce à DroGate.

Contrairement à l'asservissement visuel basé sur la *pose* (Pose-based visual servoing : PBVS), l'IBVS nous garantie de garder le *gate* dans le champs de vision de la caméra.

5.3.1 Commande et stabilité

L'objectif de la commande est de minimiser l'erreur suivante:

$$\mathbf{e}(t) = \mathbf{s}(t) - \mathbf{s}^* \tag{5.4}$$

où $\mathbf{s(t)}$ sont les 4 coordonnées dans le plan de l'image et $\mathbf{s}^*$ sont les coordonnées désirées. $\mathbf{s}^*$ est statique et est calculé de sorte à avoir le *gate* en face de la caméra à une distance d adéquatement choisie.
Soit $\mathbf{v_c} = (v_\mathbf{c}, \omega_\mathbf{c})$ la vitesse spatiale de la caméra où $v_\mathbf{c}$ et $\omega_\mathbf{c}$ sont les vitesses linéaires et angulaires respectivement .
La relation entre $\dot{\mathbf{s}}$ et $\mathbf{v_c}$ est donnée par

$$\dot{\mathbf{s}} = \mathbf{J}\mathbf{v_c} \tag{5.5}$$

$\mathbf{J}$ est la matrice jacobienne que nous expliciterons dans la section suivante.
La dynamique de e est donnée dans ce cas par:

$$\dot{\mathbf{e}} = \dot{\mathbf{s}} = \mathbf{J}\mathbf{v_c} \tag{5.6}$$

En supposant donc une plate-forme robotique holonomique[wikipédia] (hypothèse non valide pour le quadrirotor, nous y proposerons une solution ultérieurement), Les vitesses $\mathbf{v_c}$ de la caméra peuvent-être utilisés pour commander le robot avec une erreur exponentiellement décroissante (c-à-d $\dot{\mathbf{e}} = -\lambda\mathbf{e}$) comme suit:

$$\mathbf{v_c} = -\lambda\mathbf{J}^+\mathbf{e} \text{ avec } (\lambda > 0) \tag{5.7}$$

où $\mathbf{J}^+$ est la matrice pseudo-inverse de Moore-Penrose $\mathbf{J}^+ = (\mathbf{J}^T\mathbf{J})^{(-1)}\mathbf{J}^T$.
L'analyse de stabilité de cette commande est direct en choisissant comme fonction de Lyapunov $\mathcal{L} = \frac{1}{2} \parallel \mathbf{e}(t) \parallel^2$ qui a comme dérivée :

$$\mathcal{L} = \mathbf{e}^T\mathbf{e} = -\lambda\mathbf{e}^T\mathbf{J}\mathbf{J}^+\mathbf{e} \tag{5.8}$$

Le système est par conséquent globalement assymptotiquement stable [chaumette] quand la condition $\mathbf{J}\mathbf{J}^+ > 0$ est satisfaite.

5.3.2 Détermination de la matrice Jacobienne

Expression analytique

Considérons le cas d'un seul point 3D qui a pour coordonnées (x, y, z) dans le repère de la caméra. La projection de ce point sur le plan 2D de la caméra (x_p, y_p) est comme suit (voir section 4.3.1):

$$\begin{aligned} x_p &= \frac{x}{z} = \frac{u - u_0}{\alpha_u} \\ y_p &= \frac{y}{z} = \frac{v - v_0}{\alpha_v} \end{aligned} \tag{5.9}$$

En prenant $\mathbf{s}_1 = (x_p, y_p)^T$, la matrice jacobienne $\mathbf{J}_1$ est donnée par [chaumette]:

$$\mathbf{J}_1 = \begin{bmatrix} \frac{-1}{z} & 0 & \frac{x_p}{z} & x_p y_p & -(1+x_p^2) & y_p \\ 0 & \frac{-1}{z} & \frac{y_p}{z} & (1+y_p^2) & -x_p y_p & -x_p \end{bmatrix} \tag{5.10}$$

Dans ce cas on a :

$$\dot{\mathbf{s}}_1 = \mathbf{J}_1 v_c \tag{5.11}$$

La matrice jacobienne $\mathbf{J}_1$ fait apparaître la profondeur z du point relativement à la caméra; cette dernière doit donc être estimée ou approximée lors de l'élaboration de la commande. Les paramètres intrinsèques de la caméra $(\alpha_u, \alpha_v, y_0, v_0)$ doivent aussi être connues puisqu'ils sont utilisés lors du calcul de x_p et y_p.
Pour commander les $6DDL$ du mobile il est nécessaire d'avoir au minimum 3 points (pour pouvoir avoir $rang(J) = 6$). Nous utiliserons cependant 4 points pour éviter certaines configurations singulières. ces 4 points sont les 4 coins du *gate* que l'on veut franchir. on aura alors

$$\mathbf{s} = [\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4]^T \tag{5.12}$$

et

$$\mathbf{J} = [\mathbf{J}_1, \mathbf{J}_2, \mathbf{J}_3, \mathbf{J}_4]^T \tag{5.13}$$

Estimations et approximations numériques

Plusieurs estimations de la matrice Jacobienne ainsi que des approximations numériques ont été proposées pour calculer la matrice jacobienne, principalement pour obtenir des propriétés de convergence plus appréciables ou pour éviter l'estimation de la profondeur et des paramètres intrinsèques de la caméra.

Matrice Jacobienne désirée Un choix populaire pour la matrice jacobienne est celui de prendre $\hat{\mathbf{J}} = \mathbf{J}^*$, où $\mathbf{J}^*$ est la matrice jacobienne associée à la configuration $\mathbf{s} = \mathbf{s}^*$. Cette approximation quoique grossière a montré des résultats satisfaisants en pratique [chaumette]. Son intérêt réside dans le fait que souvent la profondeur z est connue pour la position désirée et ceci évite donc son estimation en temps réel.
On peut également choisir $\hat{\mathbf{J}} = \frac{1}{2}(\mathbf{J}^* + \mathbf{J})$, des propriétés intéressantes de convergence en pratique ont été observées.

Estimation directe Soit Δd_c un mouvement de caméra et Δs et un déplacement des caractéristiques dans l'image. Nous avons la relation:

$$\mathbf{J}\Delta\mathbf{d}_c \approx \Delta\mathbf{s} \tag{5.14}$$

$\mathbf{J} \in \mathbb{R}^{8\times6}$, $\Delta\mathbf{d}_c \in \mathbb{R}^{6\times1}$ et $\mathbf{s} \in \mathbb{R}^{8\times1}$. Cette relation nous fournit donc 8 équations alors que nous avons $8 \times 6 = 48$ inconnues (éléments de la matrice $\mathbf{J}$). Il nous faudrait donc mesurer au moins $N >= 6$ déplacements pour pouvoir avoir (en supposant des déplacements *petits*):

$$\mathbf{JA} = \mathbf{B} \tag{5.15}$$

où $\mathbf{A} \in \mathbb{R}^{6\times N}$ et $\mathbf{B} \in \mathbb{R}^{8\times N}$. l'estimateur des moindres carrés de $\mathbf{J}$ peut-être facilement calculée comme suit:

$$\hat{\mathbf{J}} = \mathbf{B}\mathbf{A}^+ \tag{5.16}$$

Puisque l'expression de la commande 5.7 utilise la matrice pseudo-inverse de $\mathbf{J}$, il est alors judicieux de calculer directement

$$\hat{\mathbf{J}^+} = \mathbf{A}\mathbf{B}^+ \tag{5.17}$$

Lorsque l'hypothèse des petits déplacements n'est pas valide il est alors essentiel d'estimer en ligne la matrice jacobienne, en minimisant l'erreur

$$\begin{aligned} E_{est} &= \frac{1}{2}(\dot{\mathbf{s}} - \hat{\dot{\mathbf{s}}})^2 \\ &= \frac{1}{2}(\dot{\mathbf{s}} - \hat{J}v_c)^2 \end{aligned} \tag{5.18}$$

en utilisant par exemple l'algorithme du gradient suivant:

$$\begin{aligned} \dot{\hat{J}} &= -\alpha \frac{\partial E_{est}}{\partial \hat{J}} \\ \dot{\hat{J}} &= \alpha(\dot{\mathbf{s}} - \hat{\dot{\mathbf{s}}}) \cdot v_c^T \end{aligned} \tag{5.19}$$

5.3.3 IBVS pour le quadrirotor

Le quadrirotor étant un système sous-actionné, il est donc impossible d'imposer toutes les vitesses linéaires et angulaires arbitrairement [158]. Les angles de roulis et de tangage sont commandés de manière indépendante par la boucle interne de commande. Pour palier à ce problème, nous utilisons deux méthodes différentes que nous allons comparer lors du chapitre suivant.

Séparation de la matrice jacobienne

méthode présentée dans [127] qui est de séparer les vitesses de roulis et de tangage dans l'écriture de la matrice Jacobinne comme suit:

$$\dot{\mathbf{e}} = \mathbf{J}_v[v_{c,x}, v_{c,y}, v_{c,z}, \omega_{c,z}]^T + \mathbf{J}_\omega[\omega_{c,x}, \omega_{c,y}]^T \tag{5.20}$$

En estimant $\omega_{c,x}$ et $\omega_{c,y}$ à l'aide de l'IMU par exemple, on peut procéder de manière analogue à l'équation 5.7 pour tirer finalement:

$$[v_{c,x}, v_{c,y}, v_{c,z}, \omega_{c,z}]^T = \mathbf{J}_v^+(-\mathbf{J}_\omega[\omega_{c,x}, \omega_{c,y}]^T - \lambda\mathbf{e}) \tag{5.21}$$

La convergence exponentielle de $\mathbf{e}$ est assurée à condition que la matrice pseudo-inverse de $\mathbf{J}_v$ existe.

Heuristique de guidage

En considérant que le quadrirotor est à distance raisonnable du *gate*, il est alors possible d'élaborer une commande simple dans la réalisation mais surtout efficace en pratique.
Soient (u_0, v_0) les coordonnées du centre de l'image et (u_g, v_g) les coordonnées du centre du *gate* à franchir dans le plan image (barycentre des 4 coins estimés par DroGate).

Nous notons par e_h et e_v les erreurs horizontales et verticales respectivement données par:

$$e_h = \frac{(u_0 - u_g)}{u_0} \in [-1, 1] \tag{5.22}$$

$$e_v = -\frac{(v_0 - v_g)}{v_0} \in [-1, 1]) \tag{5.23}$$

$$\tag{5.24}$$

Nous faisons apparaître le signe " $-$ " volontairement pour souligner le fait que le plan de la caméra est inversé.
Les valeurs de référence calculées à partir de ces erreurs sont les suivantes:

$$\theta_{ref}(t) = \theta_{ref} \tag{5.25}$$

$$\phi_{ref}(t) = K_{phi} e_h \tag{5.26}$$

$$T(t) = K_T e_v + g \tag{5.27}$$

$$\phi_{ref}(t) = \phi_{ref,i} \tag{5.28}$$

Le quadrirotor est commandé pour avancer à vitesse constante ($\propto \theta_{ref}$) et un cap ψ préprogrammé pour garder le *gate* dans son champs de vision. L'erreur horizontale est commandé par un régulateur proportionnel à l'aide de l'angle de roulis ϕ, tandis que l'erreur vertical est commandé à l'aide de la poussé T avec un régulateur proportionnel muni d'un terme en *feed-forward* pour compenser la gravité.
Plusieurs variantes de cette commande ont été testées à savoir:

1. L'utilisation d'un régulateur proportionnel-dérivé afin de minimiser les oscillations.

2. Régler l'angle de tangage θ inversement proportionnel à la distance du quadrirotor par rapport au gate, ceci assure la diminution de la vitesse du quadrirotor en s'approchant du *gate* et ainsi minimiser le risque de collision.

5.4 Conclusion

Plusieurs approches basées sur l'apprentissage profond ont tenté de résoudre le problème de navigation visuelle autonome pour le *drone racing* [11]. Nous estimons toutefois qu'essayer d'approximer directement les commandes à partir des images est une solution inadaptée à la problématique de commande puisque des preuves de stabilité et robustesses ne peuvent être formulées. Aussi, cette solution est difficilement généralisable à d'autres environnements en raison de la grande quantité de données nécessaire.
Nous avons alors opté pour un découplage entre la perception et la commande. Les résultats obtenues grâce à cette approche seront exposées lors du prochain chapitre.

Chapitre 6

Implémentations et résultats

6.1 Introduction

Les UAV sont des systèmes complexes composés de plusieurs sous-systèmes communiquant et interdépendants: estimation, planification, navigation, contrôle. Cela impose un certain parallélisme lors de l'élaboration de l'architecture du sytème. Aussi, le problème d'embarquement et limitation des ressources matériels pour le calcul à bord demande des solutions légères et optimisées.

Un choix naturel permettant de répondre au besoin de parallélism et la complexité qui en découle, nous avons opté pour l'utilisation de ROS (Robot Operating System)[1]. Aussi, le simulateur FlightGoggles utilisé lors d'AlphaPilot supporte nativement ROS.

Dans ce chapitre, nous allons présenter notre solution proposée pour répondre au cahier de charge de la compétition AlphaPilot, ainsi que les résultats de l'implémentation des différents systèmes développés dans les chapitres précédents dans le simulateur FlightGoggles.

6.2 Architecture globale du système

Pour répondre à la problématique des courses de drones autonomes, nous avons proposé l'architecture globale présenté dans la figure 6.1 Les différents sous systèmes de l'estimation de l'état, la commande et la navigation sont mis ensemble en adoptant une architecture asynchrone multi-processus.

[1]Plus d'informations au lien suivant : http://www.ros.org/

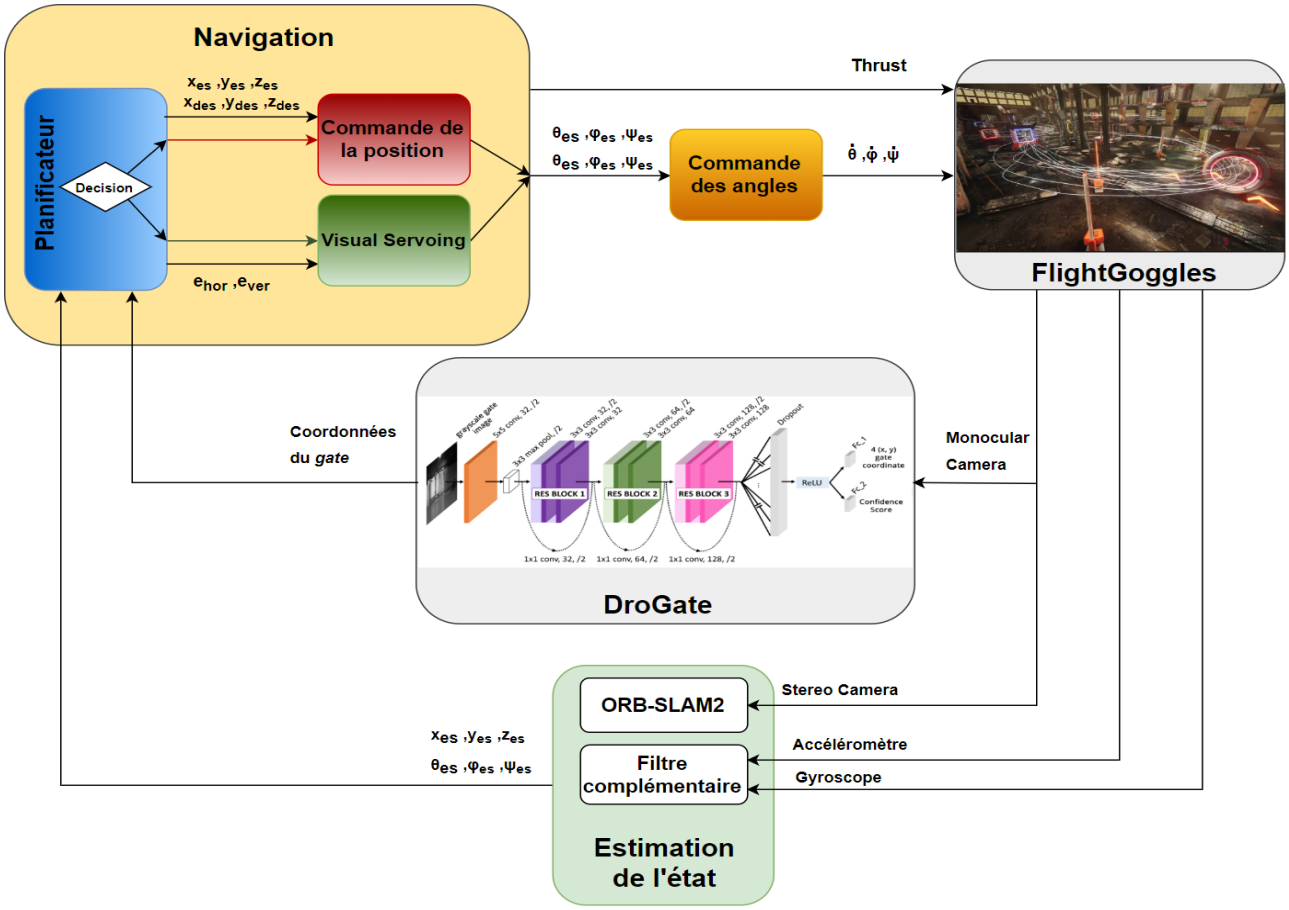


Figure 6.1: Architecture globale du système.

Estimation de l'état C'est le bloc responsable de l'exécution des algorithmes présentés dans le chapitre 4. Ce bloc reçoit en entrée les signaux issues des capteurs extéroceptifs : Caméra stéréo et signaux bruts de l'accéléromètre et du gyroscope. Sa sortie est la pose (position et orientation) du quadrirotor estimé.

DroGate Réseau de neurone convolutionnel responsable de la détection du gate dans l'image, comme présenté dans la section 5.2.2.

Navigation Le système de navigation est responsable de la commande haut niveau, Nous distinguons deux modes gérés par ce module :

- Lorsque le drone est loin du prochain *gate*, nous opérons en mode *goal reach* où nous donnons une position consigne.
- Près du gate, nous activons le mode *Visual servoing*, où nous exécutons la commande développée dans 5.3.

Ce bloc reçoit en entrée les valeurs estimées de la pose du quadcopter, ainsi que les coordonnées en pixel du prochain *gate*. En sortie, il donne les valeurs désirées des angles ainsi que le la valeur désirée de la poussée.

Commande des angles Le but de ce bloc est d'asservir l'orientation désirée. Pour les qualifications d'AlphaPilot, nous avions remplacé la loi de commande développée en 3.6 par un régulateur PID présenté dans ce rapport technique , et implémenté dans ce code source. Il est à noter que le plus bas niveau de commande des vitesses angulaires et la poussée, est assuré par le régulateur intégré dans le simulateur.

6.3 Environnement de simulation

6.3.1 FlightGoggles

FlightGoggles est un environnement de réalité virtuelle unique en son genre pour la recherche et le développement de drones, construit par Massachusetts Institute of Technology (MIT). le simulateur inclut un environnement de simulation 3D haute définition photoréaliste basé sur le moteur de réalité virtuelle Unity3D . Il inclut une liaison ROS (Robotic Operating System) pour l'intégration de logiciels autonomes et des ressources Unity pour des environnements qui ressemblent à des scénarios de course de drones.

Figure 6.2: Simulateur FlightGoggles.

Dans le cadre des qualifications de AlphaPilot, FlightGoggles comporte un circuit de course de drones ressemblant aux courses de drone à vue subjective (figure x). Il inclut aussi un quadcopter simulé avec des dynamique réaliste prenant en considération les bruits et erreurs de modélisation.

Un ensemble de capteurs extéroceptifs est aussi émulé, avec des sorties réalistes comprenant des bruits de mesure:

- Alimentation photoréaliste des caméras stéréo RGB 60Hz (comprenant des bruits et un flou de mouvement).

- Sortie de l'algorithme de détection du gate dans le repère de la caméra.
- Emplacements des marqueurs infrarouge dans le repère de la caméra.
- Données bruitées de l'IMU.
- Télémètre laser bruité orienté vers le bas pour faciliter l'estimation de l'altitude.

6.3.2 Configuration logicielle et matérielle

Afin d'implémenter notre architecture décrite précédemment, nous avons utilisé plusieurs ressources logiciels réparties sur plusieurs couches:

- *Ubuntu 16.04.6* comme système d'exploitation pour la compatibilité avec le simulateur FlightGoggles.
- *ROS Kinetic Kame* comme supéieure qui a pour but principale de gérer le parallélisme et le multiprocessing exigé dans l'application.
- Au dessus de ROS, nous avons les différents **processus** de **perception**, **navigation** et **contrôle**.

La configuration logicielle lourde et complexe demande des ressource matérielles importantes. Pour répondre à ce besoin, Nous avons utilisé l'instance payante **AWS g3s.xlarge** [2]. Cette instance permet d'accéder à des unités **GPU NVIDIA Tesla M60** disposant de 8 Go de mémoire GPU. Cette instance est idéale pour les applications à usage intensif des ressources graphiques, comme c'est le cas avec le simulateur FlightGoggles.Elle dispose aussi de pilotes nécessaires à l'entraînement des réseaux de neurones profonds.

6.3.3 implémentation de l'architecture sous ROS

La figure suivante représente un graphe relationnel entre les différents composants du système (généré en temps réel en utilisant l'outil intégré de **ROS rqt-graph**).

[2]https://aws.amazon.com/fr/ec2/instance-types/g3/

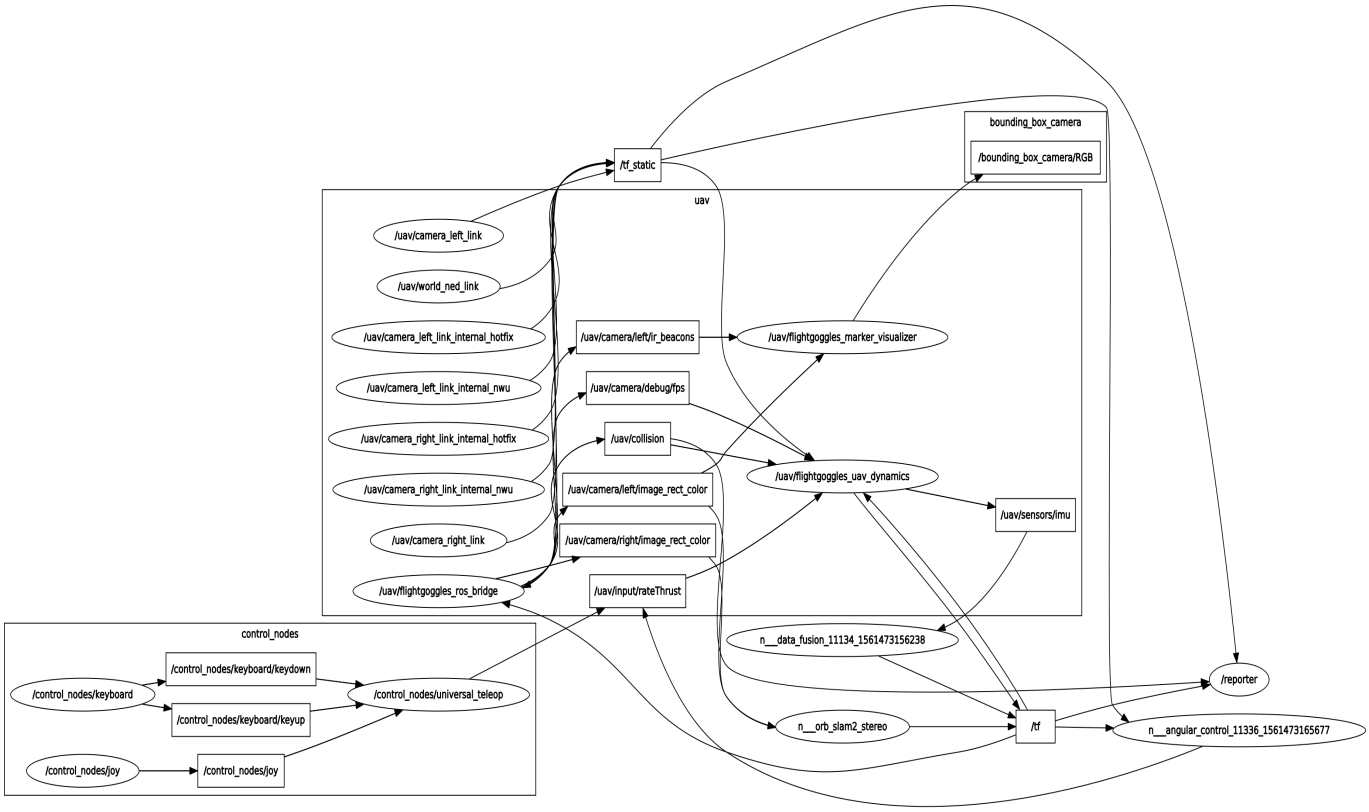


Figure 6.3: Grahe relationnel représentant l'implémentation de l'architecture.

- Le bloc **UAV** correspond au *namespace* fourni par FlightGoggles et qui regroupe les noeuds de la dynamique du quadrirotor ainsi que les noeuds des différents capteurs.
- L'estimation de l'état est effectué par les noeuds **n-orb-slam2-stereo** et **n-data-fusion** implémentant ORB-SLAM2 et le filtre complémentaire respectivement.
- Les algorithmes de navigation et de contrôle sont implémenté dans le noeud **n-angular-control**.

  Les autres noeuds sont utilisés pour l'évaluation et le calcul des performances pour les besoins de la compétition

6.4 Résultats

Le quadrirotor démarre de la position initiale $[18.0, -23.0, 5.3]$ et doit traverser 11 *gates*, dont la position est connue *grossièrement*, en un minimum de temps. Le parcours que doit accomplir le quadrirotor est illustré dans la figure 1.5.

Notre stratégie de navigation a permis au quadrirotor de franchir ces 11 *gates* comme le montre la figure suivante.

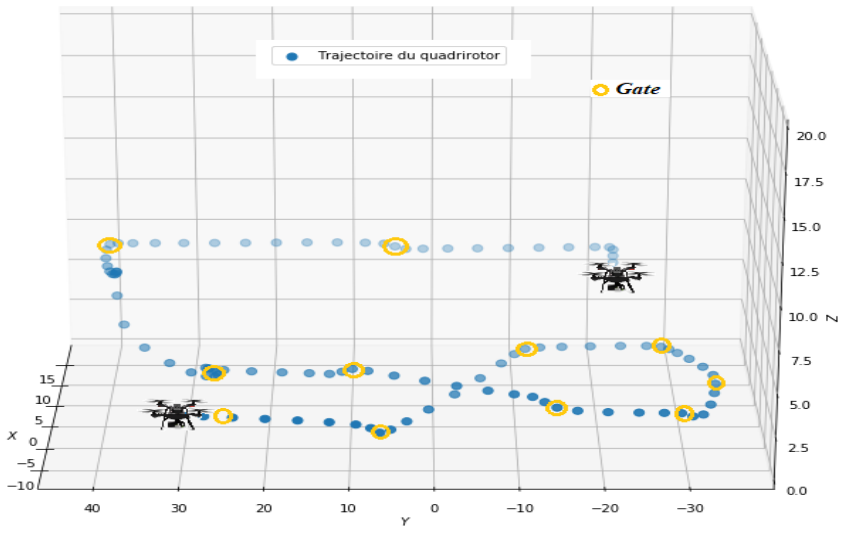


Figure 6.4: Trajectoire 3D du quadrirotor sur le parcours nominal.

La distance parcourue est de 240 m, le profil de vitesse affiché est le suivant:

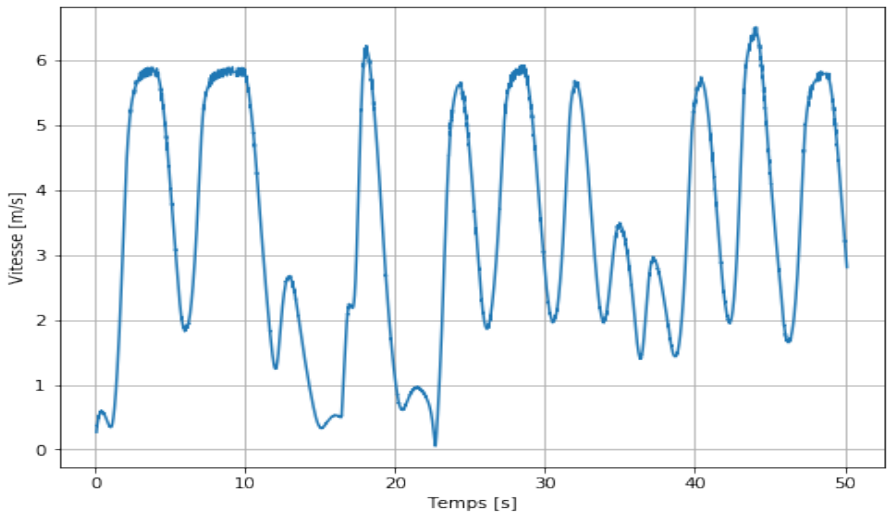


Figure 6.5: Profil de vitesse du quadrirotor sur le parcours nominal.

La vitesse maximale atteinte est $6.50\,m/s$. Le drone doit toutefois ralentir avant de franchir chaque *gate* ce qui explique les oscillations observées.

L'estimation de la position fournie par ORB-SLAM2 sont présentées ci-dessous:

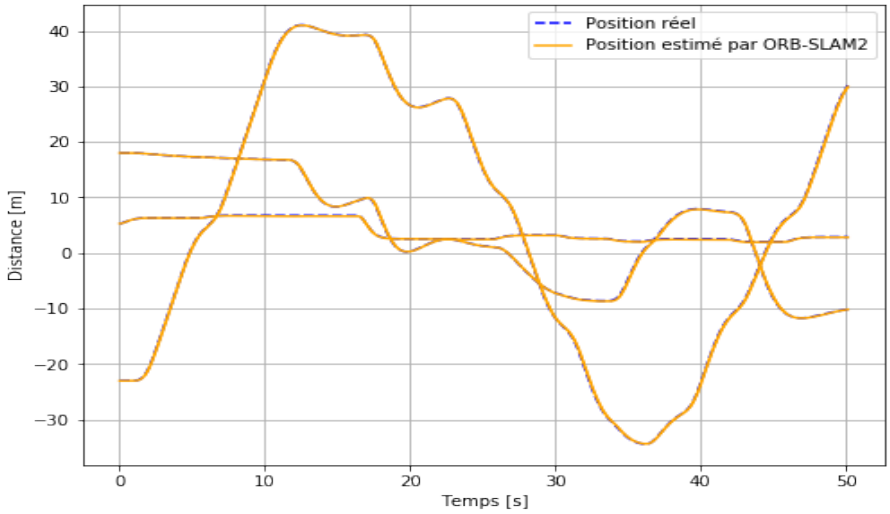


Figure 6.6: Évolution temporelle de la position réelle du quadrirotor et son estimée par ORB-SLAM2.

La distribution des erreurs d'estimation suivante montre que celles ci sont assez fiables:

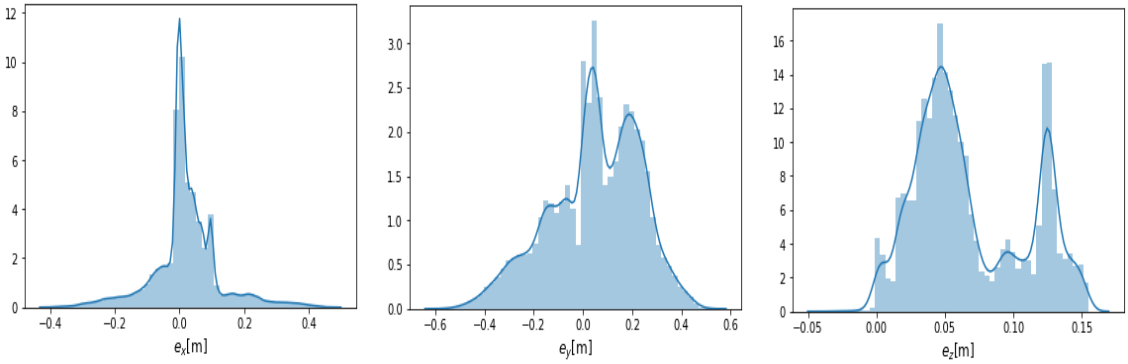


Figure 6.7: Distribution de l'erreur d'estimation par ORB-SLAM2.

Les angles estimés grâce à l'imu sont illustrés dans la figure suivante:

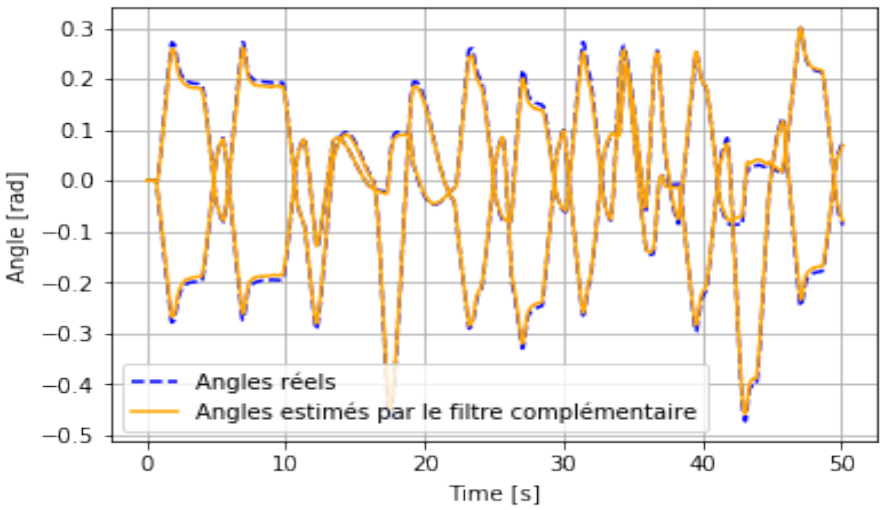


Figure 6.8: Évolution temporelle des angles réels du quadrirotor et les angles estimés par le filtre complémentaire.

Voici aussi la distribution de l'erreur entre les angles réelles et estimés:

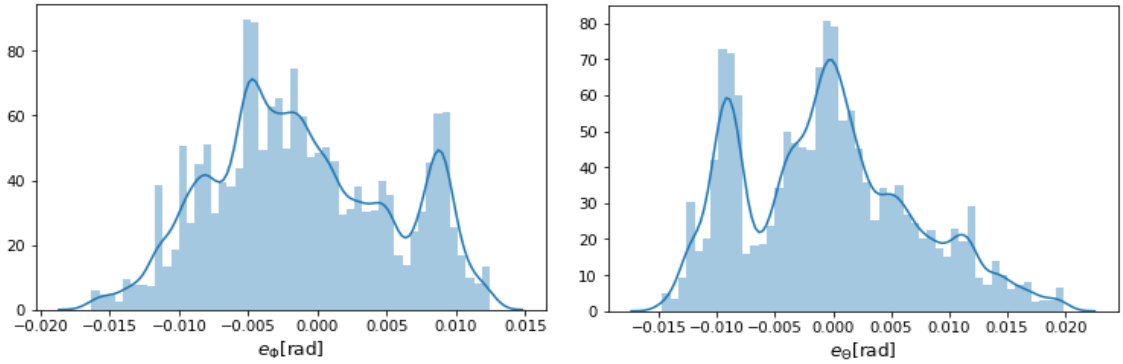


Figure 6.9: Distribution de l'erreur d'estimation par le filtre complémentaire.

Nous avons aussi récolté les données relatifs aux erreurs en pixel, horizontales et verticales, d'asservissement visuel par heuristique de guidage (section 5.3.3) pour le passage du premier *gate*. La figure suivante représente le produit de ces erreurs par la distance entre le quadrirotor et le *gate* en question.

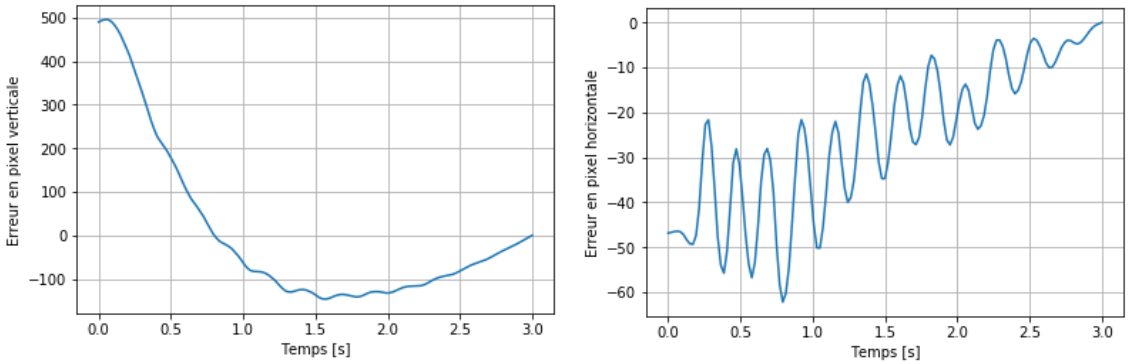


Figure 6.10: Erreurs en pixels lors de l'asservissement visuel.

La convergence exponentielle n'est pas garantie par l'heuristique, néanmoins l'erreur tend vers 0.

6.5 Conclusion

Fournir une solution d'autonomie pour des UAV fonctionnant en vol agile et rapide dans des environnement confinés nécessite une architecture modulaire divisé sur plusieurs niveaux où les différents système de perception, navigation et contrôle opère d'une façon parallèle. Des ressources logicielles et matérielles adapté à cette philosophie et supportant nativement le parallélisme sont également nécessaires.

Dans ce chapitre, nous avons présenté l'architecture globale que nous avons proposé, ainsi que les aspects techniques de l'implémentation de notre solution pour la problématique des courses de drone autonomes. Aussi, nous avons présenté quantitativement les résultats et performances des différents systèmes impliqués.

Chapitre 7

Conclusion générale

Motivés par l'engouement croissant autour de l'intelligence artificielle et la navigation visuelle, nous avons tenté à travers ce travail de contribuer à un domaine qui combine l'automatique avancée à ces deux thématiques, à savoir, les courses de drones autonomes.

Ce travail se divise en trois parties à savoir:

La commande non linéaire du quadrirotor pour permettre des vols stables, robustes et agiles, la notion d'agilité étant très importante dans le contexte des courses de drones ou alors pour la navigation dans des espaces confinés.

L'estimation de l'état du quadrirotor en utilisant seulement deux capteurs embarqués peu coûteux et pas gourmands en temps de calcul à savoir une centrale inertielle et une caméra RGB. Nous avons utilisé un filtre complémentaire pour fusionner les données du gyroscope et de l'accéléromètre. ORB-SLAM2 est quant à lui implémenté pour localiser le drone dans l'espace.

La navigation visuelle réactive en utilisant l'IBVS aidé d'un CNN pour la détection des *gates*. Le CNN entraîné a affiché des performances excellentes grâce notamment à l'architecture résiduelle.

Dans la perspective d'une continuité de ce travail, nous proposons :

- Considérer la platitude du système quadrirotor afin de générer des trajectoires injectés en *feedforward* pour obtenir un meilleur suivi de référence.
- Fusionner les informations sur l'état estimé par le filtre complémentaire et ORB-SLAM2 avec les données issues d'un modèle du quadrirotor pour une meilleure estimation, ou utiliser un algorithme de VIO qui est moins gourmand en temps de calcul et plus précis pour une application sur un quadrirotor.
- Génération et suivi d'une trajectoire 3D lors du visual servoing qui permet simultanément de garder le *gate* dans le champs de vision du quadrirotor et de satisfaire des contraintes tridimensionnelles sur la position du drone.
- Entraîner un réseau de neurones , en aval du réseau convolutionnel déjà entraîné, pour prédire une trajectoire généré en *offline* passant par les *gates* désirés.

Références

[1] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," *2013 IEEE International Conference on Robotics and Automation*, May 2013. DOI: 10.1109/icra.2013.6630809. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2013.6630809.

[2] D. Kragic and K. Daniilidis, "3-d vision for navigation and grasping," in *Springer Handbook of Robotics*, Springer, 2016, pp. 811–824.

[3] (Jun. 2019). Lockheed martin and drone racing league launch groundbreaking ai innovation challenge, [Online]. Available: https://www.herox.com/alphapilot.

[4] (Jun. 2019). Competitions — iros 2018 - international conference on intelligent robots - madrid, [Online]. Available: https://www.iros2018.org/competitions.

[5] H. Moon, Y. Sun, J. Baltes, and S. J. Kim, "The iros 2016 competitions [competitions]," *IEEE Robotics & Automation Magazine*, vol. 24, no. 1, pp. 20–29, 2017.

[6] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, *Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality*, 2019. eprint: arXiv:1905.11377.

[7] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, *Beauty and the beast: Optimal methods meet learning for drone racing*, 2018. arXiv: 1810.06224 [cs.RO].

[8] A. Bry, A. Bachrach, and N. Roy, "State estimation for aggressive flight in gps-denied environments using onboard sensing," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 1–8.

[9] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[10] S. Lynen, T. Sattler, M. Bosse, J. A. Hesch, M. Pollefeys, and R. Siegwart, "Get out of my lab: Large-scale, real-time visual-inertial localization.," in *Robotics: Science and Systems*, 2015.

[11] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," *arXiv preprint arXiv:1806.08548*, 2018.

[12] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "Pampc: Perception-aware model predictive control for quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1–8.

[13] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, "Are we ready for autonomous drone racing? the uzhfpv drone racing dataset," in *IEEE Int. Conf. Robot. Autom.(ICRA)*, 2019.

[14] S. Karaman and E. Frazzoli, "High-speed flight in an ergodic forest," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 2899–2906.

[15] A. Censi and D. Scaramuzza, "Low-latency event-based visual odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 703–710.

[16] C. Richter, W. Vega-Brown, and N. Roy, "Bayesian learning for safe high-speed navigation in unknown environments," in *Robotics Research*, Springer, 2018, pp. 325–341.

[17] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, *et al.*, "Fast, autonomous flight in gps-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[18] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," 2017.

[19] A. J. Barry, P. R. Florence, and R. Tedrake, "High-speed autonomous obstacle avoidance with pushbroom stereo," *Journal of Field Robotics*, vol. 35, no. 1, pp. 52–68, 2018.

[20] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.

[21] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2016.

[22] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 46–56, 2012.

[23] T. Özaslan, S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Inspection of penstocks and featureless tunnel-like environments using micro uavs," in *Field and Service Robotics*, Springer, 2015, pp. 123–136.

[24] J. Cacace, A. Finzi, V. Lippiello, G. Loianno, and D. Sanzone, "Aerial service vehicles for industrial inspection: Task decomposition and plan execution," *Applied Intelligence*, vol. 42, no. 1, pp. 49–62, 2015.

[25] G. Loianno, J. Thomas, and V. Kumar, "Cooperative localization and mapping of mavs using rgb-d sensors," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 4021–4028.

[26] F. Forte, R. Naldi, A. Macchelli, and L. Marconi, "Impedance control of an aerial manipulator," in *2012 American Control Conference (ACC)*, IEEE, 2012, pp. 3839–3844.

[27] J. Thomas, G. Loianno, K. Daniilidis, and V. Kumar, "Visual servoing of quadrotors for perching by hanging from cylindrical objects," *IEEE robotics and automation letters*, vol. 1, no. 1, pp. 57–64, 2015.

[28] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, *et al.*, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832–841, 2012.

[29] R. Mahony, R. W. Beard, and V. Kumar, "Modeling and control of aerial robots," in *Springer handbook of robotics*, Springer, 2016, pp. 1307–1334.

[30] M. Faessler, "Quadrotor control for accurate agile flight," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1425–1440, 2018.

[31] N. A. Chaturvedi, A. K. Sanyal, and N. H. McClamroch, "Rigid-body attitude control," *IEEE control systems magazine*, vol. 31, no. 3, pp. 30–51, 2011.

[32] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 2520–2525.

[33] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*, IEEE, 2010, pp. 5420–5425.

[34] E. Fresk and G. Nikolakopoulos, "Full quaternion based attitude control for a quadrotor," in *2013 European Control Conference (ECC)*, IEEE, 2013, pp. 3864–3869.

[35] M. Bangura, R. Mahony, H. Lim, H. Jin Kim, *et al.*, "An open-source implementation of a unit quaternion based attitude and trajectory tracking for quadrotors," 2014.

[36] E. Reyes-Valeria, R. Enriquez-Caldera, S. Camacho-Lara, and J. Guichard, "Lqr control for a quadrotor using unit quaternions: Modeling and simulation," in *CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing*, IEEE, 2013, pp. 172–178.

[37] C. De Crousaz, F. Farshidian, M. Neunert, and J. Buchli, "Unified motion control for dynamic quadrotor maneuvers demonstrated on slung load and rotor failure tasks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 2223–2229.

[38] M. Hehn and R. D'Andrea, "A flying inverted pendulum," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 763–770.

[39] D. Brescianini, M. Hehn, and R. D'Andrea, "Quadrocopter pole acrobatics," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 3472–3479.

[40] H. K. Khalil and J. W. Grizzle, *Nonlinear systems*. Prentice hall Upper Saddle River, NJ, 2002, vol. 3.

[41] S. Kumar and R. Gill, "Path following for quadrotors," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, IEEE, 2017, pp. 2075–2081.

[42] D. Brescianini, M. Hehn, and R. D'Andrea, "Nonlinear quadrocopter attitude control: Technical report," ETH Zurich, Tech. Rep., 2013.

[43] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2017.

[44] J. Ferrin, R. Leishman, R. Beard, and T. McLain, "Differential flatness based control of a rotorcraft for aggressive maneuvers," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Ieee, 2011, pp. 2688–2693.

[45] M. Kamel, M. Burri, and R. Siegwart, "Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.

[46] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *2016 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2016, pp. 1398–1404.

[47] M. Bangura and R. Mahony, "Real-time model predictive control for quadrotors," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11 773–11 780, 2014.

[48] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart, "Fast nonlinear model predictive control for multicopter attitude tracking on so (3)," in *2015 IEEE Conference on Control Applications (CCA)*, IEEE, 2015, pp. 1160–1166.

[49] G. V. Raffo, M. G. Ortega, and F. R. Rubio, "Mpc with nonlinear *H-infinity* control for path tracking of a quad-rotor helicopter," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 8564–8569, 2008.

[50] M. W. Mueller and R. D'Andrea, "A model predictive controller for quadrocopter state interception," in *2013 European Control Conference (ECC)*, IEEE, 2013, pp. 1383–1389.

[51] M. Hehn and R. D'Andrea, "Real-time trajectory generation for interception maneuvers with quadrocopters," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 4979–4984.

[52] J. W. Langelaan, "State estimation for autonomous flight in cluttered environments," *Journal of guidance, control, and dynamics*, vol. 30, no. 5, pp. 1414–1426, 2007.

[53] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 2502–2509.

[54] G. Balamurugan, J. Valarmathi, and V. Naidu, "Survey on uav navigation in gps denied environments," in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, IEEE, 2016, pp. 198–204.

[55] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[56] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2014, pp. 15–22.

[57] M. Cummins and P. Newman, "Fab-map: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

[58] R. O. Castle, G. Klein, and D. W. Murray, "Combining monoslam with object recognition for scene augmentation using a wearable camera," *Image and Vision Computing*, vol. 28, no. 11, pp. 1548–1556, 2010.

[59] S. Pillai and J. Leonard, "Monocular slam supported object recognition," *arXiv preprint arXiv:1506.01732*, 2015.

[60] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "Slam++: Simultaneous localisation and mapping at the level of objects," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 1352–1359.

[61] S. Lynen, T. Sattler, M. Bosse, J. A. Hesch, M. Pollefeys, and R. Siegwart, "Get out of my lab: Large-scale, real-time visual-inertial localization.," in *Robotics: Science and Systems*, 2015.

[62] T. Yang, P. Li, H. Zhang, J. Li, and Z. Li, "Monocular vision slam-based uav autonomous landing in emergencies and unknown environments," *Electronics*, vol. 7, no. 5, p. 73, 2018.

[63] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1052–1067, 2007.

[64] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*, Springer, 2014, pp. 834–849.

[65] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[66] D. Weikersdorfer, D. B. Adrian, D. Cremers, and J. Conradt, "Event-based 3d slam with a depth-augmented dynamic vision sensor," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 359–364.

[67] H. Rebecq, T. Horstschäfer, G. Gallego, and D. Scaramuzza, "Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real time," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 593–600, 2016.

[68] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, IEEE, 2011, pp. 155–160.

[69] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2938–2946.

[70] F. Walch, C. Hazirbas, L. Leal-Taixe, T. Sattler, S. Hilsenbeck, and D. Cremers, "Image-based localization using lstms for structured feature correlation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 627–637.

[71] R. Clark, S. Wang, A. Markham, N. Trigoni, and H. Wen, "Vidloc: A deep spatio-temporal model for 6-dof video-clip relocalization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6856–6864.

[72] K. R. Konda and R. Memisevic, "Learning visual odometry with a convolutional network.," in *VISAPP (1)*, 2015, pp. 486–490.

[73] G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia, "Exploring representation learning with cnns for frame-to-frame ego-motion estimation," *IEEE robotics and automation letters*, vol. 1, no. 1, pp. 18–25, 2015.

[74] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 224–236.

[75] J. R. Rambach, A. Tewari, A. Pagani, and D. Stricker, "Learning to fuse: A deep learning approach to visual-inertial camera pose estimation," in *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2016, pp. 71–76.

[76] A. Valada, N. Radwan, and W. Burgard, "Deep auxiliary learning for visual localization and odometry," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 6939–6946.

[77] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[78] F. Bonin-Font, A. Ortiz, and G. Oliver, "Visual navigation for mobile robots: A survey," *Journal of intelligent and robotic systems*, vol. 53, no. 3, pp. 263–296, 2008.

[79] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "Svo: Semidirect visual odometry for monocular and multicamera systems," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 249–265, 2016.

[80] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2015, pp. 298–304.

[81] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[82] V. Usenko, J. Engel, J. Stückler, and D. Cremers, "Direct visual-inertial odometry with stereo cameras," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 1885–1892.

[83] T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart, "Maplab: An open framework for research in visual-inertial mapping and localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1418–1425, 2018.

[84] M. Fehr, T. Schneider, M. Dymczyk, J. Sturm, and R. Siegwart, "Visual-inertial teach and repeat for aerial inspection," *arXiv preprint arXiv:1803.09650*, 2018.

[85] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[86] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, "A direct visual servoing-based framework for the 2016 iros autonomous drone racing challenge," *Journal of Field Robotics*, vol. 35, no. 1, pp. 146–166, 2018.

[87] D. Lee, T. Ryan, and H. J. Kim, "Autonomous landing of a vtol uav on a moving platform using image-based visual servoing," in *2012 IEEE international conference on robotics and automation*, IEEE, 2012, pp. 971–976.

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

[89] J. Thomas, G. Loianno, K. Sreenath, and V. Kumar, "Toward image based visual servoing for aerial grasping and perching," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 2113–2118.

[90] J. Pestana, J. L. Sanchez-Lopez, S. Saripalli, and P. Campoy, "Computer vision based general object following for gps-denied multirotor unmanned vehicles," in *2014 American Control Conference*, IEEE, 2014, pp. 1886–1891.

[91] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[92] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[93] P. Viola, M. Jones, *et al.*, "Rapid object detection using a boosted cascade of simple features," *CVPR (1)*, vol. 1, pp. 511–518, 2001.

[94] J. Illingworth and J. Kittler, "A survey of efficient hough transform methods.," in *Alvey Vision Conference*, 1987, pp. 1–8.

[95] S. Renals and P. Swietojanski, "Neural networks for distant speech recognition," in *2014 4th joint workshop on hands-free speech communication and microphone arrays (HSCMA)*, IEEE, 2014, pp. 172–176.

[96] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[97] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[98] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[99] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[100] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[101] T. Shen, D. Xu, W. Lu, and J. Yang, "Robust visual servoing with spheres projected in cameras obeying the unified model," in *2017 Chinese Automation Congress (CAC)*, IEEE, 2017, pp. 5602–5607.

[102] T. Hamel and R. Mahony, "Visual servoing of an under-actuated dynamic rigid-body system: An image-based approach," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 2, pp. 187–198, 2002.

[103] N. Guenard, T. Hamel, and R. Mahony, "A practical visual servo control for an unmanned aerial vehicle," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 331–340, 2008.

[104] E. Altug, J. P. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 1, 2002, pp. 72–77.

[105] L. Mejias, S. Saripalli, P. Campoy, and G. S. Sukhatme, "Visual servoing of an autonomous helicopter in urban areas using feature tracking," *Journal of Field Robotics*, vol. 23, no. 3-4, pp. 185–199, 2006.

[106] R. Ozawa and F. Chaumette, "Dynamic visual servoing with image moments for a quadrotor using a virtual spring approach," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 5670–5676.

[107] H. Jabbari Asl, G. Oriolo, and H. Bolandi, "Output feedback image-based visual servoing control of an underactuated unmanned aerial vehicle," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 228, no. 7, pp. 435–448, 2014.

[108] C. Potena, D. Nardi, and A. Pretto, "Effective target aware visual navigation for uavs," in *2017 European Conference on Mobile Robots (ECMR)*, IEEE, 2017, pp. 1–7.

[109] Wikipédia, *Apprentissage profond — wikipédia, l'encyclopédie libre*, [En ligne; Page disponible le 10-mai-2019], 2019. [Online]. Available: http://fr.wikipedia.org/w/index.php?title=Apprentissage_profond&oldid=159156662.

[110] A. Loquercio, A. I. Maqueda, C. R. del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[111] D. K. Kim and T. Chen, "Deep neural network for real-time autonomous indoor navigation," *arXiv preprint arXiv:1511.04668*, 2015.

[112] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodrıéguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2015.

[113] M. Muller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Teaching uavs to race: End-to-end regression of agile controls in simulation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 11–29.

[114] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 1433–1440.

[115] G. Li, M. Mueller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Oil: Observational imitation learning," *arXiv preprint arXiv:1803.01129*, 2018.

[116] T. Hamel, R. Mahony, R. Lozano, and J. Ostrowski, "Dynamic modelling and configuration stabilization for an x4-flyer.," *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 217–222, 2002.

[117] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, IEEE, vol. 5, 2004, pp. 4393–4398.

[118] R. Mahony and V. Kumar, "Aerial robotics and the quadrotor [from the guest editors]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 19–19, 2012.

[119] E. Altug, J. P. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 1, 2002, pp. 72–77.

[120] P. Pounds, R. Mahony, P. Hynes, and J. M. Roberts, "Design of a four-rotor aerial robot," in *Proceedings of the 2002 Australasian Conference on Robotics and Automation (ACRA 2002)*, Australian Robotics & Automation Association, 2002, pp. 145–150.

[121] L. R. G. Carrillo, A. E. D. López, R. Lozano, and C. Pégard, "Modeling the quad-rotor mini-rotorcraft," in *Quad Rotorcraft Control*, Springer, 2013, pp. 23–34.

[122] R. W. Prouty, *Helicopter performance, stability, and control*. 1995.

[123] J. G. Leishman, "Principles of helicopter aerodynamics, 2006," *Google Scholar*, pp. 730–731,

[124] G. J. Leishman, *Principles of helicopter aerodynamics with CD extra*. Cambridge university press, 2006.

[125] P.-J. Bristeau, P. Martin, E. Salaün, and N. Petit, "The role of propeller aerodynamics in the model of a quadrotor uav," in *2009 European Control Conference (ECC)*, IEEE, 2009, pp. 683–688.

[126] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a large quadrotor robot," *Control Engineering Practice*, vol. 18, no. 7, pp. 691–699, 2010.

[127] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[128] (Jul. 2019). Euler's laws of motion - wikipedia, the free encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Euler%27s_laws_of_motion.

[129] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*, IEEE, 2010, pp. 5420–5425.

[130] R. M. Murray, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[131] (Jun. 2019). Rotation matrix : Wikipedia , the free encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Rotation_matrix.

[132] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.

[133] (Jun. 2019). Vicon : Motion capture systems, [Online]. Available: https://www.vicon.com/.

[134] (Jun. 2019). Optitrack : Motion capture systems, [Online]. Available: https://optitrack.com/.

[135] K. Tuck, "Tilt sensing using linear accelerometers," *Freescale semiconductor application note AN3107*, 2007.

[136] J. L. Marins, X. Yun, E. R. Bachmann, R. B. McGhee, and M. J. Zyda, "An extended kalman filter for quaternion-based orientation estimation using marg sensors," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, IEEE, vol. 4, 2001, pp. 2003–2011.

[137] L. Rossi, C. De Gaetani, D. Pagliari, E. Realini, M. Reguzzoni, L. Pinto, *et al.*, "Comparison between rgb and rgb-d cameras for supporting low-cost gnss urban navigation," 2018.

[138] H. Mannila and P. Orponen, *Algorithms and Applications*. Springer, 2010.

[139] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[140] (Jun. 2019). Matlab : Single camera calibrator app, [Online]. Available: https://www.mathworks.com/help/vision/ug/single-camera-calibrator-app.html.

[141] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, 2000.

[142] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[143] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, Springer, 2006, pp. 404–417.

[144] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[145] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[146] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*, Springer, 2006, pp. 430–443.

[147] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*, Springer, 2010, pp. 778–792.

[148] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[149] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Apr. 2015, ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. [Online]. Available: http://dx.doi.org/10.1007/s11263-015-0816-y.

[150] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, 2018. arXiv: 1804.02767 [cs.CV].

[151] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, pp. 2539–2544, 2018.

[152] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[153] A. Loquercio, A. I. Maqueda, C. R. del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[154] A. Mandelbaum and D. Weinshall, *Distance-based confidence score for neural network classifiers*, 2017. arXiv: 1709.09844 [cs.AI].

[155] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[156] (Jun. 2019). Tensorflow : An end-to-end open source machine learning platform, [Online]. Available: https://www.tensorflow.org/.

[157] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[158] T. Hamel and R. Mahony, "Image based visual servo control for a class of aerial robotic systems," *Automatica*, vol. 43, no. 11, pp. 1975–1983, 2007.