Planification du mouvement 3D en temps réel de drones autonomes dans des environnements dynamiques inconnus

Julien Margraff

UNIVERSITÉ DE LIMOGES

École Doctorale 610 - SCIENCES ET INGÉNIERIE DES SYSTÈMES, MATHÉMATIQUES, INFORMATIQUE (SISMI)

Laboratoire XLIM - UMR CNRS/Université de Limoges 7252

29/03/2022

Thèse

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE LIMOGES

Discipline : Science et Ingénierie pour l'Information, Mathématiques

présentée par

Julien MARGRAFF

Planification du mouvement 3D en temps réel de drones autonomes dans des environnements dynamiques inconnus

Jury :

Vincent Frémont, Professeur, ARMEN team
José Fermi Guerrero – Castellanos , Profeseur, Benemérita Universidad Autónoma de Puebla
Pedro Castillo Garcia, chargé de recherche, Heudiasyc Laboratory
Juan Antonio Escareno Castro, Maitre de conférence , Laboratoire XLIM

Thèse dirigée par Ouiddad Labbani-Igbida, Professeur et Joanny Stéphant, Maître de conférences

Table des matières

Introduction générale

La première utilisation d'engin volant sans humain était un ballon en 1849 [Pisano, 2001]. Des zeppelins autonomes ont ensuite vu le jour pendant la Première Guerre mondiale. Les recherches sur les avions autonomes ont aussi avancé à la même période, mais ce n'est que pendant la Deuxième Guerre mondiale que les drones militaires pour la surveillance à distance ont été utilisés. Avec l'amélioration des technologies, la réduction de taille des composants et l'amélioration de la capacité et du poids des batteries, les drones sont de plus en plus petits et efficaces. Ils connaissent un essor important dans de nombreuses applications telles que la livraison, l'agriculture de précision, la recherche et le sauvetage, l'inspection et la surveillance environnementale. La technologie des drones, désormais peu coûteuse et accessible, évolue en permanence et fait l'objet de plusieurs nouvelles utilisations dans le monde, permettant d'ouvrir de nouveaux accès à des espaces très contraints pour l'humain. Au-delà des aspects technologiques, les législations se doivent de reconnaître ces nouvelles formes de robots et de s'y adapter.

L'utilisation des drones dans des environnements difficiles d'accès les amène parfois à être en dehors du champ de vision d'un être humain et ne peuvent donc plus être pilotés. C'est pourquoi il est nécessaire d'automatiser le déplacement et d'utiliser des planificateurs de trajectoires. Ces derniers, appliqués aux drones, ont évolué très vite. Initialement inspirés des travaux de la robotique mobile terrestre, des planificateurs 2D étaient utilisés sur les drones en contraignant leur déplacements dans un plan à une altitude fixe. Ensuite, la planification en 2.5D a permis d'étendre les possibilités de mouvement en projetant l'environnement 3D au sol. Avec l'amélioration des ordinateurs embarqués et des batteries, il est possible aujourd'hui de considérer, du moins localement, la géométrie 3D de l'espace et donc d'utiliser toutes les informations présentes dans l'environnement d'évolution du drone.

Problématique

Le travail présenté dans ce mémoire a pour but de construire un planificateur temps réel de trajectoire 3D compatible avec les contraintes dynamiques de vol, et capable d'éviter les obstacles en utilisant une bulle 3D de perception pour réaliser un vol autonome.

Contributions

On considère des environnements 3D dynamiques, inconnus, ou partiellement connus. Pour ce faire, un problème d'optimisation est formulé et résolu en temps réel pour déterminer une trajectoire dans l'horizon de perception du drone, tout en intégrant un ensemble de contraintes. Ces dernières tiennent compte de la dynamique de vol du drone, de la vitesse du drone et de celles des obstacles environnants. Le processus d'optimisation génère, pour chaque nouvelle position estimée du drone, une trajectoire de Bézier à suivre, et intègre les nouvelles consignes pour gérer les transitions entre deux trajectoires successives. La dynamique des obstacles est estimée afin d'affiner l'évitement d'obstacle lors de l'optimisation.

Organisation du mémoire

Chapitre 1 - Etat de l'art

Dans le premier chapitre, un état de l'art général des différents aspects liés aux drones est réalisé. On focalisera en particulier sur les problématiques de perception et de localisation, nécessaires pour construire un vol autonome. On présentera ensuite un parcours des travaux sur la génération de chemin, de trajectoire et/ou de mouvement, en lien avec la problématique centrale de la thèse.

Chapitre 2 - Modélisation et contrôle d'un drone

Le second chapitre présente le modèle du drone et les stratégies de contrôle utilisés dans le reste du manuscrit. En particulier, pour réaliser un suivi de trajectoire désirée, on considérera différents types de contrôleurs, du simple contrôleur PID sur modèle linéarisé, au contrôle par back-stepping pour assurer la stabilité et contrôle prédictif sur un certain horizon d'observation.

Chapitre 3 - Planificateur de trajectoire basé sur les courbes de Bézier

Le troisième chapitre traite du problème de la planification de trajectoire, qui est la problématique centrale de la thèse. On considérera différentes hypothèses sur l'environnement : connu, partiellement connu ou totalement inconnu. Après avoir introduit le calcul d'un chemin global, nous présenterons comment trouver une trajectoire locale optimale sous forme de courbes de Bézier. Notre approche se concentrera ensuite sur l'intégration de différents types de contraintes dans le processus d'optimisation : des contraintes liées à la dynamique du drone, des contraintes de sécurité de vol et de dynamique des obstacles. On s'intéressera également à la continuité des trajectoires consignes calculées par le processus d'optimisation, relativement

à l'état du drone et à la transition de consignes de trajectoires successives. Une estimation de la vitesse des obstacles sera réalisée et intégrée à l'optimisation afin d'adapter localement les trajectoires d'évitement.

Chapitre 4 - Validation en environnements réalistes

Les résultats de simulations et d'expérimentations sont présentés dans ce chapitre pour valider l'approche de planification de trajectoires et d'évitement d'obstacles dans des environnements variés. Dans un premier temps, chaque élément du planificateur sera validé de manière statistique, en simulation, dans des environnements réalistes. La génération de trajectoires sera ensuite testée dans un cas réel sur un ordinateur embarqué. La commande générée en simulation sera déployée sur un drone réel. Le fonctionnement informatique temps réel de chaque élément sur un ordinateur embarqué sera validé sur une plateforme 2D.

Chapitre 1

Etat de l'art

1.1 Introduction

Il existe un grand nombre de technologies de fabrication de drone autonome. Les plus agiles sont les drones à décollage vertical. Ils sont équipés de rotors et on peut encore les différencier en fonction du nombre de rotors et de leur positionnement. Les drones à voilure fixe sont plus efficaces en terme de consommation d'énergie, ce qui leur permet de voler sur de plus grandes distances mais avec moins d'agilité. Il existe également des configurations convertibles, qui prennent le meilleur des deux mondes. La dernière configuration utilise un ballon dirigeable pour réduire la consommation d'énergie du drone.

— Configuration à 2 rotors

Les hélicoptères sont parmi les objets volants à 2 rotors les plus communs. Comme le montre la Figure 1.1, ils sont en général équipés d'un rotor produisant la poussée et d'un rotor de queue pour compenser le couple généré par le rotor principal et contrôler le lacet.

FIGURE 1.1 – Hélicoptère standard avec un rotor de poussée et un rotor de queue

Pour augmenter la force de poussée, certains hélicoptères utilisent un rotor de queue

permettant de générer une force de poussée comme le Chinook présenté sur la Figure 1.2. La poussée étant plus importante, le rotor peut porter des charges plus lourdes.

FIGURE 1.2 – Exemple de drone bi-rotor vertical

L'hélicoptère de la Figure 1.3, est plus compact car la rotation des rotors est synchronisée. Il est donc plus compact et plus rapide.

FIGURE 1.3 – Exemple de drone bi-rotor très compact

— Configuration à 3 rotors
Les drones miniatures sont souvent équipés d'une plateforme centrale, supportant une batterie, un autopilote et les bras avec les rotors. Un drone à 3 rotors est présenté sur la Figure 1.4. Avec seulement 3 rotors, il est complexe de contrôler les déplacements dans l'espace et le lacet. Il est donc souvent nécessaire d'avoir un rotor qui possède une rotation en plus autour de son bras. Cela permet de contrôler le lacet, de compenser les effets gyroscopiques et les moments induits par les autres rotors.

FIGURE 1.4 – Exemple de drone tri-rotor

Il existe également une configuration à 6 rotors (Figure 1.5). Elle se trouve néanmoins dans cette partie car des hélices contrarotatives sont installées. Ce montage permet de compenser les moments sur chaque bras du drone.

FIGURE 1.5 – Schéma de drone tri-rotor à double hélice

— Configuration à 4 rotors
La configuration de drone miniature la plus commune est celle à 4 rotors (Figure 1.6) sur 4 bras. Il y a deux possibilités qui dépendent du placement de l'axe de roulis (axe X). Il peut être entre 2 bras pour une configuration en "x" ou colinéaire avec un bras pour une configuration en "+". Différentes vitesses de rotation sont appliquées pour générer

les différents moments qui contrôlent le mouvement du drone. Il est possible d'avoir des configurations avec différents angles entre les bras.

FIGURE 1.6 – Exemple de drone quad-rotor

— Configuration à 6 rotors
Les configurations à 6 rotors permettent de générer plus de poussée. De même que pour les 4 rotors, il est possible d'avoir l'axe de roulis entre 2 bras ou selon un bras. L'écart entre les bras est en général de 60 degrés mais peut être différent.

FIGURE 1.7 – Exemple de drone hexa-rotor

Comme pour les tri-rotors, il existe des configurations avec 2 hélices contrarotatives sur chaque bras pour limiter les couples indésirables (Figure 1.8).

FIGURE 1.8 – Exemple de drone hexa-rotor à double hélice

— Configuration à 8 rotors
Les octo-rotors sont des drones de plus grande dimension qui peuvent porter des charges non négligeables.

FIGURE 1.9 – Exemple de drone octo-rotor

— Configuration à 12 rotors
Il est compliqué de mettre 12 rotors sur le même plan. Il est donc préférable de mettre un certain nombre de rotor à un autre niveau. Comme sur la Figure 1.10, il y a un niveau de type quad-rotor et un autre niveau de type octo-rotor.

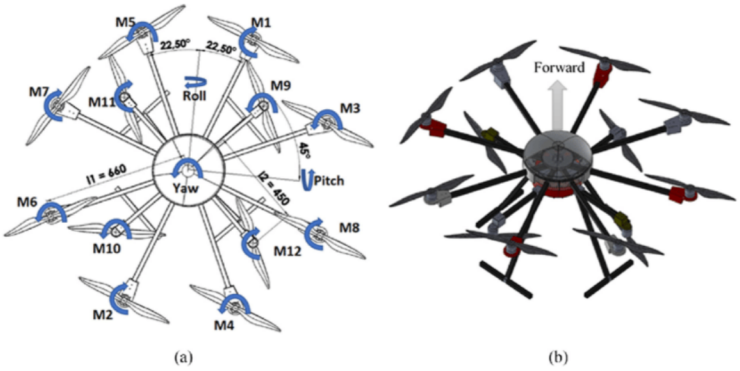


FIGURE 1.10 – Exemple de drone à 12 rotors

— Drones à voilure fixe
Ils utilisent une aile pour avoir assez de surface et créer de la portance. Il est néanmoins parfois difficile pour ce type de drones de décoller. Certains sont équipés d'un système de propulsion pour avancer et ainsi créer la portance (Figure 1.11).

FIGURE 1.11 – Exemple de drone à voilure fixe avec rotor

Les planeurs quant-à-eux n'utilisent pas forcément de système de propulsion. Il est donc nécessaire de les lancer.

FIGURE 1.12 – Exemple de drone à voilure fixe sans rotor (CAM-FLYERG)

— Configuration avec des "tilt-rotors"
Pour être plus agiles, certains drones sont hybrides. Leurs bras sont équipés de moteurs pour modifier l'orientation des hélices. Cela permet de passer d'un vol à voilure fixe à un vol par voilure tournante (Figure 1.13) pour réaliser le décollage vertical puis voler en utilisant peu d'énergie.

FIGURE 1.13 – Véhicule à rotor convertible pouvant passer d'un mode de décollage vertical à un mode à voilure fixe

Grâce au "tilt-rotor" (Figure 1.14), il est possible de se déplacer sans modifier l'attitude de la plateforme. Il n'est plus nécessaire au drone de se pencher pour faire des déplacements latéraux.

FIGURE 1.14 – Exemple de quadrotor avec tilt-rotors

— Dirigeables
Les dirigeables consomment très peu d'énergie. Le gaz présent dans le ballon leur permet de voler sur une longue durée en consommant peu d'énergie. Un système de propulsion est ensuite utilisé pour se déplacer de manière linéaire. Ils permettent le déplacement de lourdes charges.

FIGURE 1.15 – Exemple de ballon dirigeable

— Configuration modulaire à plusieurs drones
Il existe également des configurations plus originales. Par exemple un drone se déplaçant comme un serpent (Figure 1.16) a été créé avec des rotors positionnés le long de son corps. Cela lui permet de modifier la forme de son corps pour s'adapter à des passages ou encercler un objet à porter.

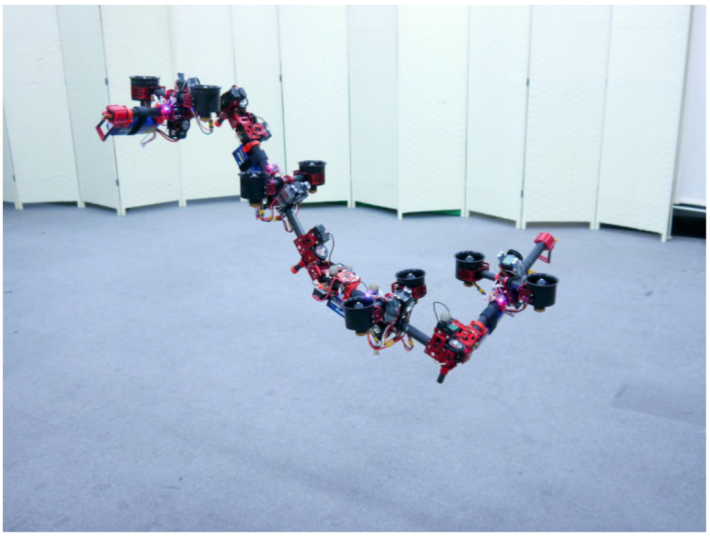


FIGURE 1.16 – Le drone dragon [Zhao et al., 2018] constitué de plusieurs segments

Une plateforme modulaire a également été imaginée. Plusieurs quad-rotors ont la possibilité de s'assembler pour créer différentes formes. Ils peuvent ainsi augmenter la portée globale de la plateforme. En adaptant la forme à un objet ils peuvent même attraper un objet.

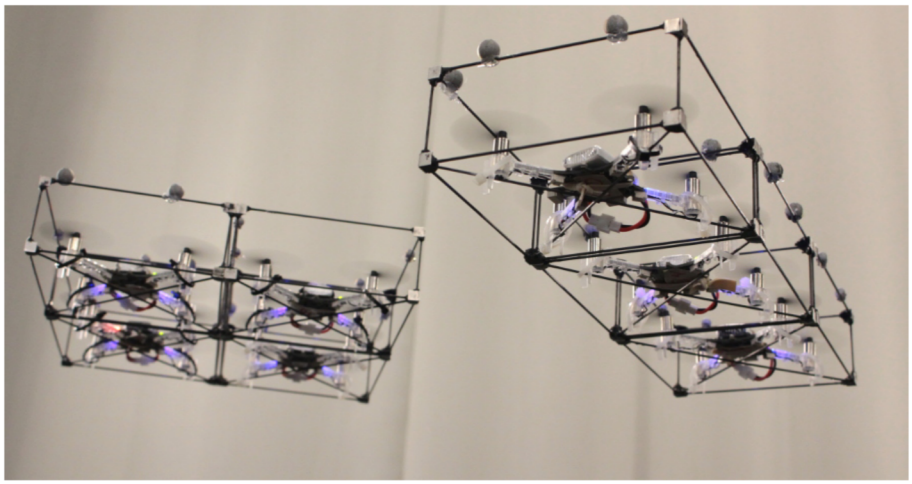


FIGURE 1.17 – Exemple de drone de configuration modulaire

De nos jours, les drones sont de plus en plus utilisés. Ils sont amenés à réaliser des missions comme de l'inspection de bâtiments [Luo et al., 2017], de l'observation de zone de catastrophe, d'incendie, de champs agricoles ou encore de la manipulation d'objets. Parfois même, des vols en intérieur sont réalisés pour explorer des grottes ou inspecter des hangars. Lors de ces missions, les drones peuvent être hors du champ de vue d'un opérateur et ne pas avoir accès à un bon réseau de communication. Dans ce type de situation, les drones ont besoin d'être autonomes.

Rendre un drone autonome demande de prendre en compte plusieurs éléments. Il faut tout d'abord pouvoir se localiser dans n'importe quelle situation. Il est ensuite nécessaire d'adapter son chemin en fonction des missions, mais aussi au regard de l'environnement qui entoure le drone. Cela doit être ajusté en fonction de la dynamique de l'environnement et des contraintes géométriques, c'est-à-dire en fonction de la vitesse de déplacement du drone, de la dynamique des obstacles, etc. Ce type d'information n'est malheureusement pas toujours disponible, pour les tâches d'inspection et d'exploration par exemple. Même avec un chemin qui prend en compte un certain nombre d'éléments, il est toujours nécessaire d'utiliser un contrôleur pour générer les commandes moteurs appliquées au drone. Par exemple, ce contrôleur devra prendre en compte un certain nombre de perturbations. [Grzonka et al., 2012] propose une analyse de la configuration nécessaire pour créer un drone autonome en intérieur. De même, [Rigter et al., 2019] présente une solution pour le vol en environnement encombré, sans localisation GPS.

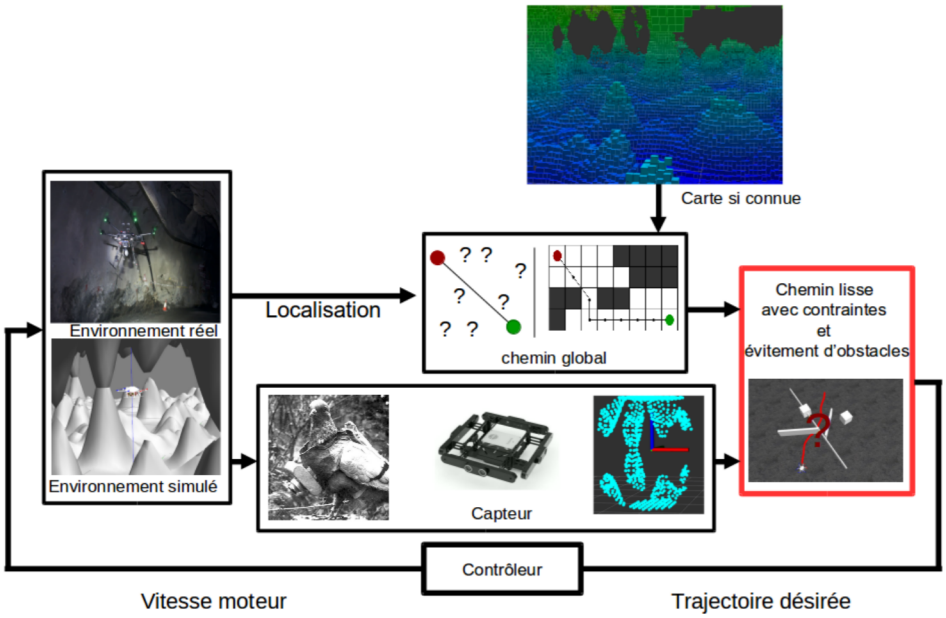


FIGURE 1.18 – Configuration complète requise pour réaliser un vol autonome de drone. Basé sur un environnement réel ou simulé, des informations capteurs sont généréespour permettre d'avoir une localisation et des observations aux alentours du drone. Si une carte est disponible, un chemin local peut-être calculé. Le cas échéant, une exploration ou un déplacement vers l'objectif est réalisé. Un planificateur local réactif s'adapte à la dynamique du drone et à celle de l'environnement. Des commandes moteurs sont calculées par le contrôleur en utilisant le résultat du planificateur et les informations capteurs.

La Figure 1.18 présente une architecture complète nécessaire à réaliser un vol autonome de drone. A partir d'un environnement (réel ou simulé), une perception est construite à partir

des capteurs. La localisation est estimée à partir des capteurs et d'une carte (si elle est disponible). Avec la localisation et les informations capteurs, un générateur de trajectoire calcule les mouvements nécessaires pour atteindre un objectif ou réaliser la mission. Un contrôleur génère ensuite les commandes moteurs pour exécuter ces mouvements dans l'environnement.

1.2 Perception

Pour prendre une décision et pour toute mission, une perception de l'environnement est nécessaire. Les capteurs permettent d'obtenir une information nécessaire pour se localiser, percevoir des obstacles ou construire une représentation de l'environnement pour différentes missions. Les nuages de points permettent de représenter des environnements complexes à partir de plusieurs capteurs, ils sont donc souvent utilisés pour analyser une scène. Un nuage de points ou un "mesh" peut être rigide ou non-rigide dépendant de la nature et de la dynamique des objets. Plusieurs algorithmes d'analyse de nuages de points, rigides ou non, sont proposés dans [Tam et al., 2013]. L'utilisation de la sémantique associée à une base de données [Hmida et al., 2012] permet d'identifier des objets et donc de mieux estimer leurs structures.

Il est possible d'utiliser la variation d'un nuage de points par rapport au temps pour déterminer un déplacement et donc estimer la vitesse d'une partie des objets constituant le nuage. [Vaquero et al., 2017] utilisent des nuages de points et un réseau convolutionnel pour détecter et traquer des véhicules. "L'Iterative Closest Point" (ICP) est une technique très souvent utilisée avec les nuages de points pour trouver des associations entre 2 nuages. L'ICP dans sa version multi-corps [Youngji Kim et al., 2015] permet de détecter le déplacement des objets dans plusieurs nuages successifs. Le "deep learning" est aussi utilisé pour détecter les déplacements dans des nuages rigides [Byravan et Fox, 2017]. Il est également possible de trouver des formes géométriques dans les nuages de points, ce qui permet par la suite d'associer ces formes [Palma et al., 2016] et détecter des déplacements, à la manière d'une détection de contours dans une vidéo. Certains algorithmes utilisent des séquences de nuages de points [Lin et al., 2018] pour analyser le déplacement sur plusieurs acquisitions successives.

L'analyse du mouvement de la scène à travers un nuage de points permet également d'obtenir une localisation. [Vakhitov et al., 2016] suit le mouvement de points d'intérêt et de lignes pour se localiser. Au-delà des formes géométriques, la sémantique permet plus généralement d'extraire des concepts de la scène comme identifier une chaise, un arbre ou même un style de pièce en fonction des éléments. Ce qui permet par la suite d'avoir de meilleures associations et donc une meilleure localisation [Wong et al., 2017].

En général, la perception permet également de savoir qu'elles sont les zones atteignables [Yel et al., 2017] par le drone pour réaliser sa mission. Ou alors dans un horizon plus court, avec des caméras par exemple, il est possible de choisir la prochaine commande [Zhang et Scaramuzza, 2018] et donc de naviguer en utilisant la perception.

L'essor des capteurs de profondeur qui fournissent des nuages de points permet de prendre en compte des environnements complexes pour trouver un chemin [Zheng et al., 2020]. Il est cependant nécessaire d'analyser la grande quantité d'informations produite de ce nuage. En ajoutant le déplacement et la perception d'une caméra, il serait possible de calibrer [Castorena et al., 2020] ces capteurs en fonction de l'environnement et de détecter les déplacements relatifs de chacun des points. Il est également possible de rassembler des points en zone d'intérêt [Kulathunga et al., 2020], ayant la même dynamique à l'intérieur d'un nuage.

1.3 Localisation

Lors d'un vol autonome, le drone doit être capable de déterminer où il se trouve. Certaines solutions disponibles sur étagère comme les capteurs GPS permettent d'obtenir une localisation rapidement. Néanmoins, dans certains environnements comme les grottes ou les environnements d'intérieur, les satellites ne sont pas toujours disponibles. De plus, pour des missions d'inspection ou de cartographie par exemple, il est nécessaire d'avoir une localisation plus précise, basée sur la perception. Dans [Panchpor et al., 2018] un ensemble d'algorithmes sont utilisés pour la localisation dans un environnement intérieur dynamique. Différents articles de SLAM (Simultaneous Localization And Mapping) y sont présentés ainsi que des algorithmes basés sur l'apprentissage, en représentant l'environnement par des graphes ou avec des informations visuelles.

Le SLAM [Durrant-Whyte et Bailey, 2006],[Durrant-Whyte et Bailey, 2006] est une méthode très répandue de nos jours, elle consiste en la création de la carte et de la localisation de manière entrelacée. Il existe plusieurs algorithmes pour le SLAM, basés sur des caméras utilisant des points d'intérêt comme des caractéristiques ORB pour "l'ORB-slam" [R.l et D., 2017], des contours [Pumarola et al., 2017] ou des capteurs 3D [Engelhard et al., 2011]. Comme pour beaucoup d'algorithmes utilisant des caméras, choisir des images clés permet de réduire la charge de calcul [Leutenegger et al., 2013].

L'odométrie visuelle consiste à suivre les déplacements de primitives visuelles dans l'image pour estimer le déplacement du robot [Delmerico et Scaramuzza, 2018], [Forster et al., 2017]. Comme pour beaucoup d'algorithmes basés sur la vision, des points d'intérêts sont extraits de l'image à l'exemple des contours [Vakhitov et al., 2016]. A ces points d'intérêt correspondent des descripteurs qui permettent d'associer les informations provenant d'images successives. Pour ce faire, diffférentes variantes de filtrage de Kalman ont été utilisées, comme des filtres à plusieurs états contraints [Mourikis et Roumeliotis, 2007] ou des filtres de Kalman étendus [Bloesch et al., 2015], des estimateurs d'états [Qin et al., 2017] apportent également certains avantages dans le cadre de l'odométrie visuelle. Certains proposent d'améliorer l'estimateur d'état en utilisant différents modèles de déplacement et en gardant le plus performant à chaque étape [Elzoghby et al., 2018]. L'utilisation de deux caméras [Sun et al., 2017] permet

d'avoir une information sur l'échelle dans l'image et ainsi de calculer un déplacement métrique. Sinon, pour une seule caméra, d'autres informations provenant d'une centrale inertielle peuvent être utilisées pour donner une première estimation de déplacement entre les images clé. Des méthodes d'optimisation comme les méthodes par intervalles [Kenmogne et al., 2017] permettent également d'obtenir une estimation du déplacement entre deux images. Il est aussi possible de combiner des informations provenant de différents capteurs comme une caméra de type fish-eye et une caméra perspective [Eynard et al., 2010]. Au-delà des caméras 3D, les capteurs lidar permettent également des générer une localisation [Zhen et al., 2017]. A noter que ces algorithmes ont aussi été exploités dans le cadre de la réalité augmentée [Klein et Murray, 2007], [Marchand et al., 2016], pour la téléopération par exemple.

1.4 Navigation

Parmi les principales missions d'un drone, on compte l'exploration, l'inspection et la navigation vers un point objectif. Explorer toute une zone de manière efficace n'est pas simple. Se déplacer de manière aléatoire [Moysis et al., 2020] dans la zone est une solution facile, mais cela peut prendre énormément de temps pour des zones importantes ou encombrées. Pour optimiser l'exploration, il est nécessaire de trouver la prochaine zone qui donnerait le meilleur point de vue [Bircher et al., 2015]. D'autres informations peuvent être incluses dans une fonction objectif [Stachniss et al., 2005], par exemple privilégier les zones proches de celles où une bonne localisation a été obtenue [Papachristos et al., 2019] pour créer une carte précise. Certaines méthodes d'apprentissage [L. et T., 2020] cherchent également à optimiser l'exploration de zones inconnues. De nombreuses missions telles que l'inspection environnementale [Yu et al., 2017], [Yingkun, 2018], le transport [Andrade et al., 2016] et la manipulation [Zhang et al., 2018], nécessitent des positionnements et des déplacements vers des points objectifs.

La Figure 1.19 résume un ensemble d'algorithmes de planification de mouvement dans des environnements dynamiques [Mohanan et Salgoankar, 2018]. Un planificateur de chemin utilise des cartes, des perceptions ou d'autres informations pour trouver le trajet permettant au drone d'accomplir sa mission, qui consiste le plus souvent à atteindre un point ou un état dans l'espace. Un planificateur de trajectoire doit, quant-à-lui intégrer la dynamique du drone et du monde dans les déplacements générés. La planification peut également se dérouler à plusieurs niveaux. Un exemple simple serait qu'un gestionnaire de tâches détermine des points de passage pour réaliser une mission et observe l'état du monde et du drone pour déclencher des comportements de récupération si le drone se perd. Ensuite, un planificateur global se déclenche à chaque nouveau point de passage reçu et calcule un chemin ou une trajectoire vers ce point en fonction d'une carte. Le planificateur de trajectoire gère ensuite la dynamique et l'aspect local du déplacement.

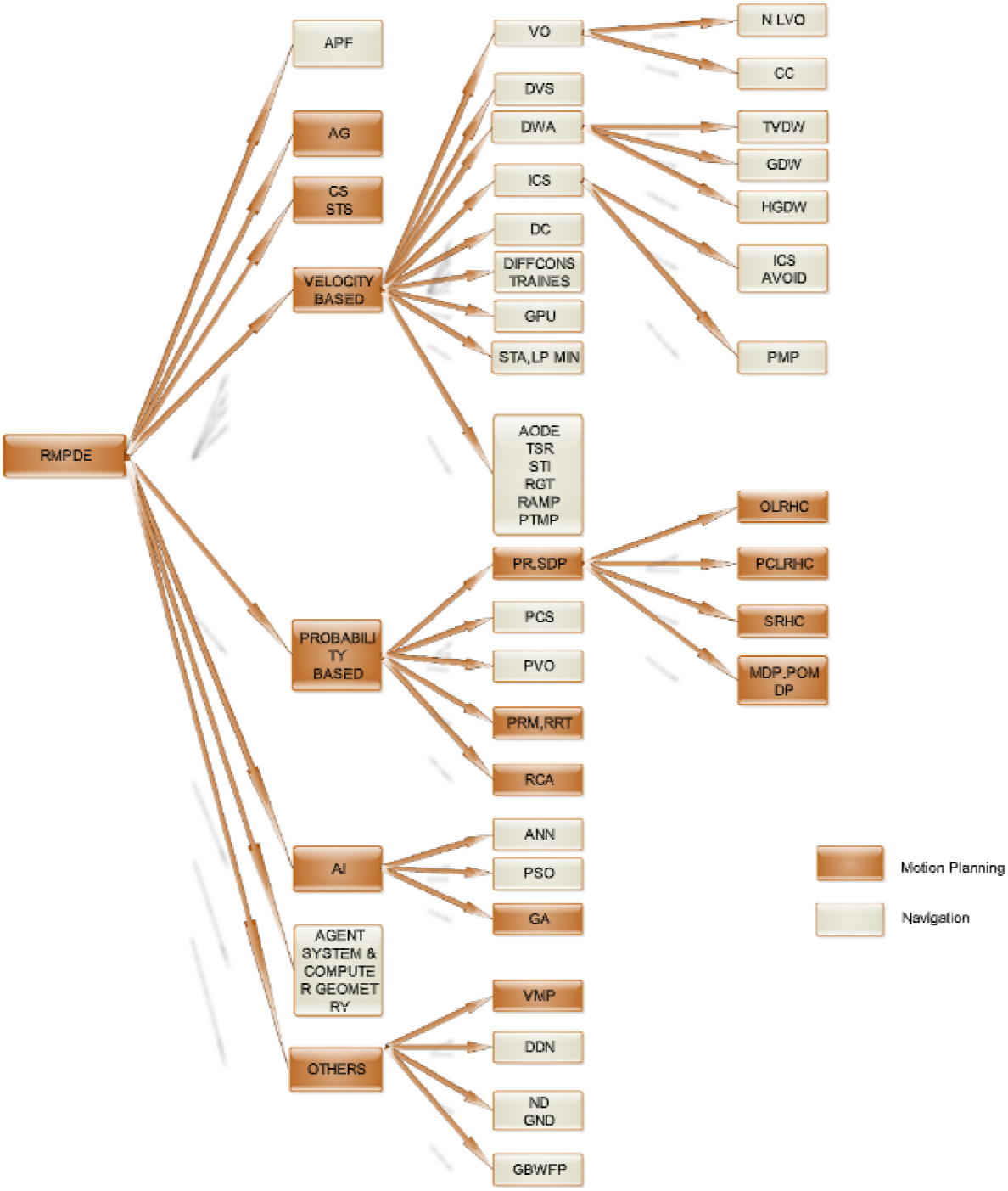


FIGURE 1.19 – Résumé des algorithmes de planification de trajectoire dans des environnements dynamiques [Mohanan et Salgoankar, 2018]

Les algorithmes de planification de déplacement discrétisent de manière uniforme la carte et cherchent un chemin sur une grille, soit échantillonnent la carte de manière aléatoire.

Dans le cas de la planification 3D dans des environnements dynamiques, il est nécessaire de réduire la complexité. Certains algorithmes fonctionnent localement en utilisant seulement la perception ou une carte locale. D'autres cherchent des primitives ou des actions dans une table de correspondance en fonction de la situation. Certains apprennent les bonnes actions ou les meilleurs chemins grâce à des réseaux de neurones [Yan et al., 2020]. D'autres approches consistent à utiliser des procédés d'optimisation comme des champs de potentiel, des essaims de particules ou des algorithmes génétiques pour trouver la meilleure solution au regard de certains critères. Les plus répendues sont basées sur les champs de potentiel (Artificial Potentiel Fields, APF), les cartes probabilistes (Probabilistic Roadmaps, PRM), les arbres aléatoires rapides (Rapidly Random Trees, RRT), les algorithmes génétiques (Génétic Algortithms, GA), les essaims de particules (Particle Swarm Optimization, PSO), les réseaux de neurones (Artificial Neural Networks, ANN) et bien d'autres que nous verrons en détail par la suite.

Pour réaliser un vol autonome, lors de la planification de chemin, il est important de prendre en compte les obstacles et de les éviter. En particulier, lorsque le vol est effectué en milieu urbain, [Ferguson et al., 2008], lors de vol en formation [Bucki et Mueller, 2019] ou en intérieur. Il existe un certain nombre de solutions [Radmanesh et al., 2018].

Beaucoup de travaux utilisent des caméras pour éviter les obstacles ou même suivre des cibles, certains prennent même en compte les occultations générées par l'environnement pour éviter de se diriger dans des zones où la perception serait encombrée [Penin et al., 2018]. L'évitement d'obstacle demande de modéliser tout l'environnement et cela correspond à une charge de calcul importante, mais chaque élément de l'environnement est assez simple, c'est pourquoi de plus en plus d'algorithmes basés caméra ou nuage de points utilisent des calculs sur GPU [Juelg et al., 2017], ou encore n'activent un algorithme d'évitement d'obstacle qu'en réponse à un déclencheur pour éviter un danger pour le drone [Lin et al., 2020].

Pour comparer différents planificateurs, on compte un certain nombre de critères, comme la longueur du chemin, la complexité algorithmique, la performance, mais également, si la solution obtenue est exacte (non-heuristique) ou non. Une étude comparative de différents planificateurs a été faite en considérant différentes erreurs de localisation [Kang et al., 2017].

Les algorithmes "bug" sont une classe d'algorithmes de planification de chemin sans carte a priori. La non-utilisation de carte rend difficile le déplacement, il est fréquent de se retrouver bloqué à certains endroits ou de ne jamais trouver une solution. Il existe néanmoins un certain nombre de stratégies de déplacement qui permettent d'optimiser ce type de solutions de navigation [McGuire et al., 2019].

1.4.1 Planification de chemin

Certains planificateurs utilisent une carte connue au préalable. Deux grands axes se dégagent lors de l'utilisation d'une carte. Dans un premier temps, la carte est traitée pour simplifier sa représentation, un chemin est ensuite cherché à différentes étapes de la mission dans cette carte

simplifiée. Il est en effet complexe de trouver un chemin dans la carte non simplifiée en cours de mission car cela représente une trop grande quantité d'informations à traiter et prendrait trop de temps. Le deuxième axe va néanmoins chercher un chemin dans la carte complète durant la mission, mais pour être faisable en ligne le chemin résultat sera moins optimal.

Un certain nombre de travaux se concentrent sur la simplification de l'espace en le discrétisant de différentes façons :

— en utilisant des cellules de taille uniforme ou de taille adaptative. Les cellules de taille uniforme sont plus simples et plus rapides. Alors que les cellules de taille adaptative permettent de générer des chemins plus précis en décomposant l'environnement de manière plus précise là où cela est nécessaire [Samaniego et al., 2017]. Il est également possible de faire une décomposition à plusieurs niveaux de précision, cela permet de descendre d'un niveau et d'être plus précis quand cela est nécessaire. Cela est souvent fait en utilisant une structure d'arbre pour organiser les données. Le "KD-tree" subdivise l'espace en deux parties à chaque nouvelle branche, basé sur différentes informations ou capteurs (nuage de points par exemple). "L'octree" [Hornung et al., 2013] quant à lui subdivise l'espace en 8 sous-espaces de tailles identiques à chaque opération. La racine étant souvent l'espace entier et la feuille représentant la division la plus précise de cet espace. Une recherche de chemin peut ensuite être réalisée plus simplement dans cette représentation de l'environnement. En utilisant par exemple des algorithmes A* [Khuswendi et al., 2011], D* ou Dijkstra. Sur la Figure 1.20, une discrétisation de l'espace uniforme est réalisée puis on cherche de manière itérative le déplacement de cellule en cellule qui minimise une fonction d'utilité. La plus simple étant le déplacement le plus petit qui nous rapproche le plus de l'objectif. Le chemin en vert représente le chemin final trouvé avec cette méthode.

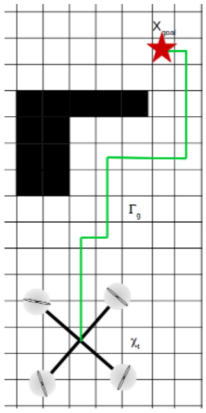


FIGURE 1.20 – Exemple de chemin généré en utilisant un algorithme A*. A chaque étape, l'algorithme choisit la cellule qui le rapproche le plus de l'objectif en se déplaçant le moins possible. Le chemin en vert représente le chemin final trouvé.

— En reliant des points choisis aléatoirement dans l'espace. Ce type de méthodes est plus

souvent utilisé en embarqué, car il est plus rapide pour décrire l'environnement. Mais il le fait de manière moins précise et les chemins trouvés sont souvent plus longs et moins efficaces. C'est ce qui est fait par exemple pour les méthodes RRT (rapidly random tree). Un point aléatoire est choisi dans l'espace puis il est relié au point le plus proche dans l'espace jusqu'à trouver un chemin vers l'objectif.

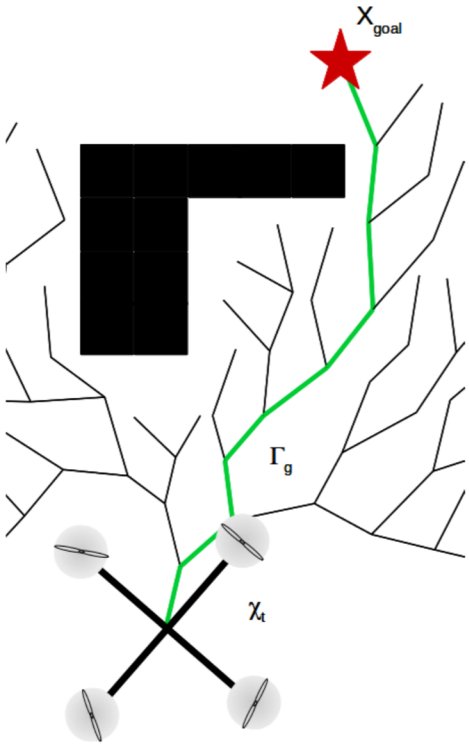


FIGURE 1.21 – Exemple de fonctionnement d'un RRT. Chaque branche est générée aléatoirement jusqu'à relier le point de départ au point d'arrivée. Le chemin en vert représente le chemin final trouvé.

— Les approches "roadmap" permettent de générer des chemins en fonction de certaines connaissances a priori sur l'environnement. Par exemple, créer des chemins qui cherchent à passer par des endroits où il y a de bonnes chances de pouvoir calculer une localisation précise [Shan et Englot, 2017], comme montré sur la Figure 1.22, ou encore éviter les endroits où il est très probable de rencontrer un obstacle [Kaufman et al., 2016]. D'autres utilisent des cartes hybrides pour à la fois utiliser des informations géométriques et la compréhension topologique de l'environnement [Zhang et al., 2015].

FIGURE 1.22 – Roadmap permettant de chercher un chemin tout en optimisant la localisation.

La Figure 1.23 donne un exemple de "probabilistic roadmap (PRM)". Ces cartes sont également utilisables en 3D, en se servant de voxels de différentes tailles pour représenter l'environnement [Yan et al., 2012].

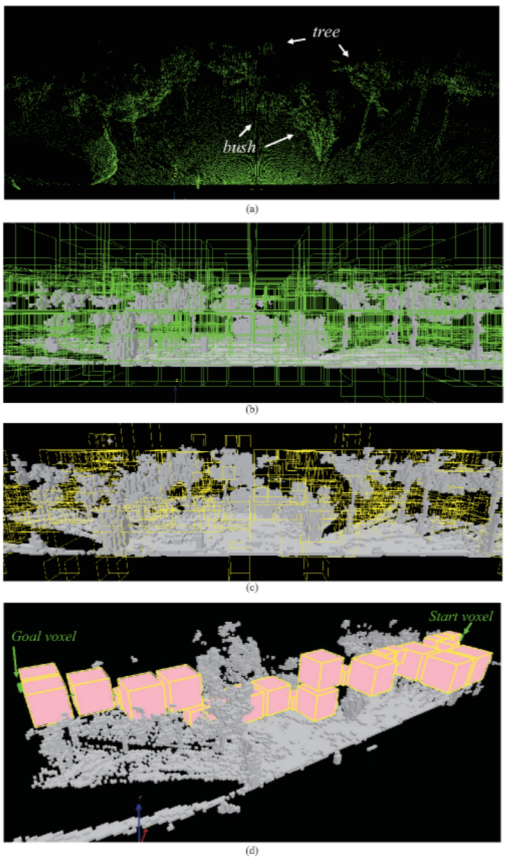


FIGURE 1.23 – Exemple de "PRM" utilisant des voxels pour trouver un chemin.

— En utilisant d'autres types d'informations que l'espace, pour assurer la sécurité, il est possible de créer un diagramme de Voronoï [Candeloro et al., 2016], [Benzaid et al., 2018] qui soit équidistant de tous les obstacles proches [Kaufman et al., 2016]. Les diagrammes de Laguerre [Wei et al., 2016] sont une autre façon de décrire l'espace de manière sûre dans le but d'assurer un vol sécuritaire.

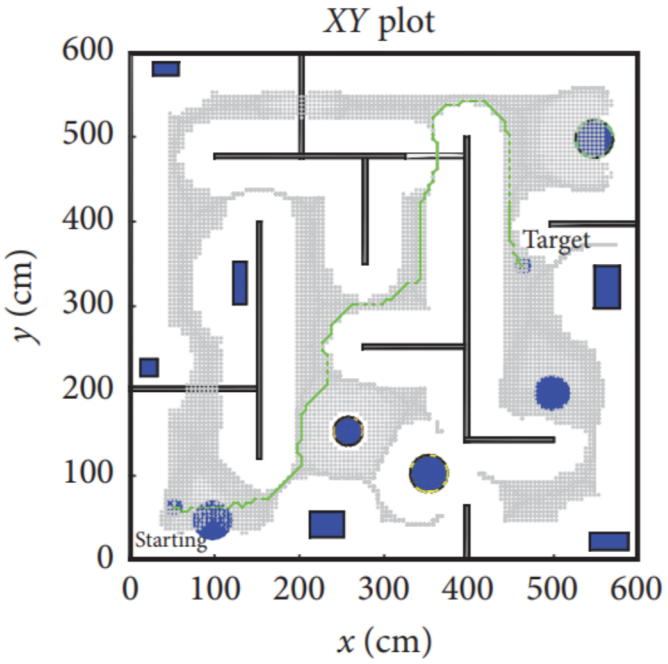


FIGURE 1.24 – Exemple de planificateur utilisant le diagramme de Voronoï.

Certains algorithmes utilisent la traversabilité, les risques dans des zones peuplées [Primatesta et al., 2020], la possibilité ou non de se relocaliser de manière efficace, ou encore dans des zones peu connues [Bircher et al., 2018] pour l'exploration ou non visitées depuis longtemps.

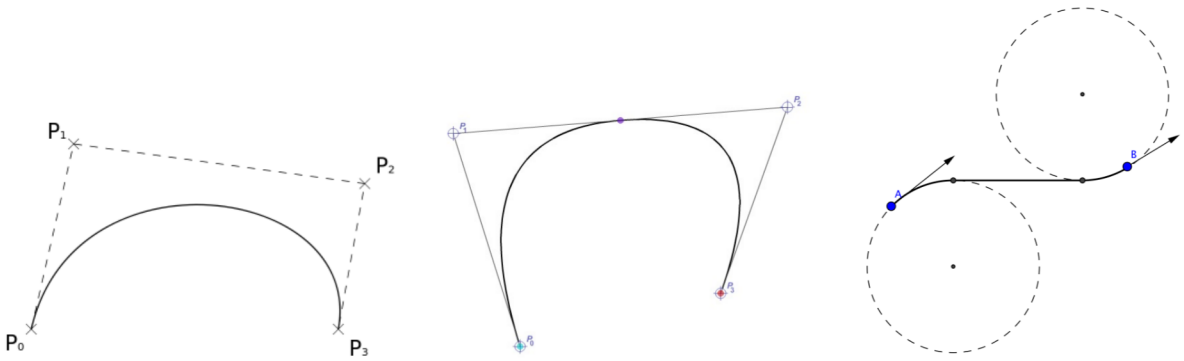
1.4.2 Planification de trajectoire

Plusieurs solutions peuvent être proposées pour la planification en considérant les contraintes physiques, cinématiques ou dynamiques du drone.

- Méthodes basées RRT-A*-Dijstra.
  Certains travaux ajoutent les contraintes dynamiques du robot [Wu et al., 2015] pour s'assurer que le chemin soit réalisable. D'autres lissent l'arbre après sa création [Yang et Sukkarieh, 2008a] pour le rendre faisable. Certains cherchent à optimiser l'étape de création de l'arbre pour qu'il soit dirigé vers l'objectif afin de réduire le nombre d'itérations pour créer un chemin [Tang et al., 2020]. Il est également possible de créer

deux arbres : un qui part du robot et un qui part de l'objectif pour les relier plus facilement.

- Méthodes basées courbes de Bézier.
  Générer un chemin ou une trajectoire continue demande une puissance de calcul importante comparativement aux méthodes discrètes. Une solution consiste à calculer des primitives. Des trajectoires faisables sont alors préalablement calculées. A chaque instant, la meilleure primitive est choisie en fonction de l'état du drone et de l'environnement. Cela limite les déplacements possibles, mais la faculté de calcul rend la méthode embarquable sur un drone et prend en compte l'environnement sans délai dû au calcul de la trajectoire [Mueller et al., 2015].
  Les courbes permettent de décrire une trajectoire de manière simple avec pour avantage d'être continue et d'obtenir simplement les dérivées successives. Elles sont souvent utilisées dans des processus d'optimisation pour obtenir le chemin vers un point objectif minimisant une fonction coût sous certaines contraintes [Van Parys et Pipeleers, 2017].
  Il est possible d'utiliser les courbes pour lisser des chemins non-faisables par le drone. Dans [Yang et Sukkarieh, 2008b], trois types d'algorithmes basés sur des courbes pour lisser des chemins sont comparés. Les sigmoïdes ou courbes logiques [Upadhyay et al., 2020], représentent une forme de "s", ce qui permet de relier un point origine et un point final avec une transition douce.
  Parmi les courbes disponibles, l'on compte les courbes de Bézier. Elles ont plusieurs avantages :
  — le premier et le dernier point correspondent au début et à la fin de la courbe,
  — la simplicité de calcul des dérivées notamment au début et à la fin de la trajectoire,
  — et très peu de points de contrôle sont nécessaires pour décrire une courbe continue.



(a) (b) (c)

FIGURE 1.25 – Exemples de courbes d'interpolation utilisant (a) une courbe de Bézier avec 4 points de contrôle, (b) une courbe B-spline, et (c) une courbe de Dubins.

Les B-splines sont une généralisation des courbes de Bézier, elles permettent plus d'agilité en ajoutant un paramètre qui impacte l'influence de chaque point de contrôle sur la courbe, et cela réduit leurs nombres. Mais cela augmente la complexité par rapport aux courbes de Bézier [Babaei et Karimi, 2018].

Les courbes de Bézier sont utilisées pour générer des chemins depuis les années 1990 [Nowacki et al., 1990]. Beaucoup de travaux [Ingersoll et al., 2016] se concentrent sur l'optimisation d'une fonction coût qui représente en général la longueur du chemin, l'énergie utilisée [Artemenko et al., 2016] ou le temps de réalisation de la trajectoire. Pour les missions de surveillance, l'objectif pourrait être de passer par certaines zones clé [Faigl et Váňa, 2018]. Modifier une trajectoire avec des courbes d'ordre 3 est assez simple, le premier et le dernier point étant sur la courbe existante, il ne reste donc qu'un point à optimiser. Il est possible d'utiliser un essaim de particules pour le faire [Wu et al., 2017], chaque particule représente une position possible de ce point de contrôle, il suffit ensuite de trouver la meilleure particule et de faire évoluer l'essaim si nécessaire. Le même processus peut être appliqué aux algorithmes de type colonie de fourmis. Malgré la simplicité des courbes de Bézier, générer une longue trajectoire en 3D peut demander beaucoup de temps de calcul. C'est pourquoi les trajectoires sont souvent définies par morceaux. Dans [Mehdi et al., 2015], si une potentielle collision est détectée, alors le chemin global est modifié localement en utilisant des courbes de Bézier. D'autres [Choi et Huhtala, 2014a] considèrent quelques modèles de courbes acceptables (aussi appelées primitives), et sélectionnent ensuite la meilleure courbe solution. Au lieu de choisir la meilleure primitive par rapport au drone, il est possible de discrétiser l'espace puis d'utiliser des primitives comme le déplacement unitaire entre chaque cellule [Choi et Huhtala, 2014b]. Un algorithme de type A* est ensuite utilisé pour trouver le meilleur chemin dans cet espace. La primitive est plus simple à calculer, mais très locale et n'utilise en général pas de carte. Le RRT est un bon compromis entre les deux méthodes. Il est plus lent à calculer et prend en compte une carte. Il est même possible de modifier le RRT pour relier des points aléatoires par des courbes de Bézier afin d'assurer un chemin lisse et dynamiquement faisable [Seemann et Janschek, 2014].

Dans [Yang et al., 2015], un chemin simple linéaire par morceau est généré puis lissé à l'aide de courbes de Bézier d'ordre 6 ou d'ordre 4 [Jayasinghe et Athauda, 2016]. Cette méthode permet même de faire de la génération de trajectoires en respectant les contraintes de vol embarqué [Hilario et al., 2011], en modifiant localement des courbes grâce à des champs de potentiels. L'un des grands avantages des courbes est de pouvoir ajouter facilement des contraintes. Parmi ces contraintes, on compte la courbure de la trajectoire. Ce qui permet de calculer des chemins plus lisses en fonction des capacités dynamiques du drone [Wang et al., 2017].

- Méthodes basées sur le modèle d'évolution de colonies.

L'utilisation d'algorithmes bio-inspirés pour résoudre le problème d'optimisation, permet de trouver rapidement une courbe optimale. Comme tout procédés d'optimisations, il est possible d'ajouter des contraintes statiques ou dynamiques à la fonction coût. Ces algorithmes [Nikolos et al., 2003] permettent de trouver une solution en simulant un ensemble qui évolue comme le ferait une colonie animale. Les algorithmes basés sur les essaims de particules créent un ensemble d'individus choisissant le meilleur en faisant évoluer la population [Sahingoz, 2013]. Les algorithmes basés sur les colonies de fourmis se déplacent d'état en état suivant un modèle probabiliste, chaque fourmi au retour d'une source, laisse derrière elle une traînée de phéromone, qui va renforcer les chemins avec les meilleurs résultats. Ainsi plus spécifiquement, les algorithmes génétiques [Gutierrez-Martinez et al., 2020] se concentrent sur les mutations et la sélection des meilleures solutions à chaque étape. Les animaux ne sont pas les seules inspirations, les modèles d'écoulement des fluides [Liang et al., 2014] donnent aussi lieu à des planificateurs de chemin.

- Méthodes basées sur des champs de potentiel.
  Ces méthodes sont très utilisées depuis des dizaines d'années étant donné leur simplicité d'utilisation et de calcul. Elles consistent en la création de zones attractives et répulsives par rapport à l'élément réel ou une contrainte dynamique que l'on désire optimiser. Une utilisation commune et simple consiste à avoir une trajectoire de référence ou un objectif attractif et des obstacles répulsifs, les obstacles peuvent être fixes ou mobiles. Comme tout procédés d'optimisations, il est possible d'ajouter des contraintes statiques ou dynamiques à la fonction coût. Un élément peut parfois être attractif ou répulsif lorsque l'on veut maintenir une formation de drone [Zhao et al., 2017] par exemple, chaque drone va essayer de se maintenir à une certaine distance fixe des autres. Si il est loin, il sera attiré et sera repoussé si il est trop proche. Ainsi, le drone va essayer de se rapprocher de l'objectif tout en s'éloignant des dangers potentiels. Faire cela de manière localisée autour du drone permet de trouver localement un chemin optimal. Il est possible de calculer les forces d'attraction et de répulsion des différents éléments séparément et de les sommer. Ce fonctionnement se prête bien à l'utilisation du GPU [Khuswendi et al., 2011] pour accélérer l'algorithme et prendre en compte tout l'environnement 3D [Juelg et al., 2017]. L'impact des différentes forces d'attraction et de répulsion peut facilement amener à des minimums locaux ou à des oscillations. Parmi les pistes d'amélioration, on trouve souvent la forme de la force en fonction de la distance, la distance maximale et minimale d'application et l'introduction d'éléments perturbateurs sur la force et la direction.
  En donnant une forme de sigmoïde [Rivera et al., 2012] à la force en fonction de la distance, il est possible de limiter la distance maximale à laquelle un élément impacte le drone. Cela réduit également la force lorsque le drone est proche d'un élément. Les fonctions harmoniques [Panati et al., 2015] sont aussi souvent utilisées pour représenter

les forces. Elles ont pour avantage d'avoir des dérivées première et seconde continues et un Laplacien nul. [Faria et al., 2004] ont analysé précisément l'intérêt de différentes fonctions harmoniques dans les champs de potentiel appliqués au contrôle de trajectoire. L'utilisation d'un modèle de fluide [Palm et Driankov, 2014] permet de créer un champ de vecteurs où le drone est emporté par le flux. Ainsi un élément n'a pas le même impact si il est sur le flux emprunté par le drone ou derrière lui ou encore sur les côtés. L'ajout de champs rotationnels [Panagou, 2014] autour des obstacles permet de créer des turbulences et éviter les minimas locaux. Pour le vol en extérieur, il est possible d'utiliser un observateur pour avoir une mesure du vent [Oettershagen et al., 2017] et ainsi adapter le flux en fonction. Pour trouver le meilleur chemin, plusieurs algorithmes et fonctions à optimiser sont à notre disposition. On peut résoudre un problème simple de distance minimale en utilisant le gradient [Gao et al., 2017] pour se rapprocher d'une solution optimale. Cette méthode fonctionne bien lorsque le problème est convexe, linéaire et ne possède qu'une solution. Sinon, il est possible de trouver une solution sous optimale ou même de ne pas converger. D'autres modélisations permettent d'obtenir de meilleurs résultats dans ce type de situations. Il est par exemple possible d'utiliser des symboles [Zhang et Huo, 2017] et d'avoir une représentation logique du problème ou alors la logique disjonctive [Babaei et Karimi, 2018] qui permet d'associer un modèle logique et géométrique du problème. Les drones n'ayant pas une grande autonomie, il est également important d'inclure l'énergie [Jones et Hollinger, 2017] dans les critères de planification de chemin, pour s'assurer d'avoir des chemins faisables, mais également allonger les chemins faisables en réduisant la consommation, et ce, malgré d'éventuelles perturbations. Pour réduire la consommation et la charge de calcul, certains auteurs [Lin et al., 2020] cherchent à optimiser l'instant t à partir duquel les obstacles obligent à recalculer une trajectoire.

1.4.3 Contrôle du mouvement

Le contrôle du mouvement nécessite de créer une loi de commande qui envoie les bonnes valeurs aux actionneurs (rotors dans notre cas) pour générer les forces et les moments créant les déplacement souhaités pour réaliser la mission, tout en prenant en compte des contraintes telles que la saturation des actionneurs, les retards ou la consommation d'énergie. La première étape du contrôle consiste en la création d'un modèle du système réel. Il existe deux grandes familles de modèles, les non linéaires [Marchand et al., 2013] et les modèles linéarisés autour d'un point de fonctionnement.

Il existe un certain nombre de contrôleurs qui permettent de générer les consignes moteurs pour suivre une trajectoire. Les premiers contrôleurs utilisés sont basés sur les PID [Åström et Hägglund, 1995]. Les PID ont des gains simples à régler et sont assez simples à interpréter et implémenter. La commande linéaire quadratique (LQ) [HOW et al., 2008] utilise

un retour d'état pour calculer les gains. Elle peut donc être appliquée sur des systèmes plus complexes. Le retour d'état [Mellinger et Kumar, 2011] est également très utilisé pour transformer un modèle non-linéaire en un modèle linéaire et donc lui appliquer une plus large gamme de lois de commande. L'utilisation des fonctions de Lyapunov [Kokotovic, 1992] est importante dans le domaine car cela permet d'assurer une certaine convergence et stabilité de l'erreur et prend en compte les systèmes non-linéaires. La commande par mode glissant [Bouadi et Tadjine, 2007] est réputée très robuste aux perturbations qui pourraient venir nuire à la commande.

Parmi les contrôleurs récents les plus utilisés [Nasr et al., 2018], on compte les modèles prédictifs, les modes glissants, le "backstepping", et les contrôleurs basés sur la logique floue. Le "Backstepping" et les "modes glissants" peuvent être utilisés de pair [Dolatabadi et Yazdanpanah, 2015], cela permet entre autres de contrôler les moteurs en prenant en compte d'éventuelles pannes [Zhu et Cao, 2019]. Les commandes prédictives [Shim et Sastry, 2002] utilisent le modèle et les commandes pour prédire le comportement du système sur plusieurs itérations et optimisent ainsi la séquence de commandes. Concevoir un contrôleur basé modèle prédictif (MPC) peut s'avérer complexe [Raemaekers, 2007]. Ce type d'algorithmes permet le contrôle de systèmes complexes contraints, mais également de systèmes multi-agents [Mansouri et al., 2015]. L'ajout de contraintes facilite la prise en compte de l'environnement pour le vol en intérieur [Marzat et al., 2017] ou encore le vol en zone urbaine et peuplée [D'Amato et al., 2020]. Il est également possible d'intégrer des informations capteurs ou de localisation [Dentler et al., 2019] dans le processus du MPC. Parmi les modèles de drone, il existe les "tilts" rotor qui ajoutent des degrés de liberté et augmentent ainsi la dimension de la commande. Le MPC est de ce fait un outil tout indiqué pour ce type de plateformes, car il permet d'ajouter des contraintes [Andrade et al., 2016] sur ces degrés de liberté et donc d'affiner la stratégie de commande.

D'autres types d'optimisation sont utilisés pour prendre en compte l'environnement [Jang et al., 2017] lors du contrôle. Il est par exemple possible d'utiliser un filtre particulaire pour déterminer les meilleures commandes permettant de suivre une trajectoire [Rendón et Martins, 2017].

1.5 Conclusion

Dans ce chapitre, les différentes structures de l'architecture globale de drones ont été présentées, puis une introduction à l'ensemble des aspects nécessaires au vol, a été réalisée. En commençant par différentes perceptions et capteurs qui servent à créer des cartes et à se localiser, deux éléments indispensables à la navigation. Nous avons ensuite introduit les différents algorithmes de planification de chemin géométrique ou de trajectoire qui ajoute le temps et la dynamique au chemin. Finalement, différentes solutions pour contrôler les drones

ont été présentées. Tous ces aspects sont indispensables à la réalisation d'un vol autonome. L'aspect matériel évolue très rapidement, ce qui permet de résoudre des problèmes de plus en plus complexes pour des volumes physiques de plus en plus petits.

Chapitre 2

Modélisation et contrôle d'un drone

2.1 Introduction

Ce chapitre s'intéresse au contrôle d'un drone dans le but de suivre une trajectoire générée par un processus d'optimisation. La première étape consiste en la recherche d'un modèle pour notre drone. Les angles d'Euler et les quaternions sont très souvent utilisés pour décrire l'attitude d'un drone. Les équations de Lagrange ou de Newton sont utilisées pour exprimer les dynamiques de translation et de rotation du système. A partir du modèle et des positions désirées de la trajectoire, une commande peut être proposée. Une façon bien connue de procéder est d'utiliser une architecture hiérarchique avec un étage de contrôle en position et un étage de contrôle en attitude. Le contrôle en position et en attitude peut utiliser un simple régulateur PID. Dans le cadre de cette thèse, nous avons testé différents contrôleurs. Dans un premier temps, on utilisera un simple contrôleur PID, puis un contrôleur en position de type "back-stepping" pour améliorer la stabilité, et enfin, un contrôle par modèle prédictif qui utilise en même temps un modèle et un procédé d'optimisation.

2.2 Modélisation dynamique d'un drone

Etant donné qu'un drone peut se déplacer en 3D, il est important de définir les différents repères de travail avant toute tentative de modélisation. Le repère monde ("world") noté $R_w = (O_w, x_w, y_w, z_w)$ est fixe, et constitue le repère global de référence. Le repère du drone ($R_d = (O_d, x_d, y_d, z_d)$) représente le référentiel de la perception et de la commande du drone, ayant pour origine son centre de gravité.

Pour passer du repère monde au repère associé au drone, une série de trois rotations est utilisée (Figure 2.1). Les angles de ces rotations sont appelés les angles d'Euler et représentent les rotations propres autour des axes x, y, z du monde, respectivement nommés angles de roulis, de tangage et de lacet. Les transformations correspondantes s'expriment :

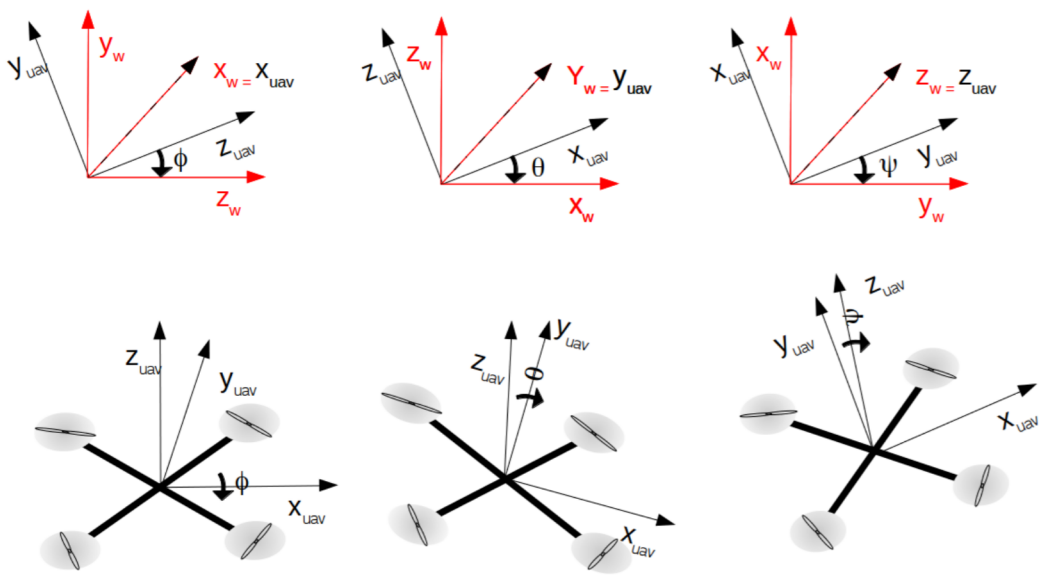


FIGURE 2.1 – Représentation de l'attitude du drone par les angles d'Euler.

— pour l'angle de roulis qui représente la rotation autour de l'axe x :

$$\begin{cases} x_d = x_w \\ y_d = cos(\phi)y_w - sin(\phi)z_w \\ z_d = sin(\phi)y_w + cos(\phi)z_w \end{cases} \equiv R_\phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} \tag{2.1}$$

— pour l'angle de tangage qui représente la rotation autour de l'axe y :

$$\begin{cases} x_d = cos(\theta)x_w + sin(\theta)z_w \\ y_d = y_w \\ z_d = -sin(\theta)x_w + cos(\theta)z_w \end{cases} \equiv R_\theta = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \tag{2.2}$$

— pour l'angle de lacet qui représente la rotation autour de l'axe z :

$$\begin{cases} x_d = cos(\psi)x_w - sin(\psi)y_w \\ y_d = sin(\psi)x_w + cos(\psi)y_w \\ z_d = z_w \end{cases} \equiv R_\psi = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

Les matrices R_ϕ, R_θ et R_ψ représentent les rotations successives pour passer du repère monde au repère drone. L'équation (2.4) récapitule la matrice de rotation complète obtenue par multiplication des trois matrices. Par souci de lisibilité, les *cosinus* seront notés c suivis de

l'angle et les *sinus*, s.

$$R = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \tag{2.4}$$

Sans perturbation, seuls deux efforts s'appliquent sur le drone, celui dû à son poids et celui dû à la force de poussée qu'il crée. La force de poussée est la somme de chaque force générée par les moteurs comme indiqué sur la figure 2.2. La force de poussée d'un rotor dépend du carré de la vitesse de rotation et du coefficient aérodynamique de poussée qui est donné par la géométrie de l'hélice. L'équation (2.5) donne la forme de la force de poussée totale générée par le drone.

$$\mathbf{F} = b \sum_{i=1}^{4} w_i^2 \ \mathbf{e}_{z_d} \tag{2.5}$$

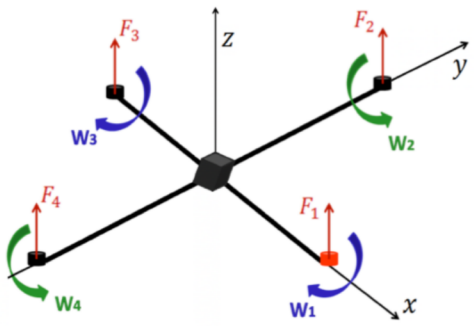


FIGURE 2.2 – Forces de poussée et moments généralisés pour un quadri-rotor.

La variation des vitesses de rotation des rotors génèrent des couples de roulis, de tangage et de lacet, exprimés respectivement par (2.6) :

$$\begin{aligned} \Gamma_x &= bl(w_2^2 - w_4^2) \\ \Gamma_y &= bl(w_1^2 - w_3^2) \\ \Gamma_z &= d(w_2^2 + w_4^2 - w_1^2 - w_3^2) \end{aligned} \tag{2.6}$$

avec w_i les vitesses angulaires, b et d les coefficients de poussée et de traînée, et l le paramètre géométrique de demi-envergure du drone.

Notons q le vecteur d'état rassemblant les 6 variables représentant l'état du drone dans l'espace par rapport au référentiel fixe. Ce vecteur est constitué de 3 positions dans l'espace $\boldsymbol{\xi}$ et des 3 angles d'Euler $\boldsymbol{\eta}$ qui représentent l'attitude du drone dans le repère monde.

$$\mathbf{q} = (x, y, z, \phi, \theta, \psi)^T = (\boldsymbol{\xi}^T, \boldsymbol{\eta}^T)^T \in \mathbb{R}^6 \tag{2.7}$$

Le Lagrangien L est défini dans l'équation (2.8) à partir de $\mathbf{q}$ et $\dot{\mathbf{q}}$. Il est égal à la somme des énergies cinétiques de translation E_{trans} et de rotation E_{rot} et de la somme des énergies potentielles U.

$$L(\mathbf{q}, \dot{\mathbf{q}}) = E_{trans} + E_{rot} - U \tag{2.8}$$

L'énergie cinétique de translation est donnée dans (2.9), en fonction de la masse m du drone et de sa vitesse en translation $\dot{\boldsymbol{\xi}}$.

$$E_{trans} = \frac{m}{2}\dot{\boldsymbol{\xi}}^T\dot{\boldsymbol{\xi}} \tag{2.9}$$

L'énergie cinétique de rotation s'exprime en fonction de la vitesse de rotation $\boldsymbol{\Omega}$ et de la matrice d'inertie I du drone.

$$E_{rot} = \frac{1}{2}\boldsymbol{\Omega}^T I \boldsymbol{\Omega} \tag{2.10}$$

Le vecteur vitesse de rotation dans le repère monde, $\boldsymbol{\Omega}$, s'exprime en fonction des angles d'Euler et des vitesses de rotation propre comme dans l'équation (2.11).

$$\boldsymbol{\Omega} = \begin{pmatrix} \dot{\phi} - \dot{\psi} sin\theta \\ \dot{\theta} cos\phi + \dot{\phi} cos\theta sin\phi \\ \dot{\psi} cos\phi cos\theta + \dot{\theta} sin\phi \end{pmatrix} \tag{2.11}$$

Pour un drone bien équilibré, la matrice d'inertie I s'exprime sous forme d'une matrice diagonale.

$$I = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} \tag{2.12}$$

La seule énergie potentielle appliquée au drone est celle créée par sa masse. L'équation (2.13) donne l'énergie potentielle de pesanteur en fonction de la masse m, de l'altitude z du drone et de l'accélération de pesanteur g.

$$U = mgz \tag{2.13}$$

Le modèle d'Euler-Lagrange s'obtient à partir des dérivées du Lagrangien et des forces qui s'appliquent sur le système. $\mathbf{F}_\xi$ représente les forces de translation essentiellement (dues à la gravité et à la force de poussée) et $\boldsymbol{\tau}$ les couples généralisés de rotation.

$$\frac{d}{dt}\frac{dL}{d\dot{q}} - \frac{dL}{dq} = \begin{Bmatrix} \mathbf{F}_\xi \\ \boldsymbol{\tau} \end{Bmatrix} \tag{2.14}$$

En remplaçant le Lagrangien (2.8) dans (2.14), on obtient l'équation (2.15) avec la partie translationnelle et la partie rotationnelle. La partie translationnelle dépend donc de $\mathbf{F}_\xi$ la force due à la force de poussée ramenée dans le repère du monde, de la gravité et des accélérations du drone. La partie rotationnelle quant-à elle dépend de la matrice d'inertie I ramenée dans le

repère monde, des moments créés par le drone, ses vitesse et accélération angulaires.

$$\begin{aligned} m\,\ddot{\boldsymbol{\xi}} + mg\,\mathbf{e}_z &= \mathbf{F}_\xi \\ I\ddot{\boldsymbol{\eta}} + \dot{I}\dot{\boldsymbol{\eta}} - \frac{1}{2}\frac{d}{d\boldsymbol{\eta}}(\dot{\boldsymbol{\eta}}^T I\dot{\boldsymbol{\eta}}) &= \boldsymbol{\tau} \end{aligned} \tag{2.15}$$

Dans (2.16), la dynamique de translation s'obtient en écrivant la force de poussée totale dans le référentiel monde. La dynamique de rotation est plus complexe. Pour le roulis et le tangage, le premier terme est le moment généré par les forces de poussée. Pour le lacet, chaque rotor génère une force de traînée en plus de celle de poussée. Ces forces de traînée créent un moment autour de l'axe de lacet z_d. Les termes qui suivent sont les termes de couplage des vitesses et les effets gyroscopiques dépendant de l'inertie j_r de chaque rotor.

$$\begin{cases} m\ddot{x} = (sin(\phi)sin(\psi) + cos(\phi)cos(\psi)sin(\theta))F \\ m\ddot{y} = (cos(\phi)sin(\psi)sin(\theta) - sin(\phi)cos(\psi))F \\ m\ddot{z} = (cos(\theta)cos(\phi))F - mg \\ I_{xx}\ddot{\phi} = bl(w_2^2 - w_4^2) + (I_{yy} - I_{zz})\dot{\theta}\dot{\psi} - j_r\dot{\theta}(-w_1 + w_2 - w_3 + w_4) \\ I_{yy}\ddot{\theta} = bl(w_3^2 - w_1^2) + (I_{yy} - I_{zz})\dot{\phi}\dot{\psi} + j_r\dot{\phi}(-w_1 + w_2 - w_3 + w_4) \\ I_{zz}\ddot{\psi} = d(w_2^2 + w_4^2 - w_1^2 - w_3^2) + (I_{xx} - I_{yy})\dot{\theta}\dot{\phi} \end{cases} \tag{2.16}$$

A partir de ces équations, et en ignorant les effets gyroscopiques et les termes de couplage d'ordre 2, on obtient la dynamique simplifiée suivante (2.17). On note U_1 une variable intermédiaire qui représente la poussée générée par le drone, U_2, U_3 et U_4 représentent les moments associés aux angles d'Euler. A noter que la dynamique de translation dépend, de manière non-linéaire, de différents angles d'attitude, ce qui rend le contrôle plus complexe.

$$\begin{cases} m\ddot{x} = U_1(sin(\phi)sin(\psi) + cos(\phi)cos(\psi)sin(\theta)) \\ m\ddot{y} = U_1(cos(\phi)sin(\psi)sin(\theta) - sin(\phi)cos(\psi)) \\ m\ddot{z} = U_1 cos(\theta)cos(\phi) - mg \\ I_{xx}\ddot{\phi} = U_2 \\ I_{yy}\ddot{\theta} = U_3 \\ I_{zz}\ddot{\psi} = U_4 \end{cases} \tag{2.17}$$

2.3 Commande pour le suivi de trajectoires

On considère une trajectoire désirée représentée par une courbe de Bézier, la méthode d'obtention de cette courbe sera présentée dans le chapitre suivant. L'architecture du contrôleur est schématisée Figure 2.3. Le contrôleur prend en entrée la courbe consigne de Bézier avec les profils correspondant de vitesse et d'accélération. Les entrées du contrôleur sont $x_d(t), y_d(t), z_d(t)$ pour la position désirée, $\dot{x}_d(t), \dot{y}_d(t), \dot{z}_d(t)$ pour la vitesse désirée et

$\ddot{x}_d(t), \ddot{y}_d(t), \ddot{z}_d(t)$ pour l'accélération désirée. Par souci de lisibilité, la variable t sera omise. Un premier niveau de contrôle dans le plan (x,y) est utilisé, il se base sur la partie translation de la dynamique (2.17). Les angles de tangage désiré ϕ_d et de roulis désiré θ_d sont obtenus à la fin de cette première étape. Des variables intermédiaires sont ensuite utilisées dans un second contrôleur pour générer les commandes U_2 et U_3. La consigne z_d est utilisée pour contrôler l'altitude, la commande U_1 est générée dans ce but. La commande U_3 de contrôle de lacet, dans notre cas, est laissée fixe.

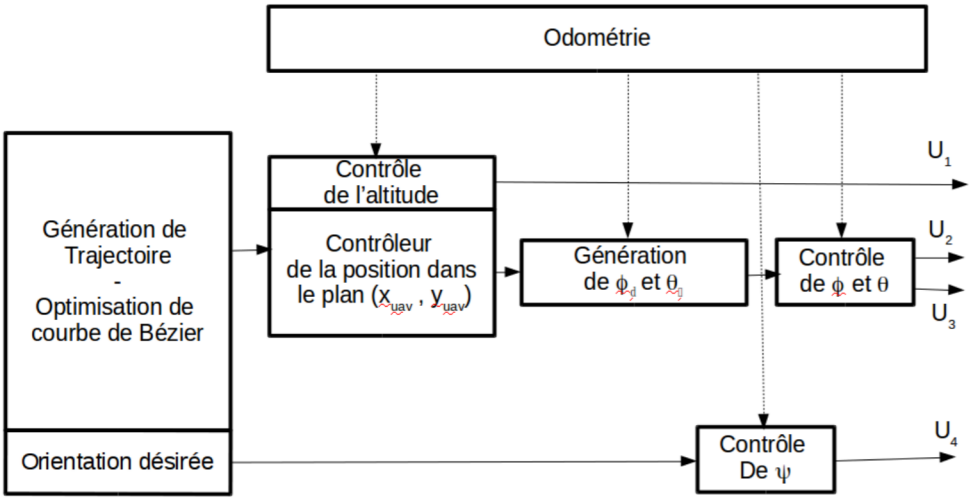


FIGURE 2.3 – Architecture de contrôle du drone. La courbe de Bézier en entrée, comprend la position, la vitesse et l'accélération désirées. Le contrôleur calcule les commandes de déplacement dans le plan (x, y), la commande de l'altitude et du lacet.

Le modèle (2.17) est non linéaire. Pour simplifier le problème, on rassemble les termes non linéaires en deux commandes intermédiaires, u_x et u_y (2.18).

$$\begin{aligned}
m\ddot{x} &= U_1(sin(\phi)sin(\psi) + cos(\phi)cos(\psi)sin(\theta)) &&= U_1 u_x \\
m\ddot{y} &= U_1(cos(\phi)sin(\psi)sin(\theta) - sin(\phi)cos(\psi)) &&= U_1 u_y \\
m\ddot{z} &= U_1(cos(\theta)cos(\phi)) - mg &&= U_1(cos(\theta)cos(\phi)) - mg \\
u_x &= sin(\phi)sin(\psi) + cos(\phi)cos(\psi)sin(\theta) \\
u_y &= cos(\phi)sin(\psi)sin(\theta) - sin(\phi)cos(\psi)
\end{aligned} \tag{2.18}$$

Il est ensuite possible d'utiliser ces deux commandes intermédiaires pour calculer les angles de roulis et de tangage désirés (2.19). A noter qu'avec cette méthode, une singularité apparaît, ce qui rend la commande impossible pour des angles de 90°. Ce n'est pas une commande souhaitable étant donné que le drone se retrouverait à la verticale.

$$\begin{aligned} sin(\phi_d) &= u_x sin(\psi_d) - u_y cos(\psi_d) \\ sin(\theta_d) &= \frac{u_x cos(\psi_d) + u_y sin(\psi_d)}{cos(\phi_d)} \end{aligned} \tag{2.19}$$

Sous l'hypothèse de petits angles, on peut linéariser le modèle (2.17) autour d'une attitude horizontale,

$$\begin{cases} m\ddot{x} = U_1(\phi sin(\psi) + cos(\psi)\theta) \\ m\ddot{y} = U_1(sin(\psi)\theta - \phi cos(\psi)) \\ m\ddot{z} = U_1 - mg \end{cases} \tag{2.20}$$

Il est alors possible d'exprimer les angles de roulis et de tangage en fonction des déplacements dans le plan horizontal et de la force de poussée.

$$\begin{aligned} \theta_d &= \frac{cos(\psi)m\ddot{x} + sin(\psi)m\ddot{y}}{U_1} \\ \phi_d &= \frac{sin(\psi)m\ddot{x} - cos(\psi)m\ddot{y}}{U_1} \end{aligned} \tag{2.21}$$

2.4 Contrôleur PID en position du modèle linéarisé

Un PID est utilisé en premier lieu pour générer la commande U_1. Il est associé à une commande en boucle ouverte pour compenser le poids. En particulier, pour un vol stationnaire, la force n'est utilisée qu'à compenser le poids. En cas de perturbation ou d'un changement de commande, U_1 est déterminée par,

$$\begin{aligned} e_z &= z_d - z \\ \dot{e}_z &= \dot{z}_d - \dot{z} \\ U_1 &= m(g + \ddot{z}_d + k_{pz}e_z + k_{dz}\dot{e}_z + k_{iz}\int_0^t e_z d\tau) \end{aligned} \tag{2.22}$$

La génération d'un angle de roulis désiré dépend des déplacements en x-y comme suit,

$$\begin{aligned} &e_x = x_d - x \\ &\dot{e}_x = \dot{x}_d - \dot{x} \\ &m\left(\ddot{x}_d + k_{px}e_x + k_{dx}\dot{e}_x + k_{ix}\int_0^t e_x d\tau\right)cos\psi + m\left(\ddot{y}_d + k_{py}e_y + k_{dy}\dot{e}_y + k_{iy}\int_0^t e_y d\tau\right)sin\psi = U_1\theta_d \end{aligned} \tag{2.23}$$

De même que pour le roulis, le tangage est défini par les déplacements en x-y, le lacet et la

force de poussée, par,

$$\begin{aligned} &e_y = y_d - y \\ &\dot{e}_y = \dot{y}_d - \dot{y} \\ &m\left(\ddot{y}_d + k_{py}e_y + k_{dy}\dot{e}_y + k_{iy}\int_0^t e_y d\tau\right) sin\psi - m\left(\ddot{x}_d + k_{px}e_x + k_{dx}\dot{e}_x + k_{ix}\int_0^t e_x d\tau\right) cos\psi = U_1\psi_d \end{aligned} \tag{2.24}$$

Les gains des contrôleurs k_{ij} sont déterminés empiriquement.

2.5 Contrôleur de position par back-stepping

L'utilisation d'une technique de back-stepping permet de garantir la stabilité [Benzaid et al., 2018], en utilisant une fonction de Lyapunov pour assurer la convergence désirée. La Figure 2.4 résume le déroulement du contrôle en altitude : une première erreur de position e_{z_1} est calculée, une commande virtuelle est ensuite obtenue et comparée à $\dot{z}$ pour construire l'erreur en vitesse e_{z_2}. Finalement, la commande U_1 est générée pour contrôler l'altitude. Des fonctions de Lyapunov sont utilisées à chaque étape pour assurer la convergence de e_{z_1} et de e_{z_2} vers 0.

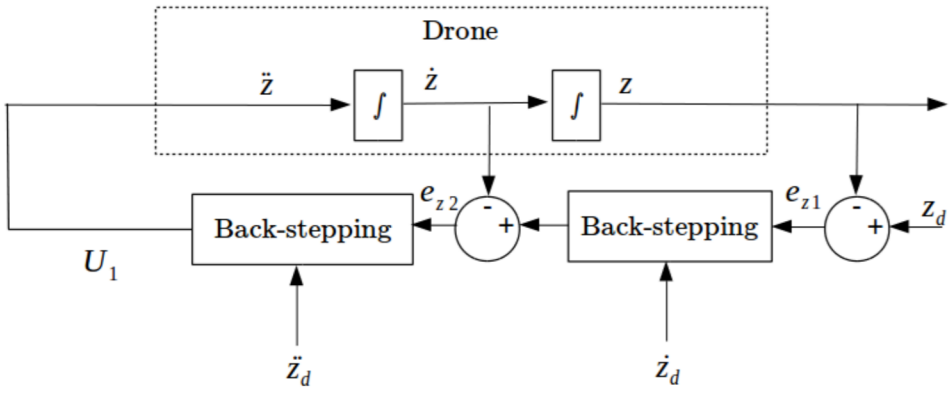


FIGURE 2.4 – Schéma de commande de l'altitude du drone par Back-stepping.

La première étape consiste à calculer e_{z_1}, l'erreur de suivi entre la position désirée pour suivre la trajectoire z_d et la position du drone z. L'équation (2.25) présente la formulation de $V_{e_{z_1}}$, une première fonction de Lyapunov pour l'erreur e_{z_1}.

$$\begin{aligned} &e_{z_1} = z_d - z, \quad \dot{e}_{z_1} = \dot{z}_d - \dot{z} \\ &V_{e_{z_1}} = \frac{1}{2}\left[e_{z_1}^2 + \alpha_1\left(\int_0^t e_{z_1}(\tau)d\tau\right)^2\right] \end{aligned} \tag{2.25}$$

$V_{e_{z_1}}$ est définie positive et sa dérivée temporelle $\dot{V}_{e_{z_1}}$ (2.26) doit être semi-définie négative pour

garantir la convergence de l'erreur e_{z_1} vers 0 et ainsi de z vers z_d. α_1 est une constante positive qui servira de paramètre pour choisir la vitesse de convergence désirée.

$$\dot{V}_{e_{z_1}} = e_{z_1}\dot{e}_{z_1} + \alpha_1 e_{z_1}\int_0^t e_{z_1}(\tau)d\tau = e_{z_1}\left(\dot{z}_d - \dot{z} + \alpha_1\int_0^t e_{z_1}(\tau)d\tau\right) \tag{2.26}$$

La commande virtuelle $\dot{z}$ est construite de telle sorte à s'assurer que $\dot{V}_{e_{z_1}}$ soit semi-définie négative. Le cœur du backstepping est présent à cette étape. Comme nous pouvons le constater dans l'équation (2.27), la première étape consiste à compenser la partie non maîtrisée de $\dot{V}_{e_{z_1}}$ puis un terme est ajouté pour assurer la convergence désirée. β_1 est une constante positive à régler (comme pour α_1) en fonction de la vitesse désirée de convergence de l'erreur e_{z_1}.

$$\dot{z} = \dot{z}_d + \alpha_1\int_0^t e_{z_1}(\tau)d\tau + \beta_1 e_{z_1} \tag{2.27}$$

En remplaçant (2.27) dans (2.26), on obtient (2.28). $\dot{V}_{e_{z_1}}$ est bien semi-définie négative, la convergence de l'erreur e_{z_1} vers 0 est donc assurée.

$$\dot{V}_{e_{z_1}} = -\beta_1 e_{z_1}^2 \quad \leq 0 \tag{2.28}$$

Dans la deuxième étape, e_{z_2} est définie comme étant l'erreur entre la vitesse virtuelle telle que choisie dans (2.27) et la vitesse de l'altitude $\dot{z}$, soit,

$$e_{z_2} = \dot{z}_d + \alpha_1\int_0^t e_{z_1}(\tau)d\tau + \beta_1 e_{z_1} - \dot{z} \tag{2.29}$$

Sa dérivée temporelle s'écrit comme suit :

$$\dot{e}_{z_2} = \ddot{z}_d + \alpha_1 e_{z_1} + \beta_1\dot{e}_{z_1} - \ddot{z} \tag{2.30}$$

Dans (2.29), nous pouvons reconnaître $\dot{z}_d - \dot{z}$ et donc exprimer $\dot{e}_{z_1}$ en fonction de e_{z_2} et e_{z_1}.

$$\dot{e}_{z_1} = \dot{z}_d - \dot{z} = e_{z_2} - \alpha_1\int_0^t e_{z_1}(\tau)d\tau - \beta_1 e_{z_1} \tag{2.31}$$

Dans (2.30), il est possible de remplacer $\dot{e}_{z_1}$ par l'expression ci-dessus et $\ddot{z}$ par l'expression de la dynamique d'Euler-Lagrange, ce qui donne,

$$\dot{e}_{z_2} = \ddot{z}_d + \alpha_1 e_{z_1} + \beta_1\left(e_{z_2} - \alpha_1\int_0^t e_{z_1}(\tau)d\tau - \beta_1 e_{z_1}\right) + g - \frac{cos\phi cos\theta}{m}U_1 \tag{2.32}$$

Dans la première étape, une commande virtuelle $\dot{z}$ a été choisie pour assurer la convergence de z. Pour que cela fonctionne, il faut s'assurer que la vitesse réelle $\dot{z}$ converge vers cette

commande virtuelle. Une fonction de Lyapunov est donc choisie telle que :

$$V(e_{z_1}, e_{z_2}) = \frac{1}{2}\left[e_{z_1}^2 + e_{z_2}^2 + \alpha_1 \left(\int_0^t e_{z_1}(\tau)d\tau\right)^2\right] \tag{2.33}$$

Cette fonction de Lyapunov est définie positive, pour avoir une convergence de l'erreur, sa dérivée doit être semi-définie négative. Elle s'écrit :

$$\dot{V}(e_{z_1}, e_{z_2}) = e_{z_1}\left(\dot{e}_{z_1} + \alpha_1 \int_0^t e_{z_1}(\tau)d\tau\right) + e_{z_2}\dot{e}_{z_2} \tag{2.34}$$

En remplaçant $\dot{e}_{z_1}$ et $\dot{e}_{z_2}$:

$$\dot{V}(e_{z_1}, e_{z_2}) = -\beta_1 e_{z_1}^2 + \beta_1 e_{z_2}^2 + (1+\alpha_1-\beta_1^2)e_{z_1}e_{z_2} - \alpha_1\beta_1 e_{z_2}\int_0^t e_{z_1}(\tau)d\tau + \ddot{z}_d e_{z_2} + g e_{z_2} - \frac{cos\phi cos\theta}{m}e_{z_2}U_1 \tag{2.35}$$

La commande U_1 est donc choisie de sorte que $\dot{V}(e_{z_1}, e_{z_2})$ soit semi-définie négative.

$$\begin{aligned} e_{z_1} &= z_d - z \\ e_{z_2} &= \dot{z_d} - \dot{z} + \alpha_1 \textstyle\int_0^t e_{z_1}(\tau)d\tau \\ U_1 &= \frac{m}{cos\theta cos\phi}(g + \ddot{z_d} + (1+\alpha_1-\beta_1^2)e_{z_1} + (\beta_1+\beta_2)e_{z_2} - \alpha_1\beta_1 \textstyle\int_0^t e_{z_1}(\tau)d\tau) \end{aligned} \tag{2.36}$$

avec β_2 une constante positive. On obtient ainsi :

$$\dot{V}(e_{z_1}, e_{z_2}) = -\beta_1 e_{z_1}^2 - \beta_2 e_{z_2}^2 \tag{2.37}$$

$\dot{V}(e_{z_1}, e_{z_2}) < 0 \ \forall \ (e_{z_1}, e_{z_2}) \neq 0$ et $\dot{V}(0) = 0$, ce qui conclut à la stabilité asymptotique de z.

Le même procédé est appliqué pour les erreurs de suivi des positions longitudinale e_{x_1} et latérale e_{y_1}, et les erreurs de suivi des vitesses correspondantes e_{x_2} et e_{y_2},

$$\begin{aligned} e_{x_1} &= x_d - x \\ e_{x_2} &= \dot{x_d} - \dot{x} + \alpha_2 \int_0^t e_{x_1}(\tau)d\tau + \beta_3 e_{x_1} \\ e_{y_1} &= y_d - y \\ e_{y_2} &= \dot{y_d} - \dot{y} + \alpha_3 \int_0^t e_{y_1}(\tau)d\tau + \beta_5 e_{y_1} \end{aligned} \tag{2.38}$$

avec α_i et β_j sont des constantes strictement positives à régler en fonction de la vitesse désirée de convergence des erreurs.

En choisissant des fonctions de Lyapunov définies positives dont les dérivées sont semi-

définies négatives, on peut déduire les lois de commande des positions longitudinale et latérale comme suit,

$$\begin{aligned} u_x &= \frac{m}{U_1}(\ddot{x_d} + (1+\alpha_2-\beta_3^2)e_{x_1} + (\beta_3+\beta_4)e_{x_2} - \alpha_2\beta_3\int_0^t e_{x_1}(\tau)d\tau) \\ u_y &= \frac{m}{U_1}(\ddot{y_d} + (1+\alpha_3-\beta_5^2)e_{y_1} + (\beta_5+\beta_6)e_{y_2} - \alpha_3\beta_5\int_0^t e_{y_1}(\tau)d\tau) \end{aligned} \tag{2.39}$$

2.6 Contrôle des angles d'Euler

Que ce soit pour la commande avec petits angles ou avec commande intermédiaire, un PID est utilisé pour suivre les angles désirés. La première étape consiste à calculer l'erreur entre la trajectoire désirée et l'attitude actuelle du drone. Il faut ensuite calculer la dérivée de l'erreur ainsi que son intégrale. Un contrôleur PID classique découplé est ensuite appliqué sur chacun des angles, avec des gains $K_{p\bullet}$, $K_{d\bullet}$ et $K_{i\bullet}$. Cela permet de gérer la réactivité, les dépassements, la stabilité et la robustesse du contrôleur.

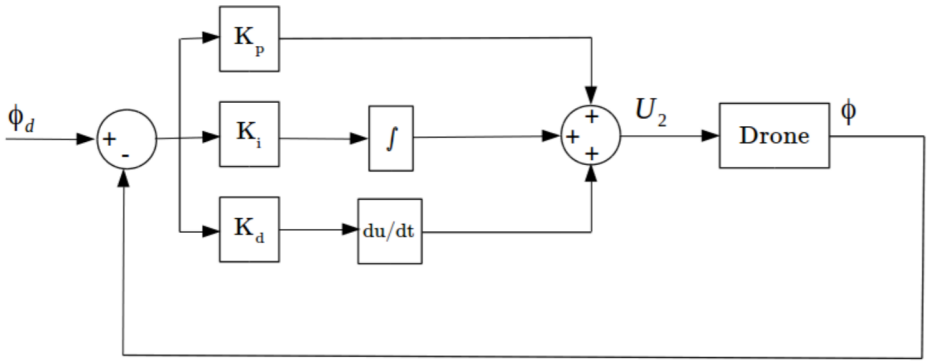


FIGURE 2.5 – Schéma du contrôleur PID appliqué à aux angles d'Euler, ici, l'angle de roulis.

$$\begin{aligned} e_\phi &= \phi_d - \phi \\ U_2 &= k_{p_\phi}e_\phi + k_{d_\phi}\frac{de_\phi}{dt} + k_{i_\phi}\int_0^t e_\phi(\tau)d\tau \end{aligned} \tag{2.40}$$

$$\begin{aligned} e_\theta &= \theta_d - \theta \\ U_3 &= k_{p_\theta}e_\theta + k_{d_\theta}\frac{de_\theta}{dt} + k_{i_\theta}\int_0^t e_\theta(\tau)d\tau \end{aligned} \tag{2.41}$$

$$\begin{aligned} e_\psi &= \psi_d - \psi \\ U_4 &= k_{p_\psi}e_\psi + k_{d_\psi}\frac{de_\psi}{dt} + k_{i_\psi}\int_0^t e_\psi(\tau)d\tau \end{aligned} \tag{2.42}$$

2.7 Commande des moteurs

Une fois la loi de commande globale U_1, U_2, U_3 et U_4 a été construite, il suffit d'utiliser la géométrie et les coefficient aérodynamiques du drone pour déterminer les consignes de vitesses à envoyer aux moteurs.

En fonction de la configuration choisie, en "+" ou en "x", définissant l'orientation de l'axe de roulis Figure 2.6, les relations entre les entrées de commande U_i et les vitesses de rotation des moteurs sont exprimées comme suit,

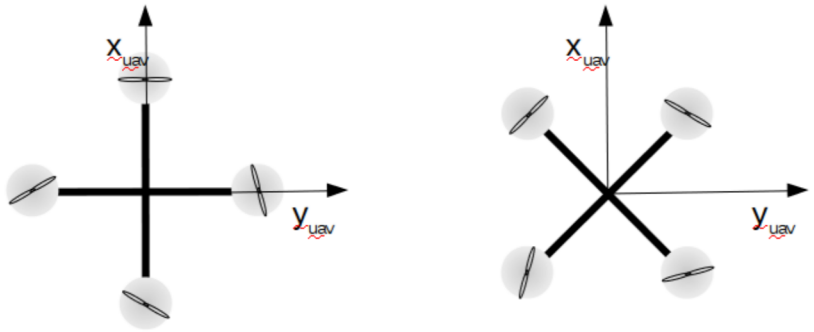


FIGURE 2.6 – Les deux configurations possibles : en "+" à gauche, l'axe de roulis est porté par un segment du drone, en "x" à droite, l'axe de roulis est la médiane de deux segments.

— Quadrotor configuration "+"

$$U = \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{pmatrix} = \begin{pmatrix} b & b & b & b \\ 0 & bl & 0 & -bl \\ -bl & 0 & bl & 0 \\ d & -d & d & -d \end{pmatrix} \begin{pmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{pmatrix} \tag{2.43}$$

— Quadrotor configuration "x"

$$U = \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{pmatrix} = \begin{pmatrix} b & b & b & b \\ bl\frac{\sqrt{2}}{2} & bl\frac{\sqrt{2}}{2} & -bl\frac{\sqrt{2}}{2} & -bl\frac{\sqrt{2}}{2} \\ -bl\frac{\sqrt{2}}{2} & bl\frac{\sqrt{2}}{2} & bl\frac{\sqrt{2}}{2} & -bl\frac{\sqrt{2}}{2} \\ d & -d & d & -d \end{pmatrix} \begin{pmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{pmatrix} \tag{2.44}$$

Les paramètres b, d et l représentent respectivement le coefficient de poussée, le coefficient de traînée et la demi-envergure du drone.

2.8 Commande prédictive (MPC)

Le commande prédictive, dite MPC (Model Predictive Control) et schématisée Figure 2.7, est un contrôleur basé sur la prédiction de modèle et sur l'optimisation des commandes sur une horizon de plusieurs étapes en utilisant une fonction coût.

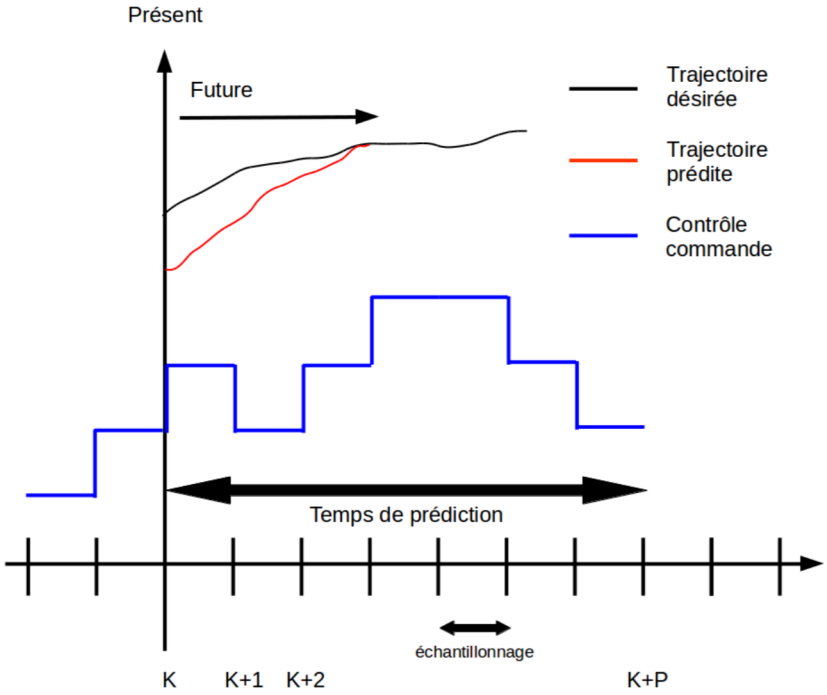


FIGURE 2.7 – Fonctionnement temporel de la commande prédictive

L'équation (2.45) donne un exemple simple d'une fonction coût à optimiser pour un MPC. Considérons une trajectoire à suivre de telle sorte qu'à chaque prédiction le MPC permet d'obtenir une position désirée ξ_{d_i}. A chaque étape, une position ξ_i est prédite par le MPC par optimisation du critère J (2.45). Ainsi, la fonction coût cherche à minimiser l'erreur entre la trajectoire désirée et l'état du drone tout en produisant la série de commandes la plus faible possible. Cette fonction coût peut prendre d'autres formes en ajoutant des saturations sous la forme de contraintes. L'évitement d'obstacles ou la minimisation d'énergie peuvent également être intégrés à cette étape.

$$J = \sum_{i=1}^{P} (\xi_{d_i} - \xi_i)^2 + \sum_{i=1}^{P} U_i^2 \tag{2.45}$$

2.9 Comparaison des différentes lois de commande

Des tests comparatifs des trois méthodes ont été réalisés sur le simulateur GAZEBO. Les Figure 2.8 et Figure 2.9 présentent des exemples de suivis de trajectoire quelconque et courbe

de Bézier, par le drone en utilisant les trois lois de commande présentées précédemment (PID, backstepping et MPC).

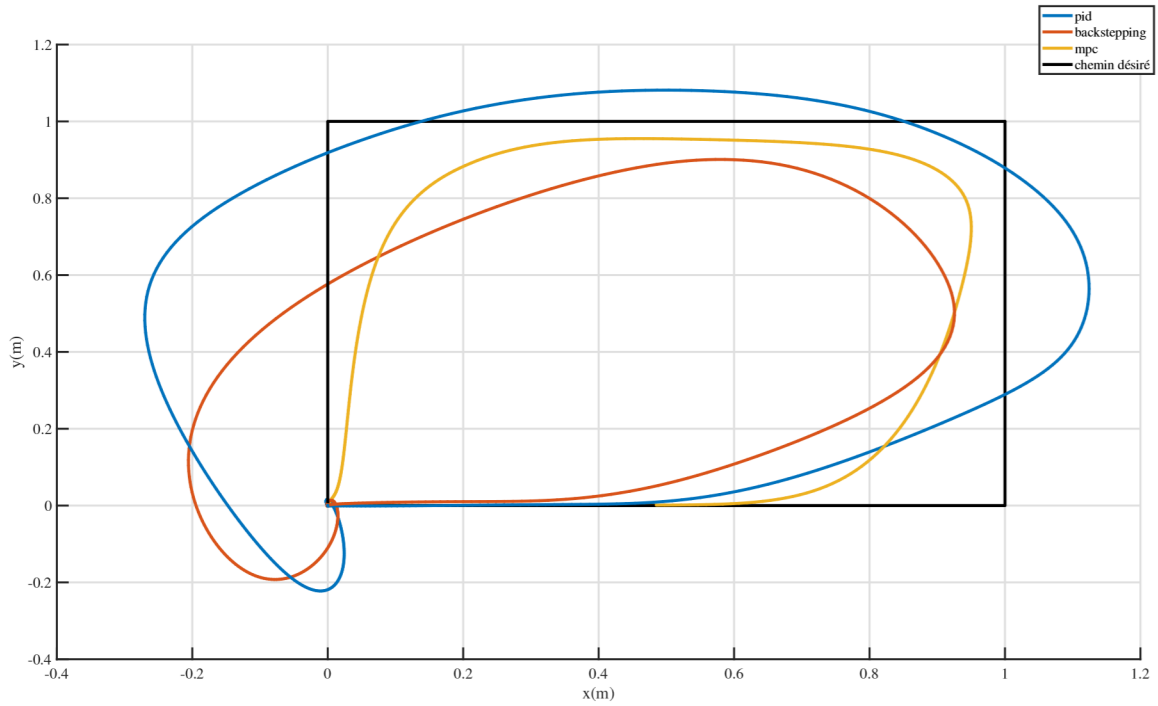


FIGURE 2.8 – Comparaison des trajectoires obtenues par les trois méthodes de contrôle : MPC en *jaune*, backstepping en *rouge* et PID en *bleu*. Le chemin consigne est ici le carré *noir*.

Le contrôleur MPC génère des trajectoires très proches de la consigne car il cherche à optimiser la commande pour réduire la distance par rapport au chemin objectif. Le contrôleur backstepping est plus éloigné de la consigne mais il sera plus robuste à d'éventuels perturbations dans l'environnement et du modèle. On constate généralement que le contrôleur PID est plus éloigné encore et il est moins robuste que le backstepping. Les vitesses moyennes des moteurs sont indiquées en Figure 2.10.

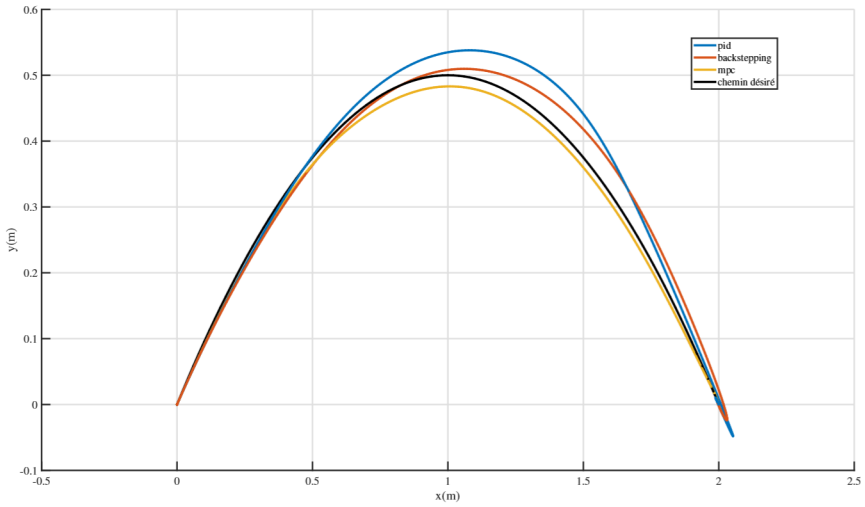


FIGURE 2.9 – Comparaison des trajectoires obtenues par les trois méthodes de contrôle : MPC en *jaune*, backstepping en *rouge* et PID en *bleu*. Le chemin consigne est ici la courbe de Bézier en *noir*.

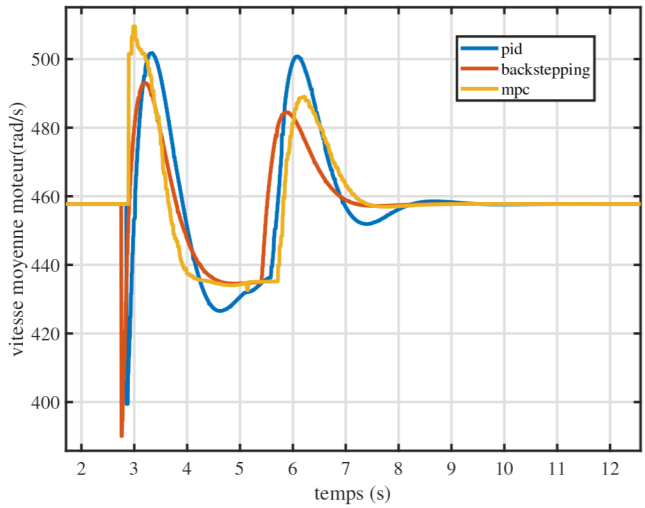


FIGURE 2.10 – Comparaison de la vitesse moyenne des moteurs lors du suivi d'une trajectoire type courbe de Bézier, le contrôleur MPC en *jaune*, le contrôleur backstepping en *rouge*, et le contrôleur PID en *bleu*.

2.10 Conclusion

Dans ce chapitre, nous avons introduit les modèles et les contrôleurs utilisés dans ce travail de thèse. La dynamique du drone a été obtenue à partir de la modélisation d'Euler-Lagrange.

Nous utilisons plusieurs contrôleurs en cascade pour piloter le drone pour suivre des trajectoires exprimées par des courbes de Bézier, ces dernières feront l'objet du chapitre suivant. Une commande non linéaire est introduite pour les déplacements plus agressifs ne respectant pas le modèle petits angles.

On a introduit trois différents contrôleurs qui seront exploités dans les résultats. Un contrôleur PID en premier lieu qui a pour avantage d'être très simple à réaliser et à interpréter. Le contrôleur de type back-stepping a pour avantage d'assurer la convergence et la stabilité des erreurs de suivi de trajectoires. Finalement, le MPC permet de prendre simplement en considération l'ajout de contraintes au contrôle.

Chapitre 3

Planificateur de trajectoire basé sur les courbes de Bézier

Ce chapitre décrit un planificateur de trajectoire et de mouvement basé sur un processus d'optimisation et une perception locale. Un chemin ne contient que des informations géométriques alors qu'une trajectoire contient aussi des informations temporelles et dynamiques. Le planificateur développé dans ce mémoire utilise des informations d'une perception omnidirectionnelle, obtenues, par exemple avec plusieurs caméras 3D embarquées sur le drone (un modèle de type Intel Realsense peut être utilisé). On supposera que l'odométrie du drone est connue. De ce fait, la position ξ_{B_w} et la vitesse $\dot{\xi}_{B_w}$ du drone sont disponibles à tout instant. On supposera également qu'un chemin global est donné pour réaliser une mission. Cette mission peut être de type exploration si on veut créer une carte ou de type surveillance en passant par différents points de passage. Lors d'une exploration, le planificateur de trajectoire peut être configuré pour suivre la direction de la plus proche zone inconnue de la carte. Pour atteindre un point désiré dans un le référentiel fixe ou mobile du drone, un algorithme heuristique de type A* [Hart et al., 1968] ou probabiliste, de type RRT (*Rapidly exploring Random Tree*) [LaValle et James J. Kuffner, 2001] peut être utilisé. On supposera donc qu'un chemin objectif est donné, qu'on notera Γ_g. L'objectif et le rôle du planificateur sont de construire localement (dans l'horizon de la sphère de perception) une trajectoire libre d'obstacles qui répond aux contraintes cinématiques et dynamiques de vol. Le processus d'optimisation utilisé dans ce chapitre vise à suivre au mieux la portion locale de Γ_g présente dans la sphère de perception. Pour se faire, on utilisera des approximations par des courbes de Bézier pour générer des trajectoires continues décrites par très peu de points de contrôle $P = \{p_0...p_i...p_n\}$. Les contraintes peuvent être naturellement exprimées en utilisant les dérivées de ces courbes paramétriques. Un certain nombre de contraintes sont ajoutées pour assurer la sécurité et la faisabilité de la trajectoire créée. Les premiers points de contrôle seront également fixés pour assurer une continuité dans la commande. Une trajectoire désirée (position, vitesse et accélération), à un temps précis sera utilisée comme consigne en entrée de l'algorithme de

contrôle du drone.

Les repères de travail

Pour des missions de vol autonome, un premier repère monde est utilisé $R_w = (O_w, x_w, y_w, z_w)$. En général, le repère monde sert de repère absolu pour avoir une base commune pour la position du drone, des obstacles, des objectifs de missions et étant un repère Galiléen, il permet de faire simplement les calculs mécaniques. L'origine O_w du repère monde peut être arbitrairement fixée dans une carte, mais dans le cadre du vol autonome, elle est généralement définie par rapport à la position initiale du drone. Le drone quant à lui possède son propre repère $B_d = (O_d, x_d, y_d, z_d)$. Ce repère est mobile avec le drone et permet de calculer la dynamique du drone et sert de référentiel commun pour les capteurs intégrés.

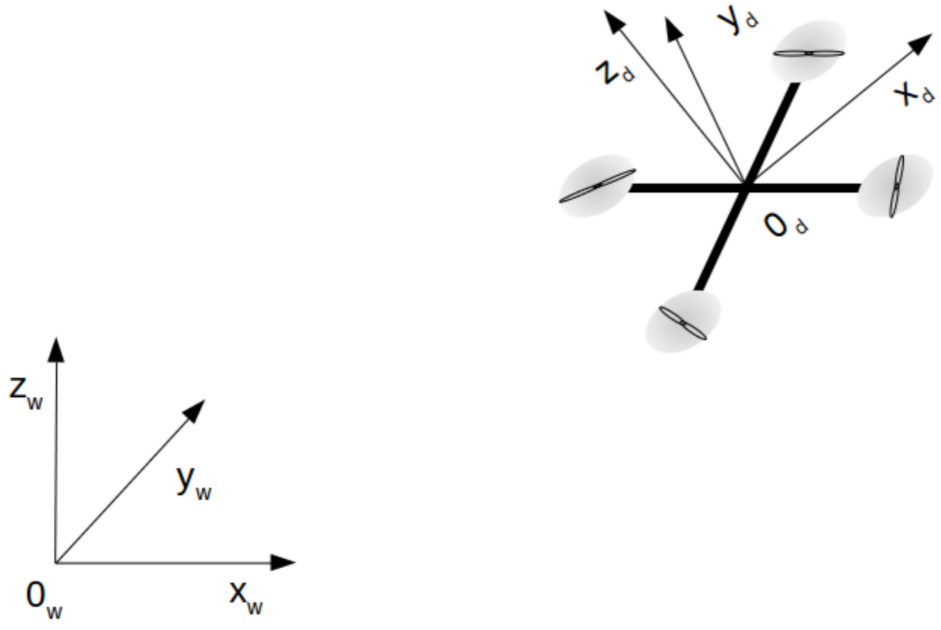


FIGURE 3.1 – Configuration des repères fixe monde R_w et mobile du drone R_d.

3.1 Définition d'un chemin global

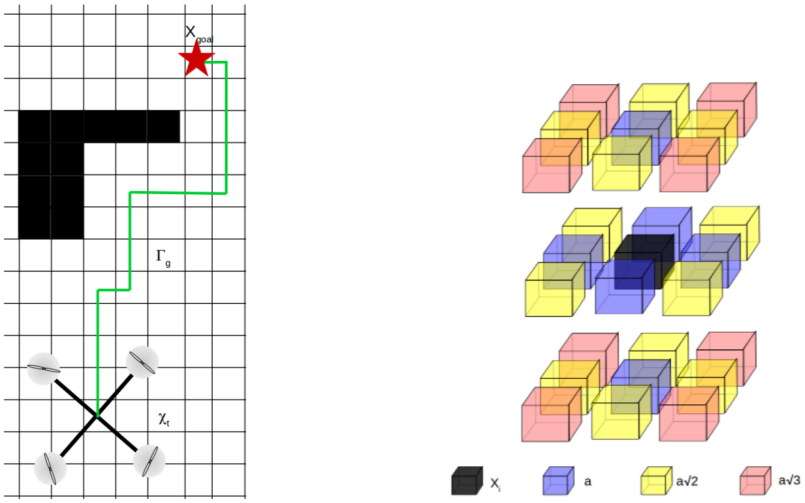
Un chemin global est initialement calculé par le gestionnaire de mission. Dans des environnements partiellement connus, il existe plusieurs façons de calculer un chemin global selon les situations, les missions et les contraintes. Dans notre cas, nous nous concentrerons sur quelques planificateurs de chemin qui relient un point de départ à un point d'arrivée.

L'algorithme A* est le plus fréquemment utilisé pour trouver le plus court chemin Γ_g entre

2 points dans une carte. À partir du point de départ $\Gamma_{g,0}$, l'algorithme cherche itérativement à trouver le voisin $\Gamma_{g,i+1}$ qui optimise une fonction coût (3.1). En général, cette fonction coût cherche à trouver le point qui nous rapproche le plus de l'objectif $(\Gamma_{g,i+1} - goal)^2$ avec le moins de déplacement $(\Gamma_{g,i+1} - \Gamma_{g,i})^2$.

$$f(\Gamma_{g,i+1}) = g(\Gamma_{g,i+1}) + h(\Gamma_{g,i+1}) = (\Gamma_{g,i+1} - goal)^2 + (\Gamma_{g,i+1} - \Gamma_{g,i})^2 \tag{3.1}$$

A gauche de la Figure 3.2, nous avons un exemple de chemin résultant d'un algorithme A*, ainsi qu'un exemple de voisinage en 3D à droite. La couleur décrit le coût de déplacement du centre du cube vers un voisin.



(a) (b)

FIGURE 3.2 – Exemple de chemin global Γ_g généré en utilisant l'algorithme A* en 2D (à gauche). En dimension 3, sont illustrés le voisinage d'un point x avec les différents coûts de déplacement vers le point voisin (à droite).

L'algorithme RRT (*Rapidly exploring Random Rree*), quant à lui, n'utilise pas une discrétisation ordonnée mais aléatoire. A chaque itération, un point aléatoire est choisi. Il est ensuite relié au point le plus proche, cela fini par créer une structure d'arbre (Figure 3.3). Ensuite, on cherche le chemin reliant les deux points en explorant l'arbre (voir exemple Figure 3.3).

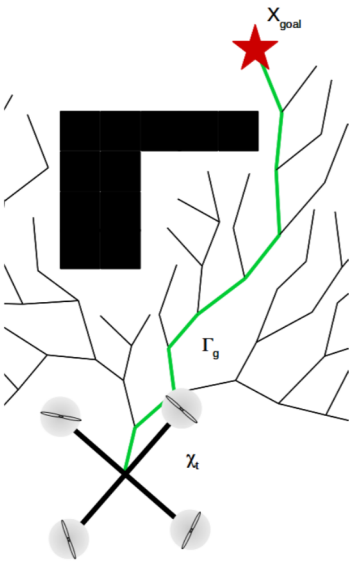


FIGURE 3.3 – Exemple de chemin global Γ_g généré en utilisant l'algorithme RRT. L'espace de recherche généré par exploration aléatoire est indiqué *en noir*, le chemin vers l'objectif est *en vert*.

Si aucune carte n'est disponible, le chemin le plus simple est une ligne droite vers l'objectif, comme sur la Figure 3.4

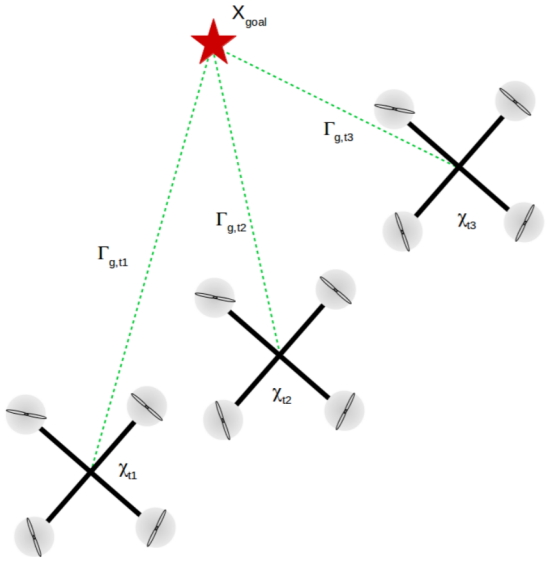


FIGURE 3.4 – Exemple de chemin global Γ_g dans le cas où aucune carte n'est disponible, à chaque étape le chemin *vert* (pointillé) est recalculé.

La Figure 3.5 résume l'interaction entre le planificateur global et le planificateur local avec

et sans carte. En haut, sans carte, le drone ne peut que se diriger vers l'objectif. Cela implique que toute la tâche d'évitement incombe au planificateur local,=. Dans ce cas, le planificateur local peut conduire à des situations de blocage (minima locaux). Dans le second cas, en bas de la figure, une carte est utilisée par le planificateur global pour avoir une première estimation du chemin à prendre pour éviter des obstacles. Cela permet de donner une bonne première estimation au planificateur local. Néanmoins, si l'environnement est très dynamique ou que le chemin global est trop agressif (changement de direction important) et infaisable par le drone (accélération et/ou vitesse supérieure à celle du drone), alors le planificateur local aura des difficultés et devra s'éloigner du chemin global. La Figure 3.6 illustre un exemple de cas où utiliser un planificateur global sans carte ne permet pas de trouver de chemin vers l'objectif.

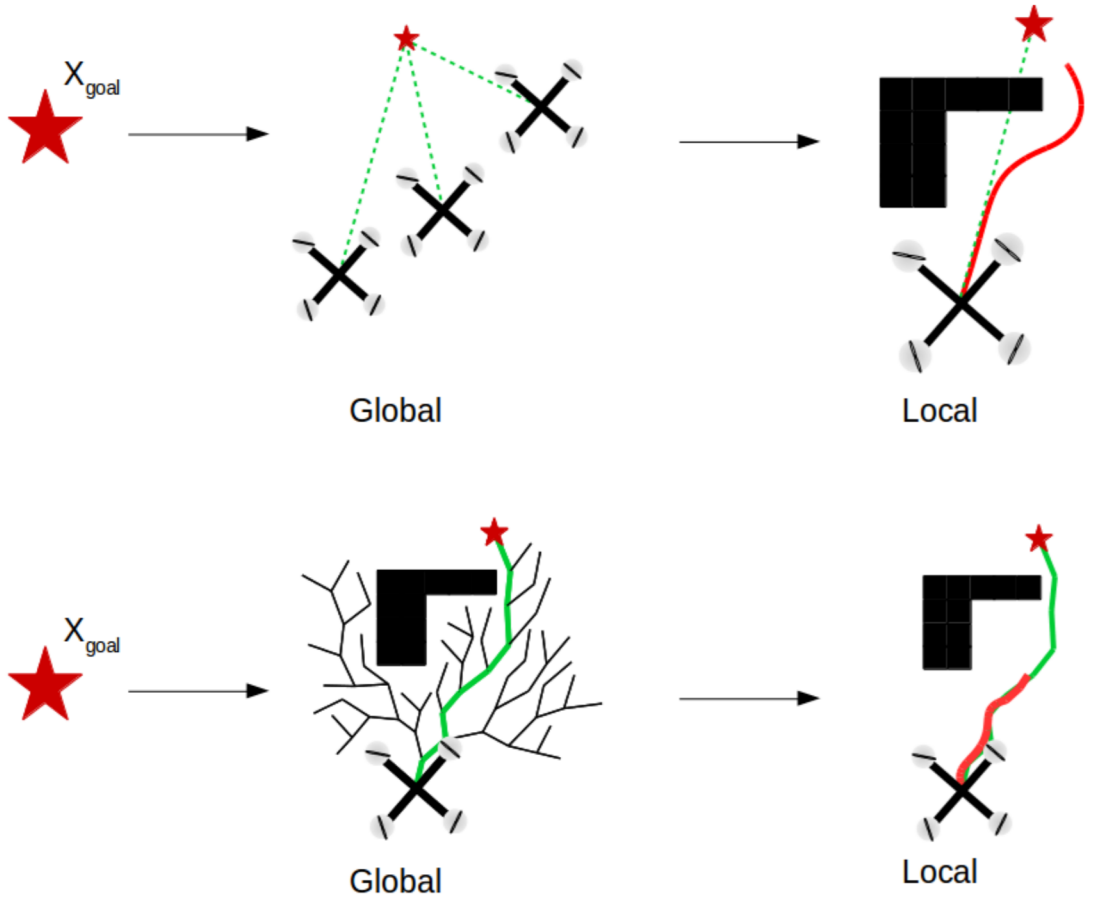


FIGURE 3.5 – Configuration du planificateur de trajectoire dans la cas où aucune carte n'est connue (partie haute) et dans la cas où une carte est disponible (partie basse).

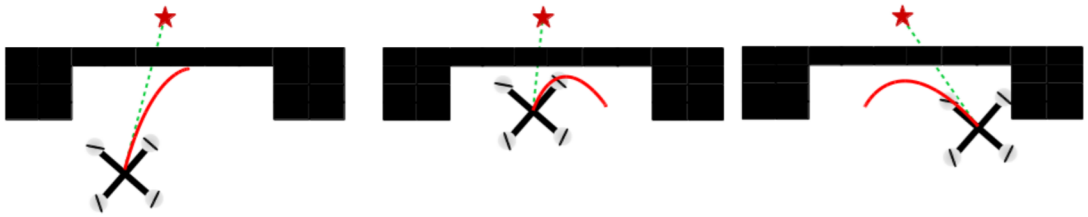


FIGURE 3.6 – Exemple de situation où la planificateur local sans carte se retrouve bloqué dans une boucle et n'arrive pas à trouver de chemin.

3.2 Génération de trajectoire par courbes de Bézier

Les courbes d'interpolation polynomiale sont utilisées en général pour la génération de chemin et/ou de trajectoire avec de bonnes propriétés de régularité. Elles offrent plusieurs avantages, et permettent de générer un chemin continu avec très peu de variables en comparaison aux algorithmes A*, RRT, etc. En effet, ces derniers opèrent sur des discrétisations de l'espace de recherche et ne garantissent pas des propriétés de continuité des vitesses, accélérations. Les approximations paramétriques par courbes polynomiales facilitent également l'intégration de contraintes sur la forme de la courbe et ses dérivées. Il existe plusieurs formulations de ces interpolations telles que les B-splines ou plus spécifiquement les courbes de Bézier utilisant des polynômes de Bernstein. Les courbes de Dubins quant à elles vont générer un chemin optimal formé d'arcs de cercle de courbure maximale reliés tangentiellement par des segments de droites.

Les courbes B-splines et Bézier constituent une base de fonctions polynomiales d'interpolation définies par un ensemble de points de contrôle. Les B-splines sont les plus agiles car un point de contrôle n'affecte qu'une partie de la courbe. Pour les courbes de Bézier, chaque point de contrôle impacte toute la courbe (Figure 3.8). Cela implique que pour une courbe de Bézier, avoir une modification locale demande de reconstruire l'ensemble de la courbe. Générer une courbe avec beaucoup d'ondulations nécessitera un nombre important de points de contrôle. Dans le cadre de génération de trajectoires locales, les oscillations créent plus de bruit sur la commande. De ce fait, les courbes de Bézier pour un drone volant consisteront en des chemins plus lisses avec un impact des points de contrôle sur la courbe plus simple à appréhender.

Pour la plupart des courbes d'interpolation, le premier et le dernier des points de contrôle correspondent au début et à la fin de la courbe. Pour les courbes de Bézier, calculer les dérivées successives et assurer des continuités entre deux courbes Bézier est assez simple en fonction du degré de dérivation. Dans la suite de ce travail, nous utiliserons des interpolations par des courbes de Bézier dans l'expression du problème d'optimisation pour la génération des trajectoires et des fonctions coûts associées.

3.2.1 Formulation des courbes de Bézier

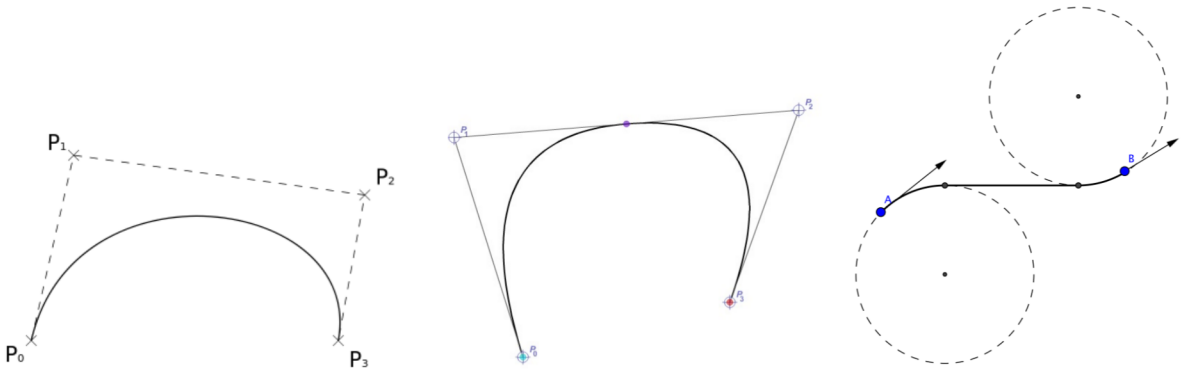
Un ensemble P de $n+1$ points de contrôle (3.2) permet de générer une courbe de Bézier en utilisant l'équation (3.3). Les $b_{i,n}$ sont les polynômes de Bernstein (3.4), de degré n exprimés en l'abscisse curviligne $s \in [0,1]$. On notera $B_t(s)$ la courbe de Bézier calculée par l'optimisation à l'instant t dans l'horizon de perception du drone. La Figure 3.7 montre un exemple de courbe décrite par 4 points de contrôle.

$$P = \{p_0 ... p_i ... p_n\} \tag{3.2}$$

$$B_t(s) = \sum_{i=0}^{n} b_{i,n}(s) p_i \tag{3.3}$$

$$b_{i,n}(s) = \binom{n}{i} (1-s)^{n-i} s^i \quad \text{avec} \quad \binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{3.4}$$

Pour comparaison, la figure 3.7 montre aussi en (b) des B-spline plus agile mais plus complexe et des courbes de Dubins. Elle ne sont pas définiee de la même façon, il est simple de forcer certaines dérivées aux extrémités.



(a) (b) (c)

FIGURE 3.7 – Exemple d'une courbe de Bézier avec 4 points de contrôle (a). B-spline en (b). Courbe de Dubins en (c).

Comme expliqué précédemment, pour les courbes de Bézier, chaque point de contrôle impacte toute la courbe comme on peut le voir sur la Figure 3.8. Le coefficient $b_{0,n}(s)$ de P_0 est très important lorsque s est proche de 0. Ce qui veut dire que la position de P_0 impacte le début de la courbe mais continue à impacter la courbe même pour une abscisse $s > 0.8$.

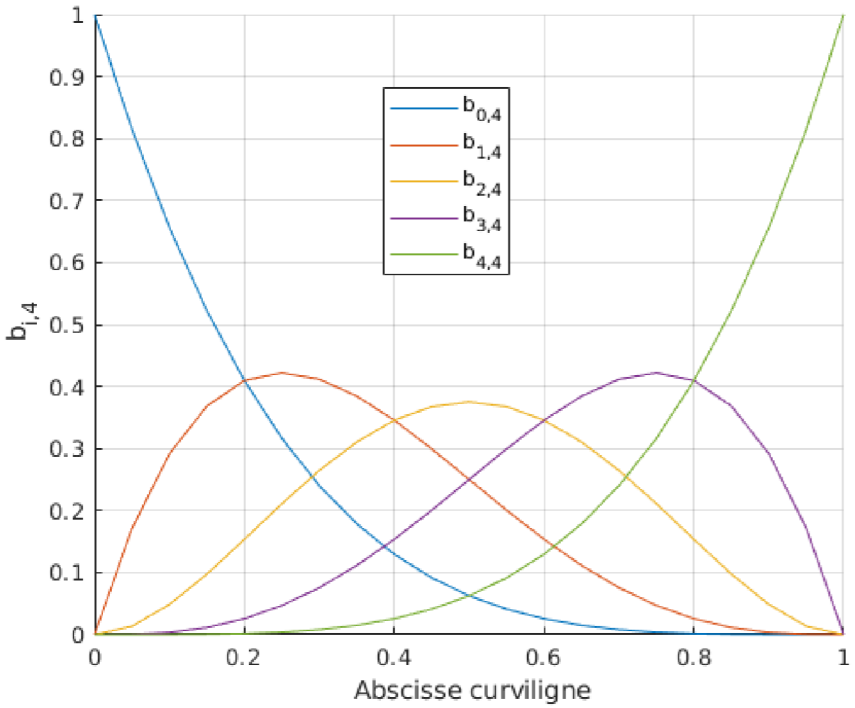


FIGURE 3.8 – Impact des points de contrôle sur une courbe de Bézier. Valeurs des coefficients $(b_{i,n}(s))$ associés aux 5 points de contrôle (p_i) le long d'une courbe de Bézier en fonction de l'abscisse curviligne s.

3.2.2 Relation entre abscisse curviligne et temps

Pour pouvoir associer les dérivées successives de la courbe de Bézier aux vitesses et accélérations du drone, il faut exprimer l'abscisse curviligne en fonction du temps. Le paramètre τ sera utilisé pour imager le temps le long de la courbe de Bézier. $B_t(s(\tau))$ représente la position désirée du système à l'instant τ. L'équation (3.5) exprime l'abscisse curviligne en fonction de $\tau_{f,t}$ et $\tau_{0,t}$, qui seront respectivement le temps désiré pour finir la trajectoire et d'instant de début de la trajectoire. La valeur $\tau_{0,t}$ correspond donc au temps au début du processus d'optimisation.

$$s(\tau) = \frac{\tau - \tau_{0,t}}{\tau_{f,t} - \tau_{0,t}} \tag{3.5}$$

Le temps désiré pour atteindre un point objectif représenté par $\tau_{f,t}$ est un paramètre de réglage. Il peut être déduit en fixant une vitesse moyenne v_{moy} le long de B_t et de la longueur de la courbe B_t.

$$\tau_{f,t} = \tau_{0,t} + \frac{\int_\tau B_t(s(\tau))}{v_{moy}} \tag{3.6}$$

La figure 3.9 montre les correspondances entre le temps physique t et le temps de chaque

courbe τ. A t et $t + \Delta t$ une courbe de Bézier est calculée. B_t va de $\tau_{0,t}$ à $\tau_{f,t}$ alors que $B_{t+\Delta t}$ va de $\tau_{0,t+\Delta t}$ à $\tau_{f,t+\Delta t}$ avec $\tau_{0,t} = t$ et $\tau_{0,t+\Delta t} = t + \Delta t$.

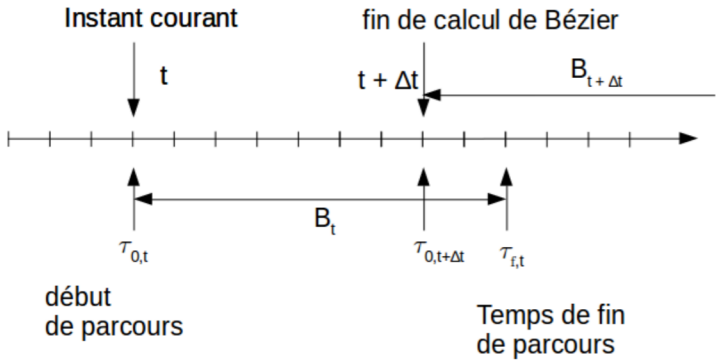


FIGURE 3.9 – Correspondance du temps t avec la variable de temps τ associée à chaque courbe de Bézier.

3.2.3 Dérivées par rapport au temps et par rapport aux points de contrôle

Maintenant que l'on a une fonction qui dépend du temps, il est possible de calculer ses dérivées successives et ainsi obtenir un profil de vitesse et d'accélération pour passer de chemin à trajectoire.

Pour dériver la courbe de Bézier par rapport à τ, on va d'abord dériver la courbe de Bézier par rapport à l'abscisse curviligne puis l'abscisse curviligne par rapport à τ (3.7).

$$B'_t(\tau) = \frac{dB_t(s(\tau))}{d\tau} = \frac{dB_t(s(\tau))}{ds}\frac{ds}{d\tau} = \frac{dB_t(s(\tau))}{ds}\frac{1}{\tau_{f,t} - \tau_{0,t}} \tag{3.7}$$

Les dérivées des courbes de Bézier relativement à l'abscisse curviligne s s'expriment simplement en fonction des polynômes de Bernstein et des points de contrôle (3.8).

$$\frac{dB_t(s)}{ds} = \sum_{i=0}^{n-1} n\ b_{i,n-1}(s)(p_{i+1} - p_i) \tag{3.8}$$

Le profil de vitesse induit par une courbe de Bézier s'écrit ainsi par (3.9), en fonction des temps au début et à la fin de la courbe, des polynômes de Bernstein et des points de contrôle. À noter que pour changer le profil de vitesse, nous ne pouvons que modifier le temps final $\tau_{f,t}$ et les points de contrôle. Le temps final est en fait un facteur d'échelle pour modifier le profil

de vitesse. Changer les points de contrôle va impacter toute la forme de la courbe.

$$\frac{dB_t(s)}{d\tau} = \frac{1}{\tau_{f,t} - \tau_{0,t}} \sum_{i=0}^{n-1} n \; b_{i,n-1}(s)(p_{i+1} - p_i) \tag{3.9}$$

On procède de même pour le profil d'accélération (3.10).

$$\frac{d^2B_t(s)}{d\tau^2} = \frac{1}{(\tau_{f,t} - \tau_{0,t})^2} \sum_{i=0}^{n-2} n(n-1) \; b_{i,n-2}(s)(p_{i+2} - 2p_{i+1} + p_i) \tag{3.10}$$

A noter que la jacobienne d'une courbe de Bézier par rapport à un point de contrôle p_i (3.11) correspond à un polynôme de Bernstein $b_{i,n}(s)$. Autrement dit, l'impact d'un point de contrôle sur la courbe est son polynôme de Bernstein associé comme présenté précédemment sur la Figure 3.8.

$$\mathbb{J}_{B_t(\tau)} = \begin{pmatrix} b_{i,n} & 0 & 0 \\ 0 & b_{i,n} & 0 \\ 0 & 0 & b_{i,n} \end{pmatrix} \tag{3.11}$$

De manière similaire, on peut calculer les jacobiennes pour les profils de vitesse (3.12) et d'accélération (3.13).

$$\mathbb{J}_{B\prime(\tau)} = \frac{n}{(\tau_{f,t} - \tau_{0,t})} \begin{pmatrix} b_{i-1,n-1} - b_{i,n-1} & 0 & 0 \\ 0 & b_{i-1,n-1} - b_{i,n-1} & 0 \\ 0 & 0 & b_{i-1,n-1} - b_{i,n-1} \end{pmatrix} \tag{3.12}$$

$$\mathbb{J}_{B\prime\prime(\tau)} = \frac{n(n-1)b_{i-2,n-2} - 2b_{i-1,n-2} + b_{i,n-2}}{(\tau_{f,t} - \tau_{0,t})^2} \; \mathbb{I}_{3\times3} \tag{3.13}$$

3.3 Optimisation de trajectoire

Un processus d'optimisation vise à optimiser une fonction coût par rapport à certains paramètres. Dans (3.14), on cherche à minimiser une fonction coût f par rapport l'ensemble P des points de contrôle.

$$\min_P f \tag{3.14}$$

3.3.1 Résolution générale

Les fonctions considérées étant différentiables, une façon simple de trouver la solution à ce type de problème d'optimisation consiste à réaliser une descente de gradient. L'équation (3.15) montre comment modifier le point de contrôle p_i en fonction du gradient de f par rapport à ce p_i. Le paramètre α est le pas de descente de gradient qui permet de gérer la vitesse de

convergence : un α trop petit réduirait la vitesse de convergence et augmente le risque de converger vers un minimum local. Un α trop grand peut déstabiliser l'optimisation, osciller autour d'une solution et donc ne plus converger.

$$p_i \leftarrow p_i - \alpha \, \nabla_{p_i} f, \quad \alpha \in \mathbb{R}^+ \tag{3.15}$$

3.3.2 Ajout de contraintes

Lors d'un processus d'optimisation, il est possible d'y ajouter des contraintes, comme dans (3.16), où les points de contrôle P sont contraints de respecter une certaine valeur maximale de contrainte $c_{k_{max}}$.

$$\begin{cases} \min\limits_{P} f \\ c_k \leq c_{k_{max}} \end{cases} \tag{3.16}$$

Une autre façon de formuler le problème consiste à ajouter une fonction qui va détériorer la fonction coût. En minimisant le nouvel ensemble de fonctions, on cherche à respecter les contraintes, mais si les contraintes sont trop fortes, le processus d'optimisation trouvera la courbe qui minimise au mieux f en respectant au mieux les contraintes.

$$\min_{P} \left(f + \sum_{k} f_{c_k}(s) \right) \tag{3.17}$$

3.3.3 Optimisation pour le suivi de trajectoire

Dans notre cas, la première partie de la fonction coût doit nous permettre de trouver la meilleure courbe de Bézier qui localement vise à suivre le chemin global. Nous choisissons donc de minimiser la distance entre la courbe de Bézier B_t calculée à l'instant t et le chemin global Γ_g limité à l'horizon de perception du drone, qu'on notera Γ_t. Si Γ_g n'est pas présent dans l'horizon de perception du drone, le gestionnaire de mission le détecte et modifier le chemin objectif, soit en recalculant un chemin si une carte est disponible, soit, si l'environnement change beaucoup, en générant une ligne droite vers l'objectif. La Figure 3.10 montre un exemple de courbe de Bézier approximant une portion du chemin objectif global dans la sphère de perception du drone (de rayon ρ). $\Gamma_t(0)$ est le point de Γ_g, le plus proche du drone. Le calcul d'approximation de Bézier est ainsi limité à la portion utile de Γ_t dans la direction de l'objectif.

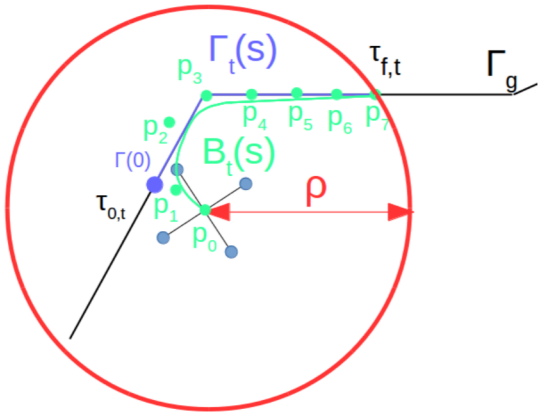


FIGURE 3.10 – Suivi de trajectoire locale. La courbe de Bézier *en vert* cherche à suivre au mieux la portion Γ_t *(en bleu)* du chemin global Γ_g *(en noir)* présente dans la sphère de perception du drone *(rouge)* de rayon ρ.

Nous formulons la fonction critère de suivi de chemin f_s à minimiser comme dans (3.18), représentant la distance entre Γ_t et et son approximation B_t.

$$\min_P f_s = \min_P \int_\tau \left\| \Gamma_t(\tau) - B_t(\tau) \right\| \tag{3.18}$$

Cette expression est discrétisée dans le cas où le chemin objectif est exprimé par un ensemble de points dans une carte discrète de l'espace. Dans ce cas, une discrétisation de la fonction B_t est utilisée avec la même résolution du chemin (3.19).

$$\min_P f_s = \min_P \sum_\tau \left\| \Gamma_t(\tau) - B_t(\tau) \right\| \tag{3.19}$$

Le gradient de la fonction critère de suivi f_s part rapport au point de contrôle p_i s'écrit alors comme suit.

$$\nabla_{p_i} f_s = -\mathbb{J}_{B_t(\tau)} \frac{\left(\Gamma_t(\tau) - B_t(\tau)\right)}{\left\| \Gamma_t(\tau) - B_t(\tau) \right\|} \tag{3.20}$$

3.3.4 Optimisation sous contrainte de courbure

Dans le but de créer une trajectoire lisse et respectant la dynamique du drone, un terme est ajouté à la fonction coût pour s'assurer que B_t ne dépasse pas une certaine courbure. La courbure en un point de la courbe est définie par le rayon du cercle passant par ce point et ayant la même tangente que la courbe. La Figure 3.11 montre le rayon de courbure r d'une courbe C en un point P, la courbure K est alors $\frac{1}{r}$.

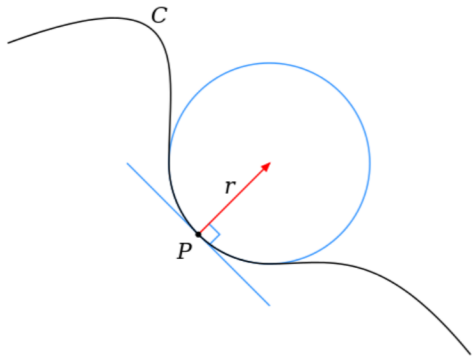


FIGURE 3.11 – Illustration du rayon de courbure r d'une courbe C au point P.

La courbure s'exprime en fonction des dérivées première et seconde de la courbe en ce point et donc en fonction des vitesses et des accélérations si le temps est pris en compte. De ce fait, la courbure peut limiter les accélérations en fonction de la vitesse et vice et versa. On peut majorer la valeur de la courbure (3.21), ce qui est suffisant dans notre cas, étant donné que l'on veut contraindre la valeur maximale de la courbure.

$$K(s(\tau)) = \frac{\left\|B'(\tau) \times B''(\tau)\right\|}{\left\|B'(\tau)\right\|^3} = \frac{\left\|B'(\tau)\right\| \left\|B''(\tau)\right\| \left|sin(\theta)\right|}{\left\|B'(\tau)\right\|^3} \leq \frac{\left\|B''(\tau)\right\|}{\left\|B'(\tau)\right\|^2} \tag{3.21}$$

Pour intégrer la contrainte de courbure maximale dans le processus d'optimisation, nous introduisons une fonction f_{c1} qui va impacter la fonction coût dans (3.17). ε_{c_1} est un paramètre qui permet de gérer la pondération de f_{c1} sur les autres composantes de la fonction coût. Cela permet de gérer les priorités entre les différentes contraintes à considérer dans le critère d'optimisation.

$$f_{c_1} = \frac{1}{2\,\varepsilon_{c_1}} \sum_{\tau} \left(\left| \frac{\left\|B''(\tau)\right\|}{\left\|B'(\tau)\right\|^2} - K_{max} \right| + \frac{\left\|B''(\tau)\right\|}{\left\|B'(\tau)\right\|^2} - K_{max} \right) \tag{3.22}$$

Dans (3.22), nous pouvons voir que si $K \leq K_{max}$ alors f_{c1} vaut 0, sinon la fonction est positive. Ainsi, lorsque l'on cherche à minimiser la fonction coût totale, cette contrainte sera respectée au mieux.

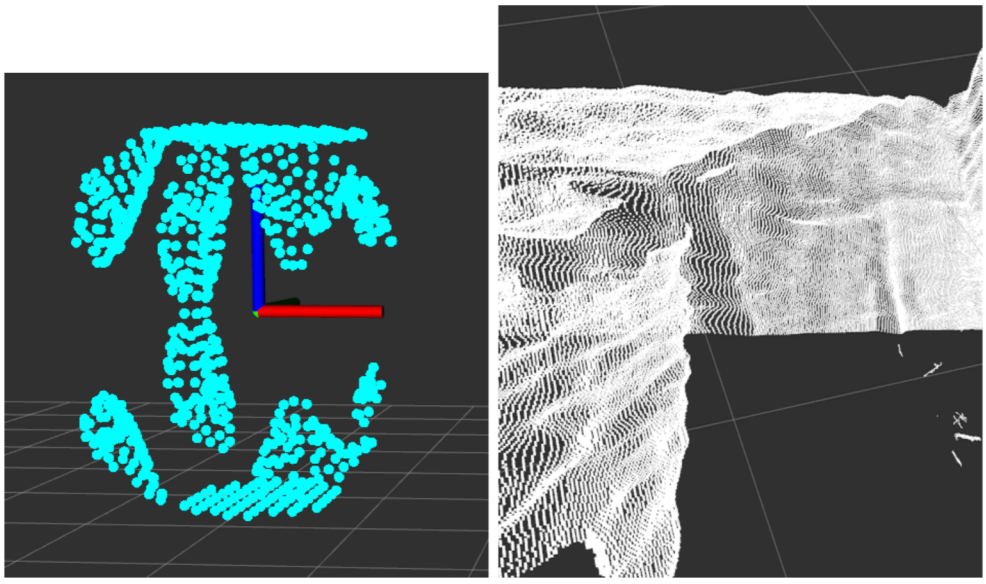
Le gradient de la fonction coût additionnel liée à la maximisation de la courbure, s'écrit :

$$\nabla_{p_i} f_{c_1} = \frac{1}{\varepsilon_{c_1}} \left(\mathbb{J}^T_{B''(\tau)} \frac{B''(\tau)}{\left\|B'(\tau)\right\|^2 \left\|B''(\tau)\right\|} + 2\, \mathbb{J}^T_{B'(\tau)} \frac{B'(\tau) \left\|B''(\tau)\right\|}{\left\|B'(\tau)\right\|^4} \right) \tag{3.23}$$

3.3.5 Evitement d'obstacles dynamiques

Pour être complètement autonome, le drone doit s'assurer d'éviter tout obstacle autour de lui en exploitant les informations disponibles des capteurs.

Dans notre étude, on considère une bulle de perception créée en équipant le drone d'un certain nombre de caméras stéréo ou caméras 3D (par exemple des caméras ©Intel Realsense) (Figure 3.12). En utilisant ce type de capteurs, il est possible d'obtenir une perception 3D des obstacles environnants et d'avoir une couverture perceptive suffisante pour réaliser la planification de chemin et de trajectoire.



(a) (b)

FIGURE 3.12 – Sphère de perception du drone de rayon ρ dans l'environnement Gazebo, en (a) un exemple de nuage de points obtenu par la fusion de données issues de plusieurs caméras de profondeur embarquées en (b).

Contrainte de distance de sécurité aux obstacles

Dans la suite, les obstacles seront décrits en utilisant un nuage de points par simplicité de calcul et d'utilisation. Une distance de sécurité entre tous les points de la trajectoire et les obstacles est définie pour garantir une trajectoire sans collision. A chaque étape, seuls les points les plus proches de la trajectoire sont utilisés pour simplifier les calculs et accélérer l'optimisation.

De manière similaire à la contrainte de courbure, la contrainte d'évitement d'obstacle est définie par la fonction (3.24) (Figure 3.13) où $D_{obs}(\tau) = \left\| Obs(\tau) - B_t(\tau) \right\|$ est la distance du point de la trajectoire $B_t(\tau)$ au point obstacle $Obs(\tau)$, étant le plus proche du point τ de la courbe. On notera que f_{c_2} vaut zéro si pour tout τ la distance à l'obstacle $D_{obs}(\tau)$ est supérieure

à la valeur de contrainte D_{min} (distance minimale de sécurité), elle prend une valeur positive sinon.

$$f_{c_2} = \frac{1}{2\,\varepsilon_{c_2}} \sum_{\tau} \left(\Big| \big\| Obs(\tau) - B_t(\tau) \big\| - D_{min} \Big| - \Big(\big\| Obs(\tau) - B_t(\tau) \big\| - D_{min} \Big) \right) \tag{3.24}$$

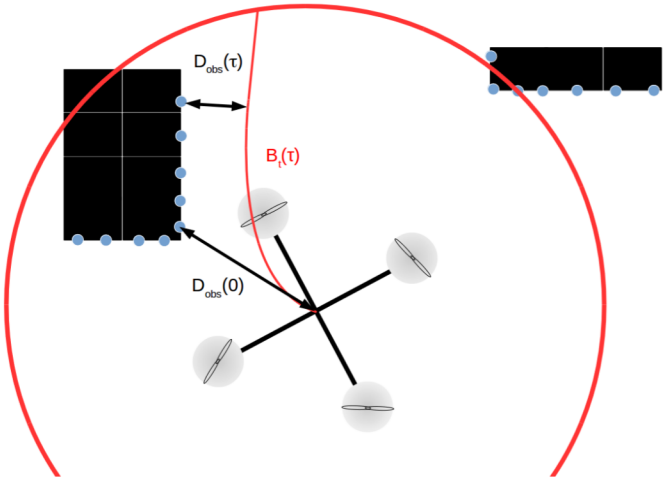


FIGURE 3.13 – Illustration de la contrainte de distance minimale de sécurité. En chaque point de $B_t(\tau)$, la contrainte cherche à maintenir une distance supérieure à D_{min} avec chaque point perçu des obstacles par les capteurs.

Le gradient de cette fonction critère additionnelle s'écrit :

$$\nabla_{p_i} f_{c_2} = \frac{1}{\varepsilon_{c_2}} \mathbb{J}^T_{[Obs(\tau) - B_t(\tau)]} \frac{Obs(\tau) - B_t(\tau)}{\big\| Obs(\tau) - B_t(\tau) \big\|} \tag{3.25}$$

Contrainte d'évitement en fonction des vitesses

La distance de sécurité est une contrainte simple qui ne prend pas en compte la dynamique du drone. Nous ajoutons une contrainte pour éviter au drone de se diriger vers des obstacles en définissant un cône de vitesse d'angle minimal autour de la direction du drone. Pour toute vitesse de consigne $\dot{B}_t$ se trouvant dans le cône de vitesse d'angle minimal dans un certain intervalle de distance, la contrainte cherche à dévier le vecteur vitesse consigne pour l'éloigner de la direction de l'obstacle. Ainsi, pour chaque τ, on cherche le point obstacle noté $O_{\beta(\tau)}$ qui se trouve dans le cône de vitesse, avec l'angle $\beta(\tau)$ le plus faible et $\overrightarrow{v}(\tau) = \dot{B}_t(\tau)$.

$$\cos\beta(\tau) = \frac{\overrightarrow{B_t(\tau)O_{\beta(\tau)}} \cdot \overrightarrow{v}(\tau)}{\big\| \overrightarrow{B_t(\tau)O_{\beta(\tau)}} \big\| \, \big\| \overrightarrow{v}(\tau) \big\|} \tag{3.26}$$

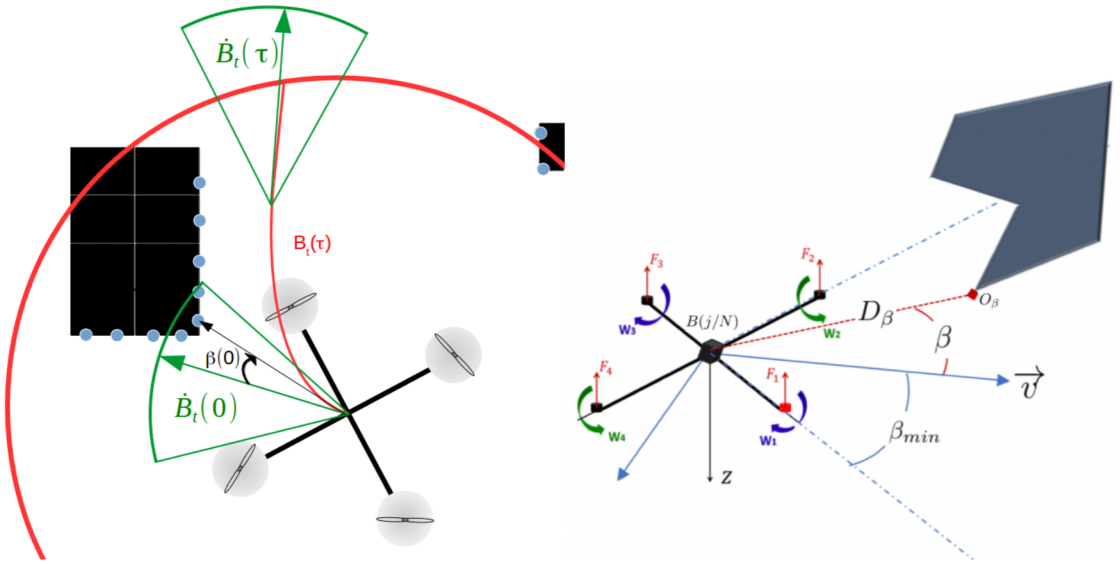


FIGURE 3.14 – Illustration de la contrainte d'évitement avec cône de vitesse. En chaque point $B_t(\tau)$ de la courbe, un cône de vitesse est généré, centré sur $\dot{B}_t(\tau)$. La contrainte s'assure qu'il n'y ait pas d'obstacle dans le cône de vitesse, le cas échéant, l'angle $\beta(\tau)$ entre le vecteur vitesse consigne et l'obstacle est modifié.

Nous introduisons ainsi (3.27) une fonction contrainte additionnelle f_{c_3} qui vise à augmenter l'angle entre la vitesse désirée et l'obstacle. Le vecteur vitesse est modifié pour sortir du cône. Si la vitesse est dans la partie droite du cône alors le drône évitera par la droite. Il y a une singularité si l'obstacle est parfaitement aligné avec la vitesse du drone.

$$f_{c_3} = \frac{1}{2\,\varepsilon_{c_3}} \sum_{\tau} \Big(\big|\cos\beta(\tau) - \cos\beta_{min}\big| + \cos\beta(\tau) - \cos\beta_{min} \Big) \tag{3.27}$$

Sa fonction gradient s'écrit :

$$\nabla_{p_i} f_{c_3} = \frac{1}{\varepsilon_{c_3}} \left[\frac{\left(\mathbb{J}^T_{B'_t(\tau)} \overrightarrow{B_t(\tau)O_{\beta(\tau)}} + \mathbb{J}^T_{B_t(\tau)} B'_t(\tau) \right)}{\left\| \overrightarrow{B_t(\tau)O_{\beta(\tau)}} \right\| \left\| B'_t(\tau) \right\|} + \right.$$

$$\left. \left(\overrightarrow{B_t(\tau)O_{\beta(\tau)}} \cdot B'_t(\tau) \right) \left(\mathbb{J}^T_{B'_t(\tau)} \frac{B'_t(\tau)}{\left\| \overrightarrow{B_t(\tau)O_{\beta(\tau)}} \right\| \left\| B'_t(\tau) \right\|^3} + \mathbb{J}^T_{B_t(\tau)} \frac{\overrightarrow{B_t(\tau)O_{\beta(\tau)}}}{\left\| \overrightarrow{B_t(\tau)O_{\beta(\tau)}} \right\|^3 \left\| B'_t(\tau) \right\|} \right) \right] \tag{3.28}$$

3.4 Optimisation avec prédiction des déplacements des obstacles

Dans l'objectif d'être encore plus agile et de mieux prendre en compte la dynamique de l'environnement, plusieurs acquisitions successives des capteurs seront utilisées pour prédire les déplacements des obstacles. La prédiction des déplacements des obstacles vise à perturber le moins possible les trajectoires générées par le processus d'optimisation. On supposera que sur la durée du processus d'optimisation (sur une itération de calcul de trajectoire locale B_t), les obstacles dans l'horizon de perception du drone à cet instant, se déplacent avec des vitesses constantes.

3.4.1 Modélisation des obstacles

Pour faciliter le traitement des nuages de points et le suivi des obstacles, les points sont regroupés en clusters par continuité. Nous ne sommes pas certains de toujours avoir un point d'impact au même endroit sur l'obstacle, mais la forme du cluster sera plus robuste d'une acquisition à l'autre. Un point de départ q_0 est choisi, puis si un autre point q_j est proche (à une distance fixée d), il est ajouté au cluster et ainsi de suite jusqu'à ce qu'il n'y ait plus de point proche (Figure 3.15).

$$obs = \{...q_j...\} \tag{3.29}$$

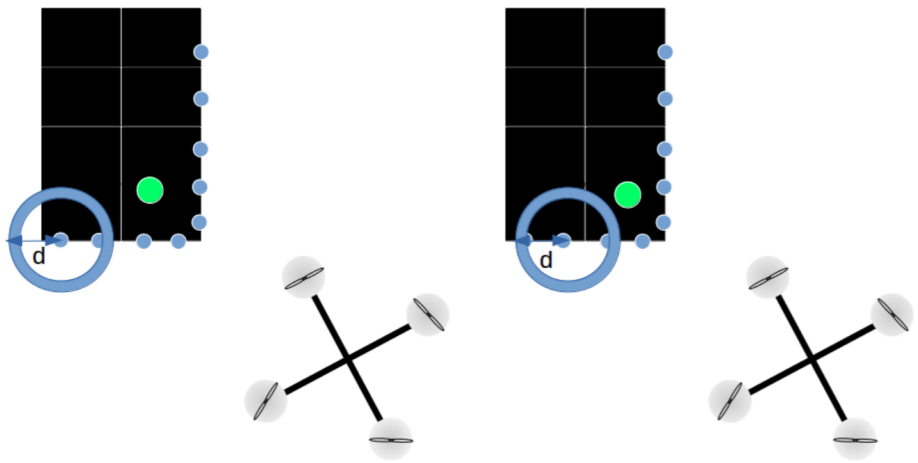


FIGURE 3.15 – Regroupement de points pour définir un obstacle en utilisant un voisinage de manière itérative, le long du bord d'un obstacle. Le point *vert* représente le centroïde de l'obstacle.

Un centroïde est calculé pour chaque cluster (3.30) et définit le centre de la perception que

l'on a de chaque obstacle et de sa position relativement au drone. Cela réduit le bruit dû aux déplacements et à l'incertitude sur la position des points sur l'obstacle.

$$C_j = moyenne(q_j) \tag{3.30}$$

3.4.2 Vitesse d'un obstacle

Une première approximation de la vitesse $v_j(t)$ (3.31) d'un obstacle (regroupement de points détectés) est réalisée avec deux acquisitions successives telle que illustré sur la Figure 3.16. Grâce au centroïde, en évitant d'avoir à associer les points 1 à 1, seul le déplacement des centroïde est utilisé. Il est plus simple d'associer 2 centroïdes. La moyenne des vitesses sur une fenêtre dynamique glissante est ensuite calculée (3.32). Cela permet de filtrer la vitesse, mais peut introduire du retard, d'où l'hypothèse de vitesse constante des obstacles sur le temps du processus d'optimisation. La Figure 3.17 montre un exemple d'estimation des vitesses des obstacles dans le simulateur Gazebo.

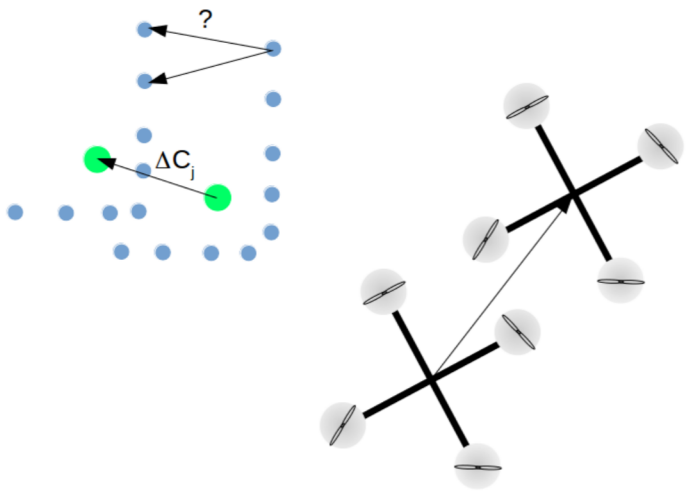


FIGURE 3.16 – Prédiction de la vitesse de déplacement des centroïdes d'obstacles. Les centroïdes permettent d'éviter d'avoir à associer les points 1 à 1 et donc de savoir quel point de l'étape t correspond à l'étape t+1.

$$v_j(t) = \frac{\Delta C_j}{\Delta t} - \dot{\xi}(t) \tag{3.31}$$

$$V_j = moyenne_{t\in[t,t+\Delta t]}(v_j(t)) \tag{3.32}$$

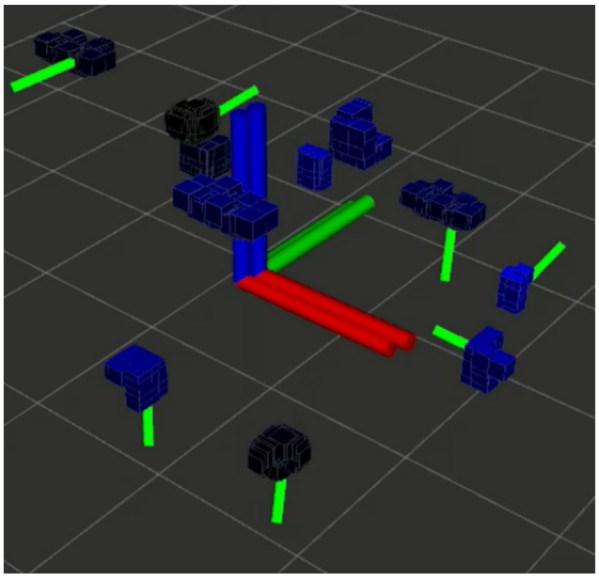


FIGURE 3.17 – Prédiction de la vitesse des obstacles. Les différentes teintes de *bleu* permettent la distinction des clusters dans le repère du drone (au centre). Les lignes *vertes* représentent les valeurs des vitesses prédites des obstacles. Dans le cas présent tous les obstacles ont la même vitesse.

Les positions des obstacles (les centroïdes des regroupements) sont propagées lors du processus d'optimisation. Ceci impacte les positions des points et des distances utilisées dans les fonctions contraintes d'évitement f_{c_2} et f_{c_3}. A chaque étape d'optimisation, le prédiction de vitesse des obstacles la plus récente est utilisée. Les obstacles sont ensuite déplacés selon τ en utilisant les vitesses prédites.

Au final, à chaque instant d'optimisation t, la solution fournie par le planificateur provient de la résolution du problème d'optimisation :

$$\min_{P} \left(f_s + \sum_{k=1}^{3} f_{c_k} \right)$$

Ceci définit une consigne pour la commande sur un horizon temporel défini $[\tau_{0,t}, \tau_{f,t}]$, et tenant compte du chemin objectif (si défini) et des contraintes de courbure maximale et d'évitement d'obstacles dynamiques.

3.5 Evaluation de l'impact des contraintes sur l'optimisation

Chaque contrainte possède un paramètre ε_{c_k} permettant d'harmoniser les dimensions, les ordres de grandeur, de pondérer une contrainte par rapport aux autres et donc de gérer quelle

sera la contrainte la plus prioritaire. La figure 3.18, montre, pour un point de contrôle, la norme du gradient créé par chaque contrainte pour chaque itération durant une optimisation. Elles sont donc bien comparables et le gradient devient très faible au bout que quelques itérations, ce qui permet de vérifier la convergence du processus.

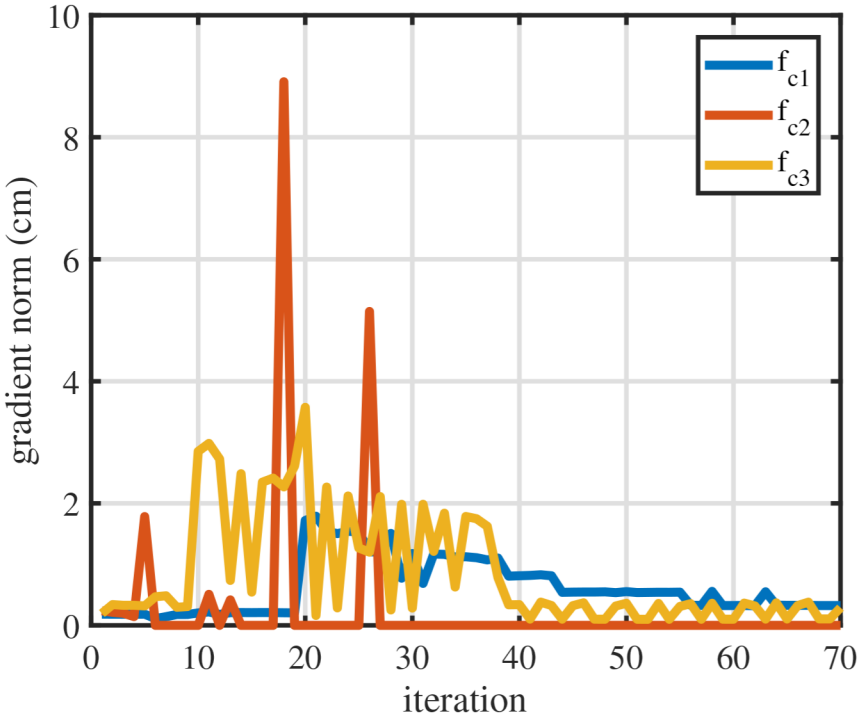


FIGURE 3.18 – Évolution de la norme du gradient créé par chaque contrainte $(1/\varepsilon_{c_k} f_{c_k})$ pour un point de contrôle d'une courbe de Bézier. Les paramètres $(1/\varepsilon_{c_k})$ valent respectivement 0.6, 2 et 0.6. Sachant que la bulle de perception est en général choisie avec 1.5m de rayon pour contenir le chemin local.

3.6 Continuité des courbes de Bézier successives

Pour assurer une régularité dans la transition entre deux trajectoires successives B_t et $B_{t+\Delta t}$, les premiers points de contrôle sont fixés en fonction de l'état du drone. Ainsi, il n'y aura pas de saut dans la commande.

Continuité en position C^0 : En utilisant la mesure de l'odométrie, on récupère la position, la vitesse et l'accélération du drone. Grâce aux courbes de Bézier, il suffit de contraindre le premier point de contrôle à la position du drone pour s'assurer que le début de B_t corresponde à la position du drone.

$$B_t(0) = p_0 = \xi(t = \tau_{0,t}) \tag{3.33}$$

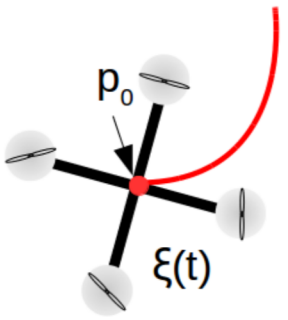


FIGURE 3.19 – Continuité d'ordre 0, le premier point de contrôle correspond à la position du drone.

Continuité en vitesse C^1 : Au point de départ de la courbe, il est possible de décrire la dérivée première en fonction des 2 premiers points de contrôle (p_0, p_1), du nombre de points de contrôle n, du temps au départ de la courbe $\tau_{0,t}$ et le temps final désiré $\tau_{f,t}$. Connaissant p_0 choisi pour assurer la continuité en position, p_1 se déduit facilement, cela permet d'assurer d'avoir une dérivée première qui respecte la vitesse du drone.

$$\frac{dB_t(0)}{d\tau} = \frac{n(p_1 - p_0)}{\tau_{f,t} - \tau_{0,t}} = \dot{\xi}(t) \tag{3.34}$$

Continuité en accélération C^2 : La dérivée seconde de la courbe à l'origine dépend des 3 premiers points de contrôle comme indiqué dans l'équation (3.35). Sachant p_0 et p_1, il est donc aisé de choisir p_2 pour assurer une continuité en accélération.

$$\frac{d^2B_t(0)}{d\tau^2} = \frac{n(n-1)(p_2 - 2p_1 + p_0)}{(\tau_{f,t} - \tau_{0,t})^2} = \ddot{\xi}(t) \tag{3.35}$$

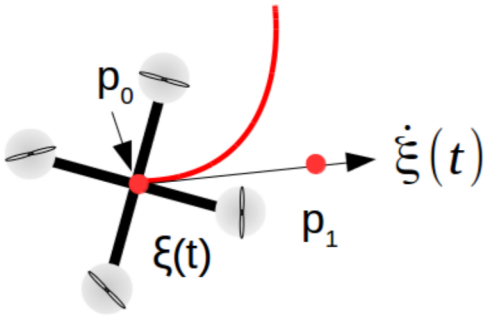


FIGURE 3.20 – Continuité d'ordre 1, le second point de contrôle est placé en fonction de la vitesse réelle du drone.

Continuité C^k : Le même procédé peut être itéré pour assurer une continuité C^k, du moment que l'on possède assez de points de contrôle.

3.7 Continuité de la commande

Pour prendre en compte le retard induit par le temps de calcul et pour éviter les sauts de commande, nous modifions le chemin objectif $\Gamma_{t+\Delta t}$ en $\tilde{\Gamma}_{t+\Delta t}$. Pour ce faire, sur un intervalle de temps très court de longueur τ_{min}, le début du chemin cible est modifié pour coïncider avec la courbe de Bézier précédente afin de maintenir une continuité durant le temps d'optimisation en cours. Le nouveau chemin objectif $\tilde{\Gamma}_{t+\Delta t}$ est alors formé par la trajectoire B_t précédente encore suivie par le drone entre $\tau_{0,t+\Delta t}$ et $\tau_{min,t+\Delta t}$ (Δt étant le temps de calcul d'une optimisation), puis du chemin cible entre $\tau_{min,t+\Delta t}$ et $\tau_{f,t+\Delta t}$ (une portion réduite de $\Gamma_{t+\Delta t}$ est utilisée tel qu'illustré Figure 3.21).

$$\tilde{\Gamma}_{t+\Delta t}(\tau) = \begin{cases} B_t(s(\tau)) \text{ pour } \tau_{0,t+\Delta t} \leq \tau < \tau_{min,t+\Delta t} \\ \Gamma_{t+\Delta t}(\tau) \text{ pour } \tau_{min,t+\Delta t} \leq \tau \leq \tau_{f,t+\Delta t} \end{cases} \tag{3.36}$$

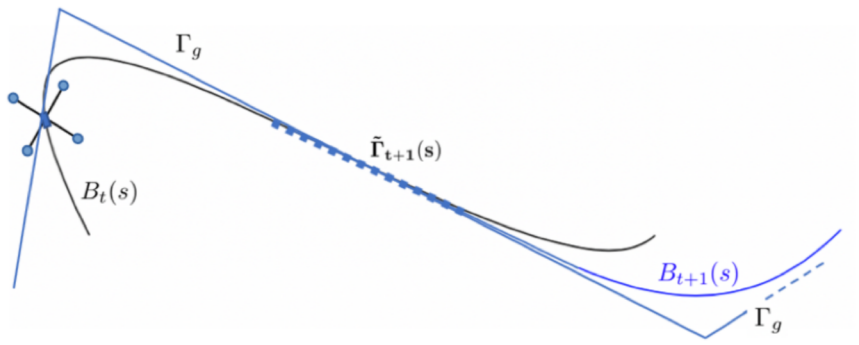


FIGURE 3.21 – Modification du chemin cible $\Gamma_{t+\Delta t}$ à l'itération suivante du processus d'optimisation pour maintenir une continuité dans la consigne de commande (dans cet exemple, $\Delta t = 1$).

Le figure 3.22, résume les correspondances des temps réels et des τ des courbes de Bézier. Nous pouvons voir la superposition des temps pour pouvoir utiliser une portion de $B_t(\tau_t)$ au début de $B_{t+\Delta t}(\tau_{t+\Delta t})$.

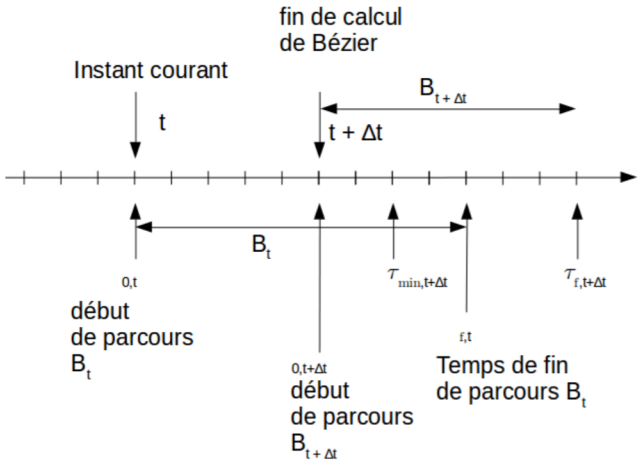


FIGURE 3.22 – Correspondance du temps réel t et des temps τ associés aux courbes de Bézier.

3.8 Conclusion

Ce chapitre a présenté la construction d'une trajectoire de vol autonome basée sur les courbes de Bézier et un processus d'optimisation. Plusieurs fonctions coûts sont définies pour la réalisation de la mission, la dynamique du drone et l'évitement d'obstacle. Ce dernier se fait de deux façons. D'abord, une distance de sécurité est assurée, ensuite pour un évitement plus agile, un cône de vitesse est utilisé pour éviter les obstacles en fonction de la vitesse du drone et de la vitesse prédite de l'obstacle.

La transition entre deux courbes successives considère la dynamique du drone aux points de contrôle et la consigne précédente l'optimisation pour réduire les sauts potentiels dans la commande. L'efficacité de la génération de trajectoires sous différentes contraintes dynamiques, de la prédiction des déplacements des obstacles et de la continuité de la commande sera testée de manière statistique et dans des environnements simulés et réalistes dans le prochain chapitre.

Chapitre 4

Validation en environnements réalistes

Dans ce chapitre, les performances de la stratégie développée sont évaluées dans des situations réalistes. Les premiers tests sont réalisés en utilisant le simulateur Gazebo. L'objectif est de comparer le comportement du planificateur de trajectoire en fonction de la connaissance et de la dynamique de l'environnement. On étudiera l'impact de différents paramètres sur les trajectoires générées par le processus d'optimisation. Les effets de la dynamique du drone et de la prédiction des vitesses des obstacles sont analysés séparément. Nous présenterons également des résultats de tests réalisés dans des environnements complexes de type grotte et hangar ainsi sur des vols plus longs (∼100 m) dans une forêt. Ces essais seront réalisés dans un contexte de simulation réaliste.

Dans le but de transposer le travail dans le monde réel, des tests intermédiaires sont réalisés avec l'environnement et les capteurs simulés alors que les calculs sont effectués sur la cible embarquée à bord du drone DJI Matrice 100.

Afin de valider vérifier le comportement de la bulle de perception dans un environnement réel, le drone DJI Matrice 100 est équipé de caméras *Intel® RealSense™*. Pour tester la robustesse au bruit de localisation, la localisation parfaite du simulateur Gazebo est remplacée par un algorithme ORB-SLAM [R.l et D., 2017] utilisant les images de la caméra frontale du drone.

Pour tester le contrôle du drone, des expérimentations sont réalisées avec le drone Parrot Bebop 2 où la commande calculée par le simulateur Gazebo est directement envoyée au drone. L'objectif ici est de tester le comportement du drone réel en suivi de trajectoires à partir des consignes générées par le planificateur proposé.

Pour finir, des résultats de tests de la chaîne complète, de la perception à la génération et au suivi de trajectoire sont obtenus avec un robot mobile terrestre.

4.1 Etude de l'influence des paramètres en environnement de simulation

Les premières expérimentations sont réalisées en utilisant le simulateur Gazebo et ont pour but de conduire une analyse statistique du comportement du drone suivant différents planificateurs de trajectoire. Le temps de calcul, le taux de réussite du planificateur ainsi que la distance entre un chemin de référence et le résultat du planificateur serviront de critère d'évaluation. La Figure 4.1 récapitule les différents scénarios et hypothèses utilisés. L'outil Gazebo simule l'environnement, produit les informations capteurs ainsi que la dynamique du drone. Le drone sera équipé de plusieurs caméras 3D ainsi que d'une centrale inertielle pour générer la bulle de perception et l'odométrie. Cela permet de faire fonctionner l'ensemble de l'algorithme, puisque, à partir des capteurs et de l'odométrie, une trajectoire est générée par le planificateur, un contrôleur calcule les commandes moteurs à envoyer au simulateur pour suivre au mieux la trajectoire consigne. Les valeurs des contraintes ainsi que le nombre de points de contrôle seront modifiés.

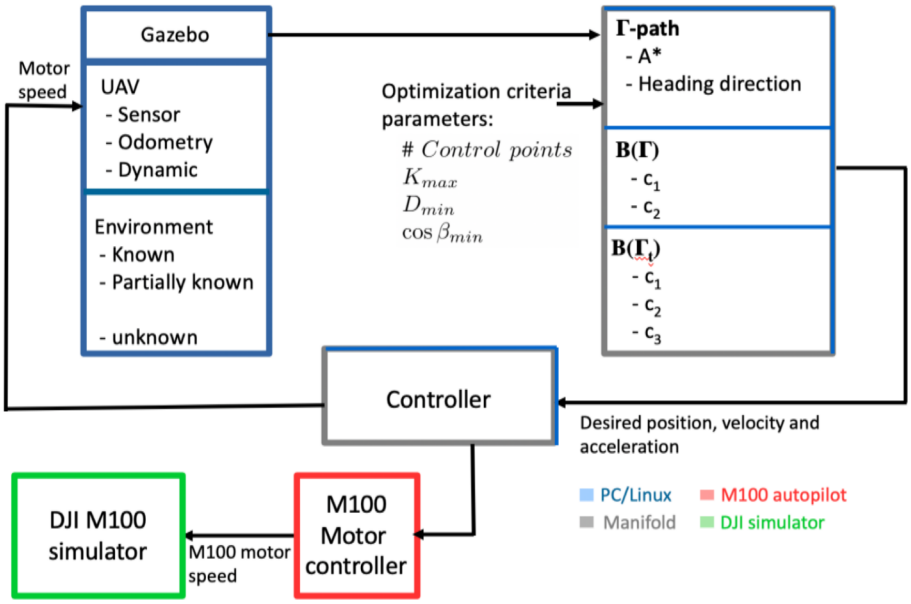


FIGURE 4.1 – Architecture globale du processus d'optimisation mis en œuvre. L'ensemble des traitements est effectué sur un ordinateur portable avec le framework ROS/Gazebo, et partiellement sur le contrôleur embarqué *(Manifold)* du drone DJI Matrice 100.

Dans chaque test, un environnement simplifié est construit (Figure 4.2) et on utilisera huit (8) points de départ et huit (8) points d'arrivée, comme indiqué dans Figure 4.2, pour générer 64 trajectoires possibles. Ce qui représentera plus de 500 trajectoires pour chaque scénario d'étude. Pour chaque valeur de paramètre, nous indiquerons en moyenne pour les 64 trajectoires, le

temps de calcul, le taux de réussite de l'optimisation ainsi que la distance entre le chemin global (optimal en distance) et la trajectoire locale générée par le planificateur.

Dans la suite de cette section, nous considérerons trois scénarios. Dans le premier, l'environnement est supposé connu et statique. Des obstacles fixes, dont les positions sont connues dans une carte, sont représentés Figure 4.2. Pour chaque couple (point départ, point d'arrivée), un chemin global optimal est calculé par un algorithme A*. Une courbe de Bézier est ensuite générée par le planificateur pour représenter au mieux le chemin A*, en respectant un ensemble de contraintes.

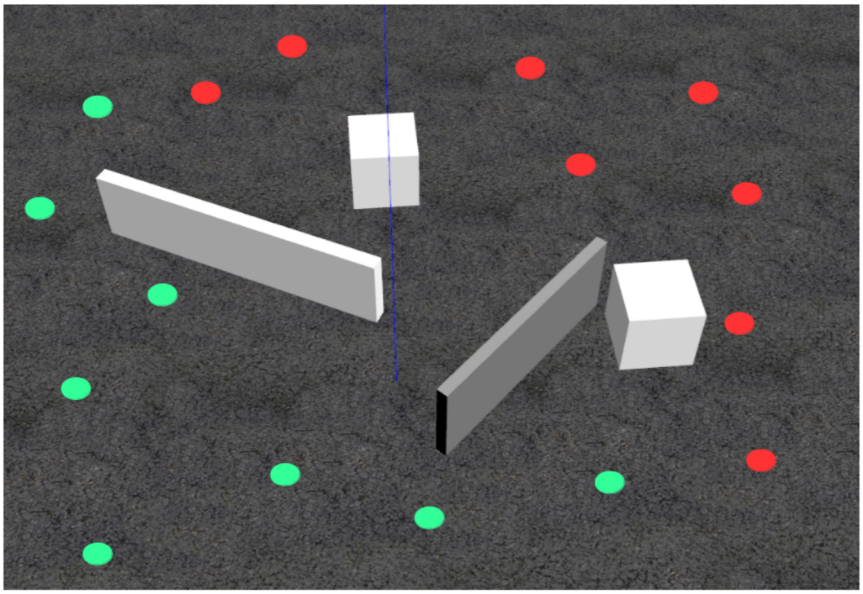


FIGURE 4.2 – Environnement de simulation pour la première étude de cas avec des obstacles statiques connus. Les *points rouges* représentent les positions de départ et les *points verts*, les positions d'arrivée.

Dans un deuxième scénario, l'environnement est supposé partiellement connu. Un nouvel objet inconnu est ajouté dans l'environnement Figure 4.2, représenté par le panneau traversant l'environnement de haut en bas (Figure 4.3). L'algorithme A* est calculé à partir de la carte initiale connue de l'environnement Figure 4.2. Le drone calcule ensuite en ligne une trajectoire locale pour éviter d'éventuels obstacles inconnus et qui devra être adaptée à la dynamique du drone en temps réel.

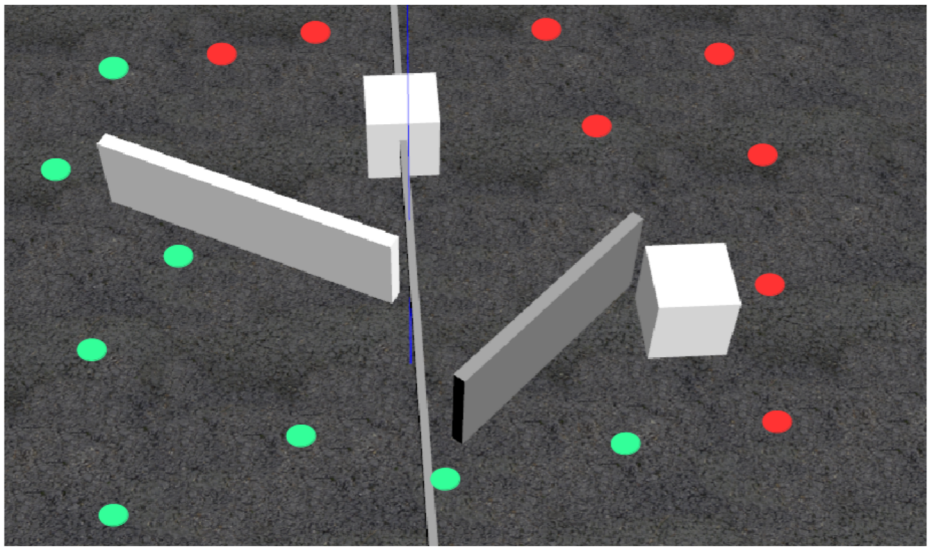


FIGURE 4.3 – Environnement de simulation pour la deuxième étude de cas avec un obstacle inconnu (la barre verticale au milieu). Les *points rouges* représentent les positions de départ et les *points verts*, les positions d'arrivée.

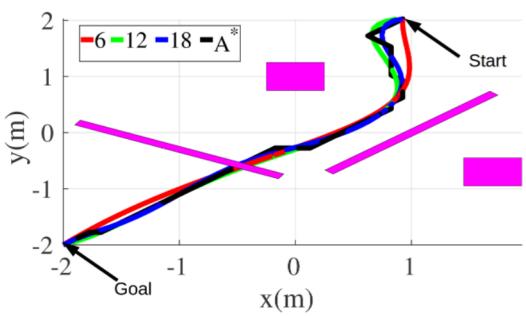
Dans le dernier scénario, le drone n'a aucune connaissance a priori de l'environnement. Il suit donc la direction globale vers l'objectif. Tout le processus d'évitement est réalisé localement basé sur les informations des capteurs et la dynamique du drone.

4.1.1 Optimisation globale dans des environnements connus

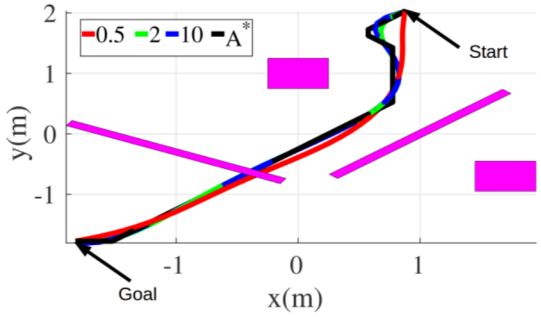
Si une carte initiale est fournie, par exemple construite à l'aide de l'algorithme OctoMap [Hornung et al., 2013], l'environnement est considéré comme connu. Un chemin global Γ_0 peut être calculé à l'aide d'un algorithme de type A* en maintenant une distance de sécurité par rapport aux obstacles, puis lissé en résolvant le problème d'optimisation sous la contrainte de courbure c_1. La Figure 4.4 montre des trajectoires comparatives en faisant varier :

— le nombre de points de contrôle
— la courbure limite
— la distance de sécurité minimale aux obstacles

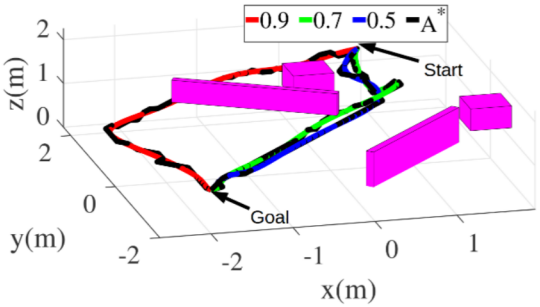
Les moyennes des performances pour les 64 trajectoires simulées pour chaque cas sont résumées dans le Tableau 4.1.



(a) Nombre de points de contrôle $= 6; 12; 18$



(b) Courbure maximale $K_{max} = 10; 2; 0.5$



(c) Distance minimale de sécurité $D_{min} = 0.5; 0.7; 0.9$

FIGURE 4.4 – Comparaison des trajectoires obtenues dans un environnement **connu** en faisant varier (a) le nombre de points de contrôle, (b) la limite supérieure de la courbure K_{max} et (c) la distance minimale de sécurité par rapport aux obstacles. Le chemin initial optimal Γ_0 est représenté en *noir*.

De toute évidence, en augmentant le nombre de points de contrôle définissant la courbe de Bézier, les trajectoires se rapprochent du chemin optimal A*. Le pourcentage de convergence réussie augmente également car il est plus facile de faire correspondre la trajectoire prédéfinie avec un nombre plus élevé de points de contrôle, ce qui diminue d'autant le temps nécessaire pour converger. Le taux de réussite correspond au pourcentage de résolution du processus d'optimisation aboutissant à une solution qui respecte les contraintes. On obtient ainsi 75% de réussite pour 9 points de contrôle de la courbe et 97% pour 21 points. En définissant uniquement 9 points de contrôle pour la trajectoire globale, la courbe n'est pas assez agile et le processus d'optimisation n'arrive pas à converger, ce qui augmente en conséquence le temps de calcul et réduit le taux de succès. Avec plus de points de contrôle, la solution est trouvée plus rapidement et la courbe de Bézier suit mieux le chemin global. En faisant varier la limite supérieure de la courbure, les résultats du Tableau (4.1) montrent que moins la courbure est contrainte, plus le temps de calcul est faible. De plus, le taux de réussite passe de 62% à près de 100% et les trajectoires générées sont plus proches de la trajectoire A* mais plus agressives vis à vis de la dynamique du drone. Enfin, en augmentant la distance minimale de sécurité, la convergence

		Temps de calcul (s)		Taux de réussite (%)	$\|\Gamma_0 - B_{\Gamma_0}\|$ (m)	
		moyenne	déviation		moyenne	déviation
# points	9	20.3	21.2	75	0.09	0.05
de	15	15.4	16.4	95.3	0.055	0.0373
contrôle	21	**11.7**	14.2	**96.9**	**0.04**	0.02
Courbure	10	**7.34**	7.5	**100**	**0.02**	0.014
K_{max}	2	7.9	7.6	**100**	0.025	0.0159
(m^{-1})	0.5	22.7	18.5	62.5	0.05	0.03
Distance	0.5	**7.2**	7.9	**100**	**0.024**	0.01
D_{min}	0.7	9.8	9.3	**100**	0.03	0.014
(m)	0.9	19.5	19.8	98.5	0.04	0.03

Tableau 4.1 – Résultats comparatifs dans un environnement **connu** : La moyenne et l'écart-type sur les 64 trajectoires sont donnés pour chaque cas, en faisant varier le nombre de points de contrôle, la borne supérieure de la courbure et la distance minimale de sécurité par rapport aux obstacles. Ces tests sont fait en globale, les résultats sont donc très impactés par la difficulté du chemin. De plus, nous utilisons plus de points de contrôle car les chemon sont plus longs en globale quand locale.

est ralentie. ceci conduit dans certains cas à des trajectoires beaucoup plus longues. On peut remarquer que le temps de calcul pour définir une trajectoire de vol réalisable prend globalement plusieurs secondes en fonction de la complexité de l'environnement et des contraintes. Ainsi, le calcul d'une trajectoire optimale globale (sur de longues distances) devra nécessairement être résolu hors ligne.

4.1.2 Optimisation locale dans des environnements partiellement connus

Ce scénario considère que l'environnement est partiellement connu, c'est-à-dire qu'une carte initiale est disponible, par exemple à l'aide d'une phase d'exploration préalable. Dans ce cas d'étude Figure 4.3, un nouvel obstacle inconnu fixe est ajouté à l'environnement précédent ; il coupe le chemin optimal prédéfini. L'obstacle obstrue une grande partie de l'espace, mais il est encore possible de passer en dessous par un passage étroit. Le problème d'optimisation est donc résolu localement pour éviter les obstacles tout en ajoutant des contraintes. On fait varier la borne supérieure de la courbure, la distance minimale aux obstacles et l'angle de sécurité dans le cône des vitesses, et on compare les trajectoires obtenues à un planificateur global, en supposant que l'environnement est connu. Un scénario typique est représenté dans la Figure 4.5. Le temps de calcul et le taux de réussite du processus d'optimisation, ainsi que la distance moyenne au chemin global sont reportés dans le Tableau 4.2 pour les 64 trajectoires.

		Temps de calcul (ms)		Taux de réussite (%)		$\sum\|B(s)-\Gamma(s)\|$ (m)	
		moyenne	déviation	moyenne	déviation	moyenne	déviation
Courbure	10	**17.9**	9.4	**99.8**	0.55	**0.08**	0.05
K_{max}	2	21.2	12.9	99.4	1.4	0.09	0.059
(m^{-1})	0.5	121.5	104.8	85.9	21.5	0.12	0.08
Distance	0.5	**17.6**	10.1	**99.3**	2.7	**0.07**	0.03
D_{min}	0.7	19.1	10.4	99.1	0.933	0.08	0.025
(m)	0.9	17.9	8.2	98.4	2	0.09	0.06
Cône des	0.9	16.65	9.4	99.8	0.2	0.064	0.04
vitesses	0.8	**15.6**	7.06	**99.9**	0.2	**0.06**	0.03
$\cos\beta_{min}$	0.6	24.1	27	98.7	4.36	0.1	0.07

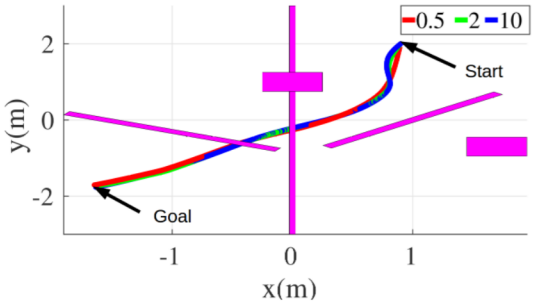
Tableau 4.2 – Résultats comparatifs dans un environnement **partiellement connu** : La moyenne et l'écart-type sur les 64 trajectoires sont donnés pour chaque cas, en faisant varier la borne supérieure de la courbure, la distance minimale de sécurité aux obstacles et l'angle minimal du cône de vitesse. Une contrainte de 0.6 pour le cône de vitesse est très difficile à respecter ce qui implique un écart type et une moyenne très important.

On peut remarquer que le taux de réussite de convergence de l'algorithme est supérieur à 99% dans la plupart des cas, avec un temps de calcul relativement faible (environ $20ms$). Lorsque la contrainte de courbure est renforcée, le temps de convergence augmente, et dans ce cas, il peut arriver que le processus d'optimisation échoue localement à respecter cette contrainte. L'écart moyen du chemin local par rapport au chemin global prédéfini (pour réaliser la mission) est en général inférieur à 10 cm. Cette valeur est fortement impactée par l'obstacle inconnu de l'environnement. On peut également remarquer que plus les contraintes sont fortes, plus il est difficile de s'adapter à la trajectoire globale.

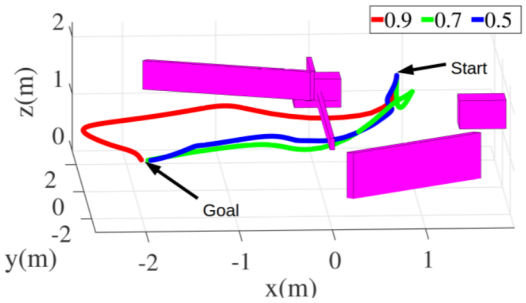
La contrainte de distance minimale de sécurité fait dévier localement la trajectoire obtenue, sauf dans les espaces encombrés, où les passages étroits seront rapidement obstrués et où des trajectoires plus longues sont alors produites. Enfin, la contrainte du cône de vitesse peut également affecter fortement la trajectoire résultante. Par exemple, avec une valeur de cosinus de contrainte de 0.6, la nouvelle trajectoire calculée est plus longue et éloignée de la trajectoire initiale souhaitée. A noter qu'une fois que le drone a passé l'obstacle, ce dernier n'a plus d'impact sur la trajectoire calculée, contrairement à la contrainte de distance minimale de sécurité qui maintient continuellement la trajectoire locale à distance.

4.1.3 Optimisation locale dans des environnements inconnus

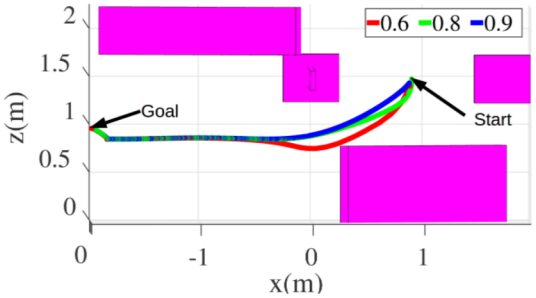
Dans ce scénario, le drone évolue dans un environnement totalement inconnu, il n'a donc que l'orientation vers la cible comme chemin objectif à suivre. La trajectoire du drone est calculée dynamiquement dans un horizon de perception. Comme précédemment, on fait varier les valeurs limites des contraintes. Les résultats sont présentés dans le Tableau 4.3 et illustrés



(a) Courbure maximale $K_{max} = 10; 2; 0.5$



(b) Distance minimale de sécurité $D_{min} = 0.5; 0.7; 0.9$



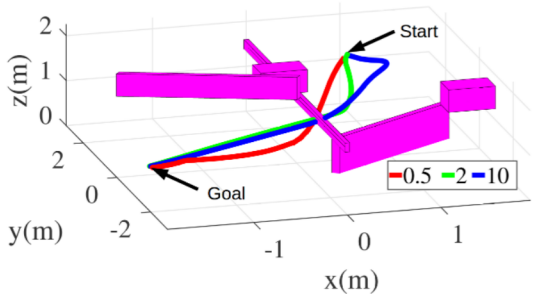
(c) Angle du cône de vitesse de sécurité $\cos\beta = 0.6; 0.8; 0.9$

FIGURE 4.5 – Comparaison des trajectoires obtenues dans un environnement **partiellement connu** en faisant varier (a) la limite supérieure de la courbure K_{max}, (b) la distance minimale de sécurité par rapport aux obstacles, et (c) l'angle minimal du cône de vitesse.

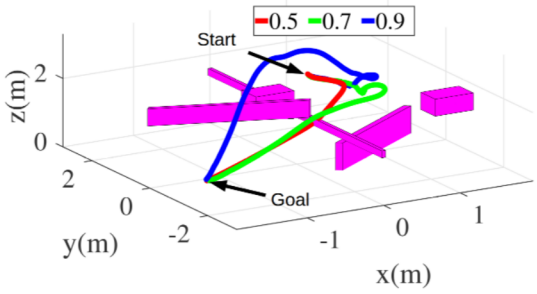
pour une étude de cas typique sur la Figure 4.6.

		Temps de calcul (ms)		Taux de réussite (%)	
		moyenne	déviation	moyenne	déviation
Courbure K_{max} (m^{-1})	10	**15.7**	5.7	**99.9**	0.16
	2	19.3	11.65	98.2	6.7
	0.5	123.5	107.8	57	40
Distance D_{min} (m)	0.5	**14.8**	4.4	**99.9**	0.23
	0.7	21.7	10.7	99.6	0.6
	0.9	45.2	36	93.8	21.1
Cône des vitesses $\cos\beta_{min}$	0.9	**16.65**	9.4	**99.9**	0.2
	0.8	19.9	14.5	99.7	1
	0.6	24.1	27	99.4	0.7

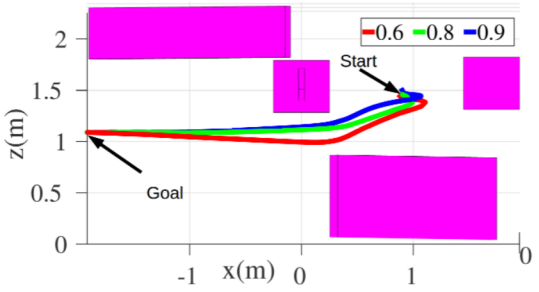
Tableau 4.3 – Résultats comparatifs dans un environnement **inconnu** : La moyenne et l'écart-type sur les 64 trajectoires sont donnés pour chaque cas, en faisant varier la limite supérieure de la courbure, la distance minimale de sécurité aux obstacles et l'angle minimal du cône de vitesse.



(a) Courbure maximale $K_{max} = 10; 2; 0.5$



(b) Distance minimale de sécurité $D_{min} = 0.5; 0.7; 0.9$



(c) Angle du cône de vitesse de sécurité $\cos\beta = 0.6; 0.8; 0.9$

FIGURE 4.6 – Comparaison des trajectoires obtenues dans un environnement **inconnu** en faisant varier (a) la limite supérieure de la courbure K_{max}, (b) la distance minimale de sécurité par rapport aux obstacles, et (c) l'angle minimal du cône de vitesse.

Dans la plupart des situations, les trajectoires du drone sont cohérentes avec les précédentes en considérant le chemin objectif connu. Dans les situations de passages étroits, plutôt que de s'en tenir à trouver le chemin de moindre longueur, l'optimisation locale conduit à prendre un chemin plus long mais plus sûr. On peut noter que la contrainte de courbure a un impact important sur la convergence du système, en particulier lorsqu'elle est fixée à de petites valeurs (par exemple, $K_{max} = 0.5\,m^{-1}$). Cela signifie que suivre un chemin en ligne droite dans un environnement encombré avec une continuité de C^2 et sous une contrainte de courbure sévère, pourrait être dans certains cas irréalisable. En moyenne, l'algorithme n'a pas respecté la contrainte de courbure sur 43% de l'ensemble du chemin. Enfin, en considérant les variations de la distance minimale de sécurité et de l'angle minimal du cône de vitesse, le comportement du système est similaire au cas précédent. L'algorithme converge avec succès dans presque toutes les trajectoires du drone. Nous notons également que le temps de convergence est légèrement augmenté puisque la trajectoire en ligne droite, même définie localement, est loin d'être optimale dans les environnements encombrés.

4.1.4 Vol de longue durée dans des environnements réalistes et complexes

De multiples expériences dans des environnements réalistes très complexes et dynamiques vont maintenant être étudiées. Un premier environnement structuré de type hangar possède un nombre important de passages étroits. Un autre environnement non structuré, très encombré de type grotte contient des obstacles mobiles représentatif d'une situation de type environnement dynamique inconnue. Les Figure 4.7 et Figure 4.8 représentent les trajectoires résultantes du drone dans ces environnements d'entrepôt et de grotte inconnus, avec des obstacles mobiles (les boîtes *noires*).

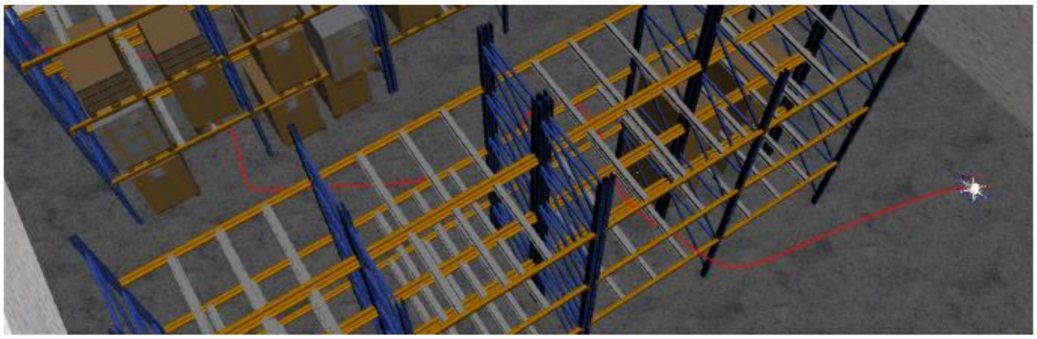


FIGURE 4.7 – Simulation dans un environnement de type entrepôt inconnu, la trajectoire résultante suivie par le drone est représentée en *rouge*.

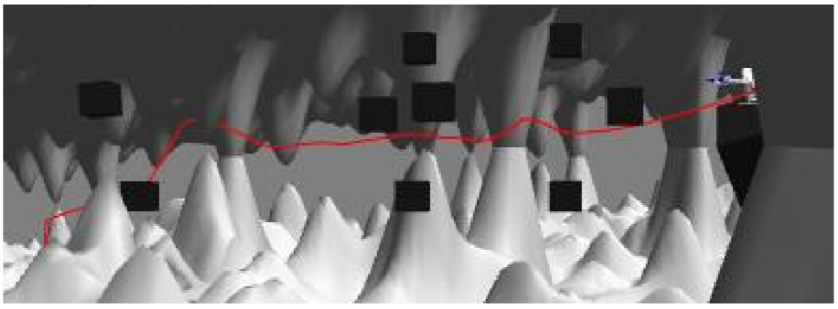


FIGURE 4.8 – Simulation dans un environnement souterrain inconnu de type grotte avec des obstacles dynamiques (les boîtes *noires* oscillent autour d'un position de référence), la trajectoire résultante suivie par le drone est représentée en *rouge*.

Une vidéo montrant les expériences simulées peut être trouvée sur ce lien [1]. Il convient de noter que la technique d'optimisation s'étend bien à un ensemble de drones volant vers des emplacements souhaités tout en évitant les collisions dans des environnements inconnus. Dans la dernière situation, chaque drone de la flotte (Figure 4.9) est contrôlé indépendamment et

1. `https://youtu.be/CMkVduikSF0`

génère de manière autonome sa trajectoire dans l'horizon de sa perception pour atteindre sa position cible et traite les autres drones comme des obstacles.

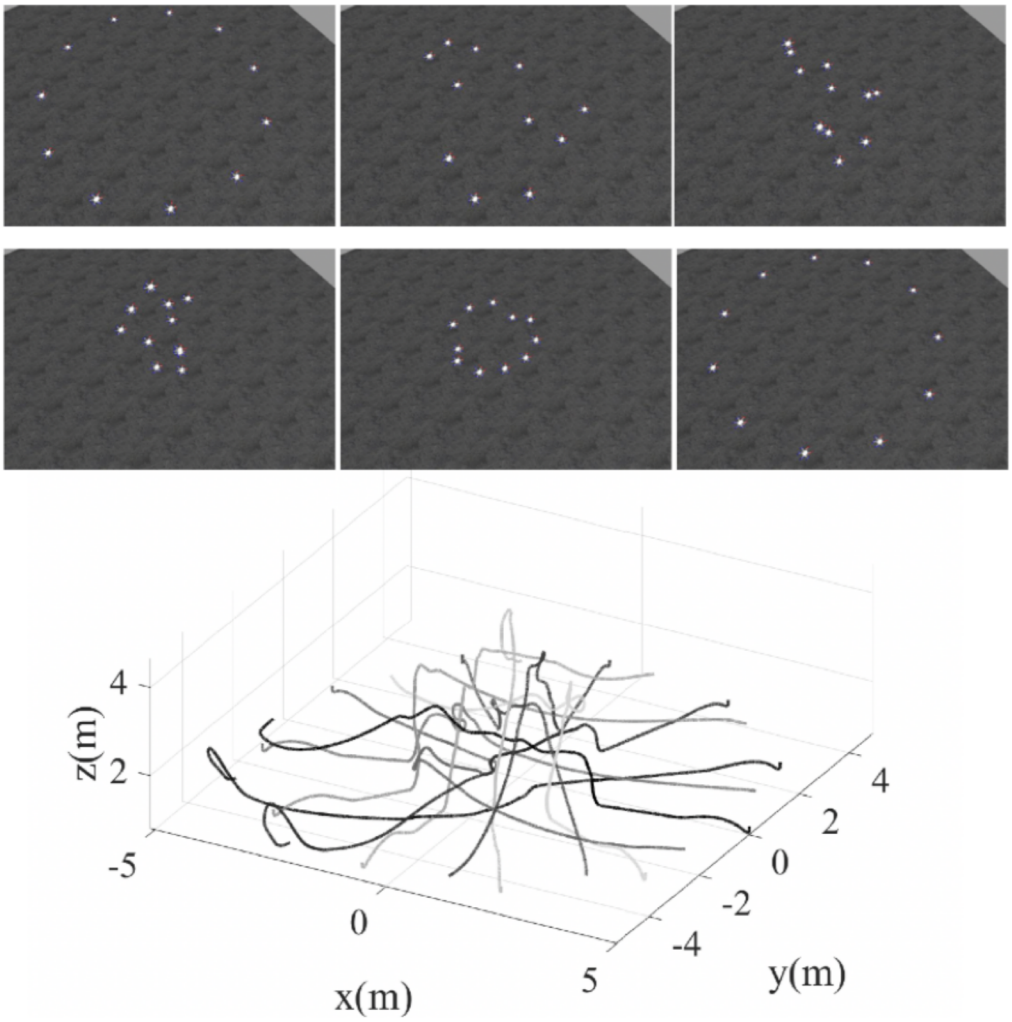


FIGURE 4.9 – Simulation avec plusieurs drones volant d'un état initial *(en haut à gauche)* à un état final *(au milieu à droite)* ; chaque drone est contrôlé indépendamment et génère sa trajectoire de manière autonome. Les trajectoires générées et suivies (sans collision) sont présentées *en bas*.

D'autres tests ont été effectués en utilisant le benchmark BOARR [Tezenas du Montcel et al., 2019] basé sur ROS et l'extension RotorS [Furrer et al., 2016] du simulateur Gazebo. BOARR vise à créer des scénarios pour tester l'évitement d'obstacles en vol long, comme celui représenté dans Figure 4.10. La Figure 4.11 montre les trajectoires résultantes du drone volant vers un point cible dans la forêt en utilisant l'ORB-SLAM comme odométrie visuelle, avec deux valeurs minimales (0.7m et 1m) pour la contrainte de distance de sécurité. Le drone réussit à atteindre la position désirée dans

des environnements forestiers avec une végétation très dense. La trajectoire est légèrement plus longue lorsque la distance minimale aux obstacles est augmentée, ce qui empêche de traverser des zones étroites.

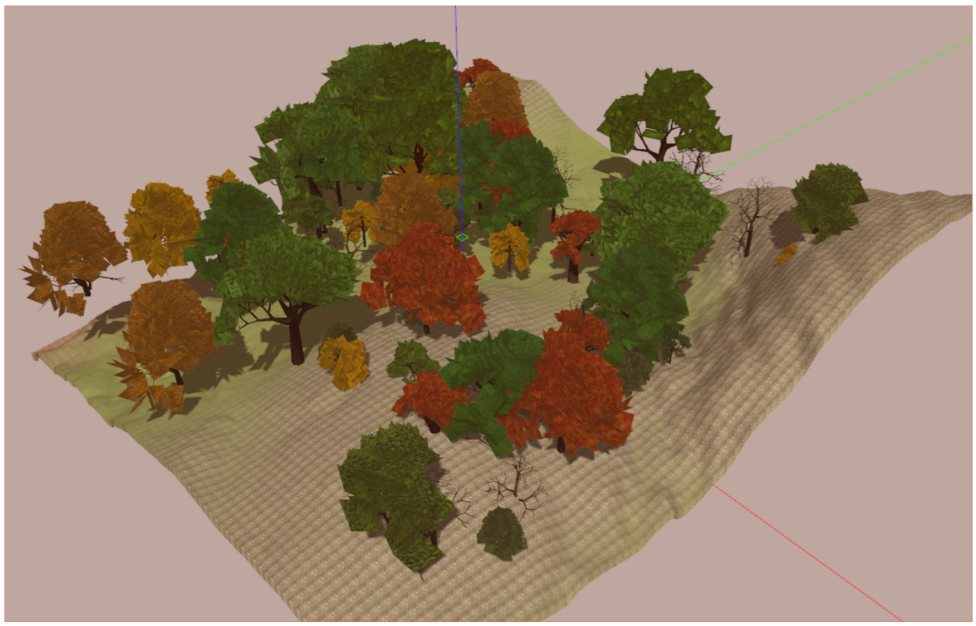
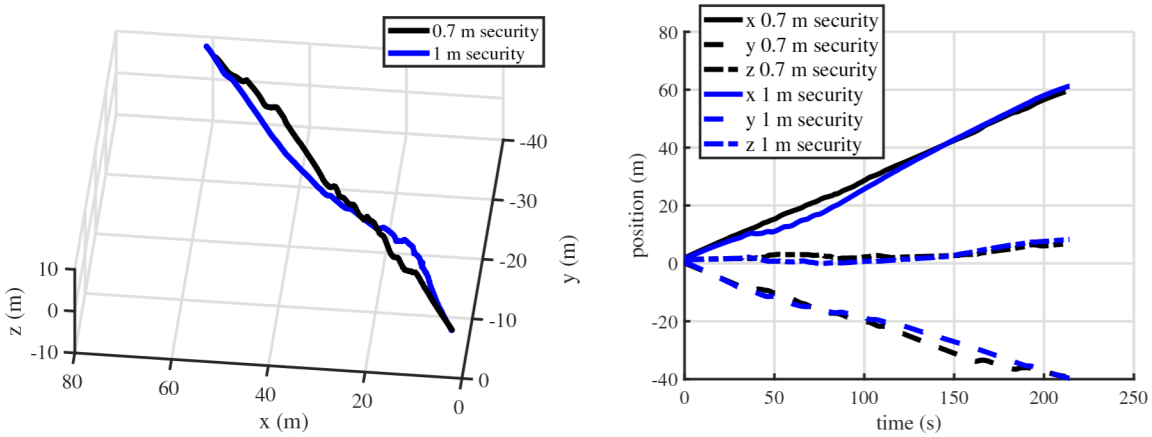


FIGURE 4.10 – Un environnement forestier réaliste pour le vol en mission longue.



(a) Position 3D du drone lors d'un vol dans la forêt (b) Position x, y, z du drone lors d'un vol dans la forêt

FIGURE 4.11 – Planification de trajectoire dans des environnements complexes. (a) La trajectoire 3D suivie avec localisation ORB-slam dans la forêt ; (b) la position (x, y, z) correspondante du drone. La trajectoire *noire* est obtenue avec une distance minimale de sécurité de 0.7 m, tandis que celle en *bleu*, est obtenue pour une distance minimale de 1m.

4.2 Optimisation avec prédiction de la vitesse des obstacles

Cette étude vise à comparer les trajectoires générées par le processus d'optimisation avec et sans prédiction de la vitesse des obstacles. Nous utilisons le scénario représenté sur la Figure 4.12. Un objet mobile se déplaçant à vitesse constante (la boîte *noire* effectue des va-et-vient de gauche à droite), et un objet statique (le mur *blanc*) constituent les obstacles dans l'environnement supposé inconnu par le drone. Un point objectif est défini pour le drone tel qu'une trajectoire directe (ligne droite) amènerait une collision avec les deux obstacles.

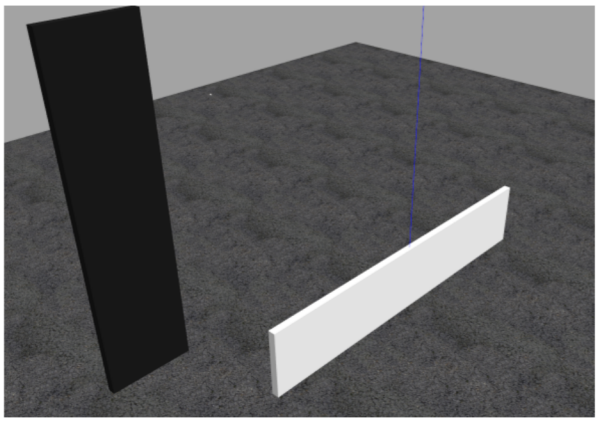
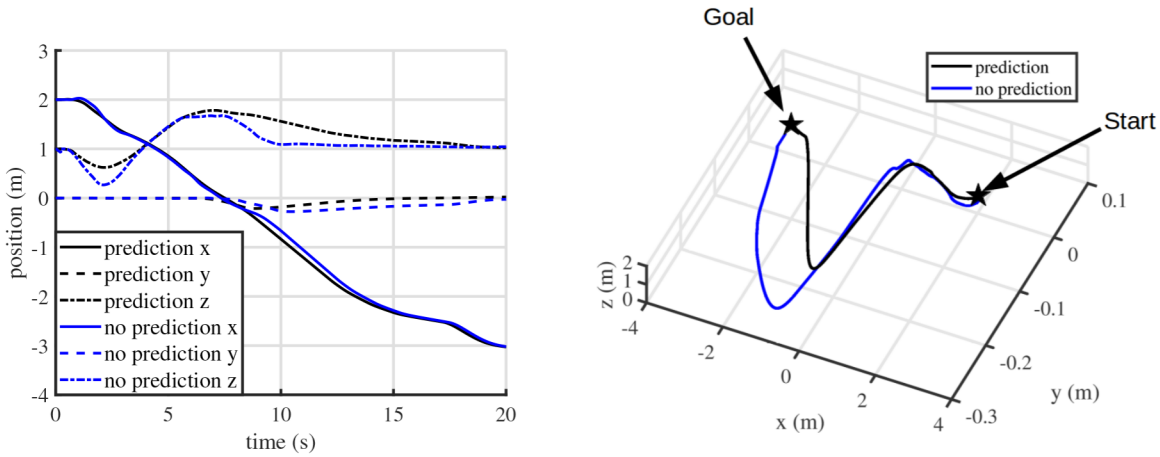


FIGURE 4.12 – Scénario de prédiction de la position des obstacles. L'objet *noir* se déplace à vitesse constante de gauche à droite, tandis que le mur *blanc* bas est immobile.

La Figure 4.13 représente les trajectoires réalisées par le drone avec et sans prédiction de la vitesse de l'obstacle. On peut remarquer que, pendant les 6 premières secondes, le drone suit à peu près la même trajectoire en survolant le mur blanc immobile. Puis, sans prédiction de la vitesse des obstacles, la trajectoire générée prend un long chemin pour éviter l'obstacle. L'extrapolation du mouvement de l'obstacle pendant le processus d'optimisation permet d'obtenir des trajectoires plus courtes et plus fluides. Dans tous les cas, comme mentionné précédemment, la distance minimale de sécurité garantit des trajectoires sans collision et ce même avec une extrapolation approximative de la trajectoire future des obstacles dynamiques.

Une analyse statistique est réalisée, pour étudier l'impact de la vitesse des obstacles sur le temps de vol avec et sans prédiction. Pour ce faire, un scénario avec 3 obstacles en mouvement est utilisé (Figure 4.14). Pour différentes vitesses d'obstacles, 200 tests sont réalisés. Les trois obstacles ont la même vitesse, mais cette vitesse peut changer en fonction du scénario. De plus, la position de départ de chaque obstacle est différente pour créer des passages ou des blocages en fonction du scénario, dans le but de changer les points de rendez-vous entre le drone et les obstacles.



(a) Position x, y, z du drone

(b) Position 3D du drone

FIGURE 4.13 – Trajectoires du drone générées avec *(en noir)* et sans *(en bleu)* prédiction de la vitesse des obstacles. (a) Position (x, y, z) du drone en fonction du temps ; (b) sa position 3D.

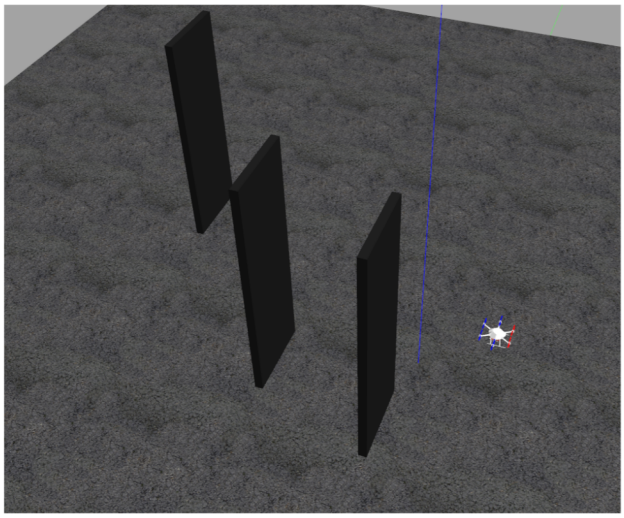


FIGURE 4.14 – Analyse statistique de l'influence de la vitesse des obstacles. Les obstacles *noirs* font des mouvements de va-et-vient de gauche à droite à vitesse constante et variable en fonction des scénarios considérés.

Sur la Figure 4.15, on peut remarquer que le temps de vol moyen sur les 200 trajectoires est inférieur avec prédiction que sans prédiction du déplacement des obstacles. En moyenne, sur une trajectoire complète, le temps de vol est aux alentours de 230 secondes avec prédiction des vitesses des obstacles, relativement à 260 secondes sans.

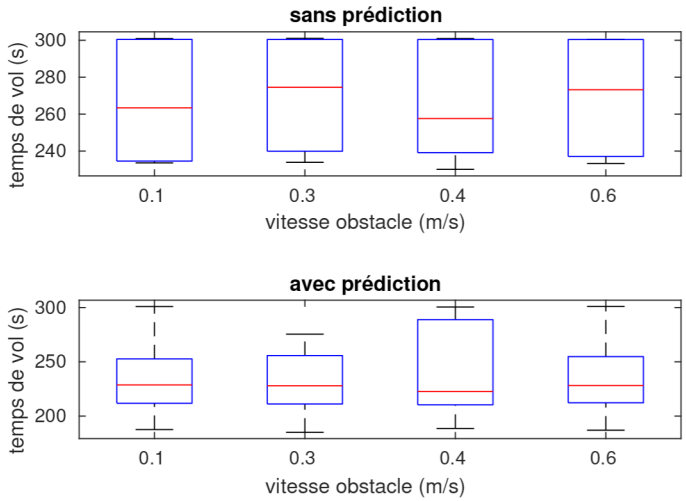


FIGURE 4.15 – Comparaison du temps de vol avec *(en bas)* et sans *(en haut)* prédiction sur 200 tests avec différentes vitesses d'obstacles.

On peut noter qu'en utilisant une approche purement réactive dans un environnement inconnu, le planificateur peut parfois échouer à produire une trajectoire réalisable sans collision. En effet, le drone peut se retrouver coincé, un obstacle peut en cacher un autre, etc. La Figure 4.16 indique le nombre de trajectoires produites avec collisions. Avec prédiction du mouvement des obstacles, il y a eu 2 collisions sur les 200 trajectoires, alors que sans prédiction, il y a eu 7 collisions sur les 200 trajectoires.

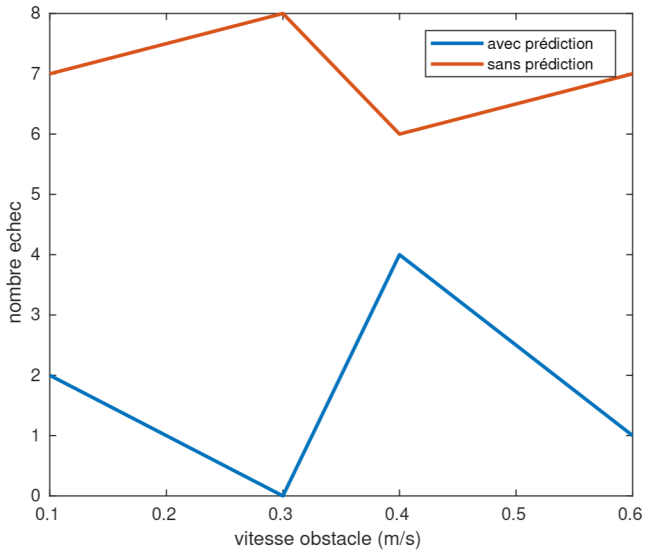


FIGURE 4.16 – Comparaison du nombre de trajectoires avec collision, avec *(en bas)* et sans *(en haut)* prédiction sur 200 tests avec différentes vitesses d'obstacles.

La Figure 4.17 montre, à titre d'exemple, l'impact de la vitesse des obstacles, avec les mêmes positions de départ et en activant la prédiction de la trajectoire des obstacles. La dynamique de l'environnement augmentant, le drone prend des chemins plus longs pour éviter les obstacles. Il évite l'obstacle soit par la gauche soit par la droite comme on peut le voir entre 0.3 et 0.4 (m/s).

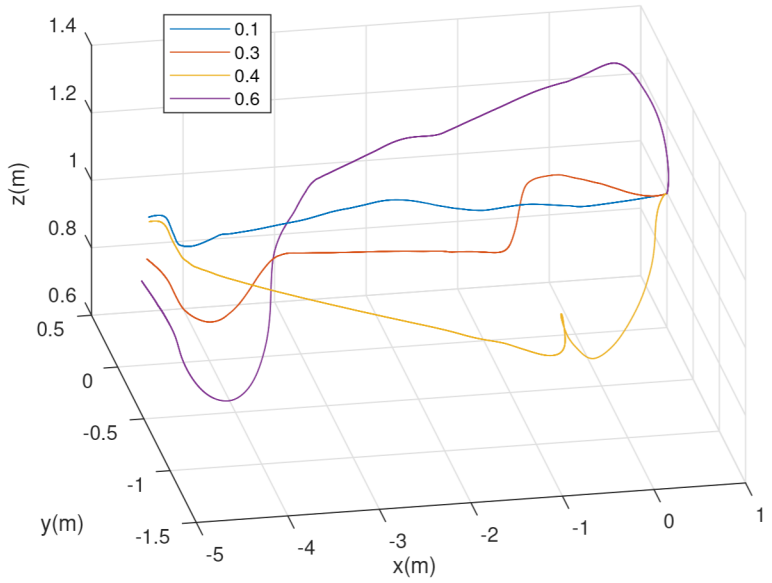


FIGURE 4.17 – Comparaison des trajectoires obtenues avec prédiction pour différentes vitesses des obstacles. La vitesse des obstacles change mais ils ont la même position de départ.

4.3 Vers le monde réel

A la différence des tests précédents où les informations du drone et de l'environnement sont considérées parfaites (observations capteurs, positions du drone et des obstacles), dans cette section, les traitements sont effectués sur le calculateur embarqué du drone DJI Matrice 100 et on utilisera ici des données réelles de capteurs embarqués. Nous considérons ci-après diverses situations qui pourraient conduire à des expériences sur des systèmes de drones réels dans des conditions réelles. L'objectif de ces expériences est de vérifier l'embarquabilité du processus de calcul et du respect des contraintes temps-réel liées au déplacement du drone volant.

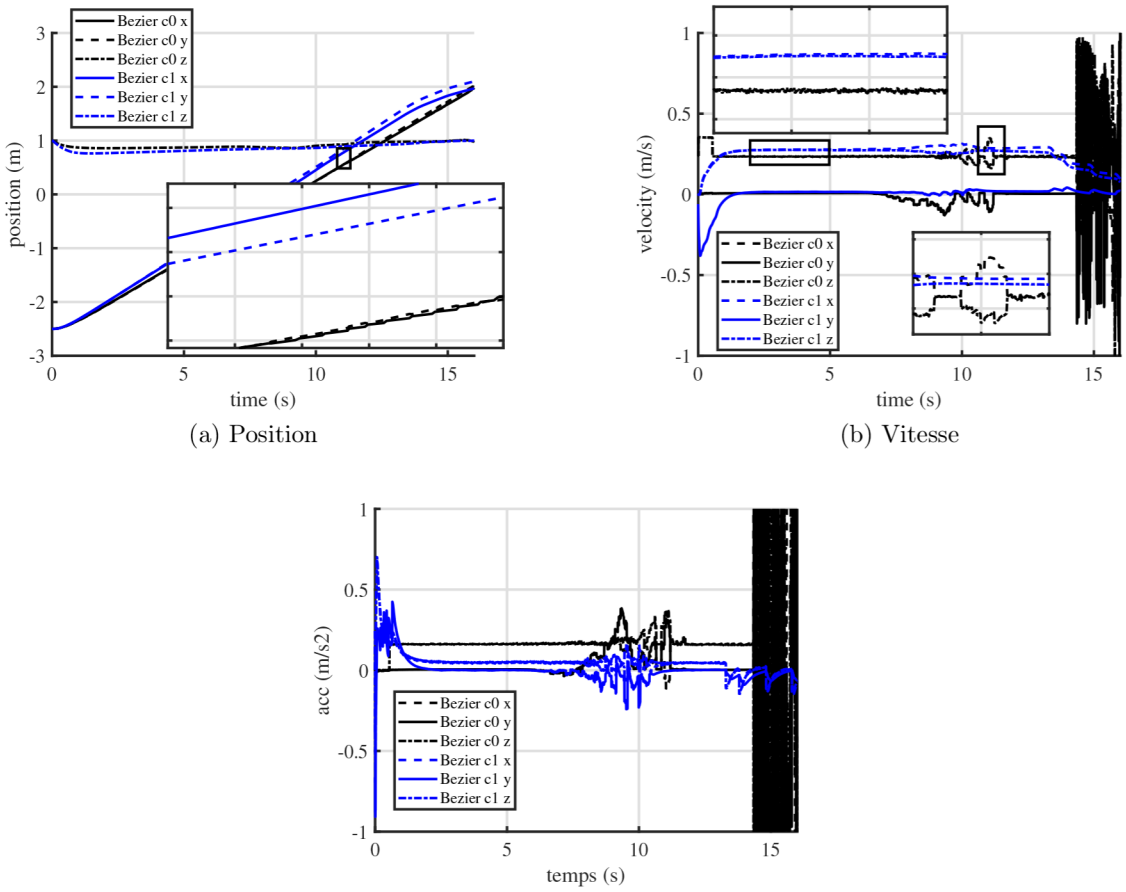
4.3.1 Optimisation avec continuité de l'état du drone pour la commande

Lors de la génération des trajectoires, le processus d'optimisation s'appuie sur la perception définie dans un horizon d'observation du drone et s'exécute avec une fréquence maximale fixée de

200Hz. Les variations de l'environnement peuvent amener des discontinuités importantes entre deux trajectoires cibles successives. Nous proposons dans cette section d'étudier les conditions de continuité entre l'état du drone et la trajectoire cible basée sur la courbe de Bézier. Cela va forcer le positionnement des premiers points de contrôle de la courbe de Bézier et peut donc impacter la convergence de l'optimisation. En considérant l'environnement Figure 4.3, la Figure 4.18 met en évidence la trajectoire générée par l'optimisation avec les conditions de continuité C_0 et C_1 sous-jacentes aux courbes de Bézier. Pour la continuité C_0, le premier point de contrôle (de départ de la trajectoire) est contraint à coïncider avec la position du drone, et pour la continuité C_1, le deuxième point de contrôle doit être également aligné avec l'orientation du drone. Comme on pourait s'y attendre, la trajectoire sans continuité de vitesse présente plus d'irrégularités et de bruit pour la commande (Figure 4.18 (a)). Dans les deux cas, l'accélération de la trajectoire cible n'est pas continue et est très bruitée (Figure 4.18 (c)), mais elle vérifie tout de même les conditions de courbure maximale. On peut remarquer sur la fin des courbes de vitesse et d'accélération, un bruit important qui correspond à un effet de bord. En effet, lorsque le drone est très proche du point objectif, la courbe de Bézier est de très courte longueur, ce qui marque des oscillations importantes. Le contrôle du drone lisse alors ces perturbations locales. Évidemment, on peut imposer des conditions de continuité supplémentaires (accélération, jerk), mais cela nécessiterait d'imposer plus de points de contrôle et donc de contraindre davantage l'évolution de la courbe.

4.3.2 Robustesse à l'incertitude de l'estimation de la pose du drone

Dans ce scénario, l'algorithme ORB-slam est utilisé comme odométrie visuelle pour l'estimation de la pose du drone au lieu de la vérité terrain du simulateur Gazebo. L'inconvénient d'utiliser des localisation basées vision dans un environnement simulé est que des patterns se répète souvent comme le sol ou les murs. Ce qui engendre des fermetures de boucle non désirées. L'objectif ici est d'évaluer l'effet des incertitudes de localisation du drone sur les trajectoires produites. La Figure 4.19 montre un exemple représentatif des trajectoires (x, y, z) obtenues. Les résultats montrent que même avec une estimation corrompue par le bruit de la position du drone, l'optimisation conduit à des trajectoires satisfaisantes.



(a) Position

(b) Vitesse

(c) Accélération

FIGURE 4.18 – Trajectoires cibles du drone générées en imposant des continuités C_0 *(en noir)* et C_1 *(en bleu)* aux courbes de Bézier. (a) Position désirée ; (b) Vitesse désirée et (c) Accélération désirée. L'important bruit à la fin vient de la contrainte de courbure quand le drone assai de s'arrêter, la courbure d'un point est infini.

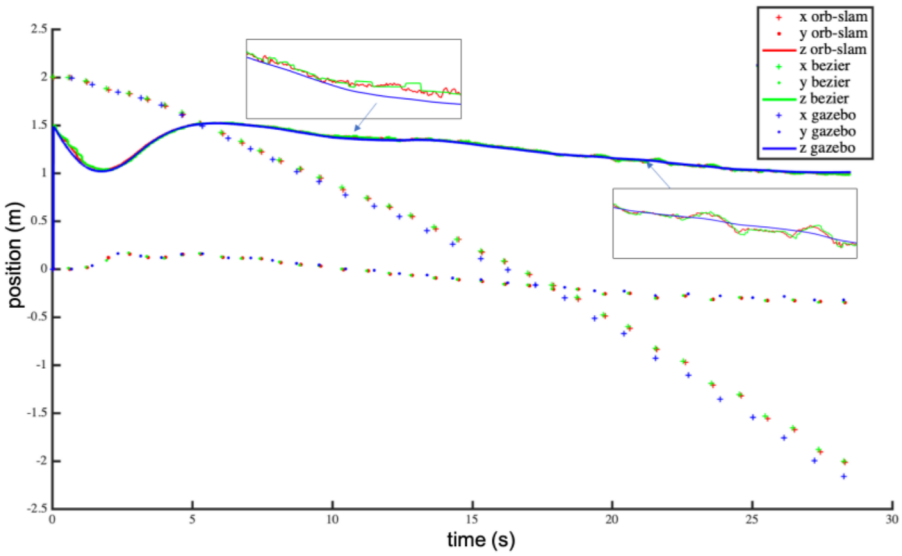


FIGURE 4.19 – Trajectoires résultantes utilisant la localisation ORB-slam comme entrée pour le planificateur de trajectoire et le contrôle. Les courbes *rouges* sont l'odométrie visuelle (x, y, z)-ORB-slam, les *bleues* représentent la pose parfaite de Gazebo, et les *vertes* sont les courbes de Bézier suivies successivement.

La Figure 4.20 montre pour deux cas, l'erreur entre la trajectoire suivie par le drone et la courbe de Bézier calculée (souhaitée). Le premier cas utilise l'odométrie visuelle ORB-slam comme entrée pour l'optimisation et le contrôle. Le second cas utilise la pose parfaite générée par Gazebo. L'erreur de suivi correspond à la norme entre le chemin global et la courbe de Bézier locale calculée à l'instant t. Évidemment, avec une localisation parfaite du drone, l'erreur de suivi est très faible. Néanmoins, même avec une localisation estimée à partir des capteurs embarqués, l'erreur de suivi est très faible et acceptable (environ 1cm).

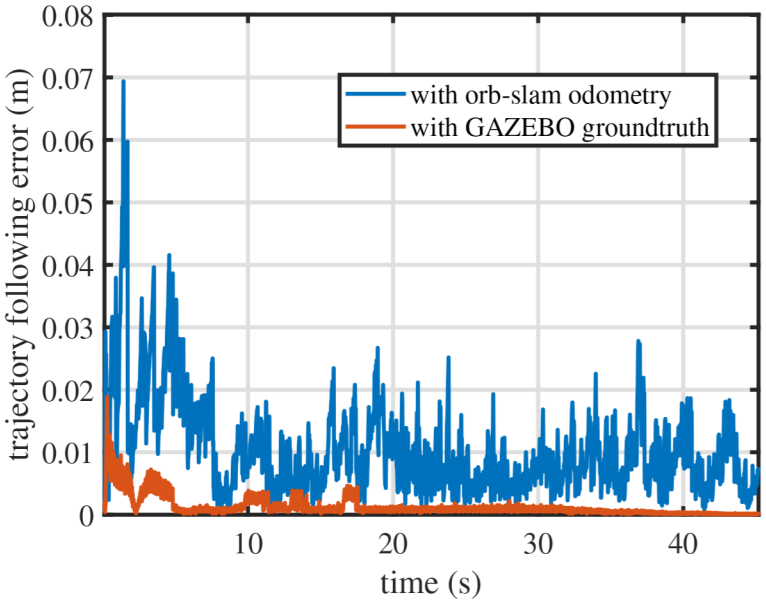


FIGURE 4.20 – Erreur de suivi entre la trajectoire suivie par le drone et la trajectoire de Bézier consigne, dans deux cas : en utilisant la localisation exacte du drone à partir du simulateur Gazebo (*rouge*), et en utilisant l'odométrie estimée par ORB-slam (*bleue*).

4.3.3 Test avec l'ordinateur embarqué du drone et le système de perception réel

L'algorithme complet de planification de mouvement est maintenant implémenté sur le DJI Manifold (ordinateur embarqué) et le pilote automatique du Matrice100 (envoyant la vitesse du moteur au simulateur DJI) dans le but de tester la charge de calcul sur un ordinateur embarqué ainsi que les commandes générées par l'autopilote d'un drone réel. Nous considérons d'abord les environnements simulés précédemment avec différentes hypothèses sur la connaissance préalable de l'environnement. Les informations capteurs sont générées par un autre ordinateur exécutant Gazebo. Les trajectoires suivies par le drone sont comparées sur la Figure 4.21, pour un cas d'étude typique avec les mêmes valeurs pour les paramètres des contraintes. La trajectoire calculée sur le contrôleur embarqué réel (Manifold) du drone est ainsi comparée à celles obtenues

lors des différentes simulations.

Pour comparer les trajectoires générées, nous calculons une trajectoire de référence globale Γ_g (en *noir* Figure 4.21) en supposant que l'environnement est connu, et son approximation lissée (en *bleu*). On peut remarquer qu'en raison des coûts résiduels, le processus d'optimisation déforme localement les trajectoires s'éloignant des obstacles. Ceci peut augmenter le temps d'exploration par rapport aux trajectoires dans les mêmes espaces libres. Ceci est également observable lorsque l'environnement est totalement inconnu, où les trajectoires peuvent localement dériver du chemin optimal. Lorsque les calculs sont embarqués sur le DJI Manifold, les tests montrent des trajectoires similaires avec un temps de calcul équivalent et donc une capacité de calcul suffisante, prouvant l'adéquation de l'approche sur des systèmes réels de drones. Un exemple de scénario réel utilisant le drone DJI M100 équipé de quatre caméras de profondeur (Intel Realsense d435i) est illustré dans Figure 4.23.

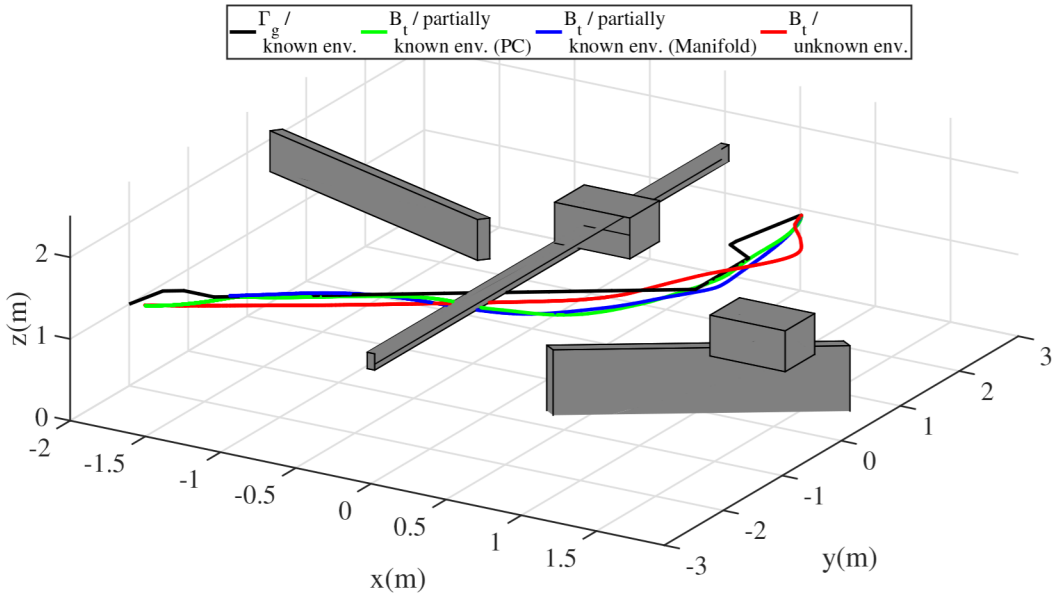


FIGURE 4.21 – Trajectoires du drone comparées en faisant varier les hypothèses sur la connaissance préalable de l'environnement et le support de calcul. Lorsqu'une carte préalable de l'environnement est connue, le chemin optimal Γ_g est représenté comme référence en *noir*. Lorsque l'environnement est partiellement connu, la trajectoire suivie est la courbe *verte*. La trajectoire suivie par le drone dans le même environnement supposé complètement inconnu, est donnée en *rouge*. La trajectoire calculée sur le contrôleur réel embarqué (DJI Manifold) du drone est montrée en *bleu*.

La Figure 4.22 indique les taux de convergence et les temps de calcul pour l'optimisation effectuée respectivement sur un ordinateur de bureau et sur le calculateur embarqué du drone. Les tests sont effectués pour différentes distances minimales de sécurité, étant la contrainte qui

a le plus fort impact sur la déformation des trajectoires par rapport aux chemins optimaux. Dans tous les cas, les taux de succès de convergence sont supérieurs à 98%, et les temps de convergence sont inférieurs à 20 millisecondes avec une légère augmentation du temps de calcul sur le système embarqué. Ce dernier est plus significatif pour les contraintes sévères de distance de sécurité avec des fréquences de mise à jour de la trajectoire moindres.

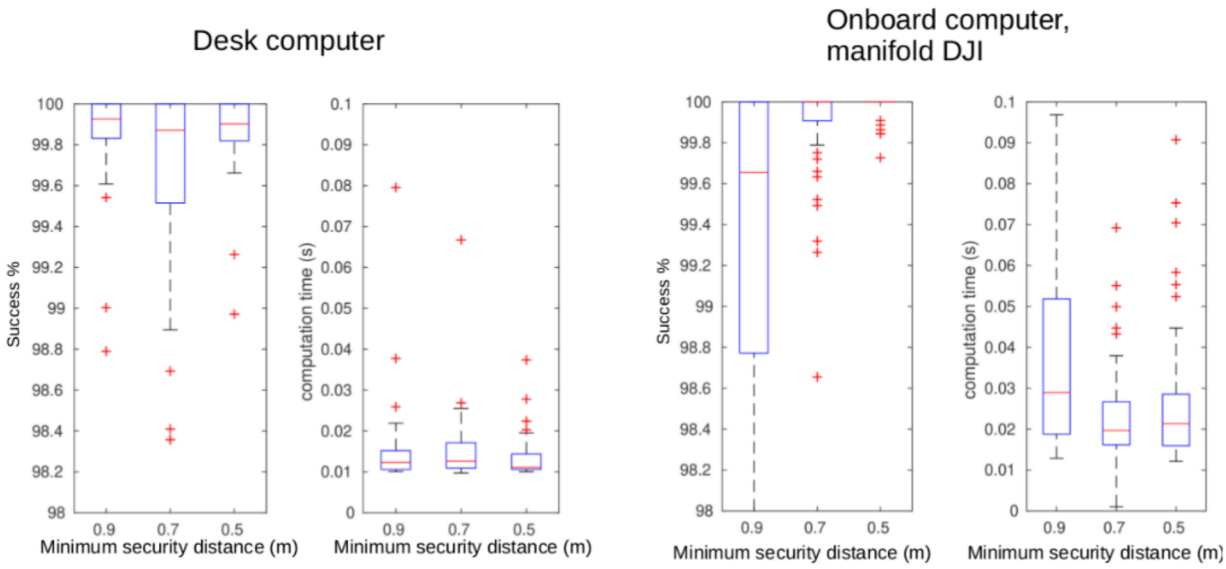
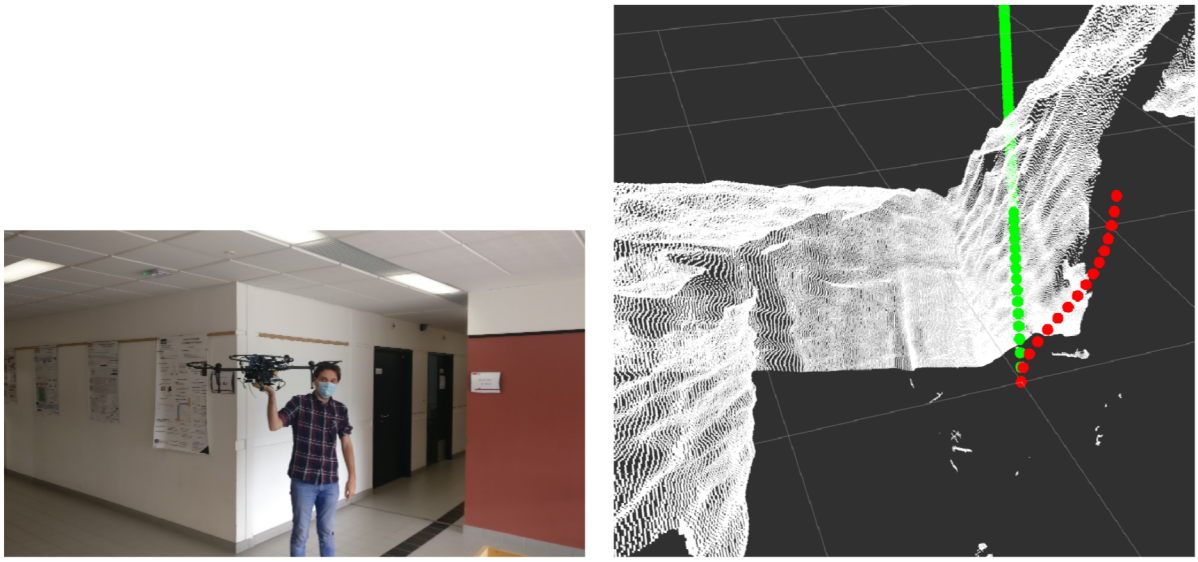


FIGURE 4.22 – Analyse quartile sur 64 trajectoires pour 3 valeurs minimales de la distance de sécurité (0.9, 0.7, 0.5). Les deux graphiques de gauche représentent les taux de réussite de l'optimisation et les temps de calcul sur un ordinateur de bureau. Les deux graphiques de droite donnent les résultats équivalents pour les mêmes scénarios, l'environnement est simulé par le logiciel Gazebo, les calculs sont réalisés sur l'ordinateur embarqué (DJI Manifold) du drone DJI Matrice 100.

Pour vérifier la partie contrôle, un test simple est réalisé à l'aide d'un drone Parrot Bebop 2 et de Gazebo. Les capteurs et l'environnement sont simulés à l'aide de Gazebo mais les commandes sont envoyées au drone réel Bebop 2. La Figure 4.24 compare la trajectoire réalisée par le drone réel, obtenue en utilisant un système Optitrack, à la portion de la courbe de Bézier consigne suivie. Les courbes de Bézier et la trajectoire du drone ne correspondent pas parfaitement mais la trajectoire du drone reste lisse.



(a) Position du drone (b) Perception réelle du drone

FIGURE 4.23 – Scénario avec la perception et la planification de mouvement embarquées. A gauche, une image de la situation du drone dans un couloir. A droite, la vue instantanée construite à partir de l'ensemble des caméras de profondeur ; la trajectoire locale de la cible (en *rouge*) est générée par le calculateur embarqué. La direction vers l'emplacement du but est indiquée par la ligne *verte*.

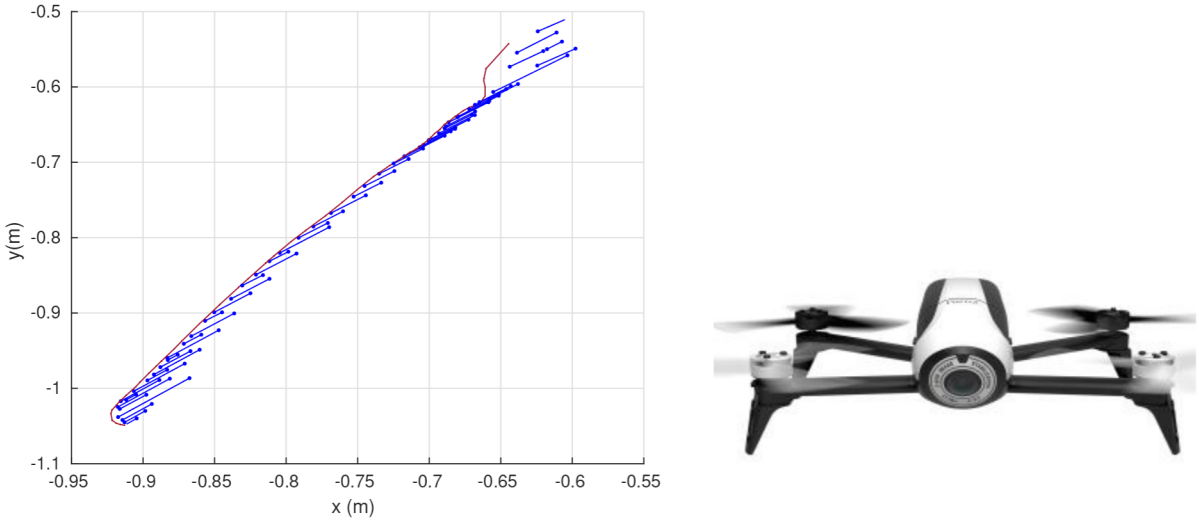


FIGURE 4.24 – Suivi de trajectoires consignes par un drone réel. Les portions de courbes de Bézier (trajectoires consignes) *(en bleu)* sont générées à partir de la perception simulée par Gazebo. La trajectoire réelle du drone, obtenue par le système Optitrack est indiquée en *rouge*. A droite un exemple de drone Bebop 2 utilisé dans cette expérience.

On notera que des tests réels de la chaine complète, de la construction de la perception

3D à la génération et suivi de trajectoires, n'ont pu être menés sur des drones réels, pour des problèmes de contrôle du drone DJI Matrice 100 d'une part, et par manque de couverture perceptive suffisante sur le drone Parrot Bebop 2 (charge utile faible pour embarquer les caméras de profondeur), d'autre part. Ainsi, pour tester la chaine complète dans des conditions réelles, nous avons utilisé un robot mobile terrestre. Un robot TurtleBot3 est équipé de plusieurs caméras 3D Intel RealSense et porte le calculateur DJI Manifold. Un point objectif est envoyé au robot. Ce dernier de manière complètement autonome, utilise les données des caméras RealSense, le planificateur embarqué génère les trajectoires à suivre et envoie les commandes aux moteurs du robot. La Figure 4.25 illustre un exemple de trajectoires résultantes obtenues en suivant les courbes de Bézier générées par le planificateur dans un environnement inconnu pour le robot.

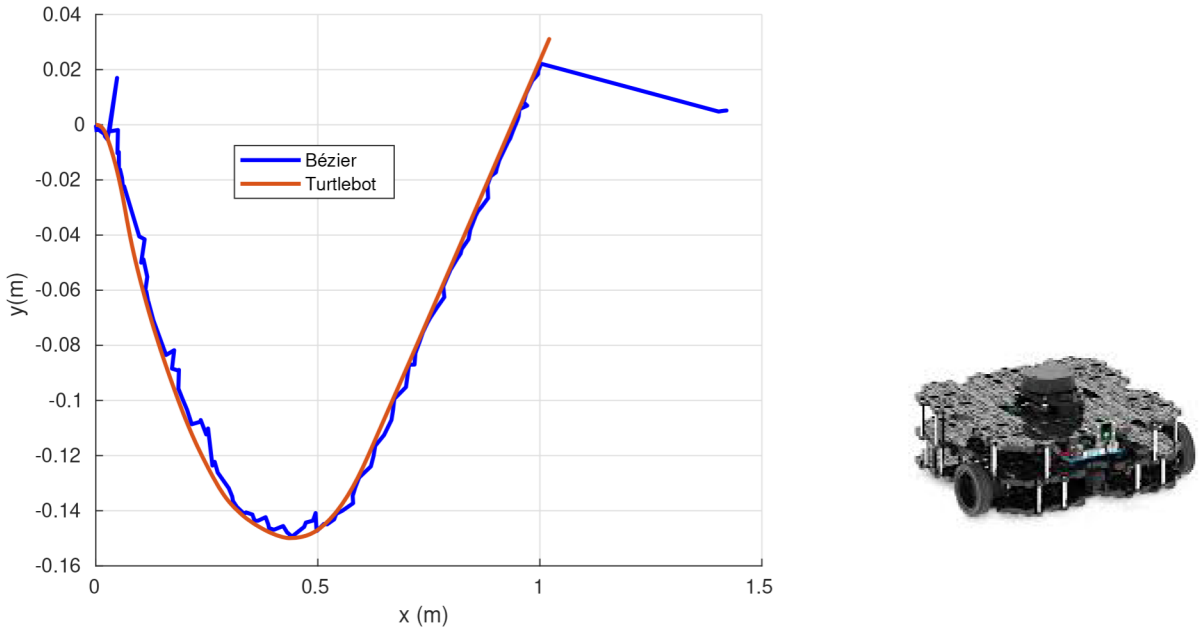


FIGURE 4.25 – Exemple de trajectoire du robot mobile TurtleBot3 *(en rouge)* suivant une trajectoire générée par le planificateur embarqué *(en bleu)*. Le robot démarre face à un obstacle. A droite le robot TurtleBot3 utilisé dans cette expérience.

4.4 Conclusion

Dans ce chapitre, les performances du planificateur de trajectoire et de mouvement que nous avons défini ont été démontrées dans différents contextes. D'abord, en simulation avec des environnements simples pour étudier l'impact des paramètres utilisés dans le processus d'optimisation, ensuite, dans des environnements simulés complexes pour démontrer l'efficacité

du planificateur dans des environnements très encombrés, dynamiques et sur de longues distances de vol.

Des tests partiels ont été effectués dans des conditions réelles de perception 3D de l'environnement et de contrôle embarqué sur des calculateurs de drones réels. Ces tests ont démontré les aspects temps réel du planificateur et la robustesse des trajectoires générées et de leur suivi dans des conditions réelles d'observation et de bruit des capteurs et des commandes moteurs. Enfin, la chaine complète, de la perception à la génération et suivi de trajectoire, a été validée avec un robot mobile terrestre autonome dans des environnements inconnus.

Conclusion générale et perspectives

Le problème de vol autonome de drones dans des environnements dynamiques inconnus a été traité au cours de ce mémoire. Le problème a été décrit en plusieurs grandes étapes : perception, localisation, planification de trajectoire, contrôle des moteurs, implémentation de la stratégie de vol sur calculateur. L'ensemble de ces éléments ont été confrontés à des expériences en simulation comme en réel.

Conclusion

Dans ce manuscrit, nous nous sommes concentrés sur le problème de vol autonome. Bien que tous les aspects nécessaires à un vol autonome soient abordés, nous nous sommes particulièrement intéressés à la génération de trajectoires lisses et faisables par le drone en toute situation.

Le premier chapitre a présenté chacune des facettes nécessaires au vol autonome d'un drone, avec des solutions actuellement apportées pour chacune des problématiques. Il a tout d'abord été montré qu'il était indispensable d'avoir des notions sur les perceptions possibles. Les caméras et les capteurs à nappe laser sont très utilisés par la communauté. L'environnement perçu par les capteurs en 3 dimensions peut ensuite être modélisé pour réduire les temps de calcul en limitant notamment les volumes de données à manipuler. Grâce aux informations des capteurs et des connaissances a priori de l'environnement, s'il en est de disponibles, il est possible d'avoir une localisation du drone sur laquelle s'appuyer.

Pour réaliser les missions qui lui sont assignées, le drone doit ensuite aller de point de passage en point de passage. Pour ce faire, un algorithme de planification de chemin est nécessaire. Une grande diversité de méthodes, de formulations et de contraintes a été présentée pour ce domaine.

Enfin, pour rendre le déplacement effectif, un contrôleur est nécessaire pour faire suivre au drone ces trajectoires en pilotant les actionneurs. Le contrôleur peut intégrer des contraintes afin de générer des commandes compatibles avec les capacités des actionneurs et les dynamiques admissibles du drone.

Dans le second chapitre, un modèle dynamique de drone a été présenté avec les repères et hypothèses de modélisation. L'approche Lagrangienne est appliquée pour obtenir ce modèle. Trois types de contrôleurs ont été présentés et testés pour passer d'un point ou d'une courbe

objectif à un vecteur de commande moteur, permettant ainsi de réaliser un contrôle de suivi de trajectoire.

Le cœur de notre contribution se trouve dans le chapitre 3 qui aborde le planificateur de trajectoire. L'approche se fait en utilisant toute l'information présente dans une sphère de perception autour du drone, basé sur les courbes de Bézier et sur une approche classique d'optimisation pour générer une trajectoire lisse. Notre travail permet de considérer différents niveaux de connaissance et la dynamique de l'environnement : environnements connus, partiellement connus ou inconnus, statiques ou dynamiques. L'entrée du processus d'optimisation peut ainsi être soit un point objectif, global ou défini dans l'horizon de perception, ou une trajectoire précalculée en utilisant une carte. Les contraintes traditionnellement utilisées comme les distances minimales aux obstacles et relatives à la dynamique du drone sont intégrées. Nous avons ajouté un cône d'évitement pour augmenter le poids des obstacles vers lesquels le drone se dirige. Nous avons également intégré des contraintes de recouvrement pour assurer une transition lisse entre deux courbes de Bézier successives. Une prédiction est également réalisée sur chaque obstacle et a été intégrée dans le processus d'optimisation. A chaque étape, le début de la courbe est contraint par la position, l'état instantané du drone. L'ensemble des développements a été testé dans des environnements réalistes. Chaque étape de l'algorithme a été validée de manière statistique, dans différentes configurations, pour différentes missions et en ajustant les poids des différentes contraintes. Des vols longs ont également été réalisés dans des environnements de type forêt ou grotte.

Finalement, des tests sur cibles réelles ont été réalisés. L'ensemble logiciel et matériel nécessaire pour rendre un drone autonome a été porté et testé sur un drone de type DJI Matrice 100 équipée d'un DJI manifold et de caméras 3D. Des tests de commande sur un drone réel avec des capteurs simulés ont été réalisés. Un robot de type char a été utilisé pour valider la génération autonome de trajectoire, sur calculateur embarqué, en utilisant les capteurs implantés sur le robot.

Perspectives

Les questions abordées et les développements réalisés dans ce travail ouvrent de nombreuses perspectives que nous discuterons brièvement ci-dessous.

— Le fait que le début des courbes respecte l'état du drone permet d'absorber quelques perturbations. Il serait intéressant d'intégrer un observateur sur des perturbations extérieures comme le vent afin d'éviter des sauts de trajectoire.

— L'optimisation cherche à atteindre l'objectif avec une trajectoire courte ou en étant le plus proche possible d'un chemin prédéfini. En l'état de la proposition développée, une vitesse de parcours moyenne est utilisée pour faire le lien entre chemin et trajectoire ce qui nous donne un profil de vitesse. Il serait intéressant d'ajuster la vitesse en cours

d'optimisation pour prendre en compte de nouvelles contraintes comme par exemple réaliser la mission en un temps minimal.

— La proposition actuelle permet d'optimiser la consommation énergétique du drone de manière détournée par la contrainte du rayon de courbure. En effet, les fortes dynamiques nécessitent plus d'énergie que des trajectoires lisses. Il serait intéressant de formaliser cette contrainte énergétique, notamment sur les drones à voilure tournante ayant peu d'autonomie énergétique.

— L'extension du travail présenté à des drones de type voilure fixe, à voilure orientable (tilt-rotor) est possible vis-à-vis de la méthodologie présentée. La typicité du drone et ses capacités dynamiques inhérentes seraient intégrées dans le seuil de contrainte admissible en terme de cône de vitesse comme de courbure.

— Par rapport à l'intégration des contraintes dans le contrôleur comme cela a été fait avec le back-stepping, il peut apparaître une redondance avec certaines contraintes du planificateur de trajectoire. Il serait intéressant de rassembler les deux aspects en utilisant, par exemple, une commande prédictive à base de modèle qui intégrerait la commande au processus d'optimisation.

— La stratégie qui a été présentée est adaptable à prendre en considération d'autres drones et de nouvelles contraintes de positionnement et/ou de vitesses relatives dans le cadre du maintien de vol en formation.

Bibliographie

[Andrade et al., 2016] Andrade, R., Raffo, G. V. et Normey-Rico, J. E. (2016). Model predictive control of a tilt-rotor UAV for load transportation. In *2016 European Control Conference (ECC)*, pages 2165–2170.

[Artemenko et al., 2016] Artemenko, O., Dominic, O. J., Andryeyev, O. et Mitschele-Thiel, A. (2016). Energy-Aware Trajectory Planning for the Localization of Mobile Devices Using an Unmanned Aerial Vehicle. In *2016 25th Int. Conf. on Computer Communication and Networks (ICCCN)*, pages 1–9.

[Åström et Hägglund, 1995] Åström, K. et Hägglund, T. (1995). *PID Controllers : Theory, Design, and Tuning*. ISA - The Instrumentation, Systems and Automation Society.

[Babaei et Karimi, 2018] Babaei, A. et Karimi, A. (2018). Optimal Trajectory-Planning of UAVs via B-Splines and Disjunctive Programming. *arXiv : Optimization and Control*.

[Benzaid et al., 2018] Benzaid, K., Marie, R., Mansouri, N. et Labbani-Igbida, O. (2018). Filtered Medial Surface Based Approach for 3D Collision-Free Path Planning Problem. *Journal of Robotics*, 2018 :1–9.

[Bircher et al., 2018] Bircher, A., Kamel, M., Alexis, K., Oleynikova, H. et Siegwart, R. (2018). Receding horizon path planning for 3D exploration and surface inspection. *Autonomous Robots*, 42(2) :291–306.

[Bircher et al., 2015] Bircher, A., Kamel, M. S., Alexis, K., Burri, M., Oettershagen, P., Omari, S., Mantel, T. et Siegwart, R. (2015). Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots. *Autonomous Robots*, 40.

[Bloesch et al., 2015] Bloesch, M., Omari, S., Hutter, M. et Siegwart, R. (2015). Robust visual inertial odometry using a direct EKF-based approach. In *2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 298–304.

[Bouadi et Tadjine, 2007] Bouadi, H. et Tadjine, M. (2007). Nonlinear observer design and sliding mode control of four rotor helicopter. *Int Journal of Eng. and Applied Science*, 3.

[Bucki et Mueller, 2019] Bucki, N. et Mueller, M. W. (2019). Rapid Collision Detection for Multicopter Trajectories. In *2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 7234–7239, Macau, China. IEEE.

[Byravan et Fox, 2017] Byravan, A. et Fox, D. (2017). SE3-nets : Learning rigid body motion using deep neural networks. In *2017 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 173–180.

[Candeloro et al., 2016] Candeloro, M., Lekkas, A. M., Hegde, J. et Sørensen, A. J. (2016). A 3D dynamic voronoi diagram-based path-planning system for uuvs. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–8.

[Castorena et al., 2020] Castorena, J., Puskoriusand, G. V. et Pandey, G. (2020). Motion guided lidar-camera self-calibration and acceleratteddepth upsampling for autonomous vehicles. *Journal of Intelligent & Robotic Systems*.

[Choi et Huhtala, 2014a] Choi, J. et Huhtala, K. (2014a). Constrained path optimization with Bézier curve primitives. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 246–251.

[Choi et Huhtala, 2014b] Choi, J. et Huhtala, K. (2014b). Constrained path optimization with Bézier curve primitives. In *2014 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 246–251.

[Delmerico et Scaramuzza, 2018] Delmerico, J. et Scaramuzza, D. (2018). A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. In *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2502–2509.

[Dentler et al., 2019] Dentler, J., Kannan, S., Bezzaoucha, S., Olivares-Mendez, M. A. et Voos, H. (2019). Model predictive cooperative localization control of multiple UAVs using potential function sensor constraints. *Autonomous Robots*, 43(1) :153–178.

[Dolatabadi et Yazdanpanah, 2015] Dolatabadi, S. H. et Yazdanpanah, M. J. (2015). MIMO sliding mode and backstepping control for a quadrotor UAV. In *2015 23rd Iranian Conference on Electrical Engineering*, pages 994–999.

[Durrant-Whyte et Bailey, 2006] Durrant-Whyte, H. et Bailey, T. (2006). Simultaneous localization and mapping : part i. *IEEE Robotics Automation Magazine*, 13(2) :99–110.

[D'Amato et al., 2020] D'Amato, E., Mattei, M. et Notaro, I. (2020). Distributed Reactive Model Predictive Control for Collision Avoidance of Unmanned Aerial Vehicles in Civil Airspace. *Journal of Intelligent & Robotic Systems*, 97(1) :185–203.

[Elzoghby et al., 2018] Elzoghby, M., Li, F., Arif, U. et Arafa, I. I. (2018). Small UAV localization based strong tracking filters augmented with interacting multiple model. In *2018 15th Int. Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 310–317.

[Engelhard et al., 2011] Engelhard, N., Endres, F., Hess, J., Sturm, J. et Burgard, W. (2011). Real-time 3D visual slam with a hand-held RGB-D camera. In *Proc. RGB-D Workshop 3-D Perception*.

[Eynard et al., 2010] Eynard, D., Vasseur, P., Demonceaux, C. et Frémont, V. (2010). UAV altitude estimation by mixed stereoscopic vision. In *2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 646–651.

[Faigl et Váňa, 2018] Faigl, J. et Váňa, P. (2018). Surveillance Planning With Bézier Curves. *IEEE Robotics and Automation Letters*, 3(2) :750–757.

[Faria et al., 2004] Faria, G., Romero, R. A. F., Prestes, E. et Idiart, M. A. P. (2004). Comparing harmonic functions and potential fields in the trajectory control of mobile robots. In *IEEE Conference on Robotics, Automation and Mechatronics, 2004.*, volume 2, pages 762–767 vol.2.

[Ferguson et al., 2008] Ferguson, D., Darms, M., Urmson, C. et Kolski, S. (2008). Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *2008 IEEE Intelligent Vehicles Symposium*, pages 1149–1154.

[Forster et al., 2017] Forster, C., Zhang, Z., Gassner, M., Werlberger, M. et Scaramuzza, D. (2017). SVO : Semidirect visual odometry for monocular and multicamera systems. *IEEE Trans. on Robotics*, 33(2) :249–265.

[Furrer et al., 2016] Furrer, F., Burri, M., Achtelik, M. et Siegwart, R. (2016). *Robot Operating System (ROS) : The Complete Reference (Volume 1)*, chapter RotorS - A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer Int. Publishing, Cham.

[Gao et al., 2017] Gao, F., Lin, Y. et Shen, S. (2017). Gradient-based online safe trajectory generation for quadrotor flight in complex environments. In *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3681–3688.

[Grzonka et al., 2012] Grzonka, S., Grisetti, G. et Burgard, W. (2012). A fully autonomous indoor quadrotor. *IEEE Trans. on Robotics*, 28(1) :90–100.

[Gutierrez-Martinez et al., 2020] Gutierrez-Martinez, M., Rojo-Rodriguez, E., Castillo, L. C.-R. L. R.-O. P. et Garcia-Salazar, O. (2020). Collision-free path planning based on a genetic algorithm for quadrotor UAVs. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J. et Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics*, 4(2) :100–107.

[Hilario et al., 2011] Hilario, L., Montés, N., Mora, M. C. et Falco, A. (2011). Real-time bézier trajectory deformation for potential fields planning methods. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1567–1572.

[Hmida et al., 2012] Hmida, H. B., Boochs, F., Cruz, C. et Nicolle, C. (2012). Knowledge Base Approach for 3D Objects Detection in Point Clouds Using 3D Processing and Specialists Knowledge. *Int. Journal on Advances in Intelligent Systems*, page 14.

[Hornung et al., 2013] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. et Burgard, W. (2013). Octomap : an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3) :189–206.

[HOW et al., 2008] HOW, J. P., BEHIHKE, B., FRANK, A., DALE, D. et VIAN, J. (2008). Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, 28(2) :51–64.

[Ingersoll et al., 2016] Ingersoll, B. T., Ingersoll, J. K., DeFranco, P. et Ning, A. (2016). UAV Path-Planning using Bezier Curves and a Receding Horizon Approach. In *AIAA Modeling and Simulation Technologies Conf.*, Washington, D.C. American Institute of Aeronautics and Astronautics.

[Jang et al., 2017] Jang, D. S., Chae, H. J. et Choi, H. L. (2017). Optimal control-based UAV path planning with dynamically-constrained TSP with neighborhoods. In *2017 17th Int. Conf. on Control, Automation and Systems (ICCAS)*, pages 373–378.

[Jayasinghe et Athauda, 2016] Jayasinghe, J. A. S. et Athauda, A. M. B. G. D. A. (2016). Smooth trajectory generation algorithm for an unmanned aerial vehicle (UAV) under dynamic constraints : Using a quadratic Bezier curve for collision avoidance. In *2016 Manufacturing Industrial Engineering Symposium (MIES)*, pages 1–6.

[Jones et Hollinger, 2017] Jones, D. et Hollinger, G. A. (2017). Planning energy-efficient trajectories in strong disturbances. *IEEE Robotics and Automation Letters*, 2(4) :2080–2087.

[Juelg et al., 2017] Juelg, C., Hermann, A., Roennau, A. et Dillmann, R. (2017). Fast online collision avoidance for mobile service robots through potential fields on 3D environment data processed on GPUs. In *2017 IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, pages 921–928.

[Kang et al., 2017] Kang, M., Liu, Y., Ren, Y., Zhao, Y. et Zheng, Z. (2017). An empirical study on robustness of UAV path planning algorithms considering position uncertainty. In *Int. Conf. on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–6. IEEE.

[Kaufman et al., 2016] Kaufman, E., Lee, T., Ai, Z. et Moskowitz, I. S. (2016). Bayesian occupancy grid mapping via an exact inverse sensor model. In *2016 American Control Conference (ACC)*, pages 5709–5715.

[Kenmogne et al., 2017] Kenmogne, I. F., Drevelle, V. et Marchand, E. (2017). Image-based UAV localization using interval methods. In *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 5285–5291.

[Khuswendi et al., 2011] Khuswendi, T., Hindersah, H. et Adiprawita, W. (2011). UAV path planning using potential field and modified receding horizon A* 3D algorithm. In *Proc. of the 2011 Int. Conf. on Electrical Engineering and Informatics*, pages 1–6.

[Klein et Murray, 2007] Klein, G. et Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM Int. Symposium on Mixed and Augmented Reality*, pages 225–234.

[Kokotovic, 1992] Kokotovic, P. (1992). The joy of feedback : nonlinear and adaptive. *IEEE Control Systems Magazine*, 12(3) :7–17.

[Kulathunga et al., 2020] Kulathunga, G., Fedorenko, R. et Klimchik, A. ((2020)). Regions of interest segmentation from lidar point cloud for multirotor aerial vehicles. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

[L. et T., 2020] L., B.-X. et T., K.-S. ((2020)). 3D map exploration via learning submodular functions in the fourier domain. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

[LaValle et James J. Kuffner, 2001] LaValle, S. M. et James J. Kuffner, J. (2001). Randomized kinodynamic planning. *The Int. Journal of Robotics Research*, 20(5) :378–400.

[Leutenegger et al., 2013] Leutenegger, S., Furgale, P., Rabaud, V., Chli, M., Konolige, K. et Siegwart, R. (2013). Keyframe-Based Visual-Inertial SLAM using Nonlinear Optimization. In *The Int. Journal of Robotics Research*. Robotics : Science and Systems Foundation.

[Liang et al., 2014] Liang, X., Wang, H., Li, D. et Liu, C. (2014). Three-dimensional path planning for unmanned aerial vehicles based on fluid flow. In *2014 IEEE Aerospace Conference*, pages 1–13.

[Lin et al., 2018] Lin, X., Casas, J. R. et Pardàs, M. (2018). Temporally Coherent 3D Point Cloud Video Segmentation in Generic Scenes. *IEEE Trans. on Image Processing*, 27 :3087–3099.

[Lin et al., 2020] Lin, Z., Castano, L. et Xu, H. ((2020)). UAV collision avoidance with varying trigger time. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

[Luo et al., 2017] Luo, X., Li, X., Yang, Q., Wu, F., Zhang, D., Yan, W. et Xi, Z. (2017). Optimal path planning for UAV based inspection system of large-scale photovoltaic farm. In *2017 Chinese Automation Congress (CAC)*, pages 4495–4500.

[Mansouri et al., 2015] Mansouri, S. S., Nikolakopoulos, G. et Gustafsson, T. (2015). Distributed model predictive control for unmanned aerial vehicles. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 152–161, Cancun, Mexico. IEEE.

[Marchand et al., 2016] Marchand, E., Uchiyama, H. et Spindler, F. (2016). Pose Estimation for Augmented Reality : A Hands-On Survey. *IEEE Trans. on Visualization and Computer Graphics*, 22 :2633–2651.

[Marchand et al., 2013] Marchand, N., Durand, S. et Castellanos, J. F. G. (2013). A general formula for event-based stabilization of nonlinear systems. *IEEE Trans. on Automatic Control*, 58(5) :1332–1337.

[Marzat et al., 2017] Marzat, J., Bertrand, S., Eudes, A., Sanfourche, M. et Moras, J. (2017). Reactive MPC for Autonomous MAV Navigation in Indoor Cluttered Environments : Flight Experiments. *IFAC-PapersOnLine*, 50(1) :15996–16002.

[McGuire et al., 2019] McGuire, K. N., de Croon, G. et Tuyls, K. (2019). A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems*, 121 :103261.

[Mehdi et al., 2015] Mehdi, S. B., Choe, R. et Hovakimyan, N. (2015). Avoiding multiple collisions through trajectory replanning using piecewise B amp ;#x00E9 ;zier curves. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 2755–2760.

[Mellinger et Kumar, 2011] Mellinger, D. et Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE Int. Conf. on Robotics and Automation*, pages 2520–2525.

[Mohanan et Salgoankar, 2018] Mohanan, M. G. et Salgoankar, A. (2018). A survey of robotic motion planning in dynamic environments. *Robotics and Autonomous Systems*, 100 :171 – 185.

[Mourikis et Roumeliotis, 2007] Mourikis, A. I. et Roumeliotis, S. I. (2007). A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation. In *IEEE Int. Conf. on Robotics and Automation*.

[Moysis et al., 2020] Moysis, L., Petavratzis, E., Volos, C., Nistazakis, H., Stouboulos, I. et Valavanis, K. (2020). A chaotic path planning method for 3D area coverage using modified logistic map and a modulo tactic. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

[Mueller et al., 2015] Mueller, M. W., Hehn, M. et D'Andrea, R. (2015). A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation. *IEEE Trans. on Robotics*, 31(6) :1294–1310.

[Nasr et al., 2018] Nasr, M., Ashraf, M., Hussein, M. S., Salem, A. S., Elias, C. M., Shehata, O. M. et Morgan, E. I. (2018). A comparitive study on the control of UAVs for trajectory tracking by mpc, smc, backstepping, and fuzzy logic controllers. In *2018 IEEE Int. Conf. on Vehicular Electronics and Safety (ICVES)*, pages 1–6.

[Nikolos et al., 2003] Nikolos, I. K., Valavanis, K. P., Tsourveloudis, N. C. et Kostaras, A. N. (2003). Evolutionary algorithm based offline/online path planner for UAV navigation. *IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 33(6) :898–912.

[Nowacki et al., 1990] Nowacki, H., Liu, D. et Lü, X. (1990). Fairing bézier curves with constraints. *Computer Aided Geometric Design*, 7(1) :43–55.

[Oettershagen et al., 2017] Oettershagen, P., Achermann, F., Müller, B., Schneider, D. et Siegwart, R. (2017). Towards Fully Environment-Aware UAVs : Real-Time Path Planning with Online 3D Wind Field Prediction in Complex Terrain. *arXiv :1712.03608 [cs]*.

[Palm et Driankov, 2014] Palm, R. et Driankov, D. (2014). Fluid mechanics for path planning and obstacle avoidance of mobile robots. In *2014 11th Int. Conf. on Informatics in Control, Automation and Robotics (ICINCO)*, volume 02, pages 231–238.

[Palma et al., 2016] Palma, G., Cignoni, P., Boubekeur, T. et Scopigno, R. (2016). Detection of Geometric Temporal Changes in Point Clouds : Detection of Changes in Point Clouds. *Computer Graphics Forum*, 35(6) :33–45.

[Panagou, 2014] Panagou, D. (2014). Motion planning and collision avoidance using navigation vector fields. In *2014 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2513–2518, Hong Kong, China. IEEE.

[Panati et al., 2015] Panati, S., Baasandorj, B. et Chong, K. T. (2015). Autonomous Mobile Robot Navigation Using Harmonic Potential Field. *IOP Conference Series : Materials Science and Engineering*, 83 :012018.

[Panchpor et al., 2018] Panchpor, A. A., Shue, S. et Conrad, J. M. (2018). A survey of methods for mobile robot localization and mapping in dynamic indoor environments. In *2018 Conference on Signal Processing And Communication Engineering Systems (SPACES)*, pages 138–144.

[Papachristos et al., 2019] Papachristos, C., Mascarich, F., Khattak, S., Dang, T. et Alexis, K. (2019). Localization uncertainty-aware autonomous exploration and mapping with aerial robots using receding horizon path-planning. *Autonomous Robots*, 43 :2131–2161.

[Penin et al., 2018] Penin, B., Giordano, P. R. et Chaumette, F. (2018). Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions. *IEEE robotics and automation letters*.

[Pisano, 2001] Pisano, D. A. (2001). Air Power in the Age of Total War. By John Buckley. (Bloomington : Indiana University Press, 1999.). *Journal of American History*, 87(4) :1563–1564.

[Primatesta et al., 2020] Primatesta, S., Scanavino, M., Guglieri, G. et Rizzo, A. (2020). A risk-based path planning strategy to compute optimum risk path for unmanned aircraft systems over populated areas. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

[Pumarola et al., 2017] Pumarola, A., Vakhitov, A., Agudo, A., Sanfeliu, A. et Moreno-Noguer, F. (2017). PL-SLAM : Real-time monocular visual SLAM with points and lines. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4503–4508. IEEE.

[Qin et al., 2017] Qin, T., Li, P. et Shen, S. (2017). VINS-Mono : A Robust and Versatile Monocular Visual-Inertial State Estimator. *arXiv :1708.03852 [cs]*.

[Radmanesh et al., 2018] Radmanesh, M., Kumar, M., Guentert, P. H. et Sarim, M. (2018). Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs : A Comparative Study. *Unmanned Systems*, 06(02) :95–118.

[Raemaekers, 2007] Raemaekers, A. (2007). Design of a model predictive controller to control UAVs. In *Computer Science, Economics*.

[Rendón et Martins, 2017] Rendón, M. A. et Martins, F. F. (2017). Path Following Control Tuning for an Autonomous Unmanned Quadrotor Using Particle Swarm Optimization. *IFAC-PapersOnLine*, 50(1) :325–330.

[Rigter et al., 2019] Rigter, M., Morrell, B., Reid, R. G., Merewether, G. B., Tzanetos, T., Rajur, V., Wong, K. et Matthies, L. H. (2019). An autonomous quadrotor system for robust high-speed flight through cluttered environments without gps. In *2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 5227–5234.

[Rivera et al., 2012] Rivera, D. M., Prieto, F. A. et Ramírez, R. (2012). Trajectory planning for UAVs in 3D environments using a moving band in potential sigmoid fields. In *2012 Brazilian Robotics Symposium and Latin American Robotics Symposium*, pages 115–119.

[R.l et D., 2017] R.l, M.-A. et D., T. J. (2017). ORB-SLAM2 : an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Trans. on Robotics*, 33(5) :1255–1262.

[Sahingoz, 2013] Sahingoz, O. K. (2013). Flyable path planning for a multi-UAV system with Genetic Algorithms and Bezier curves. In *2013 Int. Conf. on Unmanned Aircraft Systems (ICUAS)*, pages 41–48.

[Samaniego et al., 2017] Samaniego, F., Sanchis, J., García-Nieto, S. et Simarro, R. (2017). UAV motion planning and obstacle avoidance based on adaptive 3D cell decomposition : Continuous space vs discrete space. In *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, pages 1–6.

[Seemann et Janschek, 2014] Seemann, M. et Janschek, K. (2014). RRT-based Trajectory Planning for Fixed Wing UAVs using Bezier Curves. In *ISR/Robotik 2014 ; 41st Int. Symposium on Robotics*, pages 1–8.

[Shan et Englot, 2017] Shan, T. et Englot, B. (2017). Belief roadmap search : Advances in optimal and efficient planning under uncertainty. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 5318–5325.

[Shim et Sastry, 2002] Shim, D. et Sastry, S. (2002). Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles. In *Proc. of the American Control Conference*, volume 5, pages 3576 – 3581 vol.5.

[Stachniss et al., 2005] Stachniss, C., Grisetti, G. et Burgard, W. (2005). Information Gain-based Exploration Using Rao-Blackwellized Particle Filters. In *Robotics : Science and Systems*. Robotics : Science and Systems Foundation.

[Sun et al., 2017] Sun, K., Mohta, K., Pfrommer, B., Watterson, M., Liu, S., Mulgaonkar, Y., Taylor, C. J. et Kumar, V. (2017). Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight. *arXiv :1712.00036 [cs]*.

[Tam et al., 2013] Tam, G. K. L., Zhi-Quan Cheng, Yu-Kun Lai, Langbein, F. C., Yonghuai Liu, Marshall, D., Martin, R. R., Xian-Fang Sun et Rosin, P. L. (2013). Registration of 3D Point Clouds and Meshes : A Survey from Rigid to Nonrigid. *IEEE Trans. on Visualization and Computer Graphics*, 19(7) :1199–1217.

[Tang et al., 2020] Tang, Z., Chen, B., Lan, R. et Li, S. (2020). Vector field guided rrt* based on motion primitives for quadrotorkinodynamic planning. *Journal of Intelligent & Robotic Systems*.

[Tezenas du Montcel et al., 2019] Tezenas du Montcel, T., Negre, A., Gomez-Balderas, J.-E. et Marchand, N. (2019). BOARR : A Benchmark for quadrotor Obstacle Avoidance based on ROS and RotorS. working paper or preprint.

[Upadhyay et al., 2020] Upadhyay, S., Richards, A. et Richardson, T. (2020). Generation of window-traversing flyable trajectories using logistic curve. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

[Vakhitov et al., 2016] Vakhitov, A., Funke, J. et Moreno-Noguer, F. (2016). Accurate and Linear Time Pose Estimation from Points and Lines. In Leibe, B., Matas, J., Sebe, N. et Welling, M., editors, *Computer Vision – ECCV 2016*, volume 9911, pages 583–599. Springer International Publishing, Cham.

[Van Parys et Pipeleers, 2017] Van Parys, R. et Pipeleers, G. (2017). Spline-Based Motion Planning in an Obstructed 3D Environment * *This work benefits from KU Leuven-BOF PFV/10/002 Centre of Excellence. *IFAC-PapersOnLine*, 50(1) :8668–8673.

[Vaquero et al., 2017] Vaquero, V., del Pino, I., Moreno-Noguer, F., Sola, J., Sanfeliu, A. et Andrade-Cetto, J. (2017). Deconvolutional networks for point-cloud vehicle detection and tracking in driving scenarios. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–7, Paris. IEEE.

[Wang et al., 2017] Wang, X., Jiang, P., Li, D. et Sun, T. (2017). Curvature Continuous and Bounded Path Planning for Fixed-Wing UAVs. *Sensors*, 17(9) :2155.

[Wei et al., 2016] Wei, B., Zhang, Q., Xu, Z. et Wang, L. (2016). Study on real-time safe route planning method for quad-rotor UAV among buildings. In *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, pages 1067–1071.

[Wong et al., 2017] Wong, J. M., Kee, V., Le, T., Wagner, S., Mariottini, G., Schneider, A., Hamilton, L., Chipalkatty, R., Hebert, M., Johnson, D. M. S., Wu, J., Zhou, B. et Torralba, A. (2017). SegICP : Integrated deep semantic segmentation and pose estimation. In *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 5784–5789.

[Wu et al., 2017] Wu, K., Xi, T. et Wang, H. (2017). Real-time three-dimensional smooth path planning for unmanned aerial vehicles in completely unknown cluttered environments. In *TENCON 2017 - 2017 IEEE Region 10 Conference*, pages 2017–2022.

[Wu et al., 2015] Wu, X., Guo, C., Li, Y., Chen, W. et Wang, Y. (2015). The Research of RRT Route Planning Algorithm for UAV that Based on Kinematic Equation. *Int. Journal of Control and Automation*, 8(1) :10.

[Yan et al., 2020] Yan, C., iaojia Xiang et Wang, C. (2020). Towards real-time path planning through deep reinforcementlearning for a UAV in dynamic environments. *Journal of Intelligent & Robotic Systems*, 98 :297–309.

[Yan et al., 2012] Yan, F., Zhuang, Y. et Xiao, J. (2012). 3D PRM based real-time path planning for UAV in complex environment. In *IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, pages 1135–1140.

[Yang et Sukkarieh, 2008a] Yang, K. et Sukkarieh, S. (2008a). 3D smooth path planning for a UAV in cluttered natural environments. In *2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 794–800.

[Yang et Sukkarieh, 2008b] Yang, K. et Sukkarieh, S. (2008b). Real-time continuous curvature path planning of UAVS in cluttered environments. In *2008 5th Int. Symposium on Mechatronics and Its Applications*, pages 1–6.

[Yang et al., 2015] Yang, L., Song, D., Xiao, J., Han, J., Yang, L. et Cao, Y. (2015). Generation of dynamically feasible and collision free trajectory by applying six-order Bezier curve and local optimal reshaping. In *2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 643–648.

[Yel et al., 2017] Yel, E., Lin, T. X. et Bezzo, N. (2017). Reachability-based self-triggered scheduling and replanning of UAV operations. In *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 221–228.

[Yingkun, 2018] Yingkun, Z. (2018). Flight path planning of agriculture UAV based on improved artificial potential field method. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 1526–1530.

[Youngji Kim et al., 2015] Youngji Kim, Hwasup Lim et Sang Chul Ahn (2015). Multi-body ICP : Motion segmentation of rigid objects on dense point clouds. In *2015 12th Int. Conf. on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 532–536.

[Yu et al., 2017] Yu, G., Alvear, O., Yu, G. et Natalizio, E. (2017). Using UAV-based systems to monitor air pollution in areas with poor accessibility. In *Journal of Advanced Transportation.*

[Zhang et al., 2015] Zhang, Q., Rekleitis, I. et Dudek, G. (2015). Uncertainty reduction via heuristic search planning on hybrid metric/topological map. In *12th Conf. on Computer and Robot Vision*, pages 222–229.

[Zhang et Huo, 2017] Zhang, T. et Huo, X. (2017). Path planning and control of a quadrotor UAV : A symbolic approach. In *2017 11th Asian Control Conference (ASCC)*, pages 2750–2755.

[Zhang et al., 2018] Zhang, X., Lu, X., Jia, S. et Li, X. (2018). A novel phase angle-encoded fruit fly optimization algorithm with mutation adaptation mechanism applied to UAV path planning. *Applied Soft Computing*, 70 :371–388.

[Zhang et Scaramuzza, 2018] Zhang, Z. et Scaramuzza, D. (2018). Perception-aware receding horizon navigation for MAVs. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2534–2541.

[Zhao et al., 2018] Zhao, M., Anzai, T., Shi, F., Chen, X., Okada, K. et Inaba, M. (2018). Design, modeling, and control of an aerial robot dragon : A dual-rotor-embedded multilink robot with the ability of multi-degree-of-freedom aerial transformation. *IEEE Robotics and Automation Letters*, 3(2) :1176–1183.

[Zhao et al., 2017] Zhao, Y., Jiao, L., Zhou, R. et Zhang, J. (2017). UAV formation control with obstacle avoidance using improved artificial potential fields. In *2017 36th Chinese Control Conference (CCC)*, pages 6219–6224.

[Zhen et al., 2017] Zhen, W., Zeng, S. et Soberer, S. (2017). Robust localization and localizability estimation with a rotating laser scanner. In *2017 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 6240–6245.

[Zheng et al., 2020] Zheng, Z., Bewley, T. R. et Kuester, F. (2020). Point cloud-based target-oriented 3D path planning for UAVs. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

[Zhu et Cao, 2019] Zhu, Z. et Cao, S. (2019). Back-stepping sliding mode control method for quadrotor UAV with actuator failure. *The Journal of Engineering*, 2019(22) :8374–8377.

Planification du mouvement 3D en temps réel de drones autonomes dans des environnements dynamiques inconnus

Résumé :

Les drones sont utilisés dans des environnements de plus en plus complexes. Ce manuscrit traite du problème de planification de trajectoire dans des environnements dynamiques, connus ou inconnus. En utilisant la perception locale du drone ainsi que sa dynamique et celle de son environnement, des courbes de Bézier sont utilisées comme trajectoires cibles pour le déplacement autonome. De manière complètement autonome et en temps réel, une optimisation sous différentes contraintes est ainsi réalisée pour trouver à chaque instant la meilleure trajectoire lisse et faisable à suivre par le drone. Cette trajectoire respecte la dynamique du drone et son état réel estimé tout en étant un chemin sûr. L'utilisation de l'état réel du drone et de la trajectoire consigne précédente, permet d'assurer la continuité entre deux courbes successives et ainsi la régularité des trajectoires de vol. Les profils de vitesse et d'accélération sont également impactés par le processus d'optimisation.

Ce manuscrit traite du problème de planification de trajectoire de drones dans des environnements dynamiques en 3 dimensions. Les résultats consistent en la génération d'une trajectoire 3D sûre, intégrant des contraintes dynamiques pour assurer la faisabilité de celle-ci par le drone grâce à un contrôleur. Les principales contributions sont :

— Utilisation de l'information 3D complète de l'environnement pour la génération de trajectoires.
— Prédiction du déplacement des obstacles et prise en compte dans l'algorithme d'évitement.
— Évitement d'obstacles basé sur la géométrie, sur les capacités dynamiques du drone et de la variabilité de l'environnement.
— Continuité de la commande : le début d'une nouvelle trajectoire respecte l'état réel du drone et la nouvelle trajectoire dépend de la précédente.
— Embarquabilité du code, contraintes de temps de calcul respectées vis-à-vis du vol.

En plus d'une distance de sécurité, un cône de vitesse est utilisé pour améliorer l'évitement, chaque obstacle a donc une approximation de vitesse associée. Trois types de contrôleurs sont testés pour assurer le suivi des courbes générées. Des résultats en simulations montrent de manière statistique l'efficacité de notre approche dans différents environnements réalistes. L'impact des différents paramètres sur les trajectoires résultantes est aussi étudié. Différent tests réels sont réalisés pour valider la charge de calcul, les aspects temps réel et la faisabilité de l'ensemble de l'architecture de la perception au contrôle, sur un ordinateur embarqué.

Mots clés : Planification de trajectoire, 3D, courbes de Bézier, évitement d'obstacles, dynamique, Backstepping, prédiction de déplacement.

XLIM - UMR CNRS / Université de Limoges n 7252
123, rue Albert Thomas - 87060 LIMOGES CEDEX