RAPPORT FINAL

INF11 - Programmation de drones collaboratifs pour contourner des obstacles

13 avril 2020

ALBUQUERQUE SILVA Igor, FORTES MACHADO Alicia,
GALVÃO LOPES Aloysio, SOALI Mohamed Yahya, SPITZER Victor
X2018

TABLE DES MATIÈRES

1
REMERCIEMENTS

- On souhaite tout d'abord remercier l'École Polytechnique et le Pôle direction des études pour la précieuse opportunité et aubaine qu'ils nous offrent en nous proposant la réalisation de projets scientifiques collectifs, riches en enseignements, visant notre épanouissement.

- Nous tenons aussi à remercier M. Gilles SCHAEFFER, directeur de recherche au CNRS et professeur chargé de cours à l'École Polytechnique d'avoir accepté de cordonner notre projette de ses conseils lors de la réunion de cadrage qui nous ont permis de mieux cerner le sujet.

- Nous tenons à exprimer notre profonde gratitude à Mme Sylvie PUTOT et M. Eric GOUBAULT, tout deux Enseignant-Chercheur à l'École Polytechnique pour leur accompagnement bienveillant et investissement particulier en tant que tuteurs de notre projet. Ils furent d'une aide précieuse pour la recherche des différents algorithmes de planification de trajectoires et en général pour la réalisation de ce présent projet.

- On souhaite aussi remercier, à travers eux, le laboratoire d'informatique de l'École Polytechnique (LIX) de nous avoir accueilli et mis à notre disposition le matériel nécessaire (à savoir les crazyflies).

2 RÉSUMÉ

La montée en puissance de l'utilisation des drones ces dernières années dans différents domaines de production a révolutionné le progrès technologique. Ce secteur prometteur en plein essor a suscité un intérêt croissant depuis son apparition de la part des entreprises qui cherchent à en bénéficier, et l'on assiste aujourd'hui à une course contre le temps pour l'amélioration des performances de ces engins notamment l'autonomie du système de pilotage et son adaptation face aux différents imprévus rencontrés (obstacles). Un autre défi est celui de la conception d'un système multi-agents qui exploite la coordination entre différents drones afin de réaliser une tâche qui leur est dévolue. L'objet du présent projet, réalisée au sein du laboratoire d'Informatique de l'École Polytechnique (LIX) est de permettre à un ensemble de drones collaboratifs (crazyflies du laboratoire) d'évoluer et se déplacer dans un même espace et en même temps tout en contournant les éventuels obstacles rencontrés et sans collisions entre eux. Une première étape consiste à décrire numériquement le milieu dans lequel évoluent nos drones (positions des différents obstacles). Puis, il s'agit de calculer les trajectoires optimales accessibles de déplacement de chaque drone d'un point à un autre du milieu pour en conclure par une solution qui exploite la coordination entre drones en jouant sur des paramètres relatifs à ces derniers (vitesses) afin d'avoir une navigation collective d'ensemble sans aucun risque de collision.

3
INTRODUCTION

L'évolution croissante du domaine de l'aéronautique a permis à l'homme de concevoir des mini-aéronefs sans pilote à bord appelés "drones". Si leur utilisation se restreignait auparavant, lors de leur découverte, aux activités militaires (avec des aéronefs sans pilote), ces engins ont démontré récemment leur utilité dans un cadre beaucoup plus large en aidant à réaliser des missions civiles de nature très variées comme, par exemple, la réalisation de prises de vue aériennes (drones grand-public), la surveillance de zones privées (drones professionnels) et même dans le secteur d'agriculture en remplaçant les planeurs existants. Ces engins constituent ainsi un vrai potentiel en hausse dans un contexte industriel dans lequel investissent plusieurs entreprises souhaitant optimiser leurs chaînes de production. Citons ici l'exemple du secteur de logistique qui peut s'améliorer rapidement grâce à cette technologie. (exemple d'Amazon Prime Air [1]).

Avec un tel potentiel de croissance et de nombreuses possibilités passionnantes de révolutionner le marché, se posent des questions d'optimisation sur la bonne exploitation des drones et de défis qu'ils soulèvent. L'expansion du marché des drones impacte ainsi plus que les entreprises qui les utilisent et s'étend pour inclure tout les chercheurs de l'électronique et de l'informatique. De grandes recherches et études sont aujourd'hui réalisées et menées (au MIT par exemple) pour améliorer les performances de ces aéronefs, notamment les tailles des capteurs qu'ils peuvent héberger, la possibilité de les réduire (comme dans un Crazyflie [2]) mais aussi le système de pilotage utilisé et son adaptation aux différents imprévus. En effet, la diversité des environnements et des obstacles est un frein majeur à l'utilisation massive des drones qui fait que la majorité d'entre eux sont encore télépilotés à distance, mais cela n'empêchent qu'ils pourraient au fil du temps être de plus en plus autonome.

Avant d'atteindre une autonomie parfaite, un premier défi est de concevoir un système intégré qui aide au pilotage du drone, en lui permettant d'éviter les différents obstacles qu'il est censé rencontrer. Un autre est celui de la conception de systèmes multi-agents basé sur l'idée de drones collaboratifs qui utilisent la coordination entre ces derniers pour la résolution collective d'un problème. Ce dernier objectif fait partie des enjeux de la robotique et informatique moderne ; secteur attrayant et prometteur de progrès devenu aujourd'hui très actif et très médiatisé.

Arriver à contourner des obstacles est donc une condition sine-qua-non du vol réussi des drones. Tout notre travail consistera, donc, dans un premier temps au développement d'un système permettant à chaque drone de prévoir sa trajectoire optimale à suivre tout en contournant les différents obstacles rencontrés durant son vol et détectés auparavant grâce à un système de vision. Dans un second temps, on améliorera ce système pour permettre à un ensemble de drones collaboratifs d'évoluer dans un même espace et en même temps sans avoir de collisions entre eux.

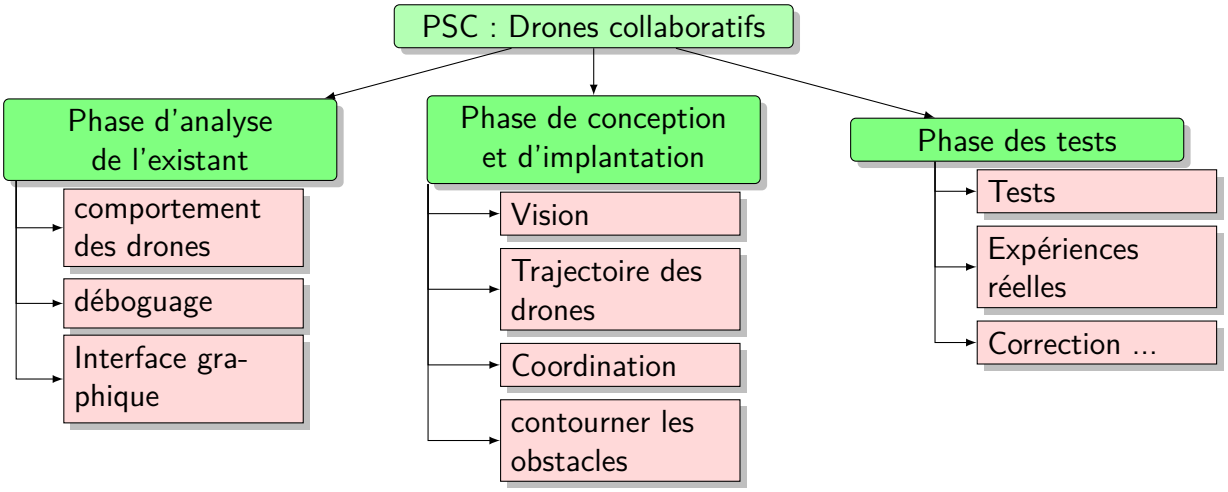
4 OBJECTIFS

4.1 Idée directrice du projet

Ce projet s'intéresse à la possibilité de coordination entre drones, chose qui peut permettre à ces derniers d'évoluer dans le même environnement et donc d'effectuer plusieurs tâches sans aucune collision entre eux ou avec les obstacles présents dans le milieu. Pour en aboutir, il parait nécessaire de construire une image 3D (discrétisation) du milieu dans lequel évoluent les drones, de calculer la trajectoire optimale de chacun d'eux (cas d'un seul drone) puis de réfléchir à une solution qui exploite la coordination entre drones afin d'avoir une navigation collective d'ensemble sans aucun risque de collision.

4.2 Révisions d'objectifs

L'objectif final et général du PSC, étant la programmation des trajectoires de drones collaboratifs pour contourner des obstacles, était divisé en plusieurs sous-objectifs concernant la compréhension du fonctionnement des drones, le système de vision, la planification de trajectoire, la coordination des drones et le test des résultats, ce qui correspondait aux trois principales phases de notre projet :



Dans ce qui suit, on décrira les trois phases de notre projet et les objectifs souhaités derrière chacune d'eux.

* Phase d'observation et d'analyse des différents algorithmes informatiques et simulateurs du fonctionnement de nos crazyflies (drones) :

L'équipe a vu qu'il était essentiel de disposer d'une interface permettant de se connecter aux drones, de les sélectionner et de déboguer leur comportement. Cela facilite et organise l'accès aux différents programmes ainsi que le contrôle et suivi instantané des différents drones. Au cours de la première moitié du PSC, l'équipe a pu finir cette interface. L'équipe considère que ce travail a été essentiel pour gagner du temps et, de plus, l'interface est facilement réutilisable pour d'autres projets, de sorte que nous pouvons apporter une forte contribution au développement de projets avec les drones du LIX. Les résultats de cette partie seront développés dans la section *Interface Graphique*.

En parallèle, une étude des techniques de planification de trajectoire à utiliser a été faite. L'équipe a choisi de passer beaucoup de temps à étudier les possibilités sans les mettre en œuvre pour distinguer les approches réalisables des plus ambitieuses. Enfin, l'équipe est parvenue à une conclusion concernant la démarche à utiliser, cette conclusion est détaillée dans la phase qui suit.

* Phase de recherche, de conception et d'implantation : Cette partie, qui est le socle de notre projet, porte sur :

- L'algorithme du système de vision d'un côté qui nous permettra de reconstruire une carte 3D du milieu des drones et la détection des obstacles.

- La planification des trajectoires : on est parti de l'analyse développé dans l'article [3] pour définir notre méthode d'aborder ce problème et qui s'articulera comme suit :

1 - Recherche des trajectoires optimales des drones tout en évitant les obstacles fixes du milieu, ceci sachant leurs points de départs et d'arrivées ainsi que la carte de l'espace avec les obstacles (obtenue à partir du système de vision). (cette partie a été réalisée avec un algorithme A* de Dijkstra).

2 - Identifier les éventuelles zones de collision entre les drones, lorsque ceux-ci évoluent dans leurs propres trajectoires.

3 - La coordination entre les drones pour qu'ils arrivent à contourner les obstacles en même temps : Jouer sur un ou plusieurs paramètres relatifs aux drones pour arriver à éviter les collisions. (ici les vitesses tout au long des trajectoires).

C'est pour cette partie qu'on a consacré le plus de temps notamment la planification des trajectoires et la coordination. Partie difficile certes mais réalisable.

* Phases de tests, d'expériences et de correction : Tester la validité des algorithmes sur un logiciel de simulation (GAZEBO) avant d'agir sur scène. Tester les algorithmes précédents sur des obstacles fixes et mouvants dans la salle. (Sachant que les obstacles mouvants ne sont autres que les drones). Notons toutefois ici qu'on voulait à la fin tester nos algorithmes sur les crazyflies du LIX si on était présents sur le campus de l'école. En attendant notre retour, on s'est contenté des simulation sur GAZEBO.

5
CONCEPTION D'ALGORITHMES

5.1 Discrétisation

La première étape de l'algorithme choisi dans le cadre de ce projet consiste à discrétiser l'espace dans lequel les drones peuvent se déplacer. Cela signifie que là où les obstacles sont présents, l'espace ne sera pas discrétisé et, par conséquent, dans l'étape de discrétisation, il est automatiquement garanti que tous les chemins possibles à générer ne vont pas entrer en collision avec les obstacles.

De manière simplifiée, l'algorithme fonctionne comme suit :

— Pour chaque drone, la trajectoire optimale est calculée dans l'espace discret avec l'algorithme A*.

— L'algorithme de coordination calcule les vitesses sur ces trajectoires afin qu'aucune collision entre drones ne se produise.

— Les drones parcourent les trajectoires, qui sont lissées par des Splines Cubiques.

Il convient de noter que la taille finale de la trajectoire est essentiellement due à la méthode de discrétisation utilisée. Au début, nous avons essayé d'utiliser une discrétisation uniforme de l'espace en cubes, cette méthode présentait deux limites majeures. Premièrement, elle fournit le même niveau de détail dans toutes les zones de l'espace, en fait, on souhaite un plus grand détail près des obstacles. L'autre problème majeur est que les trajectoires générées sont toujours de taille égale à la somme des modules des coordonnées du déplacement vectoriel à exécuter. Ce dernier facteur montre également que l'augmentation de la résolution du maillage ne diminue pas la taille des trajectoires, ce qui est extrêmement indésirable.

Pour résoudre ces deux problèmes, nous avons choisi d'utiliser le *mesh* comme méthode de discrétisation. Plus précisément, nous avons utilisé la bibliothèque CGAL [4], qui utilise la triangulation 3D de Delaunay pour générer ses *meshes*. Ainsi, la discrétisation utilisée dans ce projet est le résultat d'une triangulation de Delaunay et aucun processus de raffinement de *mesh* n'est appliqué. Les *meshes* générés sont plus détaillés aux extrémités, puisque la triangulation est pondérée et, surtout, n'ont pas la même limitation sur la taille des trajectoires.

5.2 Coordination

Nous avons utilisé l'algorithme décrit dans GRIST R. et al. [3]. Il s'agit d'une coordination entre des drones volants, il calcule ainsi des trajectoires Pareto-optimales pour éviter les

collisions et minimiser le temps de parcours.

Étant donné les trajectoires pré-calculées pour chaque drone, avec la vitesse en chaque point, et des informations sur les points d'intersection de ces trajectoires, l'algorithme ajustera les vitesses des drones afin que ceux-ci suivent toujours ces trajectoires sans collision, et que le temps nécessaire pour que le dernier drone arrive à sa destination soit minimisé.

Deux conditions sont imposées par cette méthode :

— Les trajectoires initiales doivent être linéaires par morceaux. Ainsi, elles sont une suite de points pour chaque drone ;

— L'espace de collision est cylindrique, c'est-à-dire que les intersections sont définies comme des régions sur les trajectoires de deux drones au maximum. Cela signifie que les intersections de trois drones ou plus en même temps sont définies pour chaque paire de drones.

Il faut noter que les obstacles fixes doivent déjà être pris en compte dans les trajectoires données pour chaque drone, de sorte qu'ils contiennent déjà un chemin libre.

5.2.1 • Prétraitement

Soit la trajectoire de chaque robot Γ_i, et les intersections entre elles $\mathcal{O}$. L'article définit l'espace de coordination des drones,

$$\mathcal{X} = (\Gamma_1 \times ... \times \Gamma_n) - \mathcal{O}, \tag{1}$$

et ces projections

$$\mathcal{X}_{i,j} = (\Gamma_i \times \Gamma_j) - \Delta_{i,j}, \tag{2}$$

où $\Delta_{i,j}$ est l'ensemble des intersections de Γ_i et Γ_j.

En utilisant les trajectoires et les vitesses données, il faut définir des orientations pour chacune des intersections dans les projections $\Delta_{i,j}$. Une orientation dans le sens horaire signifie que le drone j avancera l'intersection avant le drone i, et une orientation dans le sens anti-horaire signifie le contraire. L'algorithme décrit ci-après permettra de s'assurer que les trajectoires optimisées aient la même classe d'homotopie que la donné, c'est-à-dire qu'elles ne modifieront pas les orientations des intersections.

La Figure 1 montre un espace de coordination paramétré par la distance parcourue dans le chemin. Ici, le chemin en rouge est la coordination donné initialement, et le chemin en bleue est la coordination optimisé. Les rectangles noirs sont deux régions d'intersections, où CW signifie sens horaire et CCW signifie sens anti-horaire.

Pour plusieurs drones, nous déterminons le graphique de coordination entre chaque paire de drones, pour ensuite seulement suivre les étapes suivantes. Mais il a été pensé, au lieu de créer plusieurs graphes deux par deux, de créer un graphe à n dimensions avec les trajectoires paramétrées de n drones sur les axes et de mettre ensuite en œuvre cette même idée dans un seul graphe multidimensionnel. Cependant, la mise en œuvre serait complexe, car il faudrait définir comment nous orienterions les objets et autres complications.

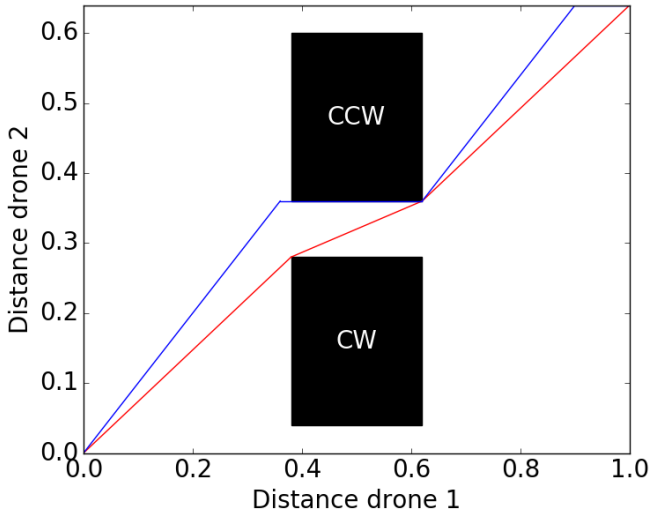


FIGURE 1 – $\mathcal{X}_{1,2}$, la projection de l'espace de coordination pour deux drones.

5.2.2 • MÉTHODE ITÉRATIF

Après ce prétraitement, GRIST R. et al. [3] décrit un algorithme itératif pour optimiser la vitesse du drone à chaque instant. Il est basé sur la définition d'événements et le calcul du prochain événement qui se produira à chaque itération. Lorsque cet événement se produit, il faut mettre à jour les vitesses pour éviter cette collision. Pour cette mise à jour, un algorithme linéaire est utilisé.

Cet algorithme linéaire consiste à maximiser la vitesse face aux contraintes suivantes :

— Contraintes de vitesse globale induites par le modèle : $0 < v_i < v_{max}$ pour chaque $0 < i < N$ (N correspondent à la quantité de drones) ;

— Pour chaque drone arrêté en attendant qu'il puisse bouger : $v_i = 0$;

— Pour chaque bord de limite d'obstacle e contenant x, soit Xi, j la projection qui contient e et que m indique l'inclinaison de e. Ensuite, afin d'éviter une collision, nous avons la contrainte $v_i - m \cdot v_j \leq 0$ si x est en dessous de l'obstacle et $v_i - m \cdot v_j \geq 0$ si x est au-dessus de l'obstacle ;

— Pour chaque drone ayant obtenu son but : $v_i = 0$.

Ainsi, on trouve la vitesse maximale possible pour chaque drone à chaque évènement et les attribuent. On répète ces étapes de prendre un évènement et changer les vitesses jusqu'à ce que tous les drones aient atteint la destination.

6 RÉSULTATS

6.1 Structure du Code

Il est essentiel pour un projet logiciel d'avoir une bonne organisation pour faciliter la compréhension et le développement de la base du code. Pour cette raison, nous avons pris le temps de réfléchir à une structure à la fois organisée et simple.

Nous l'avons divisée en cinq dossiers, comme le montre la Figure 2.

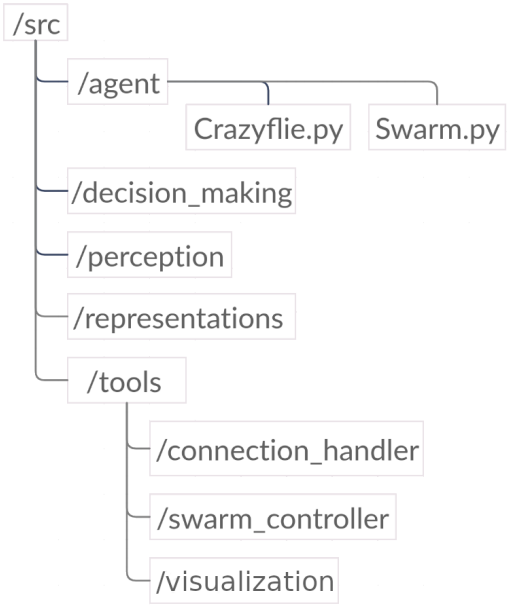


Figure 2 – Arbre des dossiers du code.

Le dossier `agent` contient les classes correspondantes aux drones et à l'essaim de drones, qui sont les classes de plus haut niveau. Le dossier `decision_making` contient les classes des algorithmes de planification de la trajectoire. Le dossier `perception` possède les classes associées à la vision et à la modélisation de l'espace. Le dossier `representations` contient les structures d'abstraction communes à plusieurs parties du code. Le dossier `tools` contient les interfaces qui seront expliquées dans la section suivante.

6.2 Interface graphique

L'équipe a identifié trois difficultés principales au début du projet :

— Connecter et déconnecter rapidement les drones, car plusieurs drones sont utilisés.
— Sélectionner les drones qui seront utilisés dans un environnement réel ou simulé.
— Pouvoir visualiser la position reçue par les drones ainsi que les trajectoires des drones.

Pour résoudre ces problèmes, l'équipe a décidé de développer une interface graphique qui intègre ces trois éléments, le but de cette interface est d'être le seul élément exécuté pour faire voler les drones. Il a été décidé de développer ces éléments séparément comme des *plugins rqt* [5], ainsi les *plugins* : *connexion_handler*, *swarm_controller* et un code pour la visualisation dans le *plugin rviz* [6] ont été créés. La Figure 3 montre l'interface graphique complète et ses trois éléments développés sont mis en évidence. Dans les prochaines sous-sections, le fonctionnement de chacune de ces parties sera détaillé.

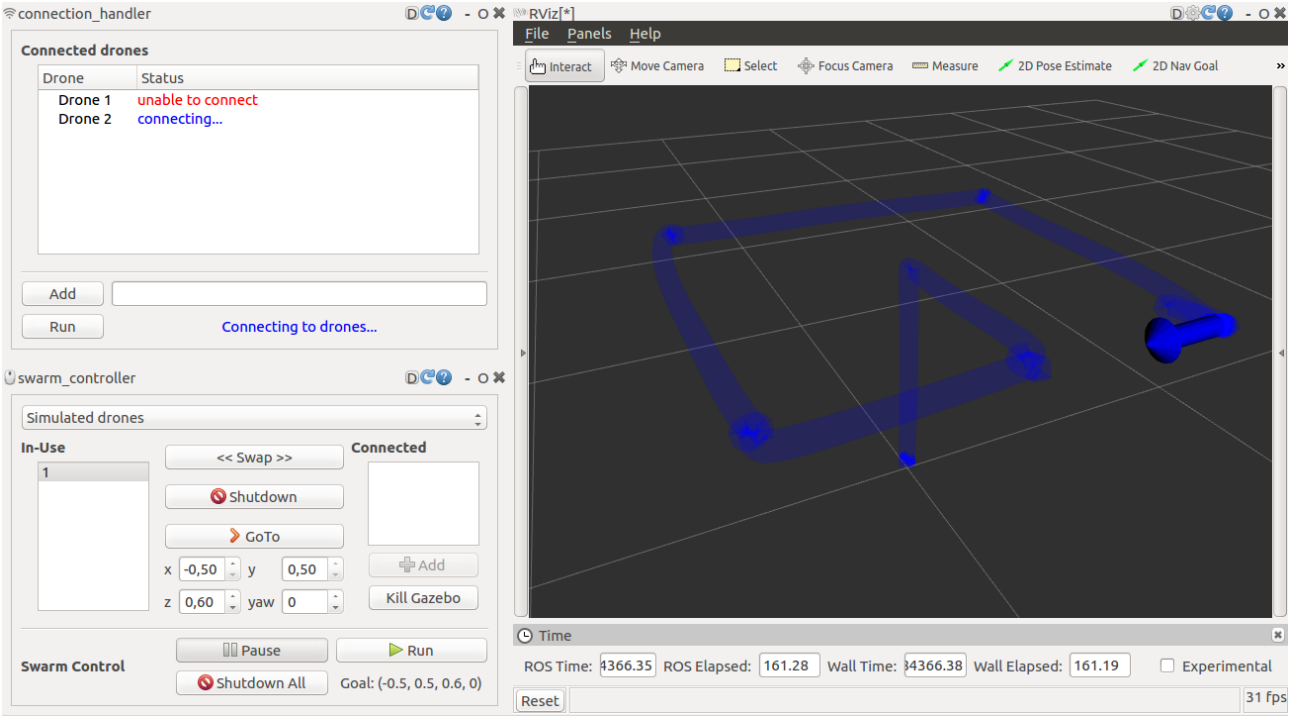


Figure 3 – Interface développée pour contrôler et visualiser les données des drones. Elle est divisée en trois parties, où la partie supérieure gauche est le *connexion_handler*, la partie inférieure gauche est le *swarm_controller*, et la partie à droite est le *rviz*.

6.2.1 • Connection handler plugin

Cet outil permet d'initialiser la connexion avec les drones souhaités. Pour ce faire, il suffit de préciser le nombre des drones auxquels vous souhaitez vous connecter et l'interface obtient les identifiants de ces drones. Dans un deuxième temps, un serveur est lancé, il sera responsable pour la communication avec les drones et recevra les messages du code exécuté par le *plugin swarm_controller*.

Il convient de noter que pour développer cet outil, il a été nécessaire d'utiliser des sous-processus pour effectuer des connexions multiples, puisque ces connexions sont effectuées par un *script* différent. Il convient également de noter qu'il était nécessaire d'utiliser des *threads* séparés pour que l'interface graphique puisse rester réactive.

Le point principal de cette interface est la liste des drones, elle indique, pour chaque drone, la numération à droite et l'état de la connexion à gauche. Sur cette interface, il y a deux boutons : le bouton *add*, qui permet de trouver les identifiants des drones placés sur la zone de texte de l'interface et le bouton *run*, qui permet d'initialiser le serveur. Si le serveur est éteint et un drone est sélectionné sur la liste, le bouton *add* se transforme en *remove* et le drone sélectionné peut être retiré de la liste des drones. De plus, si le serveur est allumé, le bouton *run* se transforme en *stop* et permet au serveur de s'arrêter. Il y a également une sortie de texte à droite du bouton *run*, qui indique l'état du serveur et qui est également utilisée pour déboguer la connexion aux drones.

6.2.2 • Swarm controller plugin

Cet outil permet de contrôler les drones de manière efficace, qu'ils soient en simulation ou en vraie expérience.

Ainsi, il contient des boutons "Run" et "Pause", qui utilisent nos algorithmes de planification de trajectoire pour contrôler l'ensemble des drones. Pour pouvoir les réorganiser manuellement, il permet de contrôler les drones individuellement lorsqu'ils sont en pause. Il permet également de sélectionner les drones qui seront utilisés, et de les éteindre si nécessaire. Pour fonctionner de manière fluide, l'outil exécute le code de planification de la trajectoire dans un thread séparé de l'interface.

Pour fonctionner en simulation, l'outil permet de sélectionner le nombre de drones utilisés et il dispose d'un bouton pour lancer gazebo [7], l'environnement de simulation, ce qui se fait selon un processus différent de celui de l'interface. Une fois le gazebo lancé, tout le reste se fait exactement comme pour les vrais drones.

6.2.3 • Visualisation dans rviz

Grâce à rviz, un outil a été créé pour visualiser les drones (en simulation et en réel) et leurs trajectoires respectives, afin que nous puissions analyser plus simplement le comportement du drone avec notre code et donc le *debugger* plus facilement.

Ainsi, par la création d'un *publisher*, les positions des drones ont été publiées afin qu'elles soient reçues par le rviz et visualisées. De même, les trajectoires et les obstacles sont publiés. Pour contrôler la quantité des drones, le bouton de *add* un drone l'ajoute à la vue, ainsi que sa trajectoire ultérieure. Le bouton de *remove* le drone l'efface du rviz (et sa trajectoire).

Comme pour la configuration de rviz, il est nécessaire de placer les *topics* et le *fixed framed* afin d'obtenir la visualisation. Pour qu'il ne soit pas nécessaire de l'ajouter à chaque fois, il a été enregistré dans un fichier comme l'est le rviz.

6.3 Système de vision

Une des premières étapes du projet avant de chercher les trajectoires optimales pour chacun des drones était de bien décrire l'espace et le milieu de travail dans lequel évolueront nos drones. Il s'agit d'être en mesure de détecter l'emplacement et la position des obstacles que pourront rencontrer les drones avec le plus de précision possible puis de dessiner une image 3D de l'espace correspondant qui servira comme input pour la partie « planification de trajectoire ».

Pour se faire, on dispose actuellement d'un système de vision basé sur l'utilisation de 4 caméras « Tracking RealSense D435i » [8], chacune orientée de façon appropriée afin de couvrir les 4 faces des milieux, équipés de capteurs de profondeur.

Les caméras de profondeur renvoient ainsi chacune une image du milieu coloré en fonction de la profondeur de ses points (voir les figures 4 et 5 ci-dessous. Cela aide, après lecture des données de différentes images, à reconstruire l'espace en prenant en compte les quatre points de vue considérés. Les données de la caméra aident dans ce sens, dans la discrétisation de l'espace accessible aux drones. Notons toutefois qu'il y'a encore des imprécisions en ce qui concerne la position de certains obstacles et que cette dernière, vu l'incertitude sur les profondeurs des objets données par la caméra, demeure approximative.

Figure 4 – Image fournie par la caméra

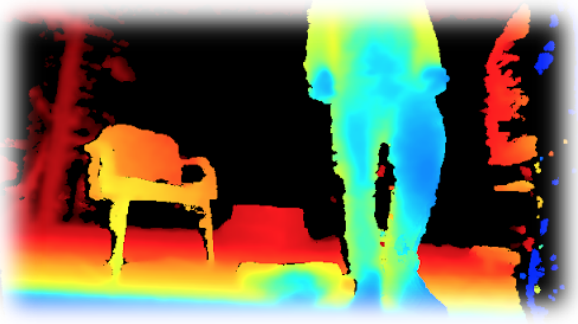


Figure 5 – Dégradé des couleurs en fonction de la profondeur

6.4 Discrétisation

Étant donné que la conception initiale du code du drone a été réalisée en Python 2 et qu'une mise à jour du code impliquerait des modifications importantes du simulateur déjà fait, il a été décidé de poursuivre le développement en Python 2. Malheureusement, il n'a pas été possible de trouver une bibliothèque Python 2 qui soit capable de générer des *meshes* et d'effectuer en même temps les opérations souhaitées. Il fallait produire un *mesh* à partir d'un domaine implicite dans le format de la salle et effectuer des opérations booléennes pour soustraire les cylindres qui représentaient les obstacles.

Il est à noter ici que le code de la plupart des bibliothèques, même en Python 3, qui calculent les *meshes* sont en fait des *Wrappers* de bibliothèques C++, puisque le processus de calcul des *meshes* est intense en calcul. En outre, il existe de nombreuses implémentations bien établies en C++.

Il a donc été décidé de créer un *Wrapper* pour Python 2 à partir de la bibliothèque CGAL [4]. Le principal outil utilisé à cette fin était le SWIG [9], qui génère l'interface entre codes de différents langages et le C++. À partir de SWIG [9] et de distutils [10], il a été possible de compiler un code en C++ qui reçoit un ensemble de cylindres (décrits par un point, un axe et un rayon) et les dimensions de la salle et calcule un *mesh*. Le résultat de la compilation est une bibliothèque partagée (.so), qui, avec un fichier Python d'interface, peut être importée directement par le reste du code.

Pour évaluer l'efficacité des *meshes*, un parallélépipède tridimensionnel représentant la salle a été discrétisé par la méthode uniforme et par des *meshes*. Après la discrétisation, 400 points différents ont été pris au hasard dans chaque espace discrétisé et la taille du plus petit chemin obtenue par l'algorithme A* a été comparée à la distance euclidienne entre les points initial et final. Pour la discrétisation uniforme, les trajectoires étaient 47% plus grandes que les trajectoires obtenues par la distance entre les points. Pour le *mesh*, l'augmentation n'a été que de 9 %.

Ensuite, nous avons procédé à l'optimisation des paramètres des *meshes*. La méthode Nelder-Mead a été utilisée et les paramètres de face et de arêtes décrits dans CGAL [4] ont été optimisés. Après cette procédure, une discrétisation a été obtenue avec une augmentation moyenne de la taille des trajectoires de 8%.

La figure 6 ci-dessous montre une vue de la surface d'un *mesh*, montrant l'obstacle cylindrique et la figure 7 ci-dessous montre un vue de l'ensemble du *mesh*.

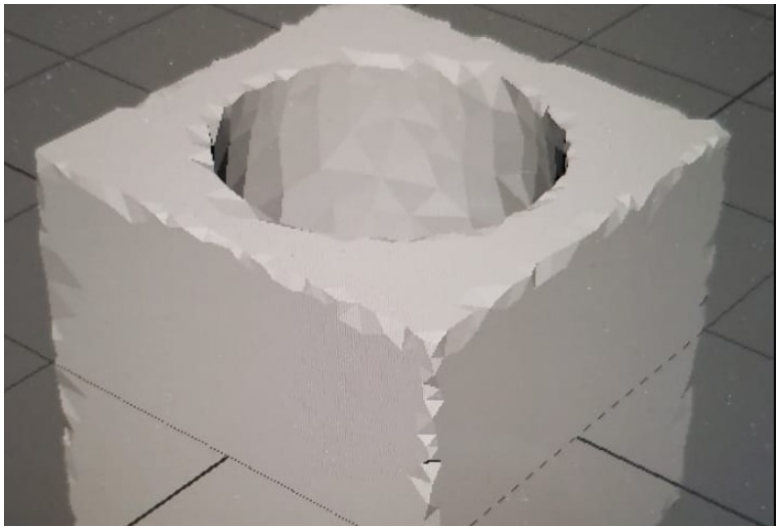


Figure 6 – Vue de la surface d'un *mesh* généré : un le trou cylindrique représente un obstacle.

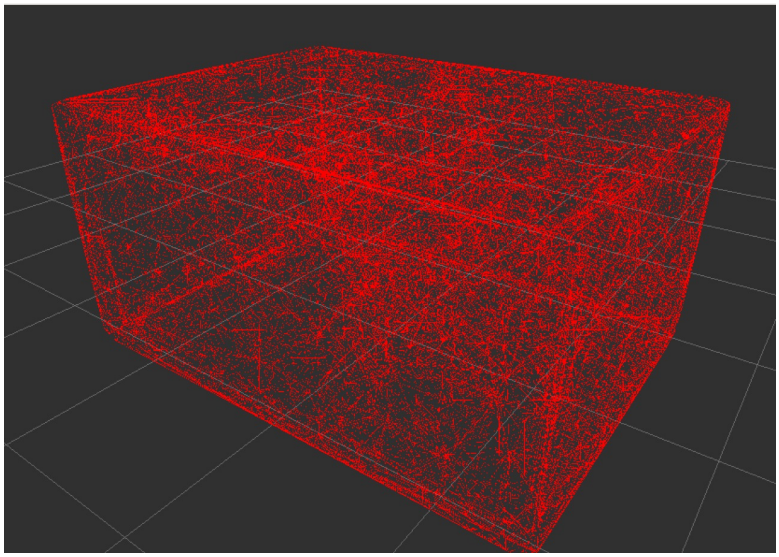


Figure 7 – Vue d'ensemble d'un *mesh* généré (des arêtes intérieures et extérieures).

6.5 Algorithme de Coordination

Nous avons utilisé l'algorithme décrit dans GRIST R. et al. [3], avec quelques adaptations pour simplifier le code.

Nous avons créé une classe en Python, `Coordinator`, avec la méthode `coordinate(paths)`, qui reçoit un dictionnaire `paths` avec des chemins pour chacun des drones individuellement, et renvoie un autre dictionnaire avec des trajectoires coordonnées. La variable `paths` que cette méthode reçoit est créée avant son appel, en faisant un A* pour chaque drone dans le maillage

décrit dans la section TODO.

6.5.1 • Prétraitement

La première partie de l'algorithme est la création d'un objet de la classe `Delta`. Cet objet gardera une matrice avec tous les $\Delta_{i,j}$, qui sont des listes d'objets de la classe `Intersection`.

Nous avons décidé de définir les intersections comme un intervalle de deux valeurs sur le premier trajet et un intervalle sur le second trajet. Chacun de ces valeurs sont une distance dans la trajectoire paramétré par la distance parcourue. Ces intervalles représentent des régions dont la distance minimale à l'autre segment est inférieure à une valeur constante ξ. Cela signifie que les intersections seront toujours des rectangles dans les projections de l'espace de coordination. Nous avons opté pour ce choix car il simplifie considérablement l'algorithme.

L'algorithme 1 montre le calcul des projections $\Delta_{i,j}$. Il analyse toutes les paires de segments dans $\Gamma_i$et Γ_j et crée des intersections si elles existent. À la fin, il combine toutes les intersections créées.

Algorithme 1: Calcul de $\Delta_{i,j}$.

$\Delta_{i,j} \leftarrow \varnothing$
$d_i \leftarrow 0$
$d_j \leftarrow 0$
for $s_i \in Segments(\Gamma_i)$ **do**
  for $s_j \in Segments(\Gamma_j)$ **do**
    if `minimum_distance`$(s_i, s_j) < \xi$ **then**
      $\overline{s_i} =$ `interval_near`$(s_i, s_j) + d_i$
      $\overline{s_j} =$ `interval_near`$(s_j, s_i) + d_j$
      $\Delta_{i,j} \leftarrow \Delta_{i,j} \cup$ `Intersection`$(\overline{s_i}, \overline{s_j})$
    end
  end
end
$\Delta_{i,j} \leftarrow$ `merge`$(\Delta_{i,j})$

Pour créer ses intersections, il utilise calcule d'abord l'intervalle dans le segment s_i, $\overline{s_i}$, et ensuite l'intervalle sur le segment s_j, $\overline{s_j}$, et il ajoute la décalage jusqu'au début de la trajectoire.

L'idée de la fonction `interval_near` est montré dans l'algorithme 2. Il contient une approche de recherche dichotomique pour le calcul du premier point des intervalles, effectuée dans $O(l \cdot log(\epsilon^{-1}))$, où l est la longueur du segment et ϵ est une petite valeur de précision. Le deuxième point des intervalles utilise un algorithme similaire.

Cet algorithme utilise le fait que le graphique des distances minimales au segment s_j, pour chaque point du segment s_i, est monotone ou peut être divisé en deux parties monotones avec un minimum au milieu. Par conséquent, l'algorithme recherche dans ce graphique une région

Algorithme 2: Recherche dichotomique du premier point d'un intervalle d'intersection.

```
l ← s_i.start
r ← s_i.end
while distance(l, r) > ϵ do
    m ← (l+r)/2
    if minimum_distance(s_j, m) < ξ then
        r ← m
    else
        m_l ← m − (r − l) · ϵ
        m_r ← m + (r − l) · ϵ
        if minimum_distance(s_j, m_l) < minimum_distance(s_j, m_r) then
            r ← m_l
        else
            l ← m_r
        end
    end
end
return (l+r)/2
```

où la distance est inférieure à xi, et une fois qu'il a trouvé cette région, il en recherche la limite gauche.

Pour la combinaison des intervalles, dans la fonction `merge`, il est nécessaire d'analyser chaque paire d'intersections et de vérifier si elles doivent être fusionnées. Elles ne doivent être fusionnées que si les intervalles des deux trajectoires se croisent. Si c'est le cas, l'intersection fusionnée est l'union des intervalles en chaque trajectoire. Le calcul total sera effectué en $\mathcal{O}(n^2)$, où n est le nombre d'intersections.

Par conséquent, l'algorithme 1 est $\mathcal{O}(n_i n_j L \cdot log(\epsilon^{-1}))$, où n_1 est le nombre de segments dans Γ_i, n_j est le nombre de segments dans Γ_j et L est la longueur maximale d'un segment.

La partie finale du prétraitement est le calcul des orientations de chaque intersection. Pour ce calcul, nous avons implémenté un algorithme que trouve un chemin non-synchronisé avec les autres drones, représenté par le chemin rouge dans la Figure 1. Cet algorithme n'est pas spécifié dans l'article.

Pour construire ce chemin, ont été considérés comme des nœuds pour déterminer le chemin, ceux correspondant aux extrémités des obstacles, l'origine et le but des drones. Ainsi, un A* a été mis en place pour déterminer le plus petit chemin de l'origine à l'objectif. La distance à A* entre un nœud et un autre a été considérée comme le maximum entre la distance projetée sur l'axe x et l'axe y de ce segment, car elle est plus appropriée si l'on considère que l'on veut minimiser le temps total.

En considérant i le nombre de intersections existants, la complexité de A* est $O(i^3)$, puisque,

pour chaque pair de chemins, on a, au $O(i)$ noeuds et, donc, $O(i^2)$ arêtes possibles. Donc, pour trouver le A* (lequel a la complexité de O(arêtes)) pour chaque pair, on prend $O(i^2)$. Comme on a, au maximum, i pairs pour trouver ce chemin rouge, la complexité totale est $O(i^3)$.

Ensuite, on suit les événements, tels que l'intersection de deux trajectoires et la sortie du voisinage de l'intersection, et on modifie les vitesses lorsque les évènement se produisent. Pour choisir le drone qui s'arrête ou qui revient, nous suivons les mêmes orientations que la trajectoire bleue. De plus, pour identifier l'orientation des obstacles par rapport à la trajectoire suivie, on crée une ligne qui va en haut. Si elle rencontre la trajectoire avant l'axe, l'obstacle est en CW (sens horaire). Sinon, c'est CCW (sens anti-horaire).

C'est pourquoi la complexité du calcul de Δ peut être écrite comme $\mathcal{O}(i^3 + N^2 n^2 L \cdot log(\epsilon^{-1}))$, où i est le nombre d'intersections des trajectoires, N est le nombre de drones et n est le nombre maximum de segments dans une trajectoire.

6.5.2 • Méthode Itératif

Nous allons suivre la même idée de GRIST R. et al [3] pour le calcul des trajectoires coordonnés, avec une simplification à cause de la définition des intersections comme des rectangles.

De cette façon, en partant d'un point en $\mathcal{X}$ où chaque drone se trouve à son point de départ, nous calculerons la prochaine vitesse maximale et le temps pendant lequel cette vitesse doit être maintenue. Ensuite, l'étape suivante de l'itération est le point si nous nous déplaçons avec cette vitesse pendant ce temps donné. Cette boucle se déroule jusqu'à ce que tous les drones aient atteint leur destination.

Pour calculer la vitesse maximale à chaque étape, notre code ne fait pas une solution de programmation linéaire. Comme les intersections sont des rectangles, à chaque point, un drone n'a que deux options : se déplacer avec la vitesse maximale ou s'arrêter. Si les intersections avaient d'autres formes, nous devrions ajuster la vitesse d'un drone en fonction des autres, afin qu'il se déplace sur le contour de cette intersection. Notez que nous montrons ici le désavantage de décrire les intersections comme des rectangles : lorsque deux drones en arrivent à une, un des drones devra attendre que l'autre passe par toute la région pour recommencer à se déplacer.

Il suffit donc de détecter les drones qui sont arrêtés. Cela peut se faire en vérifiant, à chaque étape, s'il y a un obstacle dans le sens horaire à droite (ou un prolongement de celui-ci, pour éviter de changer son orientation), ou un obstacle dans le sens anti-horaire en haut (ou un prolongement de celui-ci). La figure 8 montre la variation des vitesses dans un système à trois drones. Notez que les trois graphiques sont liés : dans le premier graphique, le premier drone s'arrête à environ 0.2 mètres à cause de l'obstacle dans le second graphique.

Pour calculer combien de temps la vitesse doit être maintenue, nous définissons les événements comme le fait d'atteindre un obstacle (ou sa projection), d'atteindre la fin de la trajectoire ou de sortir du bord d'un obstacle. À chaque itération, l'algorithme analyse tous les événements futurs qui se produiront si cette vitesse est maintenue indéfiniment, et choisit celui qui est le plus proche dans le temps.

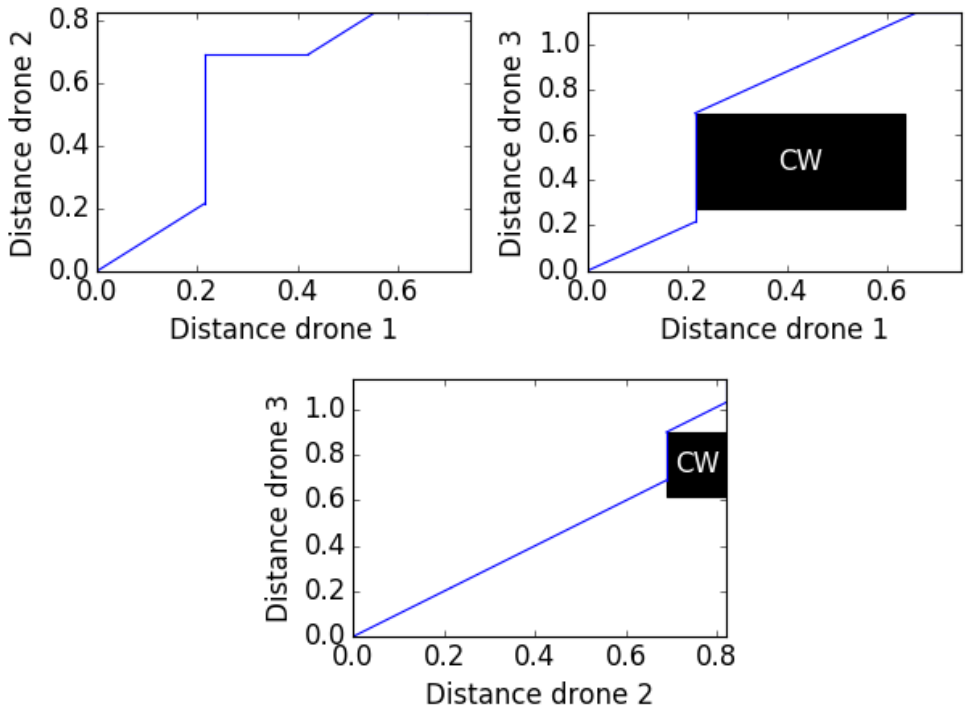


Figure 8 – $\mathcal{X}$, l'espace de coordination pour une trajectoire de trois drones, paramétré par la distance parcourue dans chaque trajectoire.

La complexité totale du processus itératif est de $\mathcal{O}(Ni^2)$, où N est le nombre de drones et i est le nombre d'intersections. Cela vient de la complexité du calcul de la vitesse maximale et de l'événement suivant, qui est $\mathcal{O}(Ni)$, et il peut y avoir au maximum $\mathcal{O}(i)$ étapes dans la boucle.

La figure 9 montre le mouvement calculé dans la figure 8.

La deuxième image représente le moment où les drones bleu et rose rencontrent la première intersection, où le drone bleu s'arrête. Sur la troisième image, le drone bleu se déplace déjà à nouveau, et le drone jaune s'arrête à cause de l'intersection avec le drone rose. Sur la dernière image, tous les drones ont atteint leur destination.

6.6 Trajectoires polynomiales

Notre objectif est de faire voler un "essaim" de drones. On a N drones partant d'une même zone restreinte initiale, et on souhaite qu'ils atteignent une zone finale restreinte prédéfinie, en un minimum de temps et en prenant en compte les obstacles statiques ou en mouvement sur leur parcours.

Si l'approche discrète a été privilégiée, en discrétisant l'espace de coordination puis en

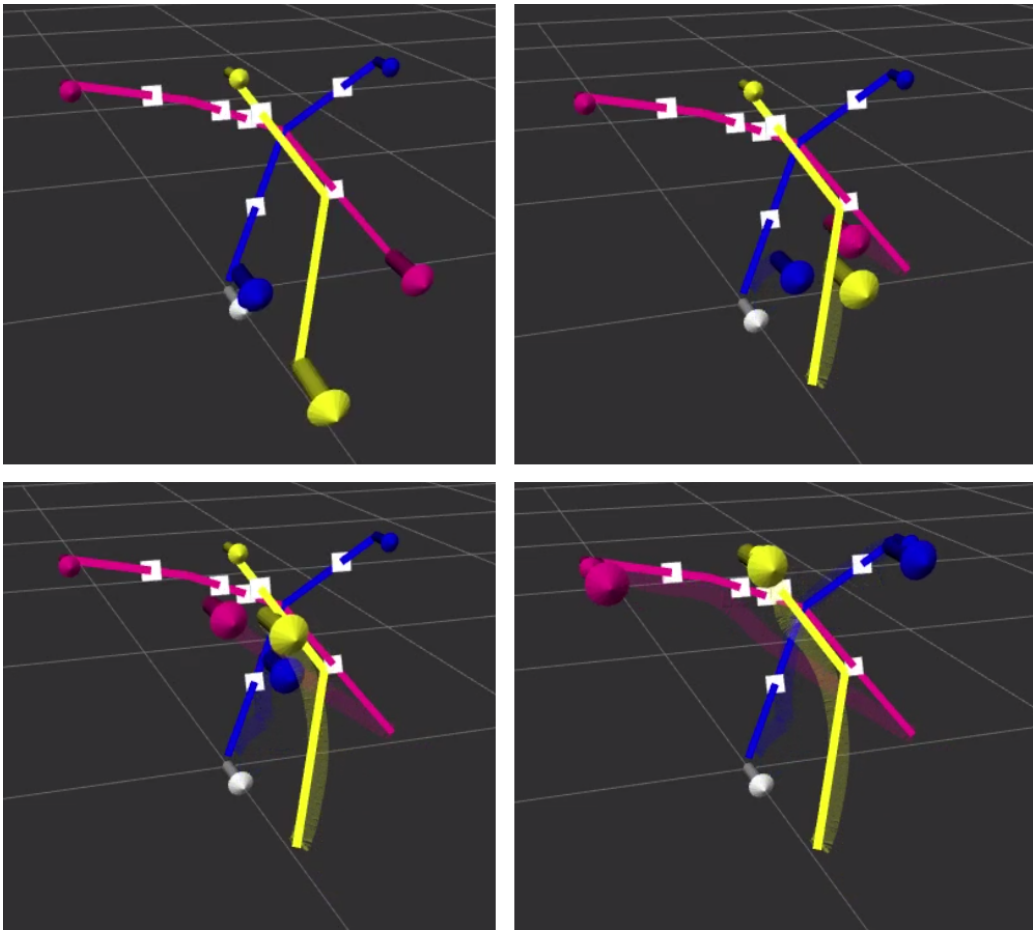


Figure 9 – Le mouvement coordonné de trois drones calculé par notre algorithme et simulé sur Gazebo. Les images sont dans l'ordre de haut en bas et de gauche à droite. Le premier drone est bleu, le deuxième est jaune et le troisième est rose. Les grandes flèches représentent les drones et les petites flèches représentent la position du but. Les points blancs dans la trajectoire calculée représentent les régions d'intersection.

obtenant les trajectoires des drones dans cet espace, certains documents trouvés par nos propres moyens évoquaient également la possibilité de représenter les trajectoires de ces drones en employant des polynômes dépendant du temps et à valeurs dans $\mathbf{R^3}$.

De nombreuses contraintes s'appliquent aux trajectoires possibles : les états initiaux et finaux des drones, les obstacles à éviter, les conditions sur les drones (vitesse, accélération, vitesse angulaire, etc...). De plus lorsqu'on souhaite faire déplacer un essaim de drones, il faut considérer chaque drone de l'essaim comme un obstacle pour tous les autres. On a alors un problème d'aspect plus combinatoire puisque la trajectoire de chacun des drones dépend directement de celle des autres.

Si plusieurs approches ont été étudiées, aucune n'ont présentées de moyens suffisamment probant et efficace pour être implémentée, le travail de documentation ayant été particuliè-

rement ardu. On présente tout de même les quelques résultats obtenus, ainsi que le principe général d'une telle démarche.

6.6.1 • Approche Par B-Splines

L'objectif d'optimisation peut s'exprimer comme un problème de programmation non linéaire. De manière naïve, on peut chercher un polynôme de degré fixé dépendant du temps, et dont on fait varier les coefficients, tel que ce polynôme respecte toutes les contraintes relatives aux caractéristiques du drone et de l'objectif fixé. On cherche un polynôme P qui minimise T la valeur en laquelle on atteint la position finale.

Soit K l'ensemble des possibilités pour les coefficients du polynôme candidat. On cherche un système de la forme :

$$\begin{aligned} &\min_{x\in K} \quad J(x) \\ &A(x) = 0, \quad \forall x \in K \\ &B(x) \leq 0, \quad \forall x \in K \end{aligned}$$

On appelle J la fonction objectif, qui donne T en fonction des coefficients du polynôme dépendant du temps et représentant la trajectoire. Les fonctions A, B sont à valeurs dans R^p, R^q et représente respectivement les contraintes sur les positions, vitesses et accélérations initiales et finales, et les contraintes sur la position (i.e. éviter les obstacles), la vitesse et l'accélération (qui sont bornées).

Toujours naïvement, le problème d'optimisation s'effectuera sur la variable $(T, a_0, ..., a_n)$ dans $\mathbb{R} \times \mathbb{R}^{3^{n+1}}$ pour un polynome P de degré n représentant la trajectoire tel que :

$$P(t) = a_0 + a_1 t + ... + a_n t^n, \ \forall t \in \mathbb{R}$$

Pour un point de départ pos_1, un point d'arrivée pos_2 et un ensemble $\mathcal{O} \subset \mathbb{R}^3$ représentant les obstacles à éviter, le problème se formalise sous la forme :

$$\begin{cases} \min \quad T \\ T > 0 \\ P(0) = pos_1 \\ (P'(0), P''(0)) = 0 \\ P(T) = pos_2 \\ (P'(T), P''(T)) = 0 \\ P(t) \notin \mathcal{O}, \quad \forall t \in [0, T] \end{cases}$$

Un problème majeur ici concerne la garanti d'obtenir le minimum global et pas seulement un minimum local. Pour cela on doit imposer des conditions sur la convexité des contraintes et de l'ensemble d'étude (Kuhn-Tucker, voir [11]).

Or les contraintes relatives aux obstacles à éviter ne sont pas convexes, et on ne peut pas se restreindre à l'étude d'un ensemble convexe des coefficients $a_1, ..., a_n$. Limiter les valeurs d'un

polynôme dans un ensemble spécifique, fut-il même convexe, en fonction de ses coefficients, ne peut pas être une contrainte convexe. On décide donc de changer de variables en utilisant une forme d'interpolation, les B-splines.

Ceux-ci sont définis par des paramètres $0 \leq t_0 \leq t_1 \leq \ldots \leq t_m \leq 1$ et des points dans l'espace, noté $P_i, i \in \{0, ..., m-n-1\}$ appelés points de contrôles. Pour de tels paramètres, on définit le B-spline de degré n tel que :

$$\mathbf{P}(t) = \sum_{i=0}^{m-n-1} b_{i,n}(t).\mathbf{P}_i \, , \; t \in [0,1]$$

Pour :

$$b_{j,0}(t) := \begin{cases} 1 & si \quad t_j \leqslant t < t_{j+1} \\ 0 & sinon \end{cases}$$

$$b_{j,n}(t) := \frac{t-t_j}{t_{j+n}-t_j} b_{j,n-1}(t) + \frac{t_{j+n+1}-t}{t_{j+n+1}-t_{j+1}} b_{j+1,n-1}(t).$$

On divise le problème en deux parties distinctes : d'une part trouver le chemin dans l'espace, d'autre part trouver la vélocité du drone sur ce chemin. Le principe de cette démarche est de construire deux B-splines, l'un noté P à valeur dans $[0,1]$ pour le chemin, l'autre noté λ à valeur dans $[0,T]$ pour la vélocité. Les paramètres $(\mu_0, ...\mu_m, P_0, ..., P_{m-n-1})$ et $(\gamma_0, ...\gamma_m, Y_0, ..., Y_{m-n-1})$ définissant ces splines forment avec T la variable d'étude de notre système d'optimisation.

L'intérêt ici est que les valeurs du B-spline représentant le chemin seront contenus dans l'enveloppe convexe de ses points de contrôles. Ainsi, en contraignant les points de contrôles à appartenir à un ensemble convexe dépendant des obstacles pour P, on obtient un polynome également dans cet ensemble convexe.

6.6.2 • B-Spline De Chemin

On peut exploiter ce caractère optimal sur un convexe pour construire un chemin constitué de polynômes par morceau, chacun optimal sur un convexe prédéterminé, tel que la suite de ces convexes dans l'espace forme un corridor de la position initiale à la position finale, et assure l'absence de collision avec des obstacles. Cette suite de convexe dans l'espace dépendra des obstacles à prendre en compte, pour l'instant considérés immobiles. En choisissant arbitrairement que ces convexes soit des polyèdres, le corridor sera lui aussi constitué de polyèdres formés à partir des arrête des obstacles à éviter.

On a donc la liste de nos paramètres à optimiser : pour des splines de degré n, on a m valeurs de temps et m-n-1 points de contrôles par polyèdres convexe du corridor, ainsi qu'un point par surface adjacente à deux polyèdres du corridor par lequel chacun des polynômes de ces deux polyèdres se rejoigne.

On fait face à une première limite de la méthode : la présence des points de contrôle dans un convexe n'est pas une condition nécessaire à l'existence d'un spline dans ce convexe. Toutes les

possibilités de trajectoires ne sont pas explorées, et celle obtenue n'est pas forcément la meilleure possible. Par ailleurs, tout comme il existe plusieurs manières de contourner un obstacle, il existe plusieurs chaînes de corridor possibles, chacune offrant un minimum local, potentiellement global. Il faut donc énumérer les possibilités ou choisir une manière bien précise de contourner les obstacles.

Par ailleurs, pour chaque polyèdre, il est nécessaire de respecter la convexité sur les contraintes. Par conséquent les m valeurs de temps du spline ne peuvent pas être variables du système d'optimisation. Il faut donc les fixer arbitrairement pour chaque polyèdre. Ce n'est ici pas un problème, et on peut arbitrairement les répartir uniformément entre 0 et 1 pour chaque polyèdre. La même spécificité sera rencontrée pour l'optimisation de la vélocité, et nous obligera à restreindre les possibilités de trajectoires. On constate qu'une fois les valeurs de temps fixés, les seuls variables du système sont les points de contrôles, dont le spline dépends de manière linéaire.

6.6.3 • Optimisation De Vélocité

En ce qui concerne l'optimisation de la vélocité sur un chemin fixé, on pourrait croire qu'il n'est plus nécessaire d'utiliser des B-splines puisque les seules contraintes sur le polynome informant la vélocité sont celles, physique, du drone : vitesse et accélération initiale, finale, et maximale sur le chemin.

On a donc commencé par étudier un système d'optimisation pour un polynome λ de degré 5 dont la variable est (T, a_3, a_4, a_5) et qui respecte :

$$\lambda(t) = a_3t^3 + a_4t^4 + a_5t^5, \forall t \in \mathbb{R}$$

L'usage d'un tel polynome permet immédiatement de fixer les contraintes initiales de temps, vitesse et accélération, telle que :

$$\lambda(0) = 0, \quad \lambda'(0) = 0, \quad \lambda''(0) = 0$$

On pose le système comme suit :

$$\begin{cases} \min \quad T \\ T > 0 \\ \lambda(T) = 1 \\ (\lambda'(T), \lambda''(T)) = 0 \\ \lambda'(t) \geq 0, \quad \forall t \in [0, T] \\ (\lambda'(t), \lambda''(t)) \leq cstr(\lambda(t)), \quad \forall t \in [0, T] \end{cases}$$

Pour *cstr* la fonction indiquant le maximum de vélocité et d'accélération en fonction du chemin P préalablement fixé.

On préfère éviter de traiter des contraintes d'égalité, les conditions sur ces contraintes pour obtenir un maximum global étant peu accommodantes (contraintes qualifiées, indépendantes

entre elles). Pour cela, on exprimera les contraintes d'égalité comme plusieurs contraintes d'inégalité.

La première contraintes est linéaire, donc convexe. Les deux contraintes suivantes sont des égalités exprimées sous forme de polynomes dépendant de T. Elles ne sont pas linéaires en fonction de T, et il a été difficile de traduire ces contraintes en inégalités.

On peut exprimer les dernières contraintes sous forme d'intégrales, par exemple la contrainte de dérivée positive peut s'écrire :

$$\int_0^T \lambda'(t)^2 \mathbb{1}_{\{\lambda'(t)\leq 1\}} dt \leq 0$$

Cependant cette contrainte n'est pas convexe par rapport à la variable T.

On a donc de nouveau recours aux splines pour vérifier nos hypothèses de convexités. Il est nécessaire de fixer les m valeurs de temps pour respecter les hypothèses de convexité, et on impose que pour chaque point de contrôle Y_i on ai :

$$Y_i \in [\tfrac{i-1}{m-n-1}; \tfrac{i}{m-n-1}], \quad \forall i \in \{1, ..., m-n-1\}$$

Pour garantir la croissance de λ.

On a alors un système d'optimisation de variable $(T, Y_1, ..., Y_n)$ tel que :

$$\begin{cases} \min \quad T \\ T > 0 \\ (\lambda(0), \lambda'(0), \lambda''(0)) = 0 \\ \lambda(T) = 1 \\ (\lambda'(T), \lambda''(T)) = 0 \\ Y_i \geq \frac{i-1}{m-n-1}, \quad \forall i \in \{1, ..., m-n-1\} \\ Y_i \leq \frac{i}{m-n-1}, \quad \forall i \in \{1, ..., m-n-1\} \\ (\lambda'(t), \lambda''(t)) \leq cstr(\lambda(t)), \quad \forall t \in [0, T] \end{cases}$$

Cependant, avoir fixé les variables de temps γ_j a pour conséquence la certitude que la valeur T minimisé respectera :$T \geq \gamma_{m-n-1}$. Il faut donc avoir les variables de temps γ_j suffisament proche de 0 car dans le cas contraire le résultat pourrait ne pas être optimal. Cependant conserver trop proche de 0 ces valeurs peut empêcher la résolution du système puisque les contraintes sur la vitesse et l'accélération maximale pourrait ne pas être respectée.

Par ailleurs il reste le problème de convexité des contraintes sur $\lambda(T)$, qui ne sont pas convexes par rapport à T.

Il reste encore à traiter la dernière inégalité. Plutôt que d'exprimer trop lourdement de manière formelle les contraintes (théoriques) des drones en fonction du chemin emprunté, il avait été envisagé de chercher le minimum des valeurs maximales de vitesse et d'accélération sur le chemin, qui s'exprimerait linéairement en fonction des points de contrôles de ces chemins. En raison du confinement, cela n'a pas pu être effectué.

6.6.4 • Conclusion

L'objectif était de combiner cette optimisation sur le chemin et la vélocité pour obtenir une trajectoire optimale. En l'absence d'expression tangible des contraintes du chemin sur la vitesse et l'accélération du drone, un algorithme complet n'a pas été écrit. On en a été également dissuadé par le manque de rigueur du résultat qui aurait été obtenu, puisqu'obtenir un optimum global n'est pas garanti. Malgré une recherche intensive d'alternative dans la documentation accessible sur internet et des tentatives de notre coté, il n'a pas été possible de trouver un méthode efficace d'être garanti d'obtenir un résultat probant.

Cette approche était pourtant attrayante car elle aurait pu apporter un soutien et des éléments de comparaison avec la méthode discrète. En particulier, il aurait été possible d'étudier les différent optimum locaux de chaque drone pour envisager des trajectoires différentes en fonction des risque de collision.

7
CONCLUSION

À travers ce travail, nous avons réussi à implémenter un algorithme de coordination entre les drones qui leur a permis d'opérer dans un milieu ensemble, sans collisions. Notons toutefois qu'on a considéré plusieurs restrictions (trajectoires linéaires par morceaux, obstacles fixes ...) pour en aboutir. Cela a simplifié la tâche mais qui demeure nécessaire pour une bonne compréhension du phénomène et qui constitue une étape primordiale avant d'attaquer le problème de manière générale.

Nous avons pris un environnement de contrôle de drones existant et nous l'avons amélioré, afin que la poursuite de ce travail puisse se faire plus facilement à l'avenir. Nous avons également mis en place des outils d'aide au développement d'algorithmes et à leur *debugging*, avec *rviz* et un contrôleur de l'ensemble de drones.

Plus important encore, nous avons calculé la trajectoire optimale pour chaque drone a partir d'une disrétisation de l'espace, et conçu un algorithme itératif exploitant la coordination entre drones, en changeant leurs vitesses pour éviter des collisions.

L'autre volet de notre travail consistait à déterminer des algorithmes pour trouver de meilleures chemins pour l'ensemble des drones. Celle ci n'était pas basé sur la même démarche et nous avons pu obtenir des théories qui demandent des contraintes supplémentaires et reposent sur des optimisations beaucoup plus compliquées.

Enfin, terminons par la suggestion de pistes d'études pour ceux voulant poursuivre notre travail. Dans ce projet, nous n'avons traité que le cas d'obstacles fixes ou ceux dont la trajectoire est connue au préalable (donc assimilé à un drone), le traitement des cas d'obstacles mouvants fait appel à la programmation de systèmes auto-adaptatifs. Ce dernier point soulève aussi la possibilité de mettre en place un système de vision qui pourra être une caméra intégré aux drones. Chose qui rendra l'optimisation en temps réel au lieu d'une optimisation faite en avance.

RÉFÉRENCES

[1] *Amazon Prime Air*. URL : https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011.

[2] *Crazyflie 2.1 | Bitcraze*. URL : https://www.bitcraze.io/crazyflie-2-1/.

[3] Robert Ghrist, Jason M. O'Kane et Steven M. LaValle. "Computing Pareto Optimal Coordinations on Roadmaps". In : *The International Journal of Robotics Research* 24.11 (2005), p. 997-1010. DOI : 10.1177/0278364905059057. eprint : https://doi.org/10.1177/0278364905059057. URL : https://doi.org/10.1177/0278364905059057.

[4] *CGAL 4.13 - 3D Mesh Generation: User Manual*. URL : https://doc.cgal.org/4.13/Mesh_3/index.html.

[5] *rqt - ROS Wiki*. URL : http://wiki.ros.org/rqt.

[6] *rviz - ROS Wiki*. URL : http://wiki.ros.org/rviz.

[7] *gazebo*. URL : http://gazebosim.org/.

[8] *Depth Camera D435i - Intel RealSense*. URL : https://www.intelrealsense.com/depth-camera-d435i/.

[9] *SWIG*. URL : http://www.swig.org/exec.html.

[10] *27.1. distutils — Building and installing Python modules — Python 2.7.18rc1 documentation*. URL : https://docs.python.org/2.7/library/distutils.html.

[11] Grégoire Allaire. *Analyse numérique et optimisation: une introduction à la modélisation mathématique et à la simulation numérique*. Google-Books-ID: vReEuE4margC. Editions Ecole Polytechnique, 2005. 482 p. ISBN : 978-2-7302-1255-7.